Documentation Technique - Projet CampSwing

Table des matières

1. Introduction

1.1 Présentation du projet

Le projet "Système de Gestion de Camp de Vacances" est une application Java qui permet la gestion complète des activités d'un camp de vacances. Elle offre différentes interfaces adaptées aux besoins spécifiques des Utilisateurs, des responsables d'activités et des administrateurs.

L'application est développée en Java avec une interface graphique Swing et utilise une base de données MySQL pour stocker les informations sur les utilisateurs, les activités, les inscriptions et les calendriers.

1.2 Objectif du projet

Le projet CampSwing est un système de gestion de camp d'activités développé en Java. Il permet la gestion des utilisateurs, des activités, des inscriptions et offre des interfaces distinctes pour trois types d'utilisateurs : les Utilisateurs (utilisateurs standard), les responsables d'activités et les administrateurs.

1.3 Contexte

Le système a été développé pour permettre une gestion efficace des camps d'activités, facilitant l'inscription des Utilisateurs et de leurs enfants aux diverses activités proposées. Le projet a été réalisé avec Eclipse, utilisant Java SE 17 et une base de données MySQL.

1.4 Technologies utilisées

- **Langage de programmation** : Java SE 17
- **IDE** : Eclipse

- **Base de données** : MySQL
- **Interface utilisateur** : Java Sw
- ing (javax.swing.*)
- **Bibliothèques externes** :
    - mysql-connector-j-9.1.0.jar (connecteur JDBC pour MySQL)
    - jbcrypt-0.4.jar (hashage sécurisé des mots de passe)

2. Architecture du projet

2.1 Structure du projet

Le projet est organisé selon une structure classique d'application Java :

```
ProjetJava-Final/
│
├── JRE System Library [JavaSE-17]/
│   └── (Bibliothèques système Java)
│
├── src/
│   └── bdd/
│       ├── DatabaseConnection.java
│   └── dao/
│       ├── ActiviteDAO.java
│       ├── CalendrierDAO.java
│       ├── InscriptionActiviteDAO.java
│       ├── NotificationDAO.java
│       └── UtilisateurDAO.java
│   └── main/
│       ├── Main.java
│   └── model/
│       ├── Activite.java
│       ├── Calendrier.java
│       ├── Evaluation.java
```

```
|       ├── InscriptionActivite.java
|       ├── Notification.java
|       ├── NotificationService.java
|       ├── Utilisateur.java
|   └── view/
|       ├── FenetreAdmin.java
|       ├── FenetreCalendrier.java
|       ├── FenetreConnexion.java
|       ├── FenetreGestionActivites.java
|       ├── FenetreInscription.java
|       ├── FenetreInscriptionsActivite.java
|       ├── FenetreNotification.java
|       ├── FenetreNotificationGlobales.java
|       ├── FenetrePrincipale.java
|       ├── FenetreResponsable.java
|       ├── FenetreTicketsIncidents.java
|
├── Referenced Libraries/
|   ├── mysql-connector-j-9.1.0.jar
|   └── jbcrypt-0.4.jar
```

Le code source est organisé en packages distincts :

- **bdd** : Contient DatabaseConnection.java
- **dao** : Objets d'accès aux données pour les opérations sur la base
- **model** : Contient les classes entités (ex : Utilisateur, Activite)
- **view** : Interfaces utilisateurs basées sur Swing

2.2 Architecture en couches

L'architecture du projet suit le modèle en couches pour une meilleure séparation des responsabilités :

1. **Couche présentation** : Interfaces graphiques développées avec Java Swing
    - Fenêtres d'interface utilisateur (classes commençant par "Fenetre...")
2. **Couche métier** : Logique d'affaires et modèles de données
    - Classes d'entités (Utilisateur, Activite, Calendrier, etc.)
    - Services (NotificationService)

3. **Couche d'accès aux données** : Interface avec la base de données
    - Classes DAO (UtilisateurDAO, ActiviteDAO, etc.)
    - DatabaseConnection pour la gestion des connexions

2.3 Bibliothèques externes

2.3.1 MySQL Connector (mysql-connector-j-9.1.0.jar)

Cette bibliothèque fournit le pilote JDBC nécessaire pour connecter l'application Java à la base de données MySQL. Elle permet d'exécuter des requêtes SQL, de gérer les connexions et les transactions avec la base de données.

2.3.2 JBCrypt (jbcrypt-0.4.jar)

JBCrypt est une implémentation Java de l'algorithme de hachage de mots de passe BCrypt. Cette bibliothèque est utilisée pour sécuriser les mots de passe des utilisateurs dans la base de données, en appliquant un hachage fort et salé.

3. Analyse

3.1 User Stories

🛡️ **User Stories - Admin**

1. **Connexion**
    - En tant qu'admin, je veux me connecter à mon interface dédiée afin de gérer les utilisateurs et superviser la plateforme.
2. **Gestion des utilisateurs**
    - En tant qu'admin, je veux voir la liste de tous les utilisateurs afin de pouvoir les gérer efficacement.
    - En tant qu'admin, je veux supprimer un utilisateur afin de révoquer son accès à la plateforme si nécessaire.
3. **Notifications**
    - En tant qu'admin, je veux voir les notifications reçues par un utilisateur afin de m'assurer du bon déroulement de ses activités.
4. **Calendrier**
    - En tant qu'admin, je veux consulter le calendrier des activités afin de avoir une vue globale de la programmation.
5. **Déconnexion**
    - En tant qu'admin, je veux me déconnecter afin de sécuriser mon compte après utilisation.

🎯 **User Stories - Responsable d'Activité**

1. **Connexion**
    - En tant que responsable d'activité, je veux me connecter à mon interface dédiée afin de gérer les activités et les inscriptions.
2. **Gestion des activités**
    - En tant que responsable d'activité, je veux ajouter, modifier ou supprimer une activité afin de maintenir une offre à jour et pertinente.
3. **Calendrier d'activités**
    - En tant que responsable d'activité, je veux ajouter et consulter des événements dans le calendrier afin de organiser les activités avec clarté.
4. **Gestion des tickets d'incident**

- En tant que responsable d'activité, je veux voir les incidents déclarés par les utilisateurs afin de prendre les mesures nécessaires.

5. **Suivi des inscriptions**
   - En tant que responsable d'activité, je veux voir les inscriptions aux activités afin de suivre la participation.
   - En tant que responsable d'activité, je veux valider, refuser ou laisser en attente les inscriptions afin de gérer les accès aux activités.
6. **Déconnexion**
   - En tant que responsable d'activité, je veux me déconnecter afin de sécuriser mon compte après usage.

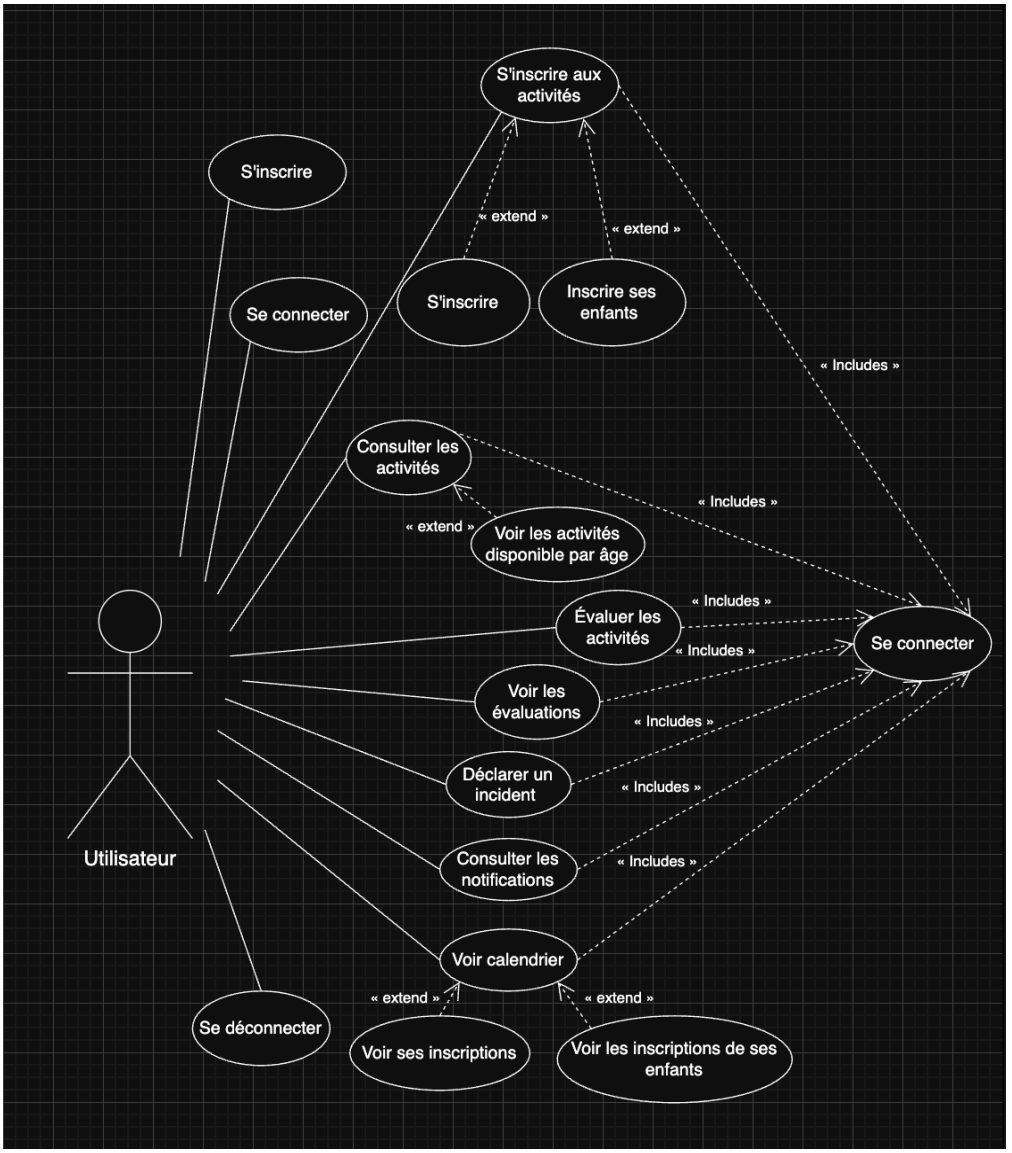
🧒 User Stories - Utilisateur

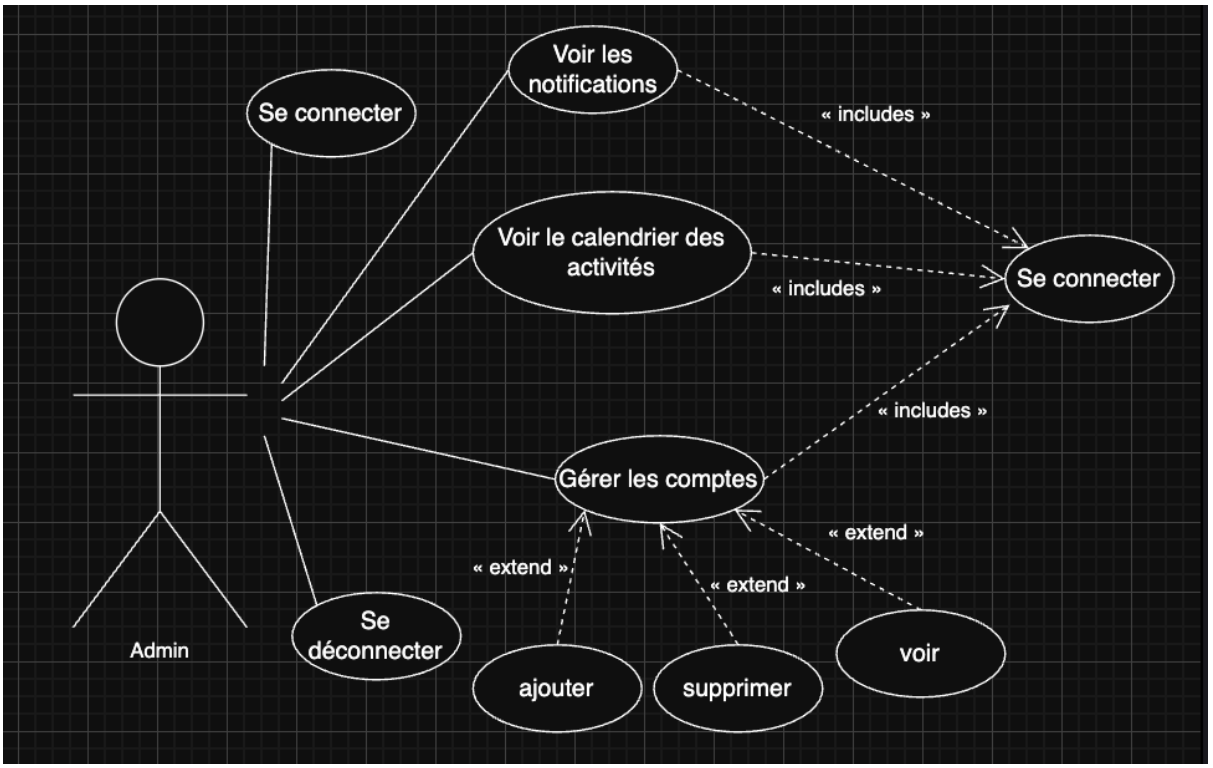
1. **Inscription et connexion**
   - En tant que Utilisateur, je veux m'inscrire et me connecter à mon compte personnel afin de accéder aux fonctionnalités de la plateforme.
2. **Consultation des activités**
   - En tant que Utilisateur, je veux voir les activités disponibles afin de choisir celles qui m'intéressent.
3. **Inscription aux activités**
   - En tant que Utilisateur, je veux m'inscrire à une activité afin de y participer.
   - En tant que Utilisateur, je veux inscrire un ou plusieurs de mes enfants à une activité adaptée à leur âge afin de les faire participer aussi.
4. **Évaluation d'activités**
   - En tant que Utilisateur, je veux évaluer les activités auxquelles j'ai participé afin de donner mon avis.
   - En tant que Utilisateur, je veux voir les évaluations des autres Utilisateurs afin de m'informer sur la qualité des activités.
5. **Calendrier et suivi**
   - En tant que Utilisateur, je veux consulter le calendrier des activités afin de connaître les dates et heures des activités.
   - En tant que Utilisateur, je veux voir les activités auxquelles moi et mes enfants sommes inscrits afin de m'organiser.
6. **Gestion des enfants**
   - En tant que Utilisateur, je veux renseigner le nombre, le nom, le prénom et l'âge de mes enfants afin de les inscrire correctement à des activités adaptées.
7. **Notifications**
   - En tant que Utilisateur, je veux voir les notifications liées à mes inscriptions et celles de mes enfants afin de ne rien manquer.
8. **Déclaration d'incident**
   - En tant que Utilisateur, je veux pouvoir signaler un problème via un ticket afin de alerter l'équipe responsable.

