



Les Toiles de Minuit

Gestionnaire d'évènements

Projet réalisé dans le cadre du test technique « Les Toiles de Minuit »

Réalisé par Alicia Kellai

Date : Septembre 2025

Sommaire

1. Introduction

- 1.1. Contexte et objectifs
- 1.2. Fonctionnalités principales

2. Installation

- 2.1. Récupération du projet
- 2.2. Prérequis
- 2.3. Installation du backend (Symfony)
- 2.4. Installation du frontend (Flutter)

3. Utilisation de l'application

- 3.1. Authentification
- 3.2. Différences entre rôles

4. Base de données

- 4.1. Modèle conceptuel (MCD) et tables
- 4.2. Gestion des relations

5. Architecture du projet

- 5.1. Communication frontend / backend
- 5.2. Structure

1. Introduction

1.1. Contexte et objectifs

Le projet « *Les Toiles de Minuit* » a été réalisé dans le cadre d'un test technique.

L'objectif est de développer une application web permettant de gérer des événements via un **système d'authentification par rôles**.

L'application doit offrir une expérience utilisateur simple, intuitive et sécurisée tout en respectant un design **sobre et moderne**, rappelant les couleurs de l'identité visuelle du projet (*bleu nuit* et *doré*).

Les objectifs principaux sont donc :

- Concevoir une interface conviviale permettant la consultation et la gestion des événements.
- Mettre en place une authentification robuste avec gestion des permissions.
- Fournir une architecture claire séparant le **frontend** (Flutter) et le **backend** (Symfony + API REST).
- Garantir la sécurité des données stockées en base MySQL.

1.2. Fonctionnalités principales

L'application propose plusieurs fonctionnalités essentielles :

- **Authentification sécurisée** : connexion via email et mot de passe, protégée par JWT.
- **Gestion des évènements** :
 - Création avec formulaire et validation des champs requis.
 - Modification avec formulaire prérempli.
 - Suppression avec confirmation.
 - Consultation d'une liste d'événements sous forme de cartes modernes et interactives.
- **Gestion des rôles** :
 - **Administrateur (ROLE_ADMIN)** : création, modification, suppression et consultation des événements.
 - **Utilisateur (ROLE_USER)** : consultation en lecture seule uniquement.
- **Interface utilisateur** :
 - Design minimaliste et élégant, avec rappels de couleurs bleu nuit et doré.
 - Navigation fluide et adaptée aux écrans réduits comme aux grands formats.

2. Installation

2.1. Récupération du projet

Avant toute installation, il vous faut récupérer le code source de l'application. Deux méthodes sont possibles :

- **Méthode 1** : Télécharger en .zip (recommandée si vous n'utilisez pas Git)
 1. Rendez-vous sur la page GitHub du projet : <https://github.com/aliciakellai/event-app>
 2. Cliquez sur le bouton vert **Code** en haut à droite.
 3. Sélectionnez **Download ZIP**.
 4. Une fois le fichier téléchargé, **décompressez-le** (clic droit -> extraire tout ou décompresser).
 5. Placez le dossier extrait dans l'emplacement de votre choix (par exemple dans *Documents/*).
- **Méthode 2** : Cloner avec Git (si Git est installé sur votre machine)
 1. Ouvrez un terminal.
 2. Tapez les commandes suivantes :

```
git clone https://github.com/aliciakellai/event-app.git|
cd event-app|
```

Quelle que soit la méthode que vous avez utilisée, vous avez maintenant une copie locale du projet. En voici la structure :

```
event-app/
|___ app/
|       |___ event_app/      #Frontend Flutter
|___ backend/
|       |___ symfony/        #Backend Symfony
|___ docs/                    #Documentation complète
|___ README.md                #Présentation et guide rapide
```

2.2. Prérequis

Avant de pouvoir utiliser l'application, assurez-vous que votre environnement de travail contient les outils suivants.

