Université de Québec À Chicoutimi

6GEN723—Réseaux d'ordinateurs

Rapport de projet de conception

Membres du groupe: KAGONE YASMINE OULIA

TIENDREBEOGO CHICKITA ESTHER

ILBOUDO SOUTONGNOMA HERMAN

INTRODUCTION

Ce présent rapport vise à implémenter une application client-serveur, permettant la communication entre un client et un serveur. Le projet se fera à partir du protocole de communication simplifié basé sur le modèle HTTP. Nous nous chargerons d'implémenter à la fois le client et le serveur de ce protocole, en se concentrant sur la communication via TCP de la couche transport. Dans ce rapport, nous présenterons notre approche de conception et d'implémentation du protocole, ainsi que les décisions prises pendant le processus.

I-CONCEPTION DU PROTOCOLE

Afin de mieux planifier notre conception, nous avons d'abord procéder à la réalisation du diagramme de classe et de cas d'utilisation suivant:

Diagramme de cas d'utilisation

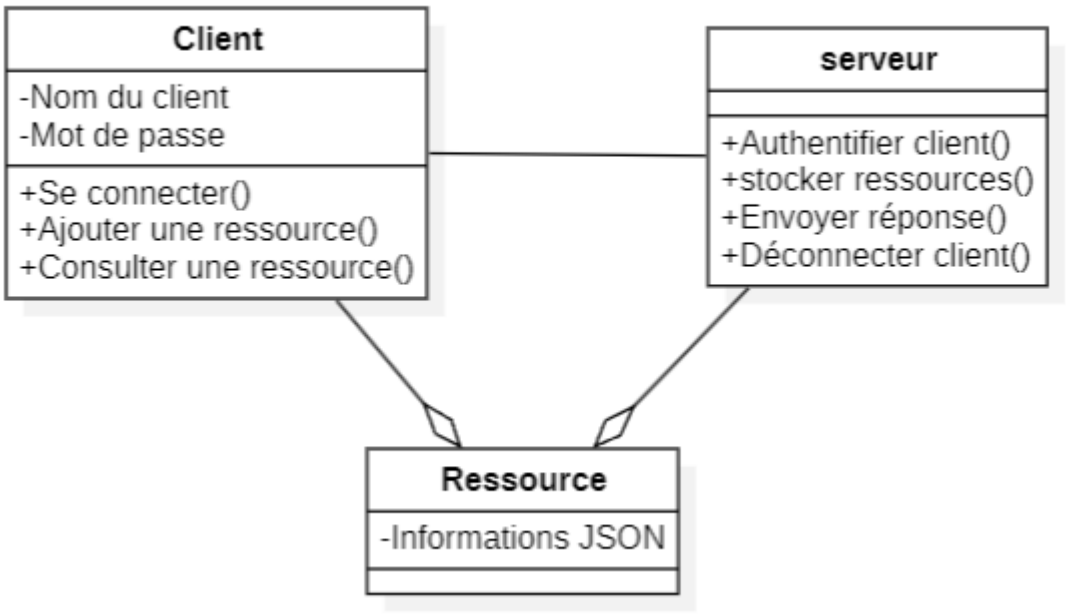

Diagramme de classes

1-Le client

Le client permet l'ajout et la consultation de ressources sur le serveur. Chaque ressource est identifiée par un identifiant unique de la forme `rdo://IP_SERVEUR:PORT/Id_Rsrc`. Il a été conçu pour interagir avec les requêtes GET et POST. Il a trois principales fonctions.

1-1-Fonctions principales du code client

- authenticate_user(): Cette fonction demande à l'utilisateur de saisir son nom d'utilisateur et son mot de passe. Elle utilise la fonction getpass.getpass() pour masquer la saisie du mot de passe.
- send_request(host, port, request): Cette fonction envoie une requête au serveur en utilisant un socket. Elle prend en entrée l'hôte, le port et la

requête à envoyer. Elle établit une connexion avec le serveur, envoie la requête encodée en JSON, reçoit la réponse du serveur et la décode en JSON avant de la renvoyer.

- handle_command(command): Cette fonction gère les commandes saisies par l'utilisateur. Elle analyse la commande pour déterminer s'il s'agit d'une requête GET ou POST. Pour une requête GET, elle extrait le protocole, l'hôte, le port et l'ID de la ressource de la commande, crée une requête GET et l'envoie au serveur à l'aide de send_request(). Pour une requête POST, elle extrait les mêmes informations, demande à l'utilisateur de s'authentifier avec authenticate_user(), crée une requête POST avec les données fournies et l'envoie au serveur.

1-2-Fonctionnement du code client

Le programme est dans l'attente que l'utilisateur saisisse une commande. Lorsque ceci est fait, handle_command(command) est appelée pour la traiter.

Si c'est une requête POST, le programme demande à l'utilisateur de s'authentifier avec authenticate_user(). Une requête POST est créée avec les données fournies et est envoyéau serveur à l'aide de send_request(). La réponse du serveur est ensuite affichée.

Pour une requête GET, cette requête est envoyée au serveur à l'aide de send_request(), puis la réponse du serveur est envoyée.