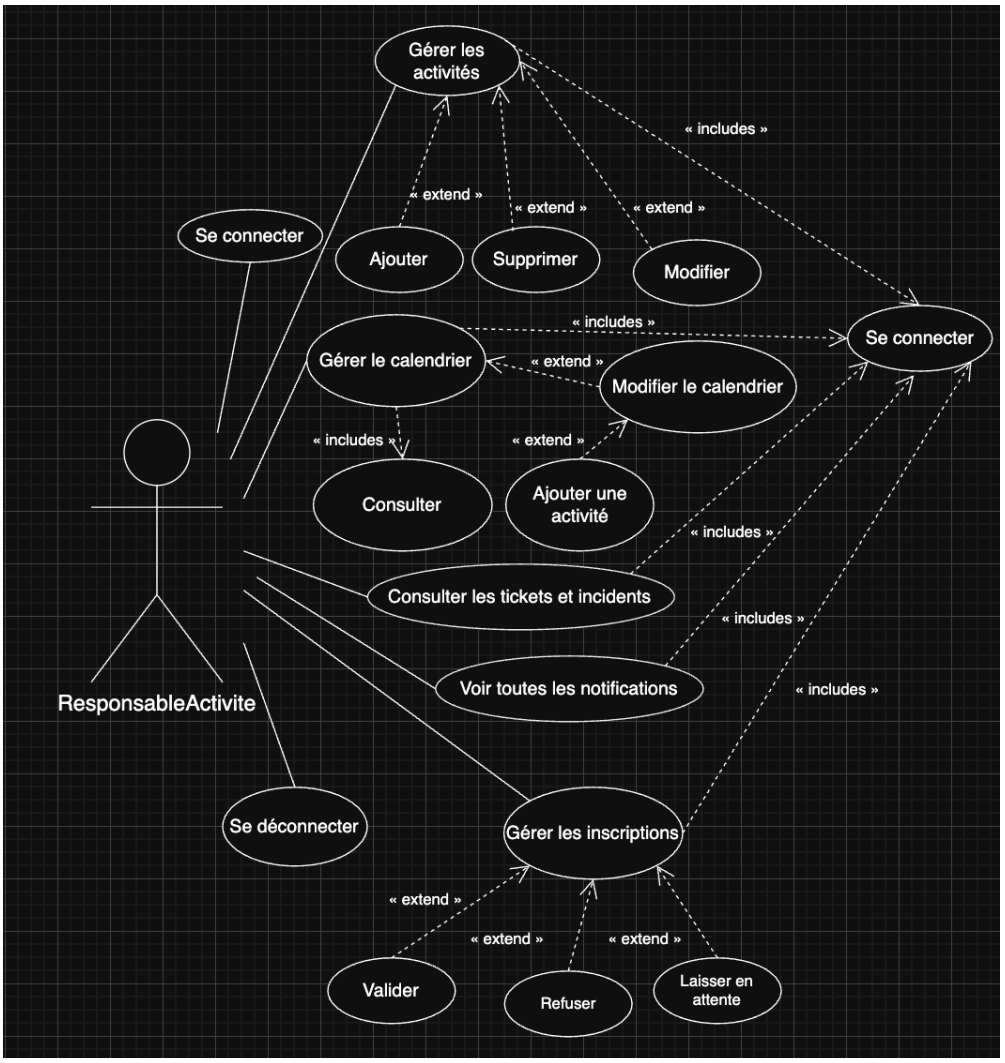
9. **Déconnexion**
   - En tant que Utilisateur, je veux me déconnecter afin de protéger mes informations personnelles.

3.2 Diagramme UML des cas d'utilisation

Le diagramme UML des cas d'utilisation illustre les interactions entre les différents acteurs (utilisateurs, responsables, administrateurs) et le système.


S'inscrire aux activités
S'inscrire
« extend »
« extend »
Se connecter
S'inscrire
Inscrire ses enfants
« Includes »
Consulter les activités
« Includes »
« extend »
Voir les activités disponible par âge
« Includes »
Évaluer les activités
Se connecter
« Includes »
Voir les évaluations
« Includes »
Déclarer un incident
« Includes »
Utilisateur
Consulter les notifications
« Includes »
Voir calendrier
« extend »
« extend »
Se déconnecter
Voir ses inscriptions
Voir les inscriptions de ses enfants


Voir les notifications
Se connecter
« includes »
Voir le calendrier des activités
« includes »
Se connecter
« includes »
Gérer les comptes
« extend »
« extend »
« extend »
Admin
Se déconnecter
ajouter
supprimer
voir


Gérer les activités
« includes »
« extend »
« extend »
« extend »
Se connecter
Ajouter
Supprimer
Modifier
« includes »
Gérer le calendrier
Se connecter
« extend »
Modifier le calendrier
« includes »
« extend »
Consulter
Ajouter une activité
« includes »
Consulter les tickets et incidents
« includes »
Voir toutes les notifications
« includes »
ResponsableActivite
Se déconnecter
Gérer les inscriptions
« extend »
« extend »
« extend »
Valider
Refuser
Laisser en attente

3.3 Backlog Produit Simplifié (Méthode Agile)

Priorité	Fonctionnalité	User Story associée	Rôle
🟥 Haute	Connexion / Déconnexion sécurisée	Connexion, Déconnexion	Admin, Responsable, Utilisateur
🟥 Haute	Gestion des utilisateurs	Voir liste, Supprimer utilisateur	Admin
🟥 Haute	Gestion des activités (CRUD)	Ajouter, modifier, supprimer activité	Responsable
🟥 Haute	Inscription aux activités (adultes/enfants)	S'inscrire, inscrire ses enfants	Utilisateur
🟧 Moyenne	Gestion des inscriptions	Voir, valider, refuser, en attente	Responsable
🟧 Moyenne	Consultation des activités et calendrier	Voir activités, calendrier, activités inscrites	Tous
🟧 Moyenne	Gestion des enfants	Renseigner info enfants	Utilisateur
🟧 Moyenne	Évaluation des activités	Évaluer, voir évaluations	Utilisateur
🟧 Moyenne	Notifications	Voir notifications	Tous
🟨 Faible	Consultation des notifications reçues par un utilisateur	Voir notifs d'un utilisateur	Admin
🟨 Faible	Déclaration et gestion des incidents	Déclarer incident, voir tickets	Utilisateur, Responsable

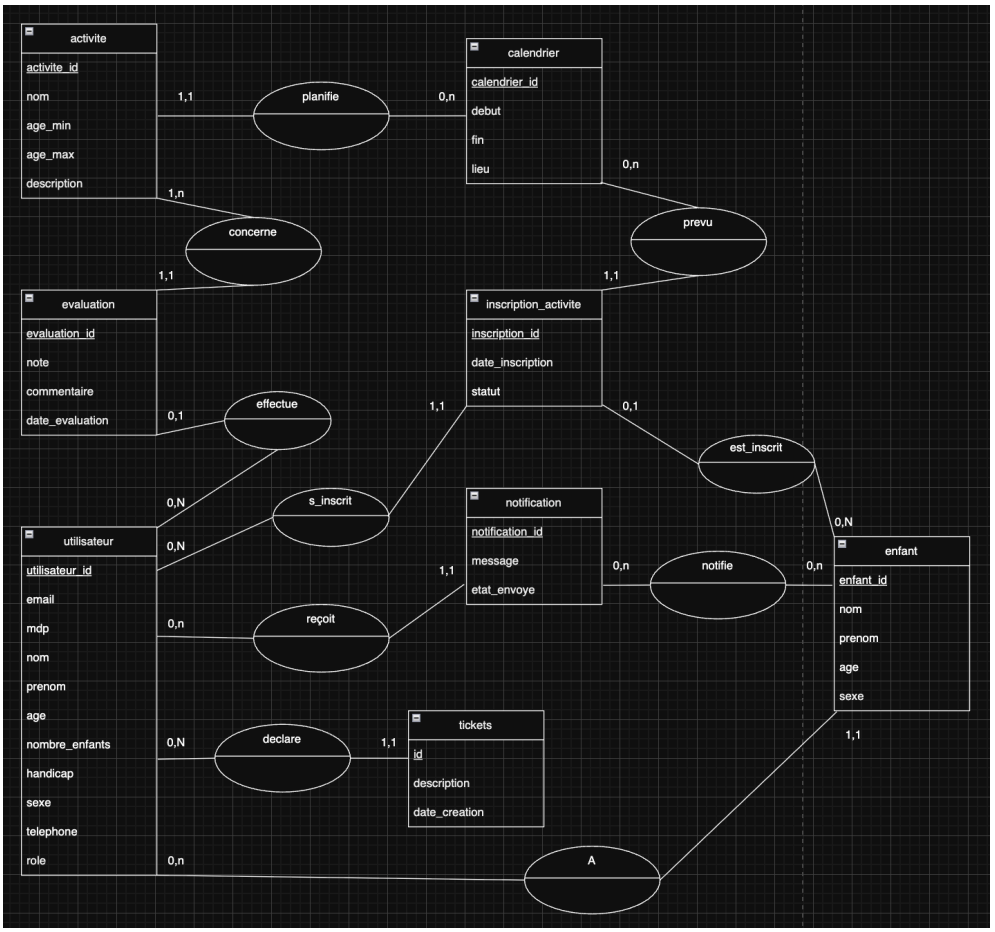
3.4 Calendrier de réalisation (Méthode Agile)

Sprint	Durée	Objectifs
Sprint 1	Semaine 1-2	Authentification, base de données utilisateurs, gestion des rôles
Sprint 2	Semaine 3-4	CRUD activités pour admin et responsable
Sprint 3	Semaine 5-6	Inscription Utilisateur, consultation activités
Sprint 4	Semaine 7-8	Gestion du calendrier et incidents
Sprint 5	Semaine 9	Notifications, évaluations, tests utilisateur
Sprint 6	Semaine 10	Déploiement, documentation, ajustements

4. Base de données

4.1 Modèle conceptuel de données

Le modèle de données comprend plusieurs entités principales et leurs relations.



4.2 Description des tables

4.2.1 Table utilisateur

Stocke les informations relatives aux utilisateurs du système.

sql

```sql
CREATE TABLE utilisateur (
    utilisateur_id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL UNIQUE,
    mdp VARCHAR(255) NOT NULL,
    nom VARCHAR(100) NOT NULL,
    prenom VARCHAR(100) NOT NULL,
    age INT NOT NULL,
    nombre_enfants INT DEFAULT 0,
    handicap TINYINT(1) DEFAULT 0,
    sexe VARCHAR(20) NOT NULL,
    telephone VARCHAR(20),
    role VARCHAR(50) NOT NULL DEFAULT 'utilisateur',
    CHECK (role IN ('utilisateur','responsable','admin'))
);
```

Champs principaux :

- **utilisateur_id** : Identifiant unique auto-incrémenté
- **email** : Adresse email de l'utilisateur (clé unique)
- **mdp** : Mot de passe hashé avec BCrypt
- **role** : Type d'utilisateur (utilisateur, responsable ou admin)

4.2.2 Table enfant

Stocke les informations sur les enfants des utilisateurs.

sql

```
CREATE TABLE enfant (

    enfant_id INT AUTO_INCREMENT PRIMARY KEY,

    utilisateur_id INT NOT NULL,

    nom VARCHAR(255) NOT NULL,

    prenom VARCHAR(255) NOT NULL,

    age INT NOT NULL,

    sexe VARCHAR(20) NOT NULL,

    handicap TINYINT(1) DEFAULT 0,

    FOREIGN KEY (utilisateur_id) REFERENCES utilisateur(utilisateur_id) ON DELETE
CASCADE

);
```

Champs principaux :

- **enfant_id** : Identifiant unique auto-incrémenté
- **utilisateur_id** : Clé étrangère vers la table utilisateur
- **age** : Âge de l'enfant, utilisé pour déterminer les activités éligibles

4.2.3 Table activite

Contient les informations sur les activités disponibles.

sql

```
CREATE TABLE activite (

    activite_id INT AUTO_INCREMENT PRIMARY KEY,

    nom VARCHAR(255) NOT NULL,

    age_min INT NOT NULL,

    age_max INT NOT NULL,

    description TEXT

);
```

Champs principaux :

- **activite_id** : Identifiant unique auto-incrémenté
- **nom** : Nom de l'activité
- **age_min** et **age_max** : Tranche d'âge pour laquelle l'activité est adaptée

4.2.4 Table calendrier

Définit les horaires et lieux des activités.

sql

```
CREATE TABLE calendrier (
    calendrier_id INT AUTO_INCREMENT PRIMARY KEY,
    activite_id INT NOT NULL,
    debut DATETIME NOT NULL,
    fin DATETIME NOT NULL,
    lieu VARCHAR(255),
    FOREIGN KEY (activite_id) REFERENCES activite(activite_id) ON DELETE CASCADE
);
```

Champs principaux :

- **calendrier_id** : Identifiant unique auto-incrémenté
- **activite_id** : Référence à l'activité concernée
- **debut** et **fin** : Date et heure de début et de fin de l'activité
- **lieu** : Emplacement où se déroule l'activité

4.2.5 Table inscription_activite

Gère les inscriptions aux activités.

sql

```
CREATE TABLE inscription_activite (

    inscription_id INT AUTO_INCREMENT PRIMARY KEY,

    calendrier_id INT NOT NULL,

    date_inscription DATETIME DEFAULT CURRENT_TIMESTAMP,

    utilisateur_id INT,

    enfant_id INT,

    statut ENUM('À valider', 'Validé', 'Refusé') NOT NULL DEFAULT 'À valider',

    FOREIGN KEY (calendrier_id) REFERENCES calendrier(calendrier_id) ON DELETE
CASCADE,

    FOREIGN KEY (utilisateur_id) REFERENCES utilisateur(utilisateur_id) ON DELETE
CASCADE,

    FOREIGN KEY (enfant_id) REFERENCES enfant(enfant_id) ON DELETE CASCADE

);
```

Champs principaux :

- **inscription_id** : Identifiant unique auto-incrémenté
- **calendrier_id** : Référence au créneau calendrier concerné
- **utilisateur_id** : Référence à l'utilisateur inscrit (si adulte)
- **enfant_id** : Référence à l'enfant inscrit (si enfant)
- **statut** : État de l'inscription (À valider, Validé, Refusé)