- Composer (gestionnaire de dépendances PHP, utilisé par Symfony)
- Téléchargement : <https://getcomposer.org/download>
- Vérification post-installation dans le terminal avec la commande : `composer -V`
- Exemple de sortie attendue : `Composer version 2.7.7 2024-08-12 11:09:12`

- Symfony CLI (pour lancer facilement le serveur Symfony)
- Téléchargement : <https://symfony.com/download>
- Vérification post-installation dans le terminal avec la commande : `symfony -v`
- Exemple de sortie attendue : `Symfony CLI version v5.8.12 (2025-01-10T12:34:56+00:00)`

- PHP 8.2 ou supérieur
- Téléchargement : <https://www.php.net/downloads>
- Vérification post-installation dans le terminal avec la commande : `php -v`
- Exemple de sortie attendue : `PHP 8.2.12 (cli) (built: Oct 7 2024 12:34:56) (NTS)`

- Docker & Docker Compose (pour lancer MySQL et phpMyAdmin en containers)
- Téléchargement : <https://docs.docker.com/get-docker>
- Vérification post-installation dans le terminal avec les commandes : `docker -v`
`docker compose version`
- Exemples de sorties attendues : `Docker version 27.0.3, build abc123`
`Docker Compose version v2.29.2`

- Flutter SDK (pour le frontend en Flutter Web)
- Téléchargement : <https://docs.flutter.dev/get-started/install>

- Vérification post-installation dans le terminal avec la commande :
`flutter --version`
- Exemple de sortie attendue :
`Flutter 3.24.3 • channel stable • https://github.com/flutter/flutter.git`
- Node.js + npm (utile si Symfony utilise encore Webpack ou pour certains outils complémentaires)
- Téléchargement : <https://nodejs.org/>
- Vérification post-installation dans le terminal avec les commandes : `node -v`
`npm -v`
- Exemples de sorties attendues : `v.20.17.0` `10.8.2`

2.3. Installation du backend (Symfony)

- Après avoir récupéré le projet (voir section 2.1.), placez-vous dans le dossier backend : `cd backend/symfony/` (via le terminal).
- Installez les dépendances avec Composer. `composer install` (via le terminal).
- Configurez les variables d'environnement. Copiez l'exemple fourni :
`cp .env.example .env` (via le terminal)
- Ouvrir le fichier `.env` dans un éditeur de texte et modifier la ligne de connexion à la base de données (exemple avec MySQL) :
`DATABASE_URL="mysql://root:password@127.0.0.1:3306/event_app"`
`root` étant votre identifiant MySQL, `password` votre mot de passe défini pour MySQL et `event_app` le nom de la base de données.
- Créez la base de données. Une fois `.env` configuré, exécutez la commande :
`php bin/console doctrine:database:create`
- Exemple de sortie attendue :
`Created database 'event_app' for connection named default`
- Appliquez les migrations pour créer les tables nécessaires (utilisateurs, événements etc.) : `php bin/console doctrine:migrations:migrate`
- Exemple de sortie attendue :
`WARNING! You are about to execute a migration in database "event_app".`
`Are you sure you wish to continue? (yes/no) [yes]:`
Tapez **yes** puis validez.
- Pour tester l'application, vous devez avoir un compte admin.

Important : Un compte admin et un compte user ont été préconfigurés. Vous pouvez les utiliser si vous ne souhaitez pas créer de compte.

Email : admin@test.com, Mot de passe : admin

Email : user@test.com, Mot de passe : user

- Si vous souhaitez créer votre compte, commencez par générer un mot de passe hashé avec la commande :

```
php bin/console security:hash-password
```

Entrez votre mot de passe et Symfony vous retournera le hash. Conservez-le.

- Ajoutez un utilisateur directement dans la base de données via **phpMyAdmin** (<http://localhost:8081> si vous utilisez Docker) ou via SQL :

```
INSERT INTO user (email, roles, password) VALUES  
( 'admin@test.com', '[ "ROLE_ADMIN" ]', 'HASH_GENERE_ICI' );
```

Quelque soit la méthode avec laquelle vous créez votre compte, prenez soin d'entrer votre **mot de passe hashé** (que le terminal vous a renvoyé précédemment).

- Lancez le serveur Symfony avec la commande : `symfony server:start`

- Exemple de sortie attendue : `[OK] Web server listening`

- Le backend est maintenant disponible sur <http://127.0.0.1:8000/api>

2.4. Installation du frontend (Flutter)

- Placez-vous dans le dossier Flutter de l'application : `cd app/event_app`
- Installez les dépendances du projet avec la commande : `flutter pub get`
- Exemple de sortie attendue : `Resolving dependencies... Got dependencies !`
 - Exécutez la commande : `cp .env.example .env`
 - Modifiez la variable **FLUTTER_API_URL** pour pointer vers le backend :
`FLUTTER_API_URL=http://127.0.0.1:8000/api`

Important : Si vous testez en local, utilisez <http://127.0.0.1:8000/api>. Si vous déployez sur un serveur, mettez l'URL publique (ex : <https://votre-domaine/api>)

