JACQUOT Thierry
EDOUARD Lucas
MIESCH Nathanaël

Projet WEBS

Master MIAGE M1
Année 2024-2025

Table des matières :

Introduction

Ce document présente le projet WEBS, un site web qui permet de visualiser des données de santé.

Dans ce site web, il est possible de visualiser les données de santé des patients. Ce traitement est effectué par une composante serveur, qui lit les données des patients et les renvoie vers la composante. Cette composante serveur peut exécuter des processus en parallèle. La composante client se charge ainsi de recevoir les données reçues par la composante serveur et de les afficher à l'écran.

Ce document possède l'introduction que vous lisez présentement, les choix de conception des deux composantes, le développement de l'application que ce soit au niveau frontend ou au niveau backend, les tests effectués et la conclusion auquel vous pouvez retrouver ainsi un résumé des fonctionnalités accomplies et non accomplies.

Conception et présentation des choix effectués

Le site a été conçu avec Flask, en utilisant une version non modifiée du démon fourni avec le sujet qui est en C++.

Composante serveur

Une partie du sujet est traitée par le backend permettant de communiquer entre le logiciel démon et le frontend de l'application.

Nous avons ainsi décidé de créer un wrapper autour de ce logiciel, qui permet au logiciel en C++ d'être accessible, de pouvoir créer des processus en parallèle et de gérer les éventuelles erreurs du côté serveur.

La partie serveur n'est composée que du logiciel démon et que du wrapper développé en Python.

Composante client

L'autre partie de l'application est le front-end de l'application web permettant de visualiser les données du patient.

Cette partie client possède deux fonctionnalités :

- Affichage de données des patients
- Appel vers la partie serveur via des requêtes asynchrones

Cette partie sera gérée avec le framework Flask, disponible en Python, qui fait office de serveur web. Flask est le moyen que nous avons pour relier la composante client à la composante serveur.

Nous avons donc Flask avec une structure HTML/CSS/JS simple. Avec ce choix, nous avons voulu prendre soin de garder le site web la plus légère possible afin de permettre au site d'être le plus rapide possible.

Développement

Pour commencer le développement, nous avons mis en place les différents outils nécessaires à l'exécution du projet : Python, Flash, GCC.

Nous avons séparé le développement en deux parties principales : la partie backend et la partie client.

Composante serveur

Nous avons commencé à utiliser le logiciel démon avec le backend. Nous avons simplement tenté de lire le contenu du fichier daemon. Cependant, nous sommes très vite partis sur la création de processus secondaires puisque l'exécution du processus démon bloquait le processus Python en cours.

Nous avons donc développé le moyen de paralléliser des processus avec cette fonction :
Nous utilisons avec la bibliothèque subprocess de Python la fonction subprocess.Popen, qui permet de créer des processus en parallèle. Nous écrivons ainsi sur subprocess.PIPE qui nous permet de récupérer les données du processus démon lancé :

```
command = ['./daemon', str(args['arg1']), str(args['arg2'])]

process = subprocess.Popen(
        command,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE
)
```

Nous traitons par après les données reçues à l'aide d'une boucle jusqu'à fin de l'exécution du processus démon :

```
while True:
    line = process.stdout.readline()
    if line:
        last_line = line.strip()
        outputs.append(last_line)
    elif last_line:
        outputs.append(last_line)
    time.sleep(args['delay_ms'] / 1000.0)

    if process.poll() is not None:
        break
```

Enfin, nous avons géré les éventuelles erreurs. Nous avons traité le cas où le processus se ferme de manière inattendue avec un code supérieur à 0, même si ce cas est rare. Nous avons donc une fonction complète pour gérer

Ainsi, nous avons donc une composante serveur simple mais fonctionnelle permettant de transmettre les données.

Composante client

Lors de la conception, nous avons fait le constat qu'il était très conseillé d'utiliser le framework Flask que nous utilisons.