4.2.6 Table evaluation

Stocke les évaluations des activités par les utilisateurs.

sql

```sql
CREATE TABLE evaluation (
    evaluation_id INT AUTO_INCREMENT PRIMARY KEY,
    utilisateur_id INT,
    activite_id INT,
    note INT NOT NULL,
    commentaire TEXT,
    date_evaluation DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (utilisateur_id) REFERENCES utilisateur(utilisateur_id),
    FOREIGN KEY (activite_id) REFERENCES activite(activite_id)
);
```

Champs principaux :

- **evaluation_id** : Identifiant unique auto-incrémenté
- **utilisateur_id** : Référence à l'utilisateur ayant fait l'évaluation
- **activite_id** : Référence à l'activité évaluée
- **note** : Note attribuée sur 5

4.2.7 Table notification

Gère les notifications système.

sql

```
CREATE TABLE notification (

    notification_id INT AUTO_INCREMENT PRIMARY KEY,

    message TEXT NOT NULL,

    etat_envoye TINYINT(1) DEFAULT 0,

    utilisateur_id INT,

    enfant_id INT,

    FOREIGN KEY (utilisateur_id) REFERENCES utilisateur(utilisateur_id) ON DELETE
CASCADE,

    FOREIGN KEY (enfant_id) REFERENCES enfant(enfant_id) ON DELETE CASCADE

);
```

Champs principaux :

- **notification_id** : Identifiant unique auto-incrémenté
- **message** : Contenu de la notification
- **utilisateur_id** : Référence à l'utilisateur destinataire
- **enfant_id** : Référence à l'enfant concerné (si applicable)

4.2.8 Table tickets

Stocke les tickets d'incidents créés par les utilisateurs.

sql

```
CREATE TABLE tickets (

    id INT AUTO_INCREMENT PRIMARY KEY,

    utilisateur_id INT NOT NULL,

    description TEXT NOT NULL,

    date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (utilisateur_id) REFERENCES utilisateur(utilisateur_id) ON DELETE
CASCADE ON UPDATE CASCADE

);
```

Champs principaux :

- **id** : Identifiant unique auto-incrémenté
- **utilisateur_id** : Référence à l'utilisateur ayant créé le ticket
- **description** : Description du problème
- **date_creation** : Date et heure de création du ticket

4.3 Triggers et événements

4.3.1 Trigger après ajout d'enfant

Ce trigger met à jour le nombre d'enfants d'un utilisateur après l'ajout d'un enfant.

sql

```
CREATE TRIGGER apres_ajout_enfant AFTER INSERT ON enfant

FOR EACH ROW

BEGIN

    UPDATE utilisateur SET nombre_enfants = nombre_enfants + 1 WHERE utilisateur_id
= NEW.utilisateur_id;

END;
```

4.3.2 Trigger après suppression d'enfant

Ce trigger met à jour le nombre d'enfants d'un utilisateur après la suppression d'un enfant.

sql

```
CREATE TRIGGER apres_suppression_enfant AFTER DELETE ON enfant

FOR EACH ROW

BEGIN

    UPDATE utilisateur SET nombre_enfants = nombre_enfants - 1 WHERE utilisateur_id
= OLD.utilisateur_id;

END;
```

4.3.3 Trigger après inscription à une activité

Ce trigger crée une notification après l'inscription à une activité.

sql

```
CREATE TRIGGER apres_inscription_activite AFTER INSERT ON inscription_activite

FOR EACH ROW

BEGIN

    DECLARE msg TEXT;

    SET msg = 'Votre inscription à l\'activité a bien été enregistrée !';

    IF NEW.utilisateur_id IS NOT NULL THEN

        INSERT INTO notification (utilisateur_id, message) VALUES
(NEW.utilisateur_id, msg);

    ELSEIF NEW.enfant_id IS NOT NULL THEN

        INSERT INTO notification (enfant_id, message) VALUES (NEW.enfant_id, msg);

    END IF;

END;
```

4.3.4 Événement de rappel d'activité

Cet événement crée des notifications de rappel 15 jours avant une activité.

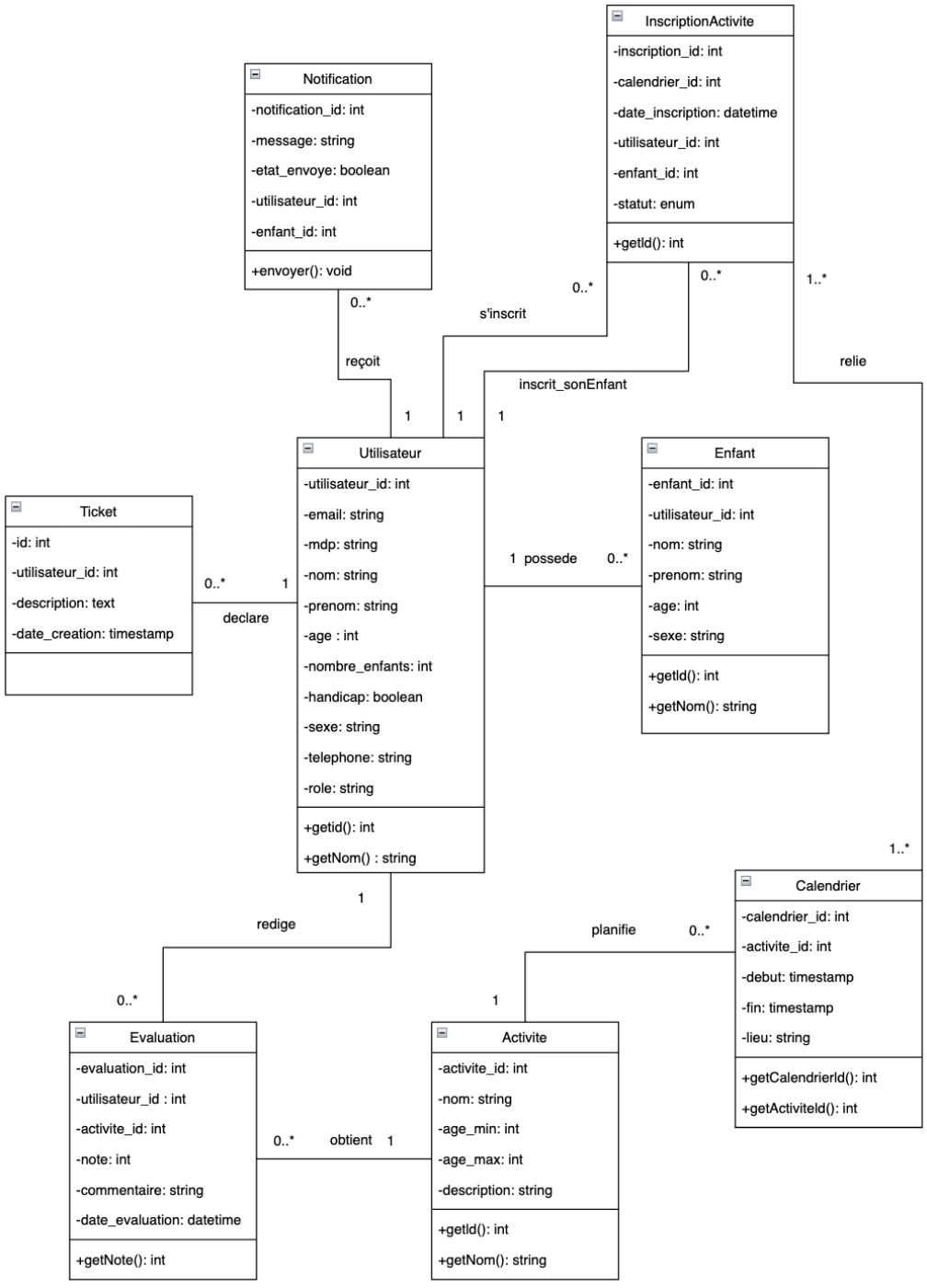
sql

```
CREATE EVENT rappel_activite

ON SCHEDULE EVERY 1 DAY STARTS CURRENT_TIMESTAMP

ON COMPLETION NOT PRESERVE ENABLE

DO

BEGIN

    DECLARE aujourd_hui DATE;

    SET aujourd_hui = CURDATE();


    INSERT INTO notification (utilisateur_id, message)

    SELECT ia.utilisateur_id,

           CONCAT('Rappel : ', u.nom, ' vous êtes inscrit à l'activité "', a.nom,
'" le ',

                  DATE_FORMAT(c.debut, '%d/%m/%Y à %H:%i'))

    FROM inscription_activite ia

    JOIN calendrier c ON ia.calendrier_id = c.calendrier_id

    JOIN activite a ON c.activite_id = a.activite_id

    JOIN utilisateur u ON ia.utilisateur_id = u.utilisateur_id

    WHERE DATEDIFF(c.debut, aujourd_hui) = 15;


    INSERT INTO notification (enfant_id, message)

    SELECT ia.enfant_id,

           CONCAT('Rappel : ', e.prenom, ' ', e.nom, ' est inscrit à l'activité "',
a.nom, '" le ',

                  DATE_FORMAT(c.debut, '%d/%m/%Y à %H:%i'))

    FROM inscription_activite ia

    JOIN calendrier c ON ia.calendrier_id = c.calendrier_id

    JOIN activite a ON c.activite_id = a.activite_id
```

```
    JOIN enfant e ON ia.enfant_id = e.enfant_id
    WHERE DATEDIFF(c.debut, aujourd_hui) = 15;
END;
```

5. Description des classes

5.1 Modèles (Model)



5.1.1 Classe Utilisateur

Représente un utilisateur du système.

Attributs principaux :

- **utilisateur_id** : Identifiant unique
- **email** : Adresse email (identifiant de connexion)
- **mdp** : Mot de passe hashé
- **nom** et **prenom** : Informations personnelles
- **age** : Âge de l'utilisateur
- **nombre_enfants** : Nombre d'enfants enregistrés
- **handicap** : Indique si l'utilisateur a un handicap
- **sexe** : Genre de l'utilisateur
- **telephone** : Numéro de téléphone
- **role** : Rôle de l'utilisateur (utilisateur, responsable, admin)

Méthodes principales :

- Getters et setters pour tous les attributs
- Méthodes de validation des données (email, mot de passe, etc.)

5.1.2 Classe Activite

Représente une activité proposée dans le camp.

Attributs principaux :

- **activite_id** : Identifiant unique
- **nom** : Nom de l'activité
- **age_min** et **age_max** : Tranche d'âge ciblée
- **description** : Description détaillée de l'activité

Méthodes principales :

- Getters et setters pour tous les attributs
- Méthode pour vérifier si un âge est dans la tranche autorisée

5.1.3 Classe Calendrier

Représente un créneau planifié pour une activité.

Attributs principaux :

- **calendrier_id** : Identifiant unique
- **activite_id** : Référence à l'activité concernée
- **debut** et **fin** : Date et heure de début et de fin
- **lieu** : Lieu où se déroule l'activité

Méthodes principales :

- Getters et setters pour tous les attributs
- Méthodes pour calculer la durée

5.1.4 Classe InscriptionActivite

Gère les inscriptions aux activités.

Attributs principaux :

- **inscription_id** : Identifiant unique
- **calendrier_id** : Référence au créneau calendrier
- **date_inscription** : Date d'inscription
- **utilisateur_id** : Référence à l'utilisateur (si adulte)
- **enfant_id** : Référence à l'enfant (si enfant)
- **statut** : État de l'inscription (À valider, Validé, Refusé)

Méthodes principales :

- Getters et setters pour tous les attributs
- Méthodes pour changer le statut de l'inscription

5.1.5 Classe Notification

Représente une notification système.

Attributs principaux :

- **notification_id** : Identifiant unique
- **message** : Contenu de la notification
- **etat_envoye** : Indique si la notification a été envoyée
- **utilisateur_id** : Référence à l'utilisateur destinataire
- **enfant_id** : Référence à l'enfant concerné (si applicable)

Méthodes principales :

- Getters et setters pour tous les attributs
- Méthode pour marquer comme lue/envoyée

5.1.6 Classe Evaluation

Représente l'évaluation d'une activité par un utilisateur.

Attributs principaux :

- **evaluation_id** : Identifiant unique
- **utilisateur_id** : Référence à l'utilisateur ayant fait l'évaluation
- **activite_id** : Référence à l'activité évaluée
- **note** : Note attribuée sur 5
- **commentaire** : Commentaire textuel
- **date_evaluation** : Date de l'évaluation

Méthodes principales :

- Getters et setters pour tous les attributs

5.2 DAO (Data Access Objects)

5.2.1 Classe DatabaseConnection

Gère la connexion à la base de données.