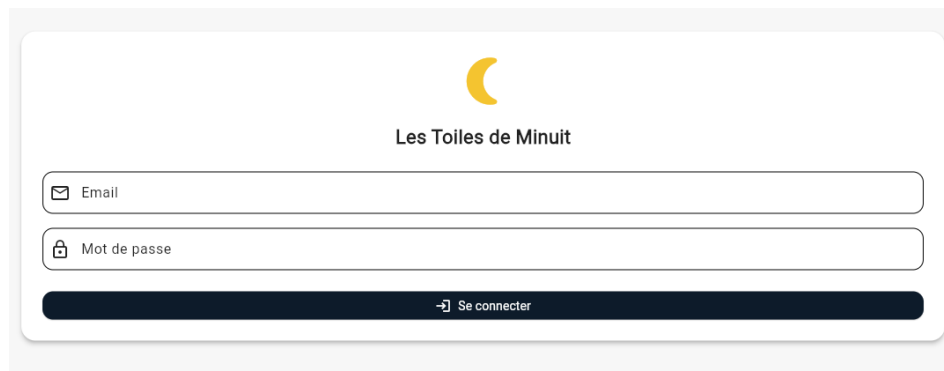
- Lancez l'application flutter avec Chrome : `flutter run -d chrome`
- L'application s'ouvre automatiquement (~ 10s) sur **Google Chrome** à l'adresse <http://localhost:5000> .

3. Utilisation de l'application

3.1. Authentification

L'accès à l'application est sécurisé grâce à un système d'authentification basé sur **JWT (JSON Web Token)**. Chaque utilisateur doit se connecter avec son **adresse email** et son **mot de passe** afin d'obtenir un jeton de session qui sera utilisé automatiquement pour toutes les requêtes vers l'API.

- Connexion :
