Rithi John Chau Benjamin Petiot Juliette Taupin

Architecture et micro-services

E5FI

Introduction

Un projet permettant de faire de nouvelles rencontres en étant fan de vélo.

Lancement du projet :

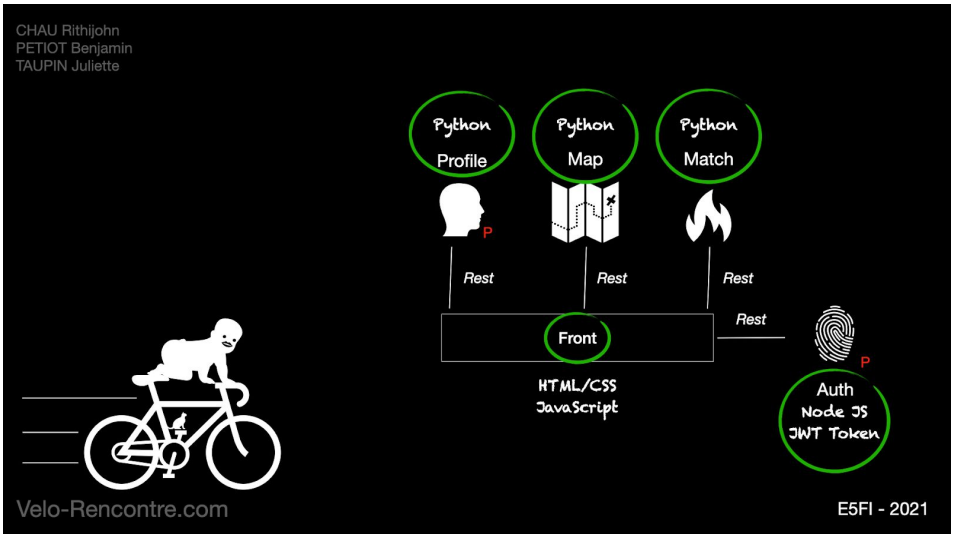
docker-compose up

Le site est accessible via l'url :

http://localhost/

Les 5 microservices ci-dessous sont disponibles et mis en place dans le projet.
- Map
- Profile
- Match
- Front
- Authentification

Notre projet suit le schéma d'architecture suivant :



Docker

Chaque microservice possède son Dockerfile permettant la mise en place de leur environnement.
A la base du projet se trouve le fichier docker-compose qui permet d'exécuter nos différentes applications (de nos différents containers).

Nous avons mis en place un volume pour le service front afin de pouvoir sauvegarder les fichiers sur notre container et donc éviter d'avoir à relancer celui-ci en cas de modifications.

Services

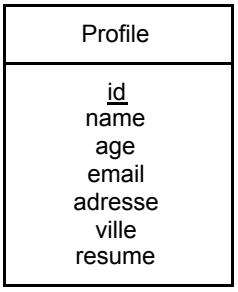
Profile

Le service profile est codé en python et sur le port 5001. Il s'agit d'une api rest qui utilise flask et est connecté à une base de données SQlite. La base de données n'est pas persistante mais cela ne pose pas de problème, nous l'avons créée avec des données par défaut, le but n'étant pas de créer un micro service pour la persistance des BDD.

Le but de l'API rest est de relier la BDD et notre front et de gérer des fonctions comme l'ajout, la récupération et la modification de données.

Côté front, le service est disponible à l'url : *http://localhost/profil/profile.html*
(Si l'utilisateur est connecté).

BDD :



Map

Le service profile est codé en python et sur le port 5000. Il s'agit d'une api rest qui utilise flask ainsi que plusieurs librairies tel que folium pour générer des cartes. Le principe est de récupérer l'adresse de l'utilisateur et l'adresse d'un autre utilisateur et d'afficher sur une carte OpenStreetMap la position des utilisateurs.

Dans notre cas, elle affiche la position de l'utilisateur connecté et une position définie à l'avance car on ne récupère pas l'adresse d'un autre utilisateur mais l'api est fonctionnelle et pourrait très bien le faire si nous adaptons le front et améliorons la fonction match. L'objectif n'étant pas de coder des micro services poussés nous nous sommes arrêté la.

Match

Le service Match est mis en place sur le port 5005 et est développé en python (flask) et fait appel à l'API OpenStreetMap afin de pouvoir récupérer les données souhaitées.

Le but est de récupérer tous les utilisateurs présents dans le même département que celui de l'utilisateur connecté et de les afficher.

Pour ce faire nous faisons un appel à notre API profile afin de récupérer tous les utilisateurs en excluant celui qui est connecté. Dans un second temps nous récupérons grâce à OpenStreetMap le département dans lequel chacun de ses utilisateurs habitent, nous le comparons ensuite au département de l'utilisateur connecté et affichons les résultats de ceux qui sont bien présents dans le même secteur.

Côté front, le service est disponible à l'url : *http://localhost/match/match.html*

Il suffit d'appuyer sur le bouton 'Matcher', attendre que les données se traitent et visionner le résultat plus bas dans la page.

Authenticate

Le service Authenticate est codé en javascript avec Node JS et Express (sur le port 5004). Il s'agit d'une api rest qui est connecté à une base de données SQlite également non persistante.