Méthodes principales :

- **getConnection()** : Retourne une connexion à la base de données MySQL

java

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    public static Connection getConnection() throws SQLException {
        String url = "jdbc:mysql://localhost:3306/camp_activites2";
        String user = "root";
        String password = ""; // À modifier selon la configuration
        return DriverManager.getConnection(url, user, password);
    }
}
```

5.2.2 Classe UtilisateurDAO

Gère les opérations CRUD pour les utilisateurs.

Méthodes principales :

- **creerUtilisateur(Utilisateur utilisateur)** : Ajoute un nouvel utilisateur
- **authentifier(String email, String mdp)** : Vérifie les identifiants de connexion
- **getUtilisateurById(int id)** : Récupère un utilisateur par son ID
- **getUtilisateurByEmail(String email)** : Récupère un utilisateur par son email
- **supprimerUtilisateur(int id)** : Supprime un utilisateur
- **getAllUtilisateurs()** : Récupère tous les utilisateurs
- **updateUtilisateur(Utilisateur utilisateur)** : Met à jour un utilisateur

5.2.3 Classe ActiviteDAO

Gère les opérations CRUD pour les activités.

Méthodes principales :

- **creerActivite(Activite activite)** : Ajoute une nouvelle activité
- **getActiviteById(int id)** : Récupère une activité par son ID
- **getActivitesByAgeRange(int age)** : Récupère les activités adaptées à un âge
- **modifierActivite(Activite activite)** : Met à jour une activité
- **supprimerActivite(int id)** : Supprime une activité
- **getAllActivites()** : Récupère toutes les activités

5.2.4 Classe CalendrierDAO

Gère les opérations CRUD pour les créneaux calendrier.

Méthodes principales :

- **ajouterCreneau(Calendrier calendrier)** : Ajoute un nouveau créneau
- **getCreneauById(int id)** : Récupère un créneau par son ID
- **getCreneauxByActiviteId(int activiteId)** : Récupère les créneaux d'une activité
- **modifierCreneau(Calendrier calendrier)** : Met à jour un créneau
- **supprimerCreneau(int id)** : Supprime un créneau
- **getAllCreneaux()** : Récupère tous les créneaux

5.2.5 Classe InscriptionActiviteDAO

Gère les opérations CRUD pour les inscriptions.

Méthodes principales :

- **inscrireUtilisateur(int utilisateurId, int calendrierId)** : Inscrit un utilisateur
- **inscrireEnfant(int enfantId, int calendrierId)** : Inscrit un enfant
- **getInscriptionsByUtilisateurId(int utilisateurId)** : Récupère les inscriptions d'un utilisateur
- **getInscriptionsByEnfantId(int enfantId)** : Récupère les inscriptions d'un enfant
- **updateStatut(int inscriptionId, String statut)** : Met à jour le statut d'une inscription
- **verifierAgeActivite(int age, int activiteId)** : Vérifie si l'âge est compatible avec l'activité

5.2.6 Classe NotificationDAO

Gère les opérations CRUD pour les notifications.

Méthodes principales :

- **creerNotification(Notification notification)** : Crée une nouvelle notification
- **getNotificationsByUtilisateurId(int utilisateurId)** : Récupère les notifications d'un utilisateur
- **marquerCommeLue(int notificationId)** : Marque une notification comme lue

5.3 Interfaces utilisateur (View)

5.3.1 Classe FenetreConnexion

Interface de connexion au système.

Composants principaux :

- Champs de saisie pour email et mot de passe
- Case à cocher pour afficher le mot de passe
- Boutons pour se connecter et créer un compte

5.3.2 Classe FenetreInscription

Interface d'inscription au système.

Composants principaux :

- Champs de saisie pour email, nom, prénom, mot de passe, etc.
- Sélecteurs pour âge, nombre d'enfants
- Options pour sexe et handicap
- Boutons pour valider et retourner à l'écran de connexion

5.3.3 Classe FenetrePrincipale

Interface principale pour les utilisateurs standards.

Composants principaux :

- Boutons pour les différentes fonctionnalités (afficher activités, s'inscrire, etc.)
- Affichage des informations personnelles
- Accès aux différentes sections de l'application

5.3.4 Classe FenetreAdmin

Interface pour les administrateurs.

Composants principaux :

- Boutons pour gérer les utilisateurs
- Boutons pour consulter les notifications
- Boutons pour accéder au calendrier des activités

5.3.5 Classe FenetreResponsable

Interface pour les responsables d'activités.

Composants principaux :

- Boutons pour gérer les activités
- Boutons pour gérer le calendrier
- Boutons pour gérer les tickets et les inscriptions

5.3.6 Autres classes d'interface

- **FenetreCalendrier** : Affiche le calendrier des activités
- **FenetreGestionActivites** : Permet de gérer les activités
- **FenetreNotification** : Affiche les notifications
- **FenetreTicketsIncidents** : Permet de gérer les tickets d'incidents
- **FenetreInscriptionsActivite** : Permet de gérer les inscriptions aux activités

5.4 Services

5.4.1 Classe NotificationService

Gère l'envoi et le traitement des notifications.

Méthodes principales :

- **envoyerNotificationInscription(int utilisateurId, int activiteId)** : Envoie une notification après inscription
- **envoyerRappelActivite(int utilisateurId, int activiteId, Date dateActivite)** : Envoie un rappel d'activité
- **getNotificationsNonLues(int utilisateurId)** : Récupère les notifications non lues

6. Fonctionnalités principales