- Au lancement de l'application, l'utilisateur arrive sur **la page de connexion**. Deux champs sont obligatoires : email (adresse enregistrée dans la base de données), et mot de passe (mot de passe associé à ce compte, haché dans la base de données).

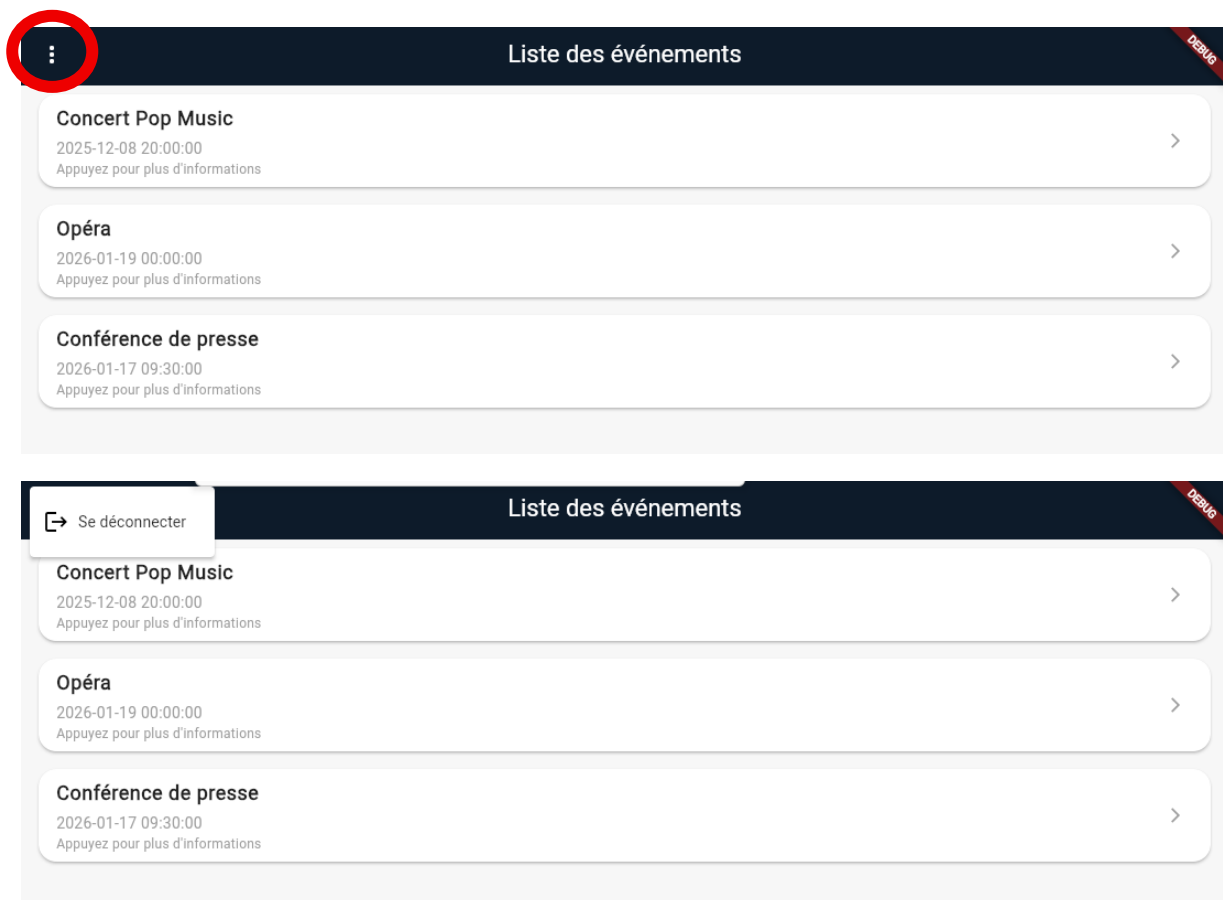
The image shows a login form for an application. At the top center is a yellow crescent moon icon. Below it is the text "Les Toiles de Minuit". The form consists of two input fields: the first is labeled "Email" with an envelope icon, and the second is labeled "Mot de passe" with a lock icon. Below these fields is a dark blue button with a white right-pointing arrow and the text "Se connecter".