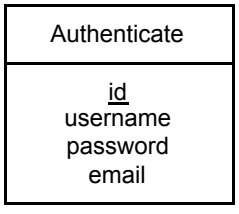
Le but de l'API est de mettre en place un système d'authentification basé sur le JSON Web Token (avec l'algorithme HS256 pour le hachage) qui est stocké en localStorage (sur la session du navigateur de l'utilisateur). Sa seule fonction est de créer ce jwtoken et de vérifier la connexion d'un utilisateur. Ce sont donc les autres micro services qui vont récupérer et vérifier la validité du token en cas de requête.

Exemple d'utilisateur déjà inscrit :

- username : smercier
- password : pwd1

L'ajout d'un utilisateur (sign up) est opérationnel et synchronisé avec la bdd présente dans le micro service de profile. Il est donc possible une fois créé, de se connecter avec et d'accéder à son profil et ses matchs.

BDD :



Côté front, le service est disponible à l'url : *http://localhost/connexion/connexion.html*

Front

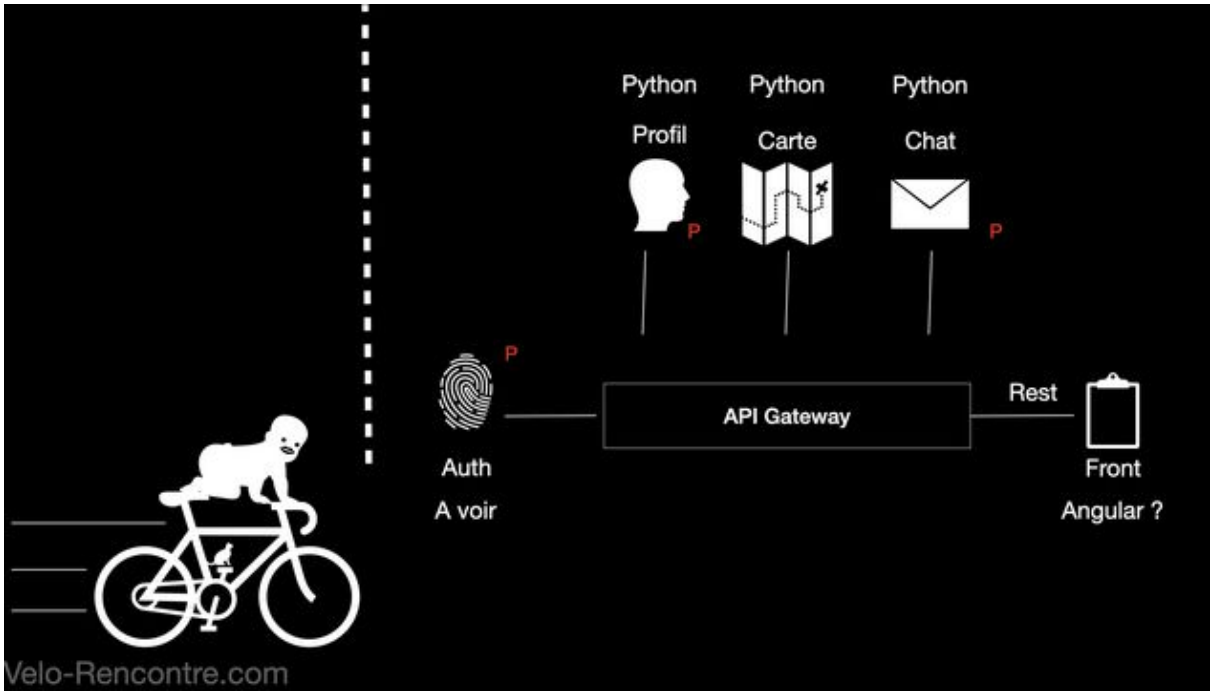
Le front a été réalisé en HTML, CSS et JS sur le port 80. Nous avons utilisé d'autres outils tels que du bootstrap ou de l'ajax. Le front se divise en différentes parties :

- Page d'accueil
    - La page d'accueil contient une introduction sur notre application ainsi que sur nous avec une magnifique mise en page et une pincée d'humour
- Profile
    - La page profil contient les informations sur l'utilisateur connecté actuellement. Elle est reliée à notre api profile et on peut éditer le profil si on le désire. Nous notons également son esthétique à couper le souffle.
- Match
    - La partie match est reliée à l'api match, il suffit de cliquer sur le bouton "matcher" pour voir toutes les personnes de son département inscrites. Nous ne commenterons pas la mise en page.
- Se connecter
    - La page nous ouvre une fenêtre élégante et fluide reliée à l'API d'authentification qui permet la connexion et la création de compte, rien à redire à part que c'est à couper le souffle.

Le service est disponible à l'url : *localhost*

Présentation orale de février

A la base, nous nous étions basé sur le schéma d'architecture ci-dessous :



Les microservices mis en place à ce moment là étaient :

- Map
- Profile (avec sa BDD)

Les dockerfiles et le docker-compose étaient également fonctionnels.
Le service d'authentification était en cours de développement.

Nous voulions mettre en place un service qui aurait relié tous les autres (avec 'l'API Gateway') mais il a été décidé que cela n'était pas utile et que notre front allait se charger de ça.