Rapport de projet : Eventfinder

Web Avancé - S.Malandain, M.Minelli

Justin Foltz, Maxime Hutinet

5 janvier 2020

Table des matières

1 Introduction

Dans le cadre d'un travail pratique pour le cours de Web avancé, nous avons créé une API grâce à *ExpressJS* ainsi qu'un front-end permettant de consommer ces données.

Le présent document a plusieurs objectifs :

1. Présenter le projet ainsi que ses fonctionnalités
2. Détailler son architecture
3. Détailler les différentes routes fournis par l'API
4. Expliquer la manière dont lancer le projet

2 Présentation du projet

2.1 Le projet

Nous avons décidé de créer une site permettant aux utilisateurs de pouvoir trouver des évènements autour d'eux et de pouvoir les ajouter dans une liste de favoris.

Afin de récupérer les données liés aux évènements, nous avons utilisé l'API du site Eventful. Ce site dispose d'un grand nombre de données pour des événements au niveau mondial.

2.2 Les fonctionnalités

Voici les fonctionnalités offertes par le site :

— Visualisation de concert/festival sur une carte
— Visualisation des détails d'un évènement en cliquant sur un marqueur
— Sauvegarde/suppression d'un évènement dans son profil
— Visualisation du profil d'un autre utilisateur
— Visualisation des évènements dans son propre profil
— Inscription au site
— Authentification sur le site
— Ajout d'évènements à son profil depuis le profil d'un autre
— Recherche de ville pour pouvoir visualiser les évènements aux alentours
— Recherche de profil d'utilisateur

3 Architecture

Afin de faciliter la portabilité et le déploiement du projet, nous avons fait le choix d'utiliser des containers pour pouvoir faire fonctionner notre projet.

Il y a ainsi deux containers :

— Web : contenant notre API ainsi que le front-end
— DB : contenant notre base de donnée MongoDB

Le diagramme représentant l'architecture générale du projet est rapporté sur la Figure 1. Les deux containers sont reliés par un réseau privé, créé via Docker. Seule l'API web est exposée sur le port 8080, mappé sur le port 80 de la machine hôte. Pour fonctionner, nous faisons appel à deux API externes, via le code JavaScript s'exécutant côté client :

— API Eventful : fournit les événements à partir d'une latitude, d'une longitude et d'un rayon
— API Leaflet : fournit la carte sur laquelle sont affichés les événements sous forme de marqueurs.

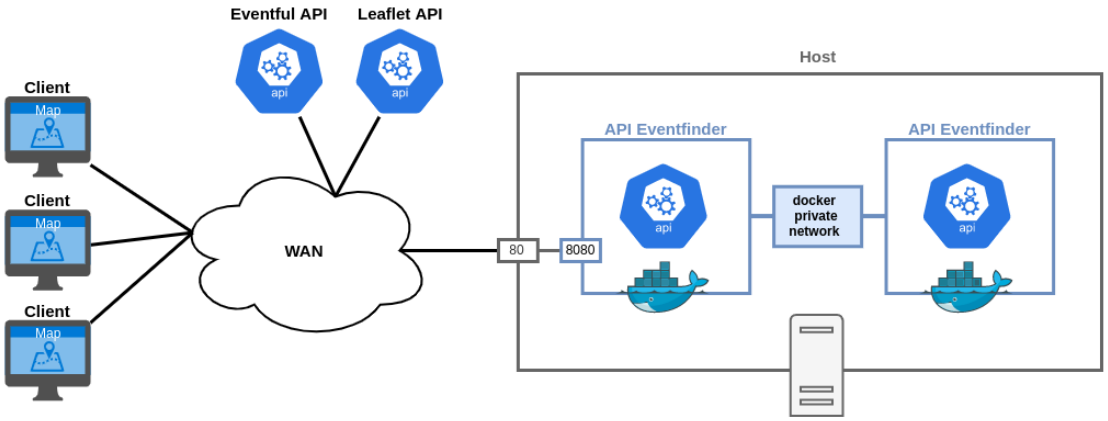

FIGURE 1 – Architecture générale du projet

4 API

Notre API dispose des routes suivantes :

```
GET /register
```

Envoi fichier du *register.html*.

```
POST /register
```

Ajoute un utilisateur au site. Le champ `id` doit être unique dans toute la base de donnée.

Paramètres :

Nom	Type	Description
username	String	ID de l'utilisateur
name	String	Nom de l'utilisateur
pass	String	Mot de passe de l'utilisateur

```
GET /login
```

Envoi du fichier *login.html*.

```
POST /login
```

Vérifie si un utilisateur a le droit de ce connecter à l'application. Si oui, un token est créé et renvoyé à l'utilisateur pour maintenir sa session.

Paramètres :

Nom	Type	Description
username	String	ID de l'utilisateur
pass	String	Mot de passe de l'utilisateur

```
GET /logout
```

Deconecte l'utilisateur en supprimant son token JWT.

```
GET /map
```

Envoi du fichier *map.html*.

```
GET /profil
```

Retourne l'ID et le nom de l'utilisateur courant.

Paramètres :

Nom	Type	Description
username	String	ID de l'utilisateur
name	String	Nom de l'utilisateur

```
GET /profil/favorite
```

Retourne une liste des évènements favoris de l'utilisateur courant.

```
POST /profil/event/:eventID
```

Ajoute un évènement à la liste des favoris de l'utilisateur courant.

Paramètres :

Nom	Type	Description
eventID	String	ID de l'évènements

```
DELETE /profil/event/:eventID
```

Supprime un évènement de la liste des favoris de l'utilisateur courant.

Paramètres :

Nom	Type	Description
eventID	String	ID de l'évènements

```
GET /event/:latitude/:longitude/:radius
```

Retourne une liste d'évènements

Paramètres :

Nom	Type	Description
latitude	String	latitude du lieu
longitude	String	longitude du lieu
radius	String	rayon dans lequel chercher (en KM)

```
GET /profil/names/:name
```

Retourne une liste de profil utilisateur matchant un mot clé

Paramètres :

Nom	Type	Description
name	String	Utilisateur recherché

```
GET /profil/:name
```

Retourne le profil de l'utilisateur matchant un mot clé

Paramètres :

Nom	Type	Description
name	String	Utilisateur recherché

```
POST /profil/edit/check
```

Vérifie si un ID d'utilisateur est libre.

Paramètres :

Nom	Type	Description
username	String	ID d'Utilisateur à vérifier

```
POST /profil/edit/activate
```

Vérifie le mot de passe de l'utilisateur courant afin d'activer l'édition de son mot de passe.

Paramètres :

Nom	Type	Description
pass	String	Mot de passe du l'utilisateur

```
POST /profil/edit
```

Applique les modifications de profil demandées par l'utilisateur.

Paramètres :

Nom	Type	Description
username	String	ID de l'utilisateur
name	String	Nom de l'utilisateur
pass	String	Mot de passe de l'utilisateur

5 Lancement du projet

Afin de faciliter le lancement du projet, nous avons mis en place un fichier docker-compose.

5.1 Pré-requis

Pour pouvoir lancer ce projet, il est nécessaire d'avoir :

— Docker
— Docker-compose

Il faut également s'assurer que le Docker engine est bien actif.

5.2 Lancement

Le lancement du projet est relativement simple, il suffit d'utiliser la commande suivante à la racine de notre projet :

```
docker-compose up -d
```

Docker-compose va ainsi créer les images, les lancer sur des containers et également créer un réseau pour que les containers puissent échanger des informations.

Il suffit ensuite simplement d'ouvrir un navigateur et d'aller sur `localhost` ou `127.0.0.1`

6 Conclusion