6.1 Système d'authentification

Le système d'authentification est géré par les classes FenetreConnexion, UtilisateurDAO et le modèle Utilisateur.

Fonctionnalités principales :

- Connexion avec vérification des identifiants
- Hashage des mots de passe avec BCrypt
- Redirection vers l'interface appropriée selon le rôle de l'utilisateur

Flux d'authentification :

1. L'utilisateur saisit son email et mot de passe
2. Le système vérifie la présence de l'email dans la base de données
3. Si l'email existe, le système compare le hash du mot de passe saisi avec celui stocké
4. En cas de correspondance, l'utilisateur est redirigé vers son interface selon son rôle

Code d'authentification (extrait) :

java
```
// Dans UtilisateurDAO
public Utilisateur authentifier(String email, String mdp) {
    // Recherche de l'utilisateur par email
    Utilisateur utilisateur = getUtilisateurByEmail(email);
    if (utilisateur != null) {
        // Vérification du mot de passe avec BCrypt
        if (BCrypt.checkpw(mdp, utilisateur.getMdp())) {
            return utilisateur; // Authentification réussie
        }
    }
    return null; // Authentification échouée
}
```

6.2 Gestion des utilisateurs

La gestion des utilisateurs est principalement assurée par l'administrateur du système. Les fonctionnalités incluent :

- Création de comptes utilisateur (inscription)
- Consultation de la liste des utilisateurs
- Suppression d'utilisateurs

- Consultation des notifications des utilisateurs

La création de compte utilisateur respecte les règles suivantes :

- L'email doit être unique et valide
- Le mot de passe doit respecter les règles de sécurité RGPD (12 caractères minimum, majuscule, minuscule, chiffre, caractère spécial)
- L'âge minimum est de 18 ans pour créer un compte

Code de validation du mot de passe (extrait) :

java
```
public boolean validatePassword(String password) {
    String regex =
"^(?=.*[A-Z])(?=.*[a-z])(?=.*\\d)(?=.*[@$!%*?&])[A-Za-z\\d@$!%*?&]{12,}$";
    return password.matches(regex);
}
```

6.3 Gestion des activités

La gestion des activités est principalement assurée par les responsables d'activités. Les fonctionnalités incluent :

- Création d'activités avec définition des tranches d'âge
- Modification des informations des activités
- Suppression d'activités
- Consultation des activités selon différents critères
- Planification des créneaux pour les activités

Processus de création d'activité :

1. Le responsable renseigne le nom, la description et la tranche d'âge de l'activité
2. Le système valide les informations et crée l'activité
3. Le responsable peut ensuite définir des créneaux dans le calendrier pour cette activité

6.4 Gestion des inscriptions

Le système permet aux utilisateurs de s'inscrire ou d'inscrire leurs enfants aux activités. Les fonctionnalités incluent :

- Inscription d'un utilisateur à une activité
- Inscription des enfants d'un utilisateur aux activités adaptées à leur âge
- Validation ou refus des inscriptions par les responsables
- Consultation des inscriptions par activité ou par utilisateur

Processus d'inscription :

1. L'utilisateur sélectionne une activité
2. Le système vérifie la compatibilité d'âge
3. Si compatible, l'inscription est créée avec le statut "À valider"
4. Une notification est automatiquement générée pour confirmer l'inscription
5. Le responsable peut ensuite valider ou refuser l'inscription

6.5 Gestion des enfants

Les utilisateurs peuvent enregistrer leurs enfants dans le système pour les inscrire à des activités. Les fonctionnalités incluent :

- Ajout d'enfants au profil utilisateur
- Modification des informations des enfants
- Consultation des activités adaptées à l'âge des enfants
- Inscription des enfants aux activités

Le système met automatiquement à jour le compte d'enfants de l'utilisateur grâce aux triggers de la base de données.

6.6 Système de notifications

Le système envoie automatiquement des notifications aux utilisateurs pour différents événements. Les fonctionnalités incluent :

- Notification de confirmation d'inscription
- Rappel d'activité 15 jours avant la date prévue
- Consultation des notifications par l'utilisateur
- Marquage des notifications comme lues

Le service de notification utilise l'événement MySQL rappel_activite qui s'exécute quotidiennement pour vérifier les activités à venir et générer les rappels appropriés.

6.7 Système d'évaluation

Les utilisateurs peuvent évaluer les activités auxquelles ils ont participé. Les fonctionnalités incluent :

- Attribution d'une note de 1 à 5 à une activité
- Ajout d'un commentaire textuel
- Consultation des évaluations par activité
- Consultation des moyennes des notes par activité

Les évaluations ne sont possibles que pour les activités auxquelles l'utilisateur a effectivement participé.

6.8 Gestion des tickets d'incident

Les utilisateurs peuvent signaler des problèmes ou incidents via des tickets. Les fonctionnalités incluent :

- Création de tickets d'incident
- Consultation des tickets par les responsables
- Suivi et résolution des problèmes signalés

Le processus de création d'un ticket est simple : l'utilisateur décrit le problème rencontré, et le ticket est enregistré avec la date de création et l'identifiant de l'utilisateur.

