REPUBLIQUE TUNISIENNE
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université de Tunis El Manar
Faculté des Sciences de Tunis

Rapport de projet :

Traité par :

Atef Amor

Table des matières :

I. Qu'est-ce qu'un système distribué ?

Un système distribué est un ensemble de programmes informatiques qui utilisent des ressources de calcul sur plusieurs nœuds de calcul distincts pour atteindre un objectif commun et partagé. Également connu sous le nom d'informatique distribuée ou de bases de données distribuées, il repose sur des nœuds distincts pour communiquer et se synchroniser sur un réseau commun. Ces nœuds représentent généralement des périphériques matériels physiques distincts, mais peuvent également représenter des processus logiciels distincts ou d'autres systèmes encapsulés récursifs. Les systèmes distribués visent à supprimer les goulots d'étranglement ou les points centraux de défaillance d'un système.

II. Les caractéristiques d'un système distribué :

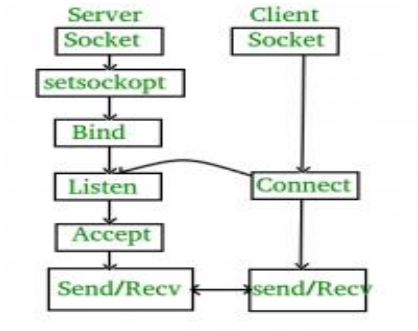
- **Partage de ressources :** Un système distribué peut partager du matériel, des logiciels ou des données.
- **Traitement simultané :** Plusieurs machines peuvent traiter la même fonction simultanément.
- **Évolutivité :** La capacité de calcul et de traitement peut évoluer selon les besoins lorsqu'elle est étendue à des machines supplémentaires.
- **Détection d'erreurs :** Les pannes peuvent être détectées plus facilement.
- **Transparence :** Un nœud peut accéder et communiquer avec d'autres nœuds du système.

III. socket:

1. C'est quoi une socket:

"Les sockets sont une API de communication de bas niveau qui permet aux applications de communiquer sur un réseau en utilisant des flux d'entrée/sortie."

2. Implémentation des sockets dans le projet de chat :



Dans notre projet, on a développé un serveur de chat en utilisant des sockets en Java. Le serveur écoute les connexions des clients sur un port spécifique et diffuse les messages reçus à tous les clients connectés. Chaque client se connecte au serveur à l'aide d'un socket et peut envoyer des messages au serveur.

Nous avons également développé un client de chat en Java qui se connecte au serveur de chat à l'aide d'un socket. Le client peut envoyer des messages au serveur et reçoit les messages diffusés par le serveur. Les messages envoyés par d'autres clients connectés au serveur sont également affichés sur la console du client.

3. Avantages des sockets:

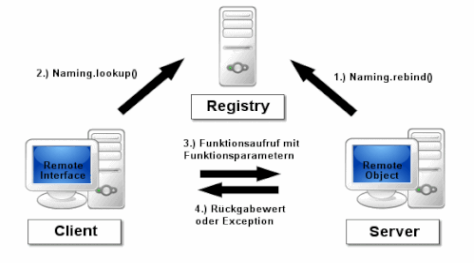
- ✓ **Interopérabilité :** Ils permettent la communication entre des applications dans divers langages et plateformes.
- ✓ **Flexibilité :** Supportent divers types de communication comme unicast, multicast et broadcast.
- ✓ **Performances :** Fournissent de bonnes performances pour une variété d'applications réseau, de la simple transmission de fichiers aux applications en temps réel.
- ✓ **Contrôle fin :** Offrent un contrôle précis sur la communication réseau, y compris les paramètres de connexion et la gestion des erreurs.

4. Limitations des sockets:

- ➢ **Nécessite une gestion manuelle des détails de la communication :** y compris l'établissement et la fermeture des connexions, la sérialisation/désérialisation des données, ce qui peut augmenter la complexité du développement et de la maintenance des applications.
- ➢ **Absence de fonctionnalités avancées :** telles que la gestion des threads et la gestion des erreurs intégrées dans d'autres technologies de communication distribuée.

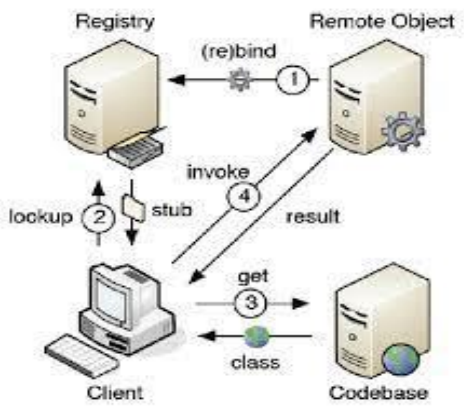
IV. Java RMI:

1. C'est quoi java RMI:



"Java RMI est un mécanisme fourni par Java pour la communication entre objets distribués dans un environnement réseau. Il permet à un objet Java d'appeler des méthodes sur un objet distant situé sur une machine distante comme s'il s'agissait d'objets locaux. Cela est rendu possible grâce à la sérialisation des objets et à l'exécution de méthodes à distance."

