Hasna Daoui

Diawara Nana

École d'Ingénierie Digitale et d'Intelligence Artificielle

I. Introduction à Java RMI

Dans un environnement distribué, les applications sont souvent réparties sur plusieurs machines. Java RMI (Remote Method Invocation) a été conçu pour répondre à ce besoin en permettant à des objets Java situés sur des **JVM différentes** de communiquer entre eux.

RMI est basé sur trois principes clés :

- **Transparence des appels** : L'appel d'une méthode distante ressemble à un appel local.
- **Sérialisation** : Les objets passés en paramètres ou renvoyés doivent être sérialisables.
- **Utilisation d'interfaces distantes** : Les objets distants implémentent des interfaces qui étendent `java.rmi.Remote`.

Les composants principaux d'une application RMI sont :

- **L'interface distante** : définit les méthodes que le client peut appeler à distance.
- **La classe d'implémentation** : implémente les méthodes de l'interface distante.
- **Le registre RMI (rmiregistry)** : permet de référencer les objets distants.
- **Le client RMI** : utilise un objet distant via l'interface distante.

II. Avantages de RMI par rapport à RPC

Critère	Java RMI	RPC
Langage	Purement orienté objet (Java)	Basé sur des procédures (non orienté objet)
Transparence objet	Permet l'appel de **méthodes sur des objets distants**	Appelle uniquement des fonctions/procédures
Passage d'objets	Supporte le **passage d'objets complets**, sérialisés	Passage limité à des types primitifs ou structures
Héritage et interfaces	Utilise l'**héritage Java** et les **interfaces distantes** (`Remote`)	Ne prend pas en charge les concepts orientés objet
Intégration avec Java	Nativement intégré au langage Java	Nécessite souvent un compilateur IDL et du code généré
Garbage Collection	Bénéficie du **ramasse-miettes distribué** (distributed GC)	Pas de gestion automatique de mémoire entre les processus distants
Sécurité	Intègre le modèle de **sécurité Java**	La sécurité dépend de l'implémentation manuelle

Simplicité de développement	Plus simple pour les développeurs Java, pas de langage IDL à apprendre	Nécessite souvent une couche d'abstraction, des définitions externes
Transparence réseau	Gère la communication via **proxies et stubs dynamiques**	Nécessite du code généré statiquement (manuellement ou via un compilateur IDL)
Extensibilité	Plus facile à étendre avec de nouvelles méthodes/objets	Moins flexible, surtout pour des systèmes évolutifs

III. Application Java RMI

Ce projet consiste à développer une application client-serveur permettant aux utilisateurs de **stocker et gérer leurs mots de passe personnels** de manière centralisée et sécurisée. L'objectif est d'offrir un outil simple d'utilisation, basé sur un système **RMI** pour faciliter l'accès et la gestion de comptes en ligne.

1. Structure du projet

1. Interface RMI (PasswordManagerRemote.java)

- Définit toutes les méthodes accessibles à distance
- Étend Remote pour la compatibilité RMI
- Méthodes type-safe avec gestion automatique des exceptions

2. Classes de réponse (ResponseClasses.java)

- Objets sérialisables pour la transmission réseau
- Plus efficace que JSON pour Java
- Type-safe avec vérification à la compilation

3. Serveur RMI (PasswordManagerRMIServer.java)

- Interface Swing améliorée avec logs en temps réel
- Registry RMI intégré (port 1099 par défaut)
- Gestion automatique des sessions

4. Client RMI (PasswordManagerRMIClient.java)

- Interface graphique complète
- Connexion transparente au serveur RMI
- Gestion des comptes avec interface intuitive

2. Structure des fichiers

```
password-manager-rmi/
├── PasswordManagerRemote.java          # Interface RMI
├── BasicResponse.java                  # Réponse simple avec success/message
```

```
├── RegisterResponse.java                    # Représente un compte utilisateur
├── LoginResponse.java                       # Ajoute le token de session
├── PasswordcResponse.java                   # Ajoute le mot de passe déchiffré
├── Account.java
├── AccountListResponse.java                 # Ajoute la liste des comptes
├── PasswordManagerRMIServer.java        # Serveur RMI avec GUI
├── PasswordManagerRMIClient.java        # Client RMI avec GUI
└── README.md                            # Ce guide
```

3. Compilation et exécution

a) Compilation

```
bash
# Compiler tous les fichiers Java
/usr/lib/jvm/java-11-openjdk-amd64/bin/javac -cp ".:gson-2.8.9.jar:sqlite-jdbc-3.42.0.0.jar" *.java
```