Le développement de la composante client a débuté par un fichier Flask dont l'un des buts est de pouvoir créer deux routes dans l'application :

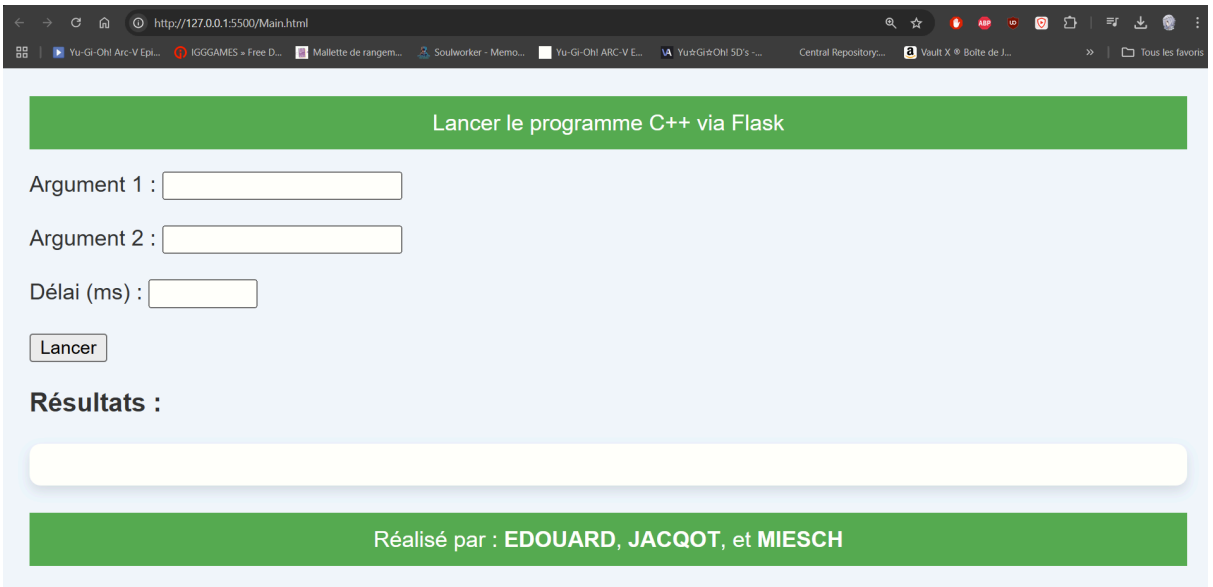
- Une route d'accueil auquel toutes les fonctionnalités,
- Une route qui appelle la composante serveur.

L'autre but est de pouvoir créer un serveur web utilisable, auquel Flask s'en charge automatiquement.

Dans la page d'accueil, auquel la route est l'accès au site, le framework envoie simplement la page HTML du site. Le site se compose d'un titre, de zones de texte correspondant aux arguments d'entrée (patient et activité physique, délai appliqué), et d'un bouton pour voir les résultats. Une case en dessous est prévue à cet effet.

La page d'appel de la composante serveur permet de récupérer la requête du client vers le serveur, en vérifiant les données transmises par le client (auquel cas nous renvoyons une erreur si les données sont erronées). Si les données sont valables, nous pouvons ainsi exécuter la fonction du serveur client et attendre le retour des données.

Le résultat final de l'interface utilisateur est la suivante :



Tests effectuées

Nous avons effectué les tests suivants :

- Nous avons effectué des tests empiriques en utilisant une page web de test. L'idée est de tester les fonctionnalités sur cette page, notamment pour tester si le serveur traitait correctement les données
- Nous avons tenté de mesurer la vitesse d'exécution de lectures de données en effectuant plusieurs requêtes à la suite.

Conclusions et perspectives

Nous avons ainsi pu mettre en place ces fonctionnalités suivantes :

- Affichage des données d'un patient sur le côté serveur
- Requêtes du client vers le serveur afin d'obtenir les données client
- Création de multiples processus en parallèle pour lire les données fournies par le daemon
- Gestion des erreurs

Nous n'avons pas réussi à mettre en place cet aspect :

- Tests sur la performance de la solution. Bien que nous avons tenté de quantifier le temps d'exécution, nous n'avons pas eu de méthode de test robuste permettant de regarder précisément le temps d'exécution au temps de l'écriture du rapport.