2. Implémentation de Java RMI dans le projet de gestion des taches :



Dans notre projet de gestion de tâches, nous avons utilisé Java RMI pour implémenter un service de gestion de tâches. Le serveur RMI offre des méthodes pour ajouter, supprimer et récupérer des tâches, tandis que le client RMI interagit avec ces méthodes pour effectuer les opérations sur les tâches.

L'interface **"Taskinterface"** définit les méthodes à distance disponibles pour la gestion des tâches. Le serveur RMI implémente cette interface et expose ces méthodes pour être appelées par les clients. Les clients RMI utilisent le registre RMI pour localiser le serveur et obtenir un stub permettant d'appeler les méthodes distantes.

3. Avantages de Java RMI:

- ✓ **Intégration transparente avec Java :** Java RMI s'intègre parfaitement avec l'écosystème Java, facilitant ainsi le développement et la maintenance.
- ✓ **Sérialisation automatique des objets :** La sérialisation automatique des objets permet de passer des objets entre les clients et les serveurs de manière transparente.
- ✓ **Gestion transparente des connexions réseau :** Java RMI gère automatiquement les détails de la communication réseau, permettant aux développeurs de se concentrer sur la logique métier.

4. Limitations de Java RMI:

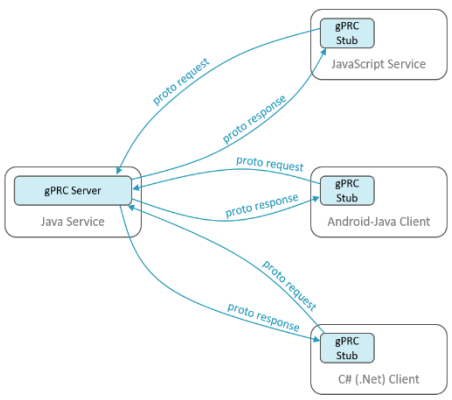
- ➢ **Nécessite une gestion manuelle des détails de la communication :** y compris l'établissement et la fermeture des connexions, la sérialisation/désérialisation des données, ce qui peut augmenter la complexité du développement et de la maintenance des applications.
- ➢ **Absence de fonctionnalités avancées :** telles que la gestion des threads et la gestion des erreurs intégrées dans d'autres technologies de communication distribuée.

V. gRPC:

1. C'est quoi gRPC:

"gRPC est l'abréviation de Google Remonte Procédure Call. gRPC est un cadre RPC open-source utilisé pour créer des API rapides et évolutives. Il permet le développement de systèmes en réseau et une communication ouverte entre les applications client et serveur gRPC. Il utilise le protocole HTTP/2 pour le transport, Protocol Buffers comme langage de description d'interface (IDL : interface description langage)."

2. Implémentation de gRPC dans le projet de service de messagerie :



Le processus d'implémentation de gRPC dans notre projet de messagerie débute par la définition du service dans un fichier. proto, suivi de sa compilation pour générer des classes Java. Ensuite, nous implémentons le serveur gRPC en utilisant ces classes pour définir le comportement des méthodes du service. Une fois le serveur prêt, il est déployé sur une machine accessible via le réseau. Parallèlement, nous développons également le client gRPC, permettant aux utilisateurs d'envoyer et de recevoir des messages en utilisant les méthodes du service de messagerie. Les interactions des utilisateurs avec le service se font via ce client, envoyant des requêtes au serveur qui y répond avec les résultats attendus. Globalement, l'utilisation de gRPC facilite une communication efficace et polyglotte au sein de notre système de messagerie distribuée.

3. Avantages de gRPC:

- ✓ **Communication Bidirectionnelle et Efficace :** gRPC utilise le protocole HTTP/2 pour la communication, permettant ainsi une communication bidirectionnelle et efficace entre les clients et les serveurs.
- ✓ **Sérialisation des Données avec Protobuf :** gRPC utilise le protocole Buffer de Google (Protobuf) pour la sérialisation des données.
- ✓ **Support Multi-langages :** gRPC prend en charge plusieurs langages de programmation, ce qui facilite l'interopérabilité entre les différentes parties d'un système distribué.

4. Limitations de gRPC:

- ➢ **Configuration Initiale Plus Complexe :** La mise en place de la sécurité, de la diffusion en continu ou d'autres fonctionnalités avancées peut nécessiter une expertise supplémentaire.
- ➢ **Pas Aussi Facile à Utiliser pour les Communications Simples :** Bien que gRPC offre des fonctionnalités avancées et des performances supérieures, il peut être moins adapté aux communications simples en raison de sa configuration et de son utilisation plus complexe.

VI. Conclusion:

En conclusion, le choix entre Java RMI, gRPC et les sockets dépend des exigences spécifiques de l'application, telles que la facilité de développement, l'interopérabilité, la performance et le contrôle. Java RMI est idéal pour les applications Java distribuées, gRPC convient aux applications nécessitant une communication efficace et multi-langages, tandis que les sockets offrent une flexibilité maximale mais nécessitent une gestion plus intensive des détails de la communication.