Lorsque la commande n'est ni GET ni POST, le programme affiche un message d'erreur.

En somme, le code client permet à l'utilisateur d'envoyer des requêtes GET et POST à un serveur en saisissant des commandes dans la console. Il permet l'authentification de l'utilisateur pour les requêtes POST et affiche les réponses du serveur.

2-Le serveur

Le serveur traite les requêtes du client en fournissant les réponses appropriées.

2-1-Fonctions du code serveur

Notre code client comprend les fonctions suivantes:

- charger_ressources(fichier_ressources='ressources.json'): cette fonction charge les ressources à partir d'un fichier JSON spécifié (par défaut ressources.json).
- sauvegarder_ressources(ressources,fichier_ressources='ressources.json'): cette fonction enregistre les ressources dans un fichier JSON spécifié (par défaut ressources.json).
- notifier_clients(rsrcId, data): cette fonction est utilisée pour notifier les clients abonnés à une ressource spécifique lorsqu'elle est mise à jour.
- traiter_requete(client, adresse_ip, port): elle est utilisée pour traiter les requêtes des clients. Elle vérifie si la ressource demandée existe dans le dictionnaire des ressources et crée ou met à jour la ressource spécifiée avec les données fournies par le client. Elle ajoute également le client à l'ensemble des clients abonnés à la ressource pour les futures notifications.
- client_handler(client, adresse_ip, port): elle est utilisée pour gérer les connexions entrantes des clients.
- demarrer_serveur(adresse='0.0.0.0', port=80): elle assure que le serveur démarre sur l'adresse IP spécifiée (par défaut en utilisant le socket.gethostbyname(socket.gethostname()) pour obtenir l'adresse IP de la machine locale) et le port spécifié (par défaut 80).

2-2-Fonctionnement du code serveur

Le code fonctionne de la manière suivante: d'abord, il charge les ressources à partir d'un fichier JSON appelé ressources.json. Ensuite, il écoute les connexions entrantes sur l'adresse IP et le port spécifiés. Lorsqu'un client se connecte, un nouveau thread est créé pour gérer la connexion. Le serveur traite les requêtes GET et POST des clients en fonction de l'opération spécifiée dans la requête.

Pour les requêtes GET, le serveur renvoie les données de la ressource demandée si elle existe, sinon il renvoie un code d'erreur 404.

Pour les requêtes POST, le serveur crée ou met à jour la ressource spécifiée avec les données fournies par le client.

Le serveur envoie ensuite une réponse au client avec un code de statut approprié. Il notifie également les clients abonnés lorsqu'une ressource est créée ou mise à jour en envoyant les nouvelles données aux clients abonnés.

Enfin, le serveur enregistre les ressources mises à jour dans le fichier ressources.json pour les conserver entre les redémarrages du serveur. Il utilise un journal pour enregistrer les événements importants, tels que les erreurs lors du traitement des requêtes. Il peut être démarré en exécutant la fonction demarrer_serveur() dans le script serveur.py.

Le serveur peut être arrêté en appuyant sur Ctrl + C dans la console où il est en cours d'exécution.

II-Difficultés rencontrées lors de l'implémentation

L'une des difficultés rencontrées est l'implémentation pour la notification. En effet, la notification est envoyée, cependant le temps qu'elle soit reçue, le serveur a déjà déconnecté le client. De ce fait nous avons ajouté un try qui permet d'exécuter même si la notification n'a pas été réceptionné par le client.

Par ailleurs, nous n'avons pas pu implémenter la connexion serveur-serveur car nous n'avons pas su quelle logique adopter à cela.

CONCLUSION

Ce projet de conception nous a permis d'user de nos connaissances théoriques afin de concevoir et d'implémenter une application client-serveur basée sur le protocole RDO, en simulant des opérations de lecture et d'écriture de ressources via un réseau. Il a représenté un grand défi pour nous, car certaines fonctions n'ont pas pu être complètement opérationnelle. Bien qu'il lui manque certaines fonctionnalités, cette réalisation est tout de même opérationnelle.

Contributions des Membres de l'Équipe

KAGONE YASMINE OULIA: les fonctions def client_handler(client, adresse_ip, port), def handle_command(command), def demarrer_serveur(adresse='0.0.0.0', port=80), def send_request(host, port, request); Tests et documentation.

TIENDREBEOGO CHICKITA : les fonctions def traiter_requete(client, adresse_ip, port), def handle_command(command), def demarrer_serveur(adresse='0.0.0.0', port=80), def send_request(host, port, request); Tests et documentation.

ILBOUDO SOUTONGNOMA HERMAN : serveur.log, les fonctions def notifier_clients(rsrcId, data), def handle_notification(notification), def authenticate_user(), def charger_ressources(fichier_ressources='ressources.json')

, def sauvegarder_ressources(ressources, fichier_ressources='ressources.json'), def demarrer_serveur(adresse='0.0.0.0', port=80), def send_request(host, port, request); Tests et documentation.

Références :

https://docs.python.org/3/library/socket.html