ARCHITECTURE

TraceIn

SOMMAIRE

I. AVANT-PROPOS

Ce document décrit l'architecture de **TraceIn**.

Ce projet s'inscrit dans le cadre du cours de Réseaux et Protocole dispensé par **Dr DIALLO**. Le but dudit projet est de développer un logiciel de diagnostique implémentant les fonctionnalités de Ping et Traceroute avec différentes options.

Ce dit document qui vise à expliquer le projet implique des choix de technologies et de modèles de décisions quitte à assurer des distribuables de bonne qualités et offrir des codes fiables et faciles à maintenir.

Ce document décrit le processus de réflexion de chaque aspect. Notons que les instructions présentes dans ce document ont été suivi pour l'obtention des distribuables que vous trouverez à ces liens ci-dessous.

Application Mobile :

- Code source : https://github.com/Master1BDGL-NetworkProject/trace_in_mobile_app

- Distribuable : https://github.com/Master1BDGL-NetworkProject/trace_in_mobile_app/releases/download/v1.0/app-armeabi-v7a-release.apk

Application Desktop :

- Code source :
- Distribuable :

Serveur d'API :

- Code source : https://github.com/Master1BDGL-NetworkProject/tracerouteAPI

II. FONCTIONNALITES

A. EXIGENCES FONCTIONNELLES

1. PING :

Cette fonctionnalité requiert :

- L'adresse IP ou le nom d'hôte
- Le nombre de paquets
- La taille du paquet
- Le TTL (Time To Live)
- Le temps mort
-

2. TRACEROUTE:

Pour implémenter celle-ci, il faut :

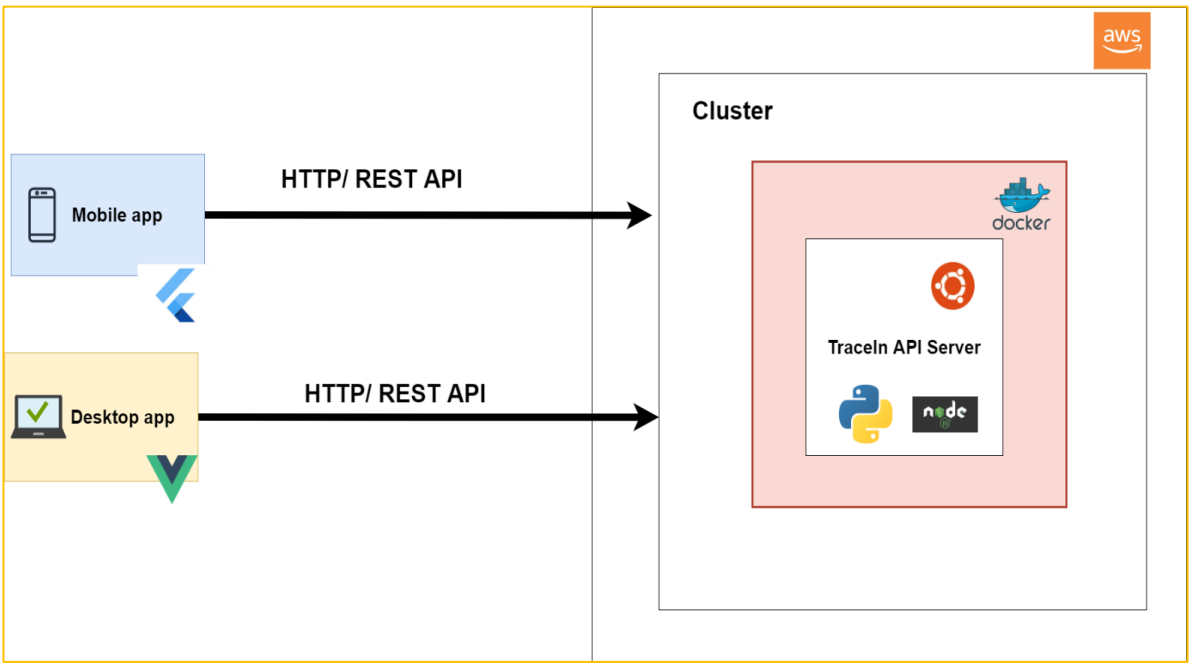
- L'adresse IP ou le nom d'hôte
- Le maximum de houblon
- Le temps d'attente (Time out)
- Le protocole : UDP, ICMP

B. EXIGENCES NON FONCTIONNELLES :

Les distribuables devront être des clients lourds.