6.9 Calendrier des activités

Le calendrier permet de visualiser les activités planifiées. Les fonctionnalités incluent :

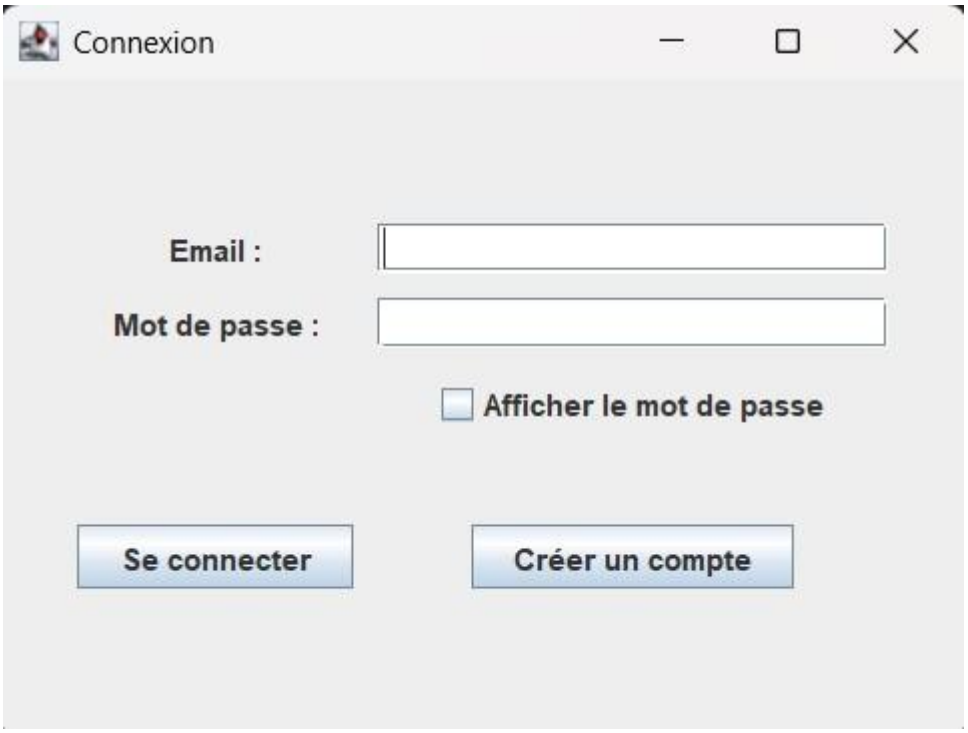
- Consultation du calendrier des activités
- Filtrage des activités par date, type ou tranche d'âge
- Consultation des activités auxquelles l'utilisateur et ses enfants sont inscrits

Le calendrier affiche les informations essentielles comme le nom de l'activité, les dates et heures de début et de fin, et le lieu.

7. Interfaces Principales

7.1 Interface de connexion

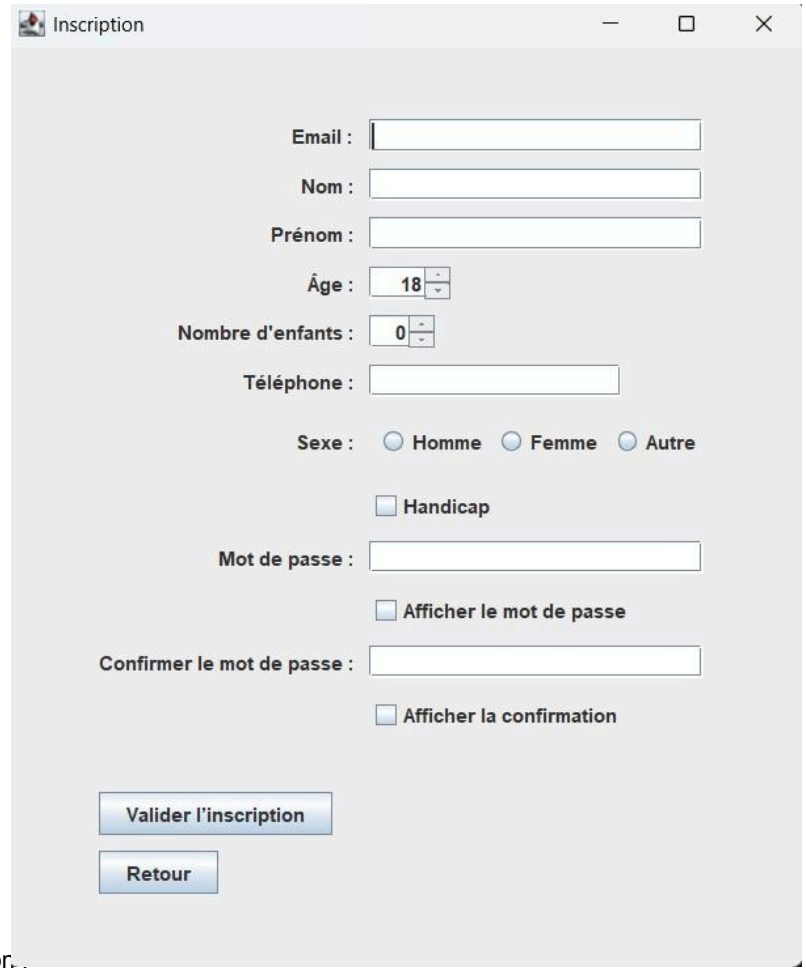
L'interface de connexion est la première page que voit l'utilisateur. Elle est gérée par la classe FenetreConnexion.



- Description :
  - L'utilisateur entre son email et son mot de passe.
  - Un bouton permet de se connecter.
  - Un lien ou bouton permet de créer un compte si l'utilisateur n'en a pas encore.

7.2 Interface d'inscription

L'interface d'inscription permet de créer un nouveau compte utilisateur. Elle est gérée par la classe FenetreInscription.



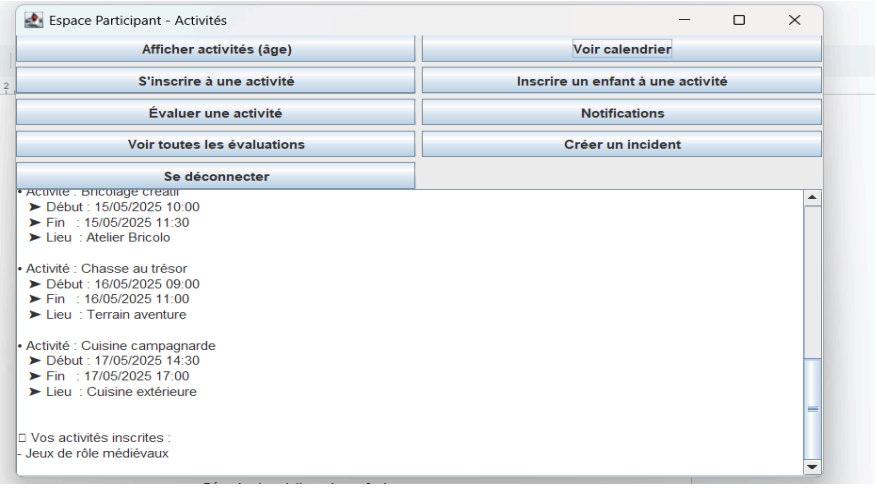
- Description :
  - Formulaire pour créer un compte avec des champs comme email, nom, prénom, âge, mot de passe, etc.
  - Boutons pour valider l'inscription ou revenir à l'écran de connexion.

7.3 Interface utilisateur

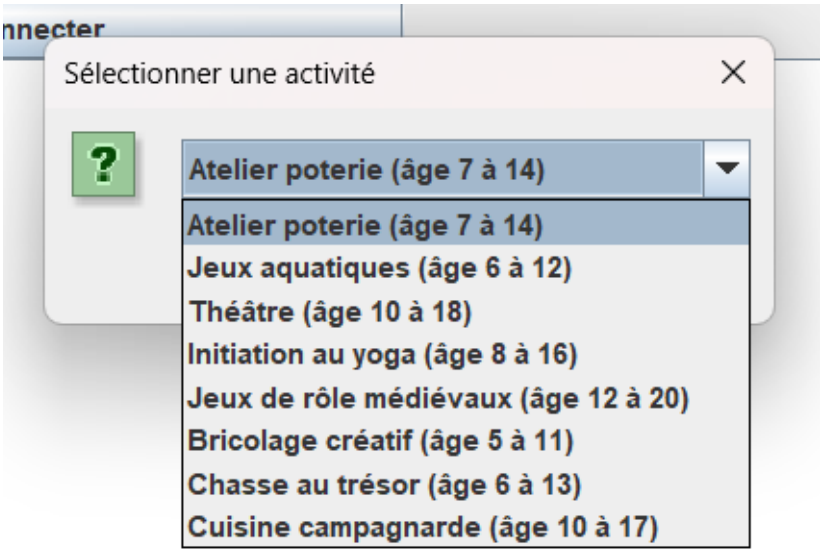
L'interface principale pour les utilisateurs standards est gérée par la classe FenetrePrincipale.

- Description :
  - Tableau de bord avec des boutons pour :
    - **Afficher les activités disponibles.**

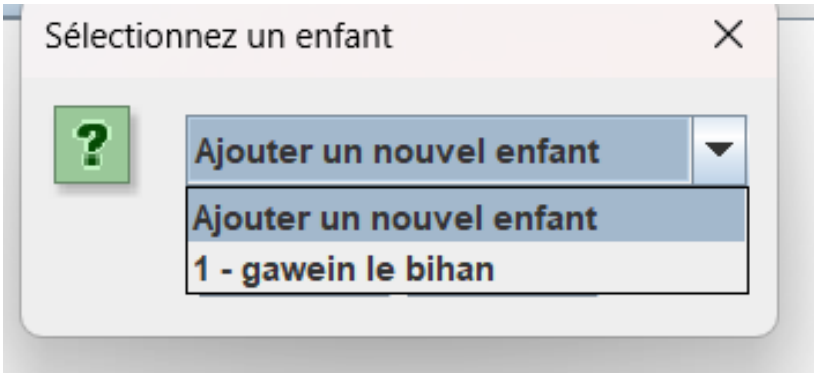
- **Voir le calendrier.**



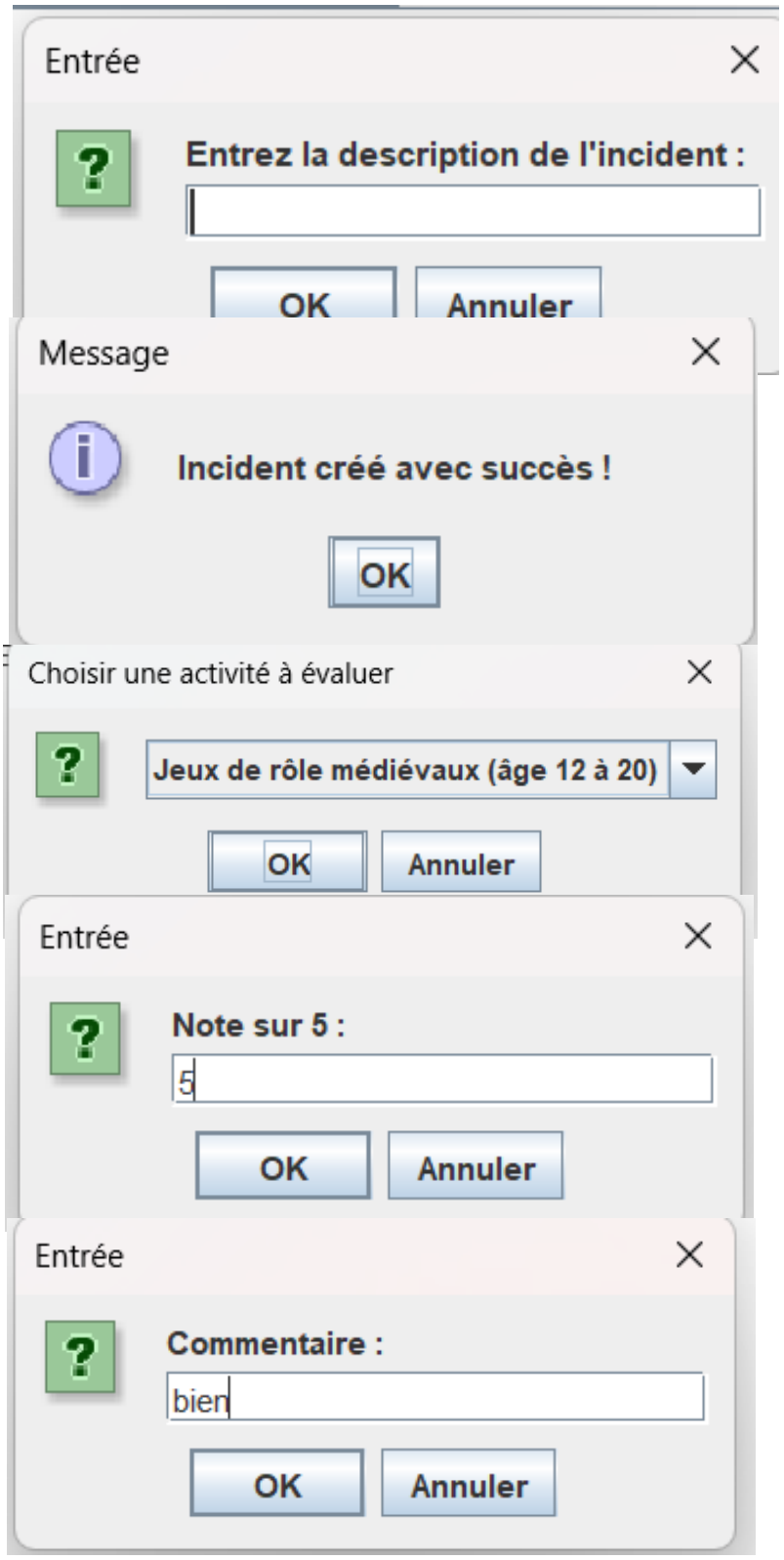
- **S'inscrire à une activité.**



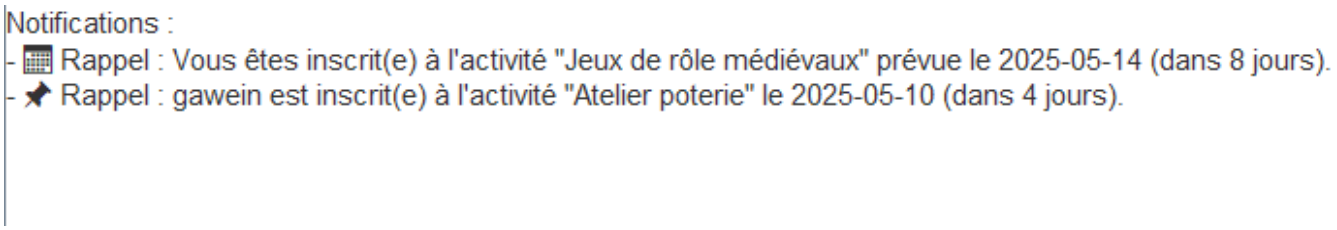
- **Gérer les inscriptions des enfants.**



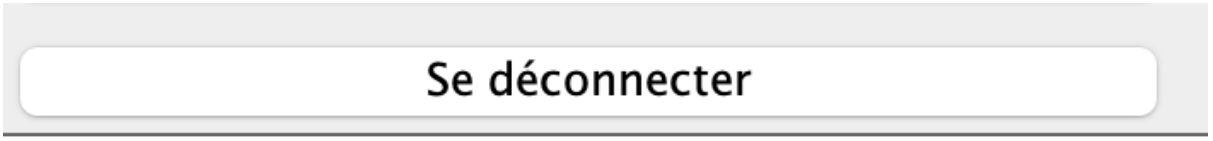
- **crée un ticket**



- **Notifications**

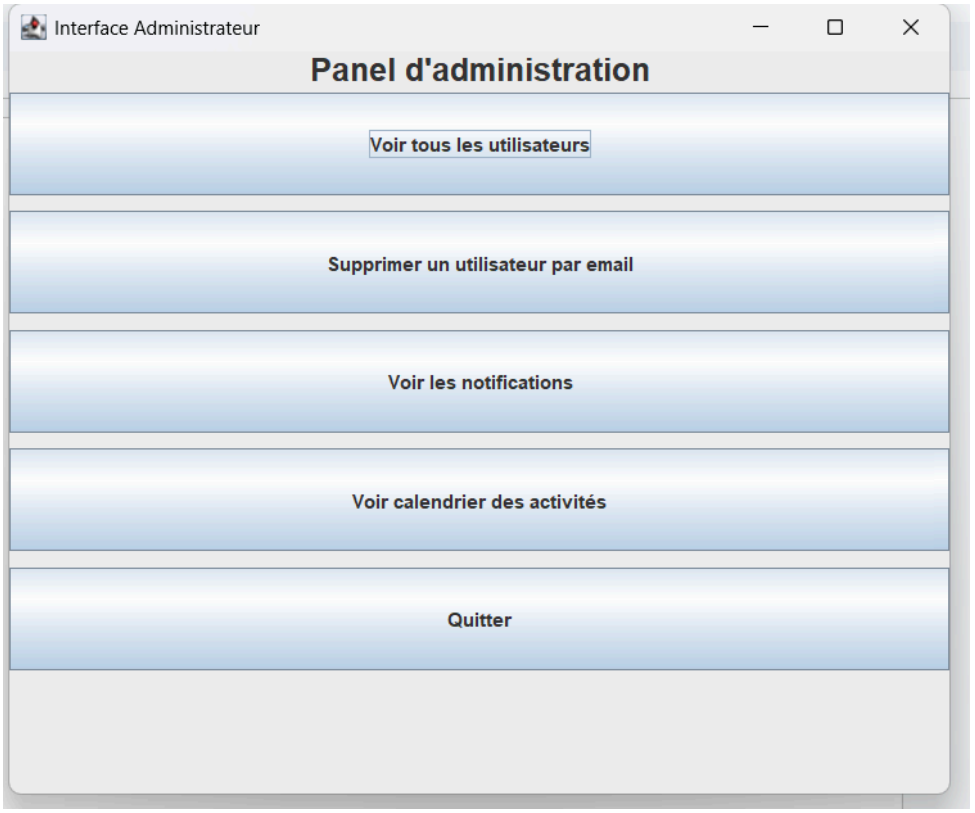


- **Déconnexion.**

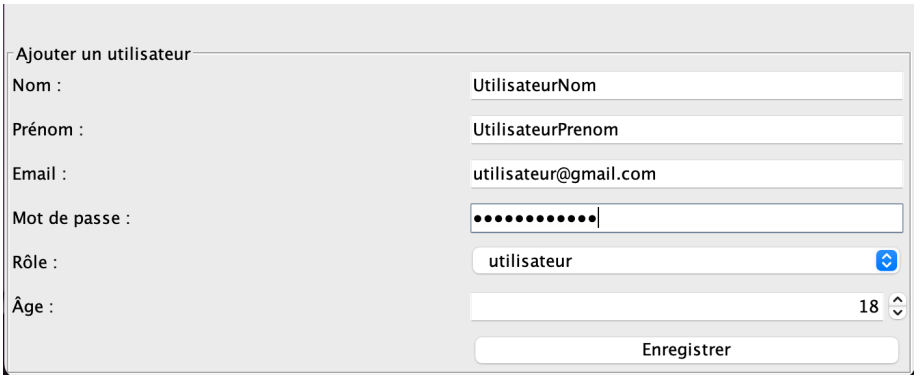


7.4 Interface administrateur

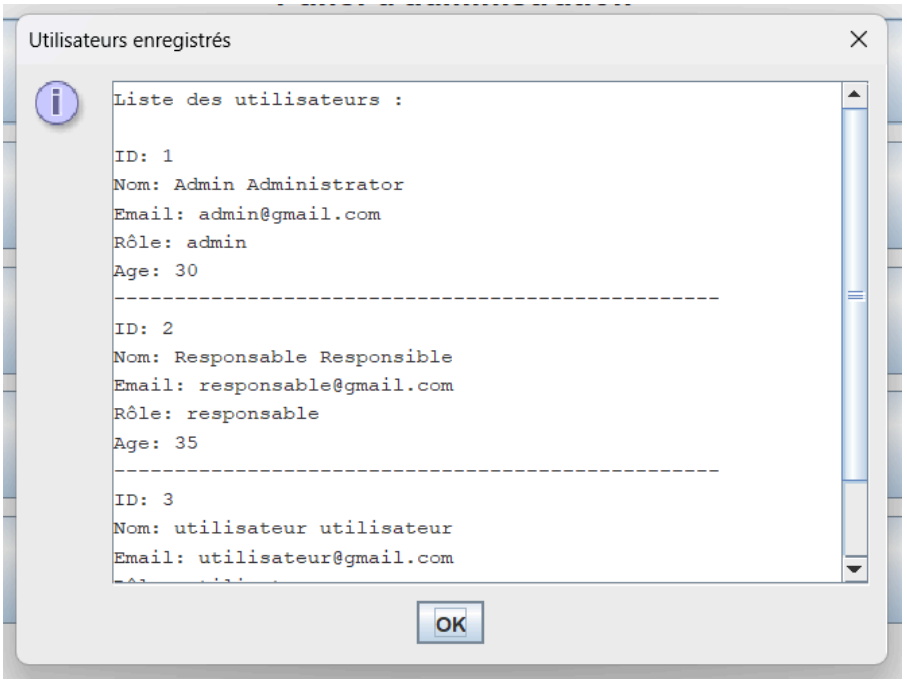
L'interface administrateur est gérée par la classe FenetreAdmin.



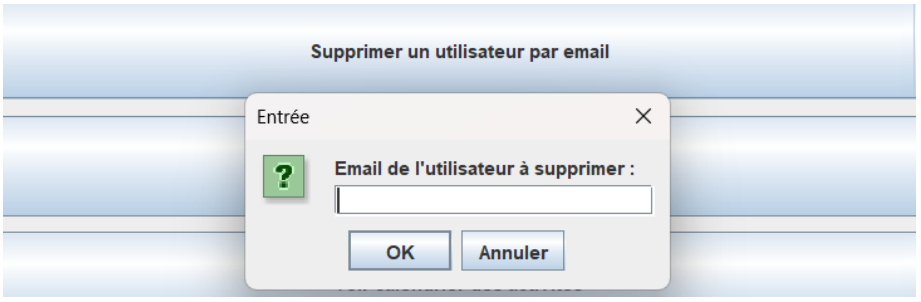
- Description :
  - Tableau de bord avec des boutons pour :
    - **Ajouter un utilisateur**



    - **Voir tous les utilisateurs.**

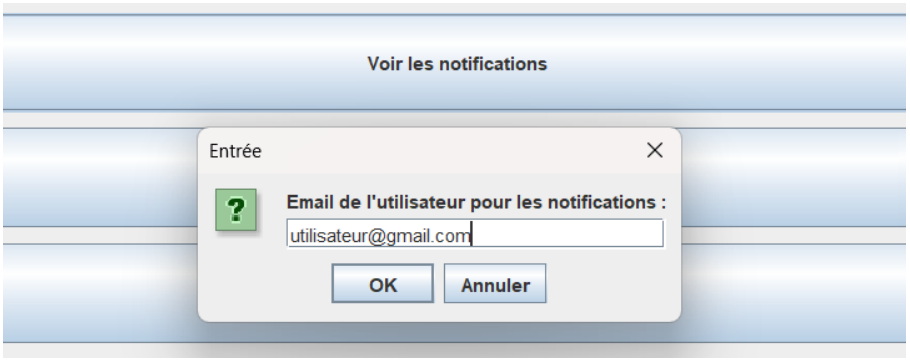


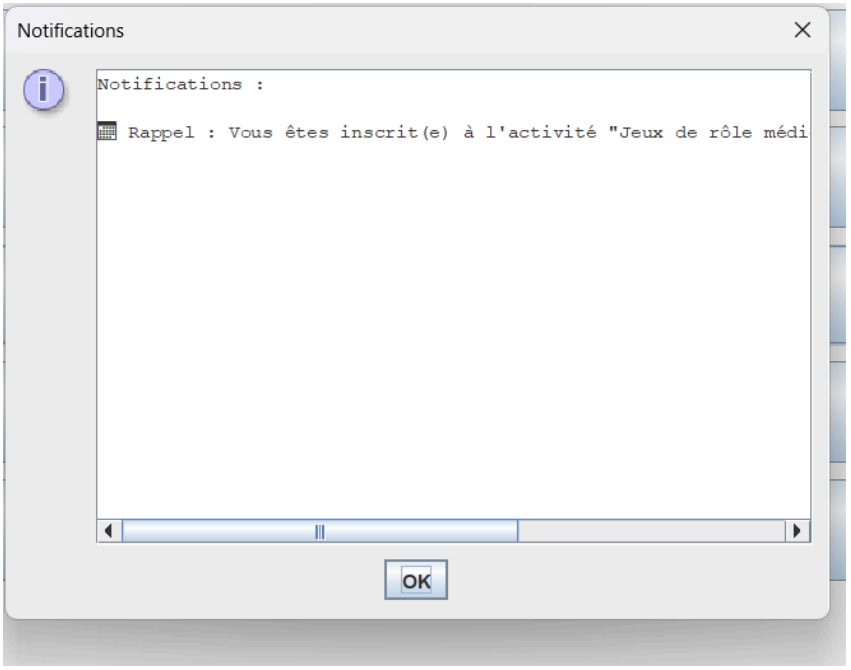
    - **Supprimer un utilisateur.**



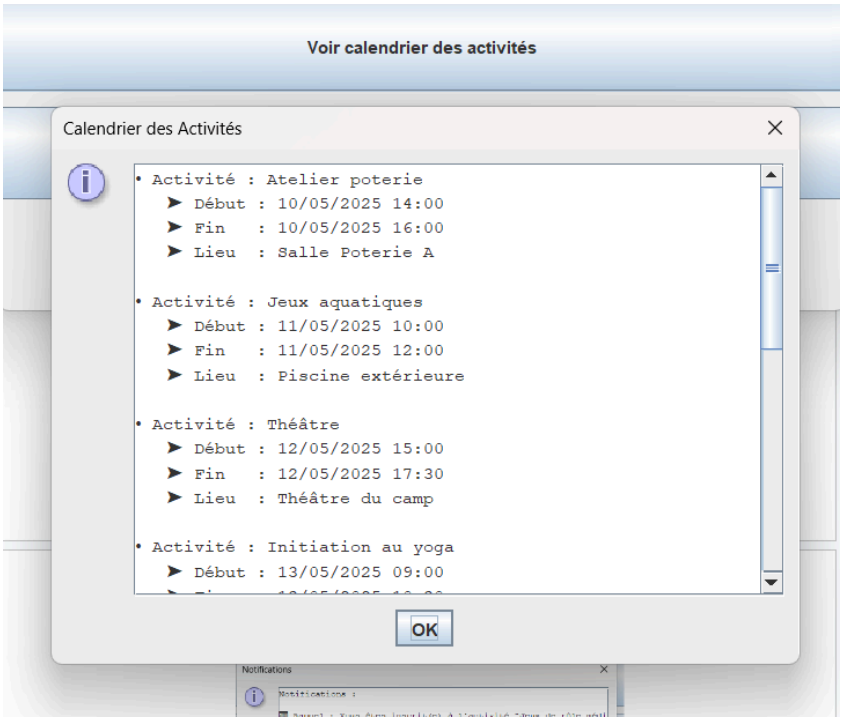
    -

- **Consulter les notifications.**

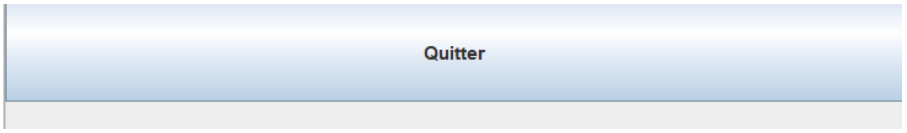




- **Voir le calendrier des activités.**



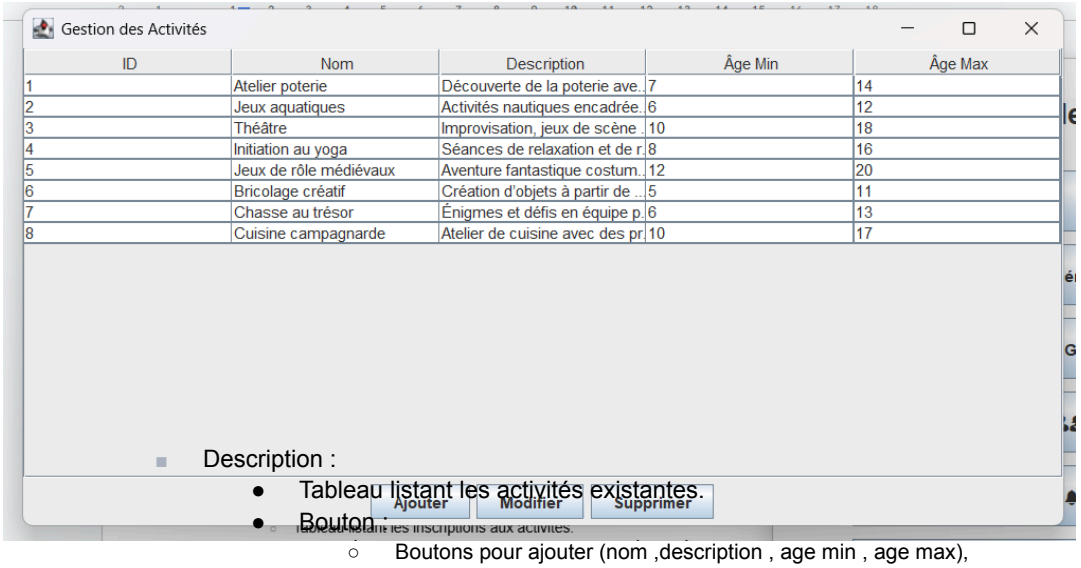
- **Quitter.**



7.5 Interface responsable d'activité

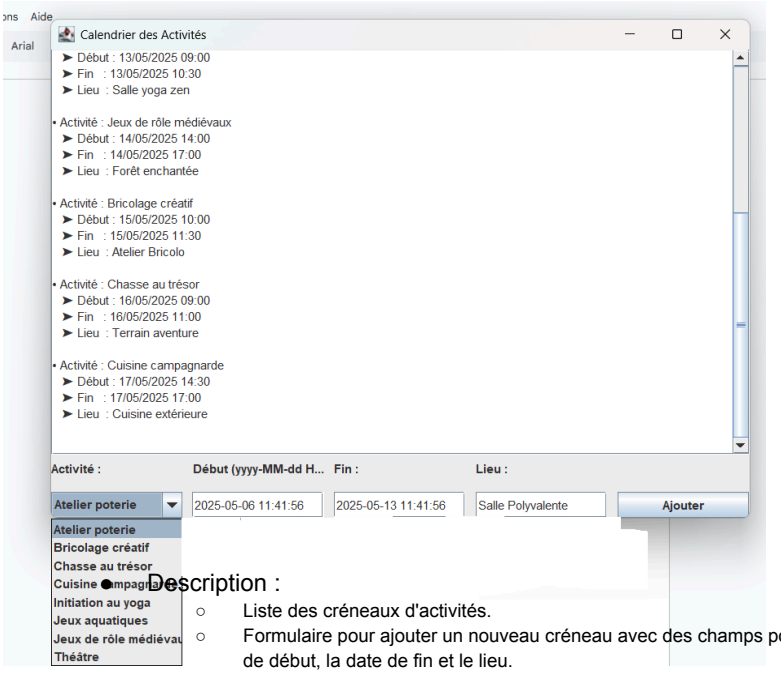
L'interface responsable est gérée par la classe FenetreResponsable.

- Description :
  - Tableau [illegible]
    - **Gérer les activités.**



    - Description :
      - Tableau listant les activités existantes.
      - Bouton :
        - Boutons pour ajouter (nom ,description , age min , age max),
        - Boutons modifier (nom ,description , age min , age max) ou
        - Boutons supprimer une activité.

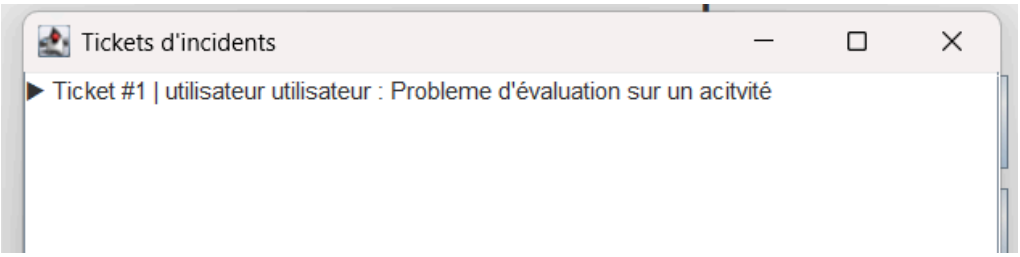
- **Gérer le calendrier.**



Description :

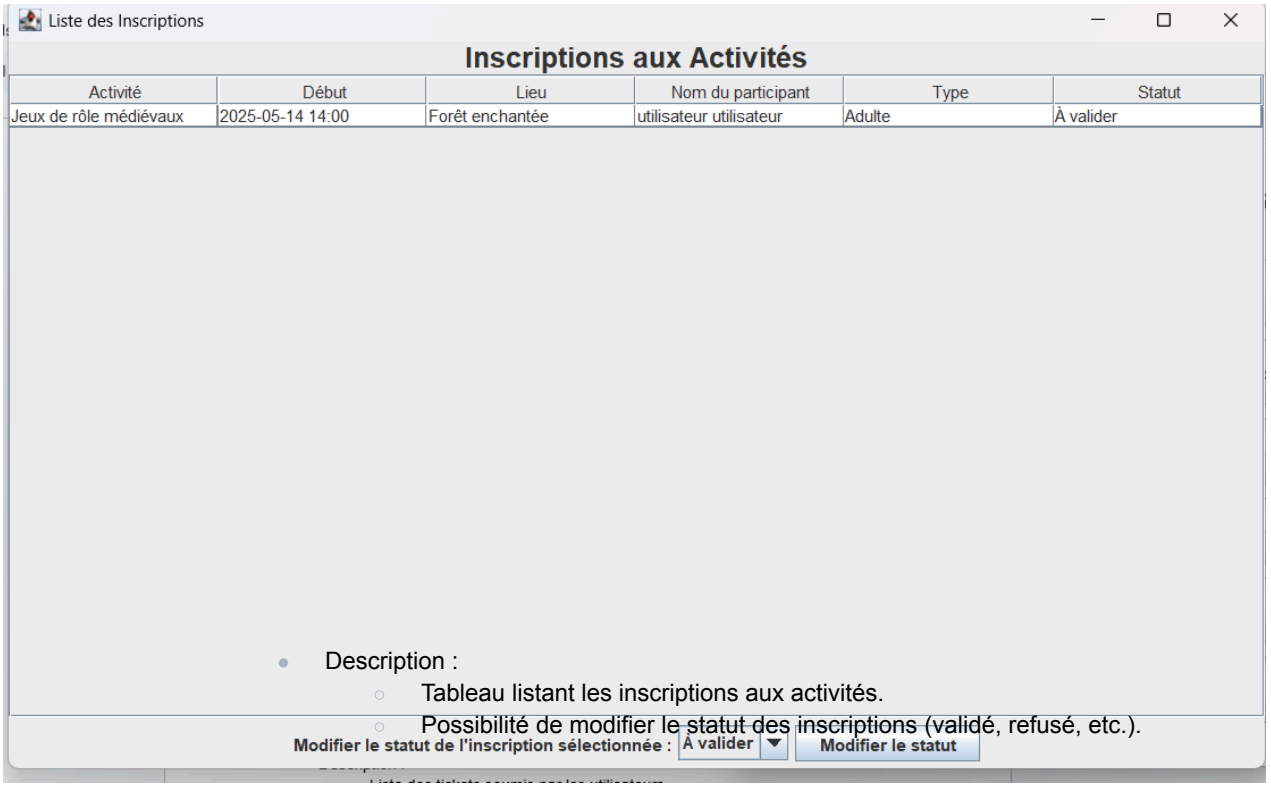
- Liste des créneaux d'activités.
- Formulaire pour ajouter un nouveau créneau avec des champs pour l'activité, la date de début, la date de fin et le lieu.

- **Voir les tickets d'incidents.**

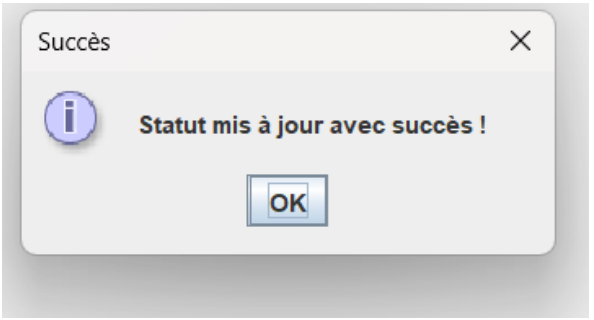


- Description :
  - Liste des tickets soumis par les utilisateurs.
  - Affichage des détails comme l'auteur, la description et la date de création.

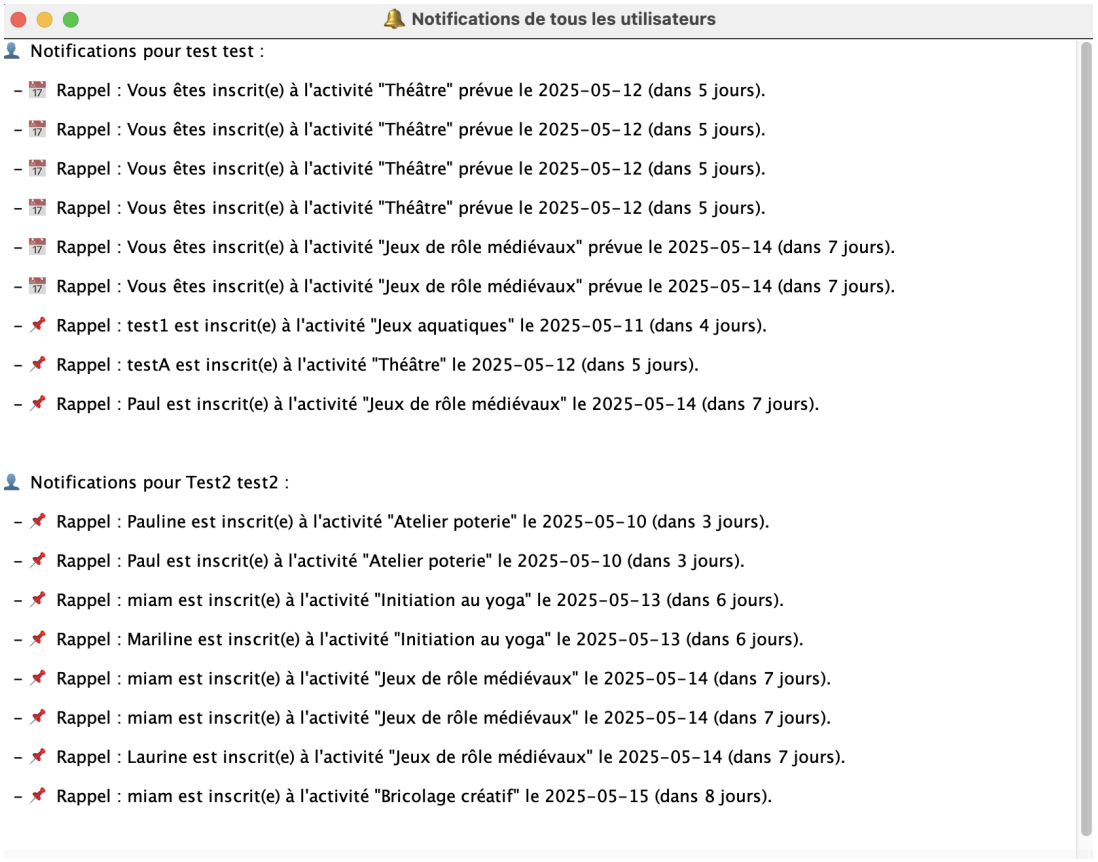
- **Consulter les inscriptions.**



- Description :
  - Tableau listant les inscriptions aux activités.
  - Possibilité de modifier le statut des inscriptions (validé, refusé, etc.).



- **Notifications en attente.**



- **Déconnexion.**

7.6 Navigation entre les écrans

- **Connexion → Inscription** : Accessible via un bouton ou un lien sur l'écran de connexion.
- **Connexion → Tableau de bord** : Après une connexion réussie, redirection vers le tableau de bord correspondant au rôle de l'utilisateur.
- **Tableau de bord → Gestion** : Les boutons du tableau de bord permettent d'accéder aux écrans de gestion (activités, calendrier, etc.).
- **Retour** : Chaque écran dispose d'un bouton ou d'une option pour revenir à l'écran précédent ou se déconnecter.

8. Sécurité

8.1 Gestion des mots de passe

La sécurité des mots de passe est assurée par plusieurs mécanismes :

1. **Règles de complexité** : Les mots de passe doivent respecter les règles RGPD :
    - 12 caractères minimum
    - Au moins une majuscule
    - Au moins une minuscule
    - Au moins un chiffre
    - Au moins un caractère spécial
2. **Hashage avec BCrypt** : Les mots de passe sont hashés avec l'algorithme BCrypt avant d'être stockés dans la base de données. L'implémentation utilisée est fournie par la bibliothèque jbcrypt-0.4.jar.
3. **Validation côté client et serveur** : Les règles de complexité sont vérifiées à la fois dans l'interface utilisateur et lors du traitement côté serveur.

Code de hashage de mot de passe (extrait) :

java