Note: Vous devez avoir les dépendances suivantes :

```
# - gson-2.8.9.jar (pour la compatibilité avec l'ancien code)
wget https://repo1.maven.org/maven2/com/google/code/gson/gson/2.8.9/gson-2.8.9.jar
# - sqlite-jdbc-3.42.0.0.jar (pour la base de données)
wget https://repo1.maven.org/maven2/org/xerial/sqlite-jdbc/3.42.0.0/sqlite-jdbc-3.42.0.0.jar
```

b) Démarrage du serveur

```
bash
# Lancer le serveur RMI (port par défaut: 1099)
/usr/lib/jvm/java-11-openjdk-amd64/bin/java ¥
  --add-opens=java.base/java.lang=ALL-UNNAMED ¥
  --add-opens=java.base/java.util=ALL-UNNAMED ¥
  --add-opens=java.rmi/sun.rmi.server=ALL-UNNAMED ¥
  -cp ".:gson-2.8.9.jar:sqlite-jdbc-3.42.0.0.jar" ¥
  PasswordManagerRMIServer
```

c) Démarrage du client

```
bash
# Lancer le client RMI
/usr/lib/jvm/java-11-openjdk-amd64/bin/java ¥
  --add-opens=java.base/java.lang=ALL-UNNAMED ¥
```

```
--add-opens=java.base/java.util=ALL-UNNAMED ¥
--add-opens=java.rmi/sun.rmi.server=ALL-UNNAMED ¥
-cp ".:gson-2.8.9.jar:sqlite-jdbc-3.42.0.0.jar" ¥
PasswordManagerClient
```

4. Configuration réseau

Pare-feu et ports

- **Port RMI Registry** : 1099 - doit être ouvert
- **Ports RMI dynamiques** : Java alloue automatiquement des ports pour les objets RMI
- **Solution** : Utiliser les propriétés système Java pour fixer les ports

Configuration SSL

Afin de sécuriser les échanges entre le client et le serveur RMI, j'ai mis en place une connexion SSL/TLS basée sur un certificat auto-signé. Cette configuration permet d'assurer la confidentialité et l'intégrité des données échangées.

```
System.setProperty("javax.net.ssl.keyStore", "keystore.jks");
System.setProperty("javax.net.ssl.keyStorePassword", "password");
System.setProperty("java.rmi.server.useCodebaseOnly", "true");
System.setProperty("javax.net.ssl.trustStore", "truststore.jks");
System.setProperty("javax.net.ssl.trustStorePassword", "motdepasse");
```

`javax.net.ssl.keyStore` : indique le fichier contenant le certificat SSL et la clé privée que le serveur va utiliser.

`javax.net.ssl.keyStorePassword` : le mot de passe pour accéder au keystore.

`javax.net.ssl.trustStore :` Le truststore contient le certificat auto-signé du serveur (server.crt)

`javax.net.ssl.trustStorePassword :` le mot de passe pour accéder au truststore.

IV. Utilisation de l'application

Interface serveur

1. Cliquez sur "Se connecter au serveur"
2. Le serveur crée automatiquement le registry RMI
3. Les logs affichent les connexions et opérations en temps réel

Interface client

1. **Connexion au serveur** : Entrez l'adresse IP (localhost par défaut) et le port du serveur(par défaut 1099)
2. **Authentification** : Créez un compte ou connectez-vous

3. **Gestion des comptes** : Ajoutez, modifiez, supprimez vos comptes de mots de passe

API RMI

Classes de réponse

- **BasicResponse** : Réponse simple avec success/message
- **LoginResponse** : Ajoute le token de session
- **PasswordResponse** : Ajoute le mot de passe déchiffré
- **AccountListResponse** : Ajoute la liste des comptes
- **Account** : Représente un compte utilisateur

Sécurité

Mesures implémentées

1. **Chiffrement AES-256** : Mots de passe chiffrés en base
2. **Hachage SHA-256** : Mots de passe utilisateur hachés avec sel
3. **Sessions sécurisées** : Tokens de session aléatoires
4. **Validation des entrées** : Sanitisation et validation
5. **Isolation utilisateur** : Chaque utilisateur ne voit que ses données
6. **Sécurisation du serveur RMI avec SSL/TLS**

Dépannage

Erreurs communes

1. **ClassNotFoundException** : Vérifiez le classpath
2. **ConnectException** : Vérifiez que le serveur est démarré
3. **NotBoundException** : Le service n'est pas enregistré dans le registry
4. **AccessControlException** : Problème de sécurité Java

Monitoring

Le serveur affiche des statistiques en temps réel :

- Nombre de connexions actives
- Operations par utilisateur
- Temps de réponse des requêtes

V. Conclusion

Ce projet a permis de mettre en place une application sécurisée de gestion de mots de passe utilisant une architecture client-serveur. La communication entre le client et le serveur est assurée via le protocole RMI sécurisé par SSL/TLS, garantissant la confidentialité et l'intégrité des données échangées.