III. OVERVIEW DE L'ARCHITECTURE

Vous trouverez ci-dessous le diagramme :



Comme vous pouvez constater sur le diagramme ci-dessus, notre architecture est une architecture deux tiers (client-serveur). Les clients et le serveur communiquent en utilisant le protocole HTTP.

Ladite architecture peut se décomposer en trois composants :

- Un client Desktop (Windows, Linux)
- Un client mobile (Android)
- Un serveur d'API

L'intérêt d'une telle architecture se situe à deux niveaux :

– Permettre à différents clients de communiquer avec le serveur. Ce qui nous donnent l'avantage d'offrir différentes

applications suivant les systèmes Linux, Windows, Android, mac OS
- Une indépendance dans les outils de développement à savoir les langages et les Frameworks utilisés. Ce qui permet d'avoir le choix d'utiliser les technologies qui conviennent.

IV. COMPOSANTS :

A. CLIENT DESKTOP

Technologies utilisées

Pour la réalisation de la version desktop nous avons optée pour le couple VueJS-NeutralinoJS.

VueJS est l'un des Framework Javasript les plus populaires et actuellement le plus côté. Il permet de concevoir des applications single page (SPA) qui offre une réactivité comme celle des applications mobiles.

NeutralinoJS est un Framework javascript qui permet de créer des applications Cross plateforme c'est-à-dire pour Linux, pour Windows et pour mac OS tout en utilisant les technologies du web.

Le choix de ces technologies est d'offrir à chaque membre du groupe l'opportunité de contribuer au projet vu que tous sont déjà des adeptes des technologies du web.

Implémentation

Forte utilisation des principes de **clean Architecture** (**SOLID**) et par donc une forte utilisation des injections de dépendances qui

a avoir un faible couplage et une forte cohésion et donc assurer une bonne maintenance du code

B. CLIENT MOBILE

Technologies utilisées

La réalisation de la version mobile fut faite avec Flutter.

Flutter est un Framework conçu par Google qui permet de concevoir les applications multiplateformes c'est-à-dire des applications Mobile (Android, IOS), Desktop (Linux, Windows, macOS), Web et systèmes embarqués (Raspberry pi). Il est actuellement le Framework mobile le plus en vogue.

Le choix de cette technologie avait pour but d'offrir toujours des applications multiplateformes et se plaçait également comme seconde alternative à l'association VueJS-NeutralinoJS pour la conception de la version Desktop.

Implémentation

Forte utilisation des principes de **clean Architecture** (**SOLID**) et par donc une forte utilisation des injections de dépendances qui a avoir un faible couplage et une forte cohésion et donc assurer une bonne maintenance du code

C. SERVEUR D'API

Technologies utilisées

Ce serveur d'API fut fait en utilisant NodeJS, Python et Docker.

L'utilisation de Python était de faciliter la manipulation de paquets en utilisant la bibliothèque Scapy.

L'utilisation de NodeJS était surtout pour la conception du serveur d'API.

Docker fut utilisé pour le déploiement sur le Cloud (AWS). C'est une technologie de virtualisation.