```
// Dans UtilisateurDAO, lors de la création d'un utilisateur
String hashedPassword = BCrypt.hashpw(utilisateur.getMdp(),
BCrypt.gensalt());

utilisateur.setMdp(hashedPassword);
```

8.2 Contrôle d'accès

Le système implémente un contrôle d'accès basé sur les rôles :

1. **Rôles utilisateur** : Trois niveaux d'accès sont définis :
    - Utilisateur standard (Utilisateur)
    - Responsable d'activité
    - Administrateur
2. **Vérification des autorisations** : Chaque action sensible vérifie que l'utilisateur a les droits nécessaires avant de l'exécuter.
3. **Redirection selon le rôle** : Après la connexion, l'utilisateur est redirigé vers l'interface correspondant à son rôle.

Code de redirection basée sur le rôle (extrait) :

java

```
// Dans FenetreConnexion, après authentification réussie
String role = utilisateur.getRole();
if (role.equals("admin")) {
    new FenetreAdmin(utilisateur);
} else if (role.equals("responsable")) {
    new FenetreResponsable(utilisateur);
} else {
    new FenetrePrincipale(utilisateur);
}

this.dispose(); // Ferme la fenêtre de connexion
```

9. Tests et validation

9.1 Tests fonctionnels

Les tests fonctionnels vérifient le bon fonctionnement des différentes fonctionnalités du système :

1. **Tests d'authentification** :
    - Connexion avec identifiants valides
    - Tentative de connexion avec identifiants invalides
    - Création de compte avec informations valides
    - Tentative de création de compte avec email déjà utilisé
    - Tentative de création de compte avec mot de passe non conforme
2. **Tests de gestion des activités** :
    - Création d'une activité
    - Modification d'une activité
    - Suppression d'une activité
    - Consultation des activités par tranche d'âge
3. **Tests d'inscription** :
    - Inscription à une activité compatible avec l'âge
    - Tentative d'inscription à une activité incompatible avec l'âge
    - Validation d'une inscription par un responsable
    - Refus d'une inscription par un responsable
4. **Tests des notifications** :
    - Génération de notification lors d'une inscription
    - Génération de rappel d'activité

9.2 Tests de base de données

Les tests de base de données vérifient l'intégrité et la cohérence des données :

1. **Tests des contraintes** :
    - Unicité de l'email utilisateur
    - Validation des clés étrangères
    - Incrémentation automatique des identifiants
2. **Tests des triggers** :
    - Mise à jour du nombre d'enfants lors de l'ajout d'un enfant
    - Mise à jour du nombre d'enfants lors de la suppression d'un enfant
    - Création de notification lors d'une inscription
3. **Tests de l'événement de rappel** :
    - Génération correcte des rappels 15 jours avant les activités

10. Déploiement

10.1 Prérequis système

Pour exécuter l'application, les prérequis suivants sont nécessaires :

1. **Java Runtime Environment (JRE)** :
    - Version 17 ou supérieure
    - Configuration correcte des variables d'environnement
2. **MySQL** :
    - Version 8.0 ou supérieure
    - Serveur MySQL configuré avec les droits d'accès appropriés
3. **Bibliothèques** :
    - mysql-connector-j-9.1.0.jar
    - jbcrypt-0.4.jar

10.2 Guide d'installation

1. **Préparation de la base de données** :
    - Créer une base de données MySQL nommée camp_activites2
    - Exécuter le script SQL fourni pour créer les tables et les triggers
    - Vérifier que l'événement de rappel est activé

2. **Configuration de l'application** :
    - Vérifier les paramètres de connexion à la base de données dans DatabaseConnection.java
    - Compiler l'application avec les bibliothèques nécessaires
3. **Lancement de l'application** :
    - Exécuter la classe Main.java
    - L'interface de connexion devrait s'afficher

11. Manuels d'utilisation

11.1 Manuel de l'Administrateur

🔧 Comment ajouter une activité

1. Connectez-vous à l'application avec vos identifiants administrateur.
2. Accédez à la section « Gérer les activités » depuis le tableau de bord administrateur.
3. Dans l'interface de gestion des activités :
    - Nom de l'activité : Saisissez le nom de l'activité (ex. : « Atelier de yoga »).
    - Description : Fournissez une brève description.
    - Âge minimum : Indiquez l'âge minimum requis.
    - Âge maximum : Indiquez l'âge maximum autorisé.
4. Vérifiez que tous les champs sont correctement remplis pour éviter les erreurs de validation.
5. Cliquez sur « Ajouter » pour enregistrer l'activité.
6. Un message de confirmation s'affichera pour indiquer que l'activité a bien été ajoutée.
7. L'activité apparaîtra alors dans la liste des activités disponibles et pourra être planifiée.

🛠️ Comment modifier ou supprimer une activité

1. Dans la section « Gérer les activités », localisez l'activité à modifier ou supprimer.
2. Pour modifier :
    - Sélectionnez l'activité dans la liste.
    - Mettez à jour les champs souhaités (nom, description, tranche d'âge).
    - Cliquez sur « Enregistrer les modifications ».
3. Pour supprimer :
    - Sélectionnez l'activité concernée.
    - Cliquez sur « Supprimer ».
    - Confirmez la suppression dans la fenêtre de dialogue.

11.2 Manuel de l'Utilisateur

📝 Comment créer un compte

1. Ouvrez l'application et accédez à l'écran « Créer un compte ».
2. Remplissez tous les champs requis :
    - Email : Adresse email valide.
    - Mot de passe : Doit contenir :
        - Minimum 12 caractères
        - Une majuscule, une minuscule, un chiffre, un caractère spécial
    - Confirmation du mot de passe : Saisir de nouveau le mot de passe.
    - Nom : Prénom et nom.
    - Âge : Sélectionnez votre âge (minimum 18 ans).
    - Téléphone : Numéro de téléphone valide.
    - Genre : Choisissez parmi Homme / Femme / Autre.
    - Handicap : Cochez la case si concerné (facultatif).
3. Vérifiez l'exactitude des informations.
4. Cliquez sur « Valider » pour envoyer le formulaire.
5. Si tous les champs sont valides, un message de confirmation s'affiche et votre compte est créé.
6. Vous serez redirigé vers l'écran de connexion.

🔐 Comment se connecter

1. Saisissez votre email et mot de passe sur l'écran de connexion.
2. Cochez « Afficher le mot de passe » si vous souhaitez le visualiser.
3. Cliquez sur « Connexion ».
4. Si les identifiants sont corrects, vous accédez à l'interface principale.

🎯 Comment s'inscrire à une activité

1. Connectez-vous à votre compte.
2. Rendez-vous dans la section « Inscription à une activité ».
3. Choisissez une activité dans le menu déroulant correspondant à votre tranche d'âge.
4. Cliquez sur « S'inscrire ».
5. Un message de confirmation apparaîtra si l'inscription est réussie. Si votre âge ne correspond pas à l'activité, un message d'erreur s'affichera.

12.Code crée

12.1.1- Création des tickets côté utilisateur

Description générale

La fonctionnalité de création et de gestion des tickets permet aux utilisateurs de signaler des incidents ou des problèmes via une interface dédiée. Ces tickets sont ensuite accessibles par les responsables, qui peuvent les consulter et les traiter.

Création des tickets côté utilisateur

L'utilisateur peut créer un ticket via un bouton dans l'interface principale. Ce bouton ouvre une boîte de dialogue où l'utilisateur peut entrer une description du problème.

Le bouton pour créer un ticket est défini dans la classe **FenetrePrincipale**. Voici le code correspondant :