Page de connexion

- L'utilisateur saisit ses identifiants puis clique sur **Se connecter**. Si les informations sont correctes, il est redirigé vers **la liste des évènements**. Si l'adresse email est inconnue **ou** si le mot de passe est incorrect, un message d'erreur s'affiche sous le champ concerné.

Message d'erreur lors de la connexion

- Déconnexion :
- Une fois connecté, l'utilisateur peut se déconnecter via le menu situé en haut à gauche de la page des événements. Lors de la déconnexion, **le jeton JWT** est supprimé de la mémoire de l'application. L'utilisateur est automatiquement redirigé vers la page de connexion.



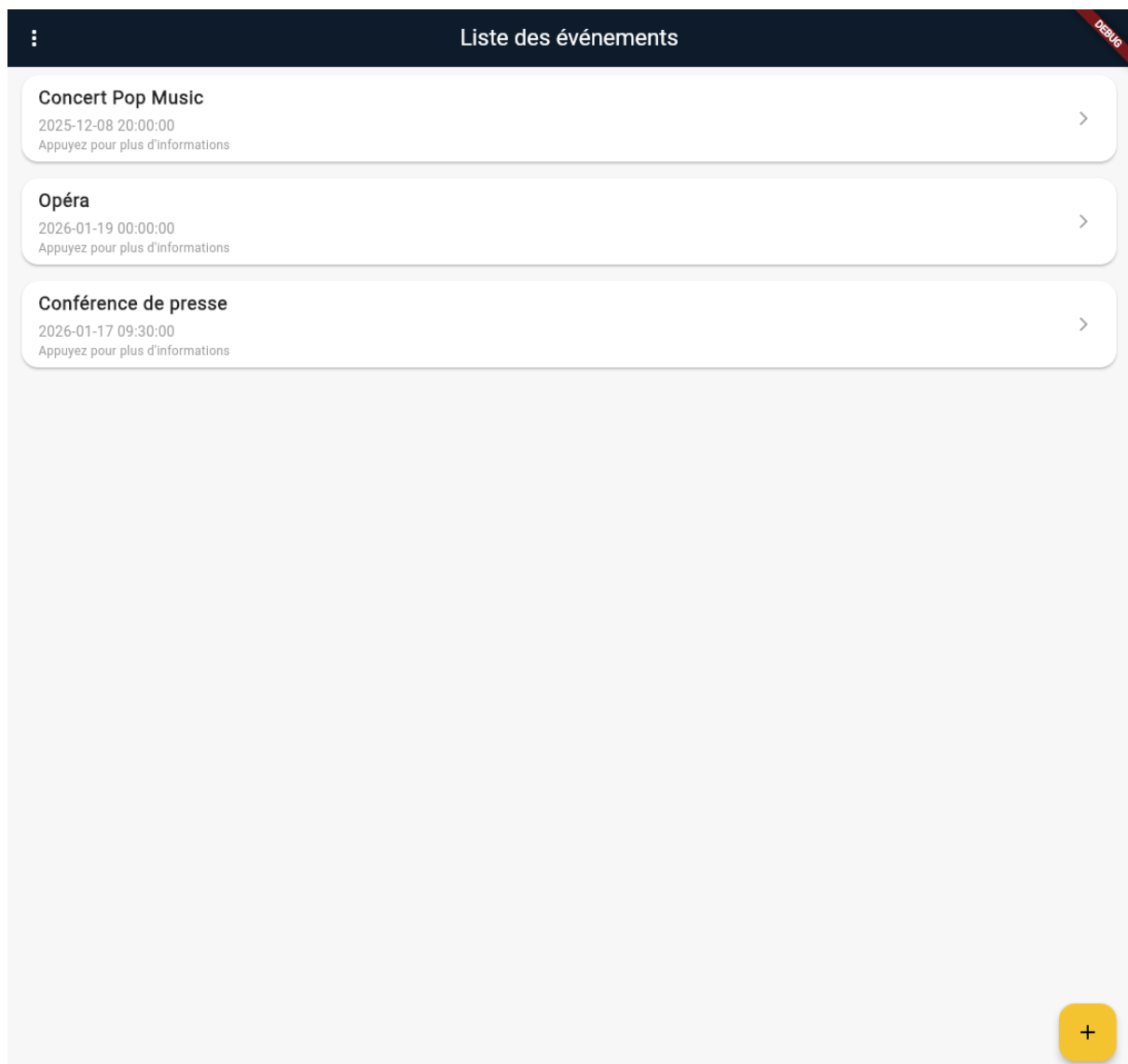
Icône de déconnexion

- Cas particuliers :
- Si un utilisateur essaie d'accéder directement à une page protégée (par exemple via l'URL) sans être connecté, il sera automatiquement redirigé vers la page de connexion.
- En cas d'expiration du jeton JWT, l'utilisateur devra simplement se reconnecter.

3.2. Différences entre rôles

L'application distingue deux types d'utilisateurs grâce à un système de rôles attribués lors de la création des comptes.

- **Rôle Administrateur** ([« ROLE_ADMIN »])
 - Peut accéder à toutes les fonctionnalités de gestion des événements (CRUD : Create, Read, Update, Delete en anglais).
 - L'admin peut **créer un évènement** depuis le bouton « + » jaune en bas à droite :



← Créer un événement

Titre

Description

Date (format: YYYY-MM-DD HH:mm:ss)

Créer

Page de création d'un Admin

- Il peut **modifier un évènement existant** depuis la page de détails :