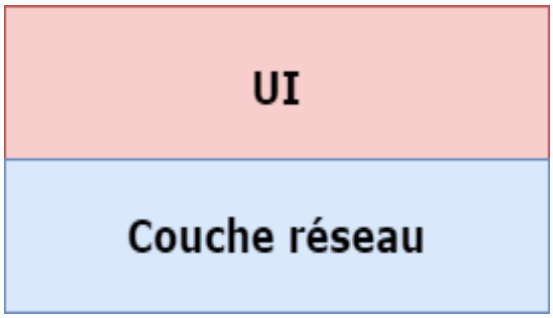
Implémentation

Forte utilisation des principes de **clean Architecture** (**SOLID**) et par donc une forte utilisation des injections de dépendances qui a avoir un faible couplage et une forte cohésion et donc assurer une bonne maintenance du code

V. ARCHITECTURE DES COMPOSANTS

A. Application Desktop

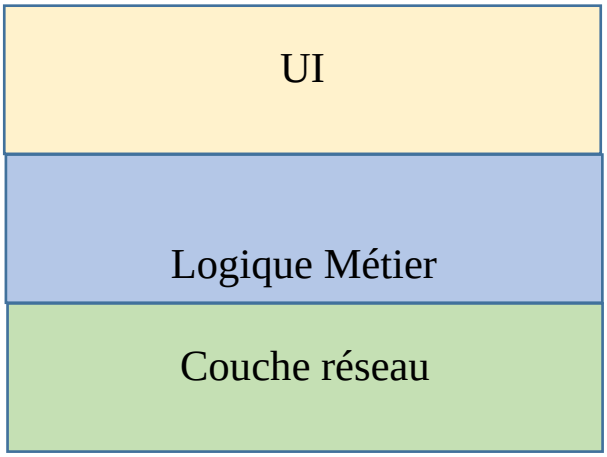
L'application se décompose en deux couches.



UI : est la couche qui contient tout le code nécessaire à la création de l'interface utilisateur

Couche réseau : Cette couche responsable de l'interaction avec le server d'API.

B. Application mobile

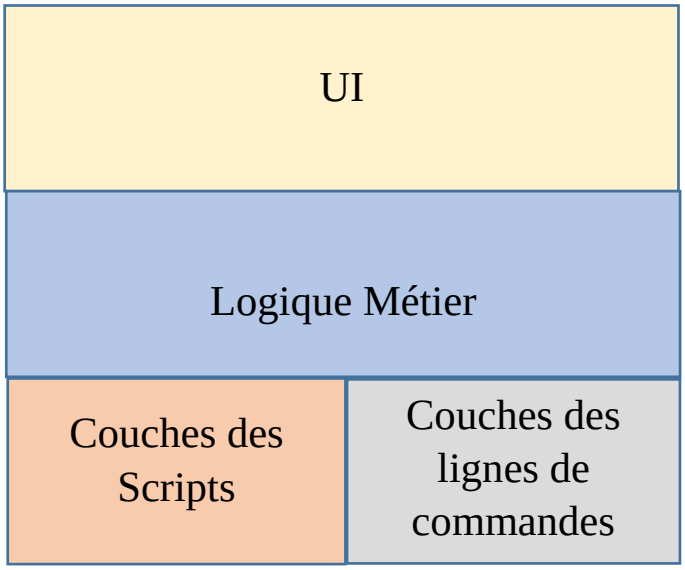


UI : est la couche qui contient tout le code nécessaire à la création de l'interface utilisateur

Logique Métier : Responsable de la validation des données entrées.

Couche réseau : Cette couche responsable de l'interaction avec le server d'API.

C. SERVEUR



UI : Interface API, responsable de recevoir les requêtes HTTP et de les acheminer vers la couche supérieure (Logique Métier).

Logique Métier : Cette couche n'est qu'un ensemble de contrôleurs qui valide les données reçus puis appelle la couche supérieure correspondante selon le type de requête.

- Pour les requêtes PING, les données reçues sont acheminées vers la couche Script
- Pour les requêtes TRACEROUTE, les données reçues sont acheminées vers la couche des lignes de commande.

Cette couche est également responsable du format des réponses à envoyer.

Couches des Scripts : Cette couche ne contient que des scripts Pythons qui permettent d'envoyer des paquets dans le réseau puis les recevoir et les déléguer à la couche Métier

Couche des Lignes de Commande : Cette couche est responsable d'exécuter des lignes de commandes pour les requêtes TRACEROUTE. Elle exécute simplement la commande **inetulis** installée sur le système (dans le container)