```
private void creerIncident(ActionEvent e) {

    String description = JOptionPane.showInputDialog(this, "Entrez la description de
l'incident :");

    if (description == null || description.trim().isEmpty()) {

        JOptionPane.showMessageDialog(this, "La description ne peut pas être vide.",
"Erreur", JOptionPane.ERROR_MESSAGE);

        return;

    }

    try {

        String sql = "INSERT INTO tickets (utilisateur_id, description) VALUES (?, ?)";

        PreparedStatement stmt = connection.prepareStatement(sql);

        stmt.setInt(1, utilisateur.getId());

        stmt.setString(2, description);

        stmt.executeUpdate();

        JOptionPane.showMessageDialog(this, "Incident créé avec succès !");

    } catch (SQLException ex) {

        ex.printStackTrace();

        JOptionPane.showMessageDialog(this, "Erreur lors de la création de l'incident.",
"Erreur", JOptionPane.ERROR_MESSAGE);
```

```
    }

}
```

- Étapes principales :
    1. Une boîte de dialogue (JOptionPane) demande à l'utilisateur d'entrer une description.
    2. Si la description est vide, un message d'erreur est affiché.
    3. Une requête SQL insère le ticket dans la table tickets avec l'ID de l'utilisateur et la description.

12.1.2- Fonctionnalités de la page Responsable

La page dédiée au rôle de Responsable permet de gérer plusieurs aspects de l'application, notamment les activités, le calendrier, les tickets d'incidents, les inscriptions et les notifications. Voici une description détaillée des fonctionsnalités disponibles dans cette interface.

---

1. Gestion des Activités

Le responsable peut gérer les activités proposées dans le camp, notamment :

- Ajouter une nouvelle activité.
- Modifier une activité existante.
- Supprimer une activité.

Code associé

La gestion des activités est implémentée dans la classe **FenetreGestionActivites**. Voici les actions principales :

- Ajouter une activité : Une boîte de dialogue demande les informations nécessaires (nom, description, âge minimum et maximum).
- Modifier une activité : Permet de modifier les informations d'une activité sélectionnée.
- Supprimer une activité : Supprime une activité et ses dépendances (calendrier, évaluations).

---

2. Gestion du Calendrier

Le responsable peut gérer les créneaux horaires des activités :

- Ajouter un créneau pour une activité.
- Visualiser tous les créneaux existants.

Code associé

La gestion du calendrier est implémentée dans la classe **FenetreCalendrier**. Les fonctionnalités incluent :

- Ajout d'un créneau : Sélection d'une activité, définition des dates de début et de fin, et du lieu.
- Affichage du calendrier : Liste des créneaux existants avec leurs détails (activité, lieu, horaires).

---

3. Gestion des Tickets d'Incidents

Le responsable peut consulter et gérer les tickets soumis par les utilisateurs :

- Afficher la liste des tickets.
- Traiter ou supprimer un ticket.

Code associé

La gestion des tickets est implémentée dans la classe **FenetreTicketsIncidents**. Les fonctionnalités incluent :

- Affichage des tickets : Liste des tickets avec les informations suivantes :

- ID du ticket.
- Nom et prénom de l'utilisateur.
- Description du problème.
- Date de création.

---

4. Gestion des Inscriptions

Le responsable peut consulter et gérer les inscriptions aux activités :

- Afficher la liste des inscriptions (utilisateurs et enfants).
- Modifier le statut d'une inscription (À valider, Validé, Refusé).

Code associé
La gestion des inscriptions est implémentée dans la classe **FenetreInscriptionsActivite**. Les fonctionnalités incluent :

- Affichage des inscriptions : Liste des inscriptions avec les informations suivantes :
  - Activité.
  - Utilisateur (nom, type : adulte ou enfant).
  - Statut de l'inscription.
- Modification du statut : Permet de valider ou refuser une inscription.

---

5. Gestion des Notifications

Le responsable peut consulter les notifications en attente, comme :

- Nouveaux tickets soumis.
- Modifications dans les activités ou le calendrier.

Code associé

La gestion des notifications est implémentée dans la classe **FenetreNotifications**. Les fonctionnalités incluent :

- Affichage des notifications : Liste des notifications importantes pour le responsable.

---

6. Déconnexion

Le responsable peut se déconnecter de son compte et revenir à l'écran de connexion.

Code associé

La déconnexion est gérée dans la classe **FenetreResponsable**. Lorsqu'il clique sur le bouton "Se déconnecter", la fenêtre actuelle est fermée et l'écran de connexion est affiché.

Résumé des Boutons dans l'Interface Responsable

Voici les boutons disponibles dans l'interface et leurs actions associées :

1. 📋 **Gérer les Activités :** Ouvre la fenêtre de gestion des activités.
2. 📅 **Gérer les Créneaux / Calendrier :** Ouvre la fenêtre de gestion du calendrier.
3. 🛠️ **Gérer les Tickets / Incidents :** Ouvre la fenêtre de gestion des tickets.
4. 👨‍👩‍👧‍👦 **Voir les Inscriptions :** Ouvre la fenêtre de gestion des inscriptions.
5. 🔔 **Notifications en attente** : Affiche les notifications importantes.
6. 🚪 **Se déconnecter** : Déconnecter le responsable et retourne à l'écran de connexion.

Code Principal de la Page Responsable

Voici un extrait du code de la classe `FenetreResponsable` qui gère ces fonctionnalités :

```
public class FenetreResponsable extends JFrame {
    public FenetreResponsable(Connection connexion, Utilisateur utilisateur) {
        super("Espace Responsable");
        setSize(600, 450);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout());
        // Titre
        JLabel titre = new JLabel("Tableau de bord du Responsable", JLabel.CENTER);
        titre.setFont(new Font("Arial", Font.BOLD, 22));
        add(titre, BorderLayout.NORTH);
        // Panneau des boutons
        JPanel panelBoutons = new JPanel(new GridLayout(6, 1, 10, 10));
        JButton boutonGestionActivites = new JButton("📋 Gérer les Activités");
        JButton boutonGestionCalendrier = new JButton("📅 Gérer les Créneaux / Calendrier");
        JButton boutonTickets = new JButton("🛠️ Gérer les Tickets / Incidents");
        JButton boutonVoirInscriptions = new JButton("👨‍👩‍👧‍👦 Voir les Inscriptions");
        JButton boutonNotifications = new JButton("🔔 Notifications en attente");
        JButton boutonDeconnexion = new JButton("🚪 Se déconnecter");
        panelBoutons.add(boutonGestionActivites);
        panelBoutons.add(boutonGestionCalendrier);
        panelBoutons.add(boutonTickets);
        panelBoutons.add(boutonVoirInscriptions);
        panelBoutons.add(boutonNotifications);
        panelBoutons.add(boutonDeconnexion);
        add(panelBoutons, BorderLayout.CENTER);
        // Actions des boutons
        boutonGestionActivites.addActionListener(e -> new FenetreGestionActivites(connexion));
        boutonGestionCalendrier.addActionListener(e -> new FenetreCalendrier(connexion));
        boutonTickets.addActionListener(e -> new FenetreTicketsIncidents(connexion));
        boutonVoirInscriptions.addActionListener(e -> new FenetreInscriptionsActivite(connexion));
        boutonNotifications.addActionListener(e -> new FenetreNotifications(connexion));
        boutonDeconnexion.addActionListener(e -> {
            dispose();
            new FenetreConnexion(connexion);
        });
        setVisible(true);
    }
}
```

---

13. Annexes

13.3 Extraits de code clés

13.3.1 Connexion à la base de données

java

```
import java.sql.Connection;
```

```java
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    public static Connection getConnection() throws SQLException {
        String url = "jdbc:mysql://localhost:3306/camp_activites2";
        String user = "root";
        String password = ""; // À modifier selon la configuration
        return DriverManager.getConnection(url, user, password);
    }
}
```

13.3.2 Validation du mot de passe

java

```java
public boolean validatePassword(String password) {
    String regex =
"^(?=.*[A-Z])(?=.*[a-z])(?=.*\\d)(?=.*[@$!%*?&])[A-Za-z\\d@$!%*?&]{12,}$";
    return password.matches(regex);
}
```

13.3.3 Inscription à une activité

java

```
public boolean inscrireUtilisateur(int utilisateurId, int calendrierId) {
    try (Connection conn = DatabaseConnection.getConnection()) {
        // Récupérer l'activité associée au calendrier
        String sqlGetActivite = "SELECT a.activite_id, a.age_min, a.age_max FROM
activite a " +
                "JOIN calendrier c ON a.activite_id = c.activite_id " +
                "WHERE c.calendrier_id = ?";
        PreparedStatement pstmtGet = conn.prepareStatement(sqlGetActivite);
        pstmtGet.setInt(1, calendrierId);
        ResultSet rs = pstmtGet.executeQuery();

        if (rs.next()) {
            int activiteId = rs.getInt("activite_id");
            int ageMin = rs.getInt("age_min");
            int ageMax = rs.getInt("age_max");

            // Récupérer l'âge de l'utilisateur
            String sqlGetAge = "SELECT age FROM utilisateur WHERE utilisateur_id =
?";
            PreparedStatement pstmtAge = conn.prepareStatement(sqlGetAge);
            pstmtAge.setInt(1, utilisateurId);
            ResultSet rsAge = pstmtAge.executeQuery();

            if (rsAge.next()) {
                int age = rsAge.getInt("age");

                // Vérifier si l'âge correspond à la tranche d'âge de l'activité
                if (age >= ageMin && age <= ageMax) {
                    // Insérer l'inscription
                    String sqlInsert = "INSERT INTO inscription_activite
(calendrier_id, utilisateur_id) VALUES (?, ?)";
                    PreparedStatement pstmtInsert =
conn.prepareStatement(sqlInsert);
                    pstmtInsert.setInt(1, calendrierId);
                    pstmtInsert.setInt(2, utilisateurId);
                    int result = pstmtInsert.executeUpdate();
                    return result > 0;
                }
            }
        }
        return false;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
```

13.3.4 Événement de rappel d'activité

sql

```
CREATE EVENT rappel_activite
ON SCHEDULE EVERY 1 DAY STARTS CURRENT_TIMESTAMP
ON COMPLETION NOT PRESERVE ENABLE
DO
BEGIN
    DECLARE aujourd_hui DATE;
    SET aujourd_hui = CURDATE();
```

```
    INSERT INTO notification (utilisateur_id, message)
    SELECT ia.utilisateur_id,
           CONCAT('Rappel : ', u.nom, ' vous êtes inscrit à l'activité "', a.nom,
'" le ',
                  DATE_FORMAT(c.debut, '%d/%m/%Y à %H:%i'))
    FROM inscription_activite ia
    JOIN calendrier c ON ia.calendrier_id = c.calendrier_id
    JOIN activite a ON c.activite_id = a.activite_id
    JOIN utilisateur u ON ia.utilisateur_id = u.utilisateur_id
    WHERE DATEDIFF(c.debut, aujourd_hui) = 15;

    INSERT INTO notification (enfant_id, message)
    SELECT ia.enfant_id,
           CONCAT('Rappel : ', e.prenom, ' ', e.nom, ' est inscrit à l'activité "',
a.nom, '" le ',
                  DATE_FORMAT(c.debut, '%d/%m/%Y à %H:%i'))
    FROM inscription_activite ia
    JOIN calendrier c ON ia.calendrier_id = c.calendrier_id
    JOIN activite a ON c.activite_id = a.activite_id
    JOIN enfant e ON ia.enfant_id = e.enfant_id
    WHERE DATEDIFF(c.debut, aujourd_hui) = 15;

END;
```

13.3.5 Authentification utilisateur

java

```
public Utilisateur authentifier(String email, String mdp) {
    try (Connection conn = DatabaseConnection.getConnection()) {
        String sql = "SELECT * FROM utilisateur WHERE email = ?";
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setString(1, email);

        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            String hashedPassword = rs.getString("mdp");

            // Vérifier le mot de passe avec BCrypt
            if (BCrypt.checkpw(mdp, hashedPassword)) {
                Utilisateur utilisateur = new Utilisateur();
                utilisateur.setUtilisateurId(rs.getInt("utilisateur_id"));
                utilisateur.setEmail(rs.getString("email"));
                utilisateur.setNom(rs.getString("nom"));
                utilisateur.setPrenom(rs.getString("prenom"));
                utilisateur.setAge(rs.getInt("age"));
                utilisateur.setNombreEnfants(rs.getInt("nombre_enfants"));
                utilisateur.setHandicap(rs.getBoolean("handicap"));
                utilisateur.setSexe(rs.getString("sexe"));
                utilisateur.setTelephone(rs.getString("telephone"));
                utilisateur.setRole(rs.getString("role"));

                return utilisateur;
            }
        }
        return null;
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}
```

13.3.6 Création d'un ticket d'incident

java

```
public boolean creerTicket(int utilisateurId, String description) {
    try (Connection conn = DatabaseConnection.getConnection()) {
        String sql = "INSERT INTO tickets (utilisateur_id, description) VALUES (?,
?)";
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setInt(1, utilisateurId);
        stmt.setString(2, description);

        int result = stmt.executeUpdate();
        return result > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
```

13.3.7 Redirection basée sur le rôle

java

```
// Dans la classe FenetreConnexion, méthode de connexion
private void btnConnexionActionPerformed(java.awt.event.ActionEvent evt) {
    String email = txtEmail.getText();
    String motDePasse = new String(txtMotDePasse.getPassword());

    UtilisateurDAO utilisateurDAO = new UtilisateurDAO();
    Utilisateur utilisateur = utilisateurDAO.authentifier(email, motDePasse);

    if (utilisateur != null) {
        // Redirection selon le rôle
        switch (utilisateur.getRole()) {
            case "admin":
                new FenetreAdmin(utilisateur).setVisible(true);
                break;
            case "responsable":
                new FenetreResponsable(utilisateur).setVisible(true);
                break;
            default:
                new FenetrePrincipale(utilisateur).setVisible(true);
                break;
        }
        this.dispose(); // Ferme la fenêtre de connexion
    } else {
        JOptionPane.showMessageDialog(this,
                "Email ou mot de passe incorrect",
                "Erreur d'authentification",
                JOptionPane.ERROR_MESSAGE);
    }
}
```

13.3.8 Validation de l'âge pour une activité

java

```
public boolean verifierAgeActivite(int age, int activiteId) {
    try (Connection conn = DatabaseConnection.getConnection()) {
        String sql = "SELECT age_min, age_max FROM activite WHERE activite_id = ?";
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setInt(1, activiteId);
```

```
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                int ageMin = rs.getInt("age_min");
                int ageMax = rs.getInt("age_max");

                return (age >= ageMin && age <= ageMax);
            }
            return false;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
```