← Concert Pop Music

Concert Pop Music

Dress code : blanc

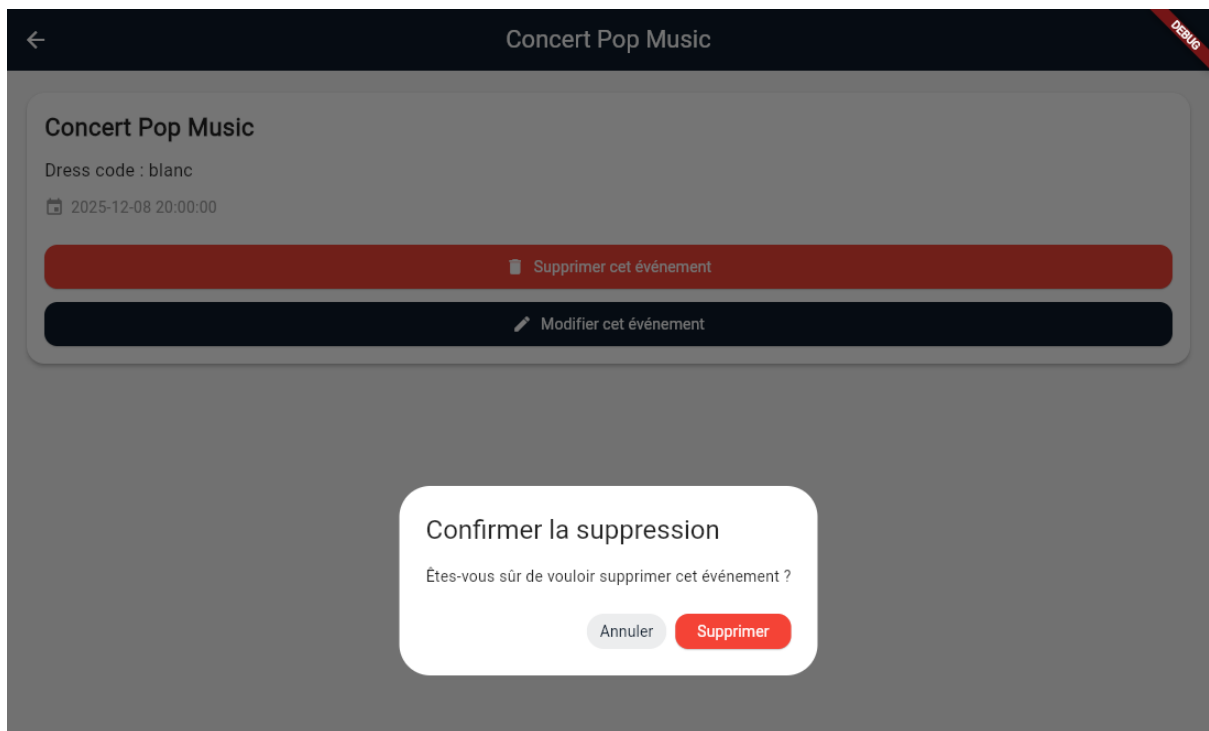
📅 2025-12-08 20:00:00

🗑 Supprimer cet événement

✎ Modifier cet événement

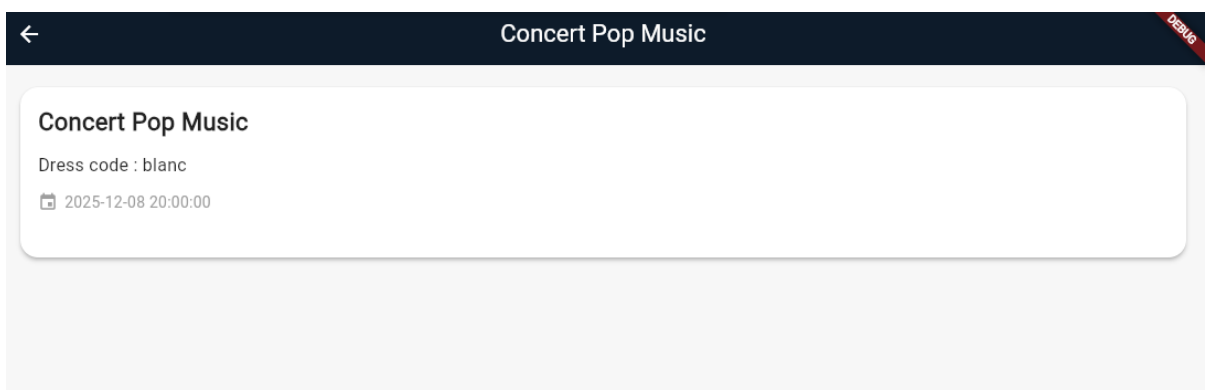
Page de détails d'un Admin

- Il peut également **supprimer un évènement** (confirmation requise avant suppression définitive) :

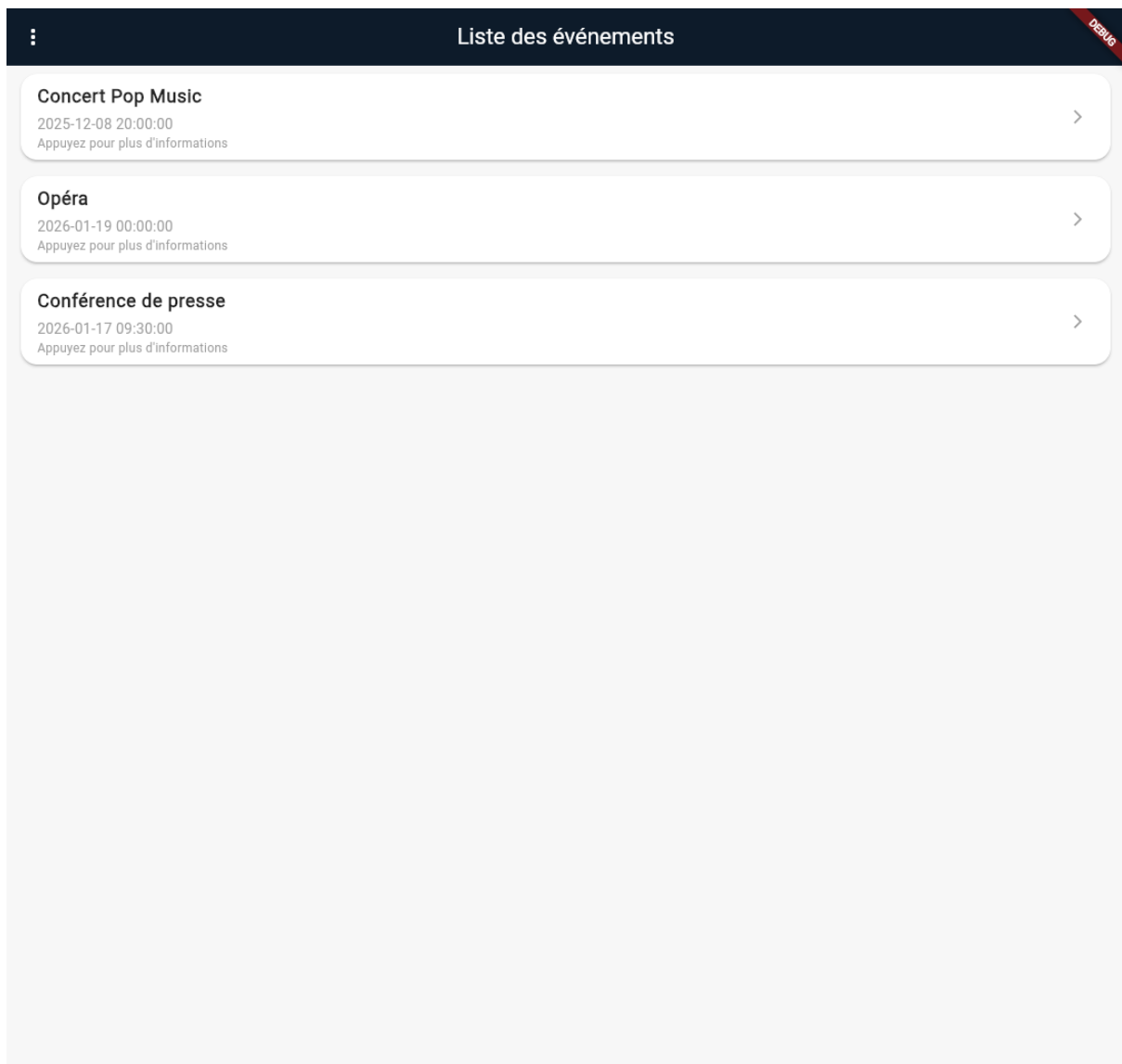


Page de suppression d'un Admin

- C'est le rôle utilisé pour gérer le contenu de la plateforme.
 - **Rôle Utilisateur** ([« ROLE_USER »])
- Dispose uniquement d'un accès en lecture seule.
- Il peut **consulter la liste** des évènements, et en **consulter les détails**.
- Il n'a pas accès aux boutons Créer / Modifier / Supprimer.



Page de détails d'un User



Page d'accueil d'un User

4. Base de données

4.1. Modèle conceptuel (MCD) et tables

Le MCD est une représentation abstraite des entités manipulées par l'application.

Dans ce projet, deux entités principales sont présentes :

- **User** : représente un utilisateur de l'application.

Attributs : id, email, password, roles.

- **Event** : représente un événement.

Attributs : id, title, description, date, createdBy.

Représentation des tables (MySQL) :

- Table **User** :

Colonne	Type	Description
id	INT (PK, auto)	Identifiant unique de l'utilisateur
email	VARCHAR(180)	Adresse email (unique)
roles	JSON	["ROLE_ADMIN"] ou bien ["ROLE_USER »]
password	VARCHAR(255)	Mot de passe haché

- Table **Event** :

Colonne	Type	Description
id	INT (PK, auto)	Identifiant unique de l'évènement
title	VARCHAR(255)	Titre de l'évènement
description	TEXT(nullable)	Description détaillée
date	DATETIME	Date et heure de l'évènement
Created_By	INT(FK -> user.id)	Identifiant de l'utilisateur créateur (admin)

4.2. Gestions des relations

- **Relation 1.N** entre user et event :
 - Un utilisateur peut créer plusieurs événements.
 - La colonne created_by de event est une **clé étrangère** pointant vers user.id.
- **Contraintes d'intégrité :**
 - Si un utilisateur est supprimé, les événements créés peuvent être supprimés en cascade (selon la config Symfony / Doctrine).
 - L'email est unique pour garantir qu'aucun doublon d'utilisateur n'existe.

5. Architecture du projet

5.1. Communication frontend / backend

L'application repose sur une architecture **client-serveur** :

- **Frontend (client)** : développé en Flutter (Web). Il est exécuté dans le navigateur et s'occupe de l'affichage et des interactions utilisateur.
- **Backend (serveur)** : développé en Symfony (API REST). Il gère la logique métier, la base de données et l'authentification.

a) Les requêtes HTTP

Toutes les interactions passent par des **requêtes HTTP** envoyées depuis le frontend via la classe ApiService (dans le dossier app/event_app/lib/).

b) Authentification via JWT

Lorsqu'un utilisateur se connecte :

- Symfony génère un **token JWT** et le renvoie au frontend.
 - Ce token est stocké côté Flutter (en mémoire).
 - Pour chaque requête suivante, Flutter ajoute le token dans l'entête HTTP :
Authorization: Bearer <token>
 - Symfony vérifie la validité du token avant de répondre :
- Si le token est valide → l'utilisateur est identifié et son rôle est reconnu.
 - Si le token est invalide/expiré → Symfony renvoie une erreur (401 Unauthorized).

c) Gestion des rôles dans l'échange

- Les **utilisateurs** peuvent seulement effectuer des requêtes GET /api/events.
- Les **admins** peuvent accéder aux routes protégées (POST, PUT, DELETE).
- Le contrôle se fait côté **backend** grâce aux rôles stockés dans la base (ROLE_USER, ROLE_ADMIN).

d) Résumé du flux

- L'utilisateur interagit avec l'interface Flutter.
- Flutter envoie une requête HTTP au backend via ApiService.
- Symfony reçoit la requête, vérifie les permissions grâce au JWT.
- Symfony accède à la base MySQL et renvoie une réponse JSON.
- Flutter met à jour l'interface en fonction de la réponse.

5.2. Structure

Le projet est organisé en trois grandes parties :

```
event-app/
├── app/                                # Frontend Flutter (interface utilisateur)
│   ├── event_app/
│   │   ├── lib/                        # Code source principal de l'application Flutter
│   │   │   ├── main.dart              # Point d'entrée de l'application
│   │   │   ├── api_service.dart       # Communication avec l'API Symfony
│   │   │   ├── login_page.dart        # Page de connexion (authentification)
│   │   │   ├── create_event_page.dart # Formulaire de création d'événements
│   │   │   ├── edit_event_page.dart   # Formulaire de modification
│   │   │   ├── event_detail_page.dart # Détails + suppression d'un événement
│   │   │   └── ...                    # Autres pages utilitaires
│   │   └── pubspec.yaml               # Dépendances du projet Flutter
│   └── ...
├── backend/                           # Backend Symfony (API REST)
│   ├── symfony/
│   │   ├── config/                   # Configuration de Symfony (routes, services, sécurité)
│   │   ├── migrations/              # Migrations de la base de données
│   │   ├── src/                      # Code source principal (contrôleurs, entités, sécurité)
│   │   │   ├── Controller/          # Contrôleurs API (EventController, AuthController...)
│   │   │   ├── Entity/              # Entités (Event, User...)
│   │   │   ├── Repository/          # Accès aux données
│   │   │   └── Security/             # Gestion JWT et authentification
│   │   ├── .env.example              # Exemple de configuration (copie vers .env)
│   │   └── composer.json             # Dépendances PHP
│   └── ...
├── docs/                              # Documentation complète (Word/PDF rendu final)
└── README.md                          # Présentation générale + guide rapide
```

a) Organisation du Frontend (Flutter)

- **main.dart** : point d'entrée de l'application, gestion du thème, redirection selon l'état de connexion.
- **api_service.dart** : centralise toutes les requêtes HTTP vers le backend (login, CRUD événements).
- **Pages** (login_page.dart, create_event_page.dart, edit_event_page.dart, event_detail_page.dart) :
 - Chaque page correspond à une étape spécifique de l'application (connexion, création, modification, visualisation).

- **pubspec.yaml** : gère les dépendances Flutter (http, intl, dotenv...).

b) Organisation du Backend (Symfony)

- **Controllers** : exposent les routes de l'API REST (/api/login, /api/events, etc.).
- **Entities** : définissent les modèles de données (Event, User).
- **Repositories** : font le lien entre les entités et la base MySQL.
- **Security** : gère l'authentification via JWT.
- **Config** : contient les routes, la config de sécurité, etc.

c) Organisation des fichiers transversaux

- **README.md** : explications rapides pour lancer le projet.
- **docs/** : documentation complète pour le jury.
- **.env / .env.example** : configuration des variables sensibles (non versionnées sur GitHub).