Kocek Tuncer 40631 Pipers Chris 39864

Directeur : Michel Willemse

Professeur : Absil Romain

RESG4 – Sécurité

Projet : Messagerie sécurisée

Mai 2018

Executive summary

Le projet messagerie est un programme qui permet de mettre en place une messagerie instantanée sécurisée.

Le but principal de ce programme est de garantir au mieux la sécurité de la communication entre deux personnes ainsi que celle des données.

Pour cela, l'utilisation de plusieurs API Java spécialisées et de techniques de sécurisation ont été mises en place et/ou utilisées.

Table des matières

Executive summary 3
Introduction 5
Projet, package, classes et leurs compositions 6
Messagerie 6
Model 6
Observer 7
Network 7
View 9
Messagerie_Server 10
MessagerieDB 10
Bl 11
Dto 11
Equestre.messagerieserver 12
Commun 14
Commun 14
Base de données 15
Table Contact 15
Table SEQUENCES 15
Table USERS 16
Table BLOCKED 16
Sécurités informatique supplémentaire 16
Base de données 16
Connexion Serveur 16
Organisation du travail 17
Conclusion 18
Bibliographie 19

Introduction

Dans ce document, nous allons aborder le développement d'une messagerie en temps réel sécurisée dans le langage de programmation Java.

Nous aborderons les différentes classes utilisées, les API Java de sécurité et certaines techniques utilisées, pour renforcer la sécurité.

En annexe, il vous sera fourni un diagramme de classes permettant d'avoir une vision globale des classes ainsi que les liens entre elles.

Projet, package, classes et leurs compositions

Les choix des classes ont été murement réfléchis afin de pouvoir convenir à plusieurs contraintes, notamment la contrainte de la connexion des différents clients entre eux et leurs communications mais aussi à la connexion et la communication avec le serveur. Par ailleurs, il y'avait un autre souci auxquelles on a dû faire face. Celui de la modification de données en concurrence. Les solutions mis en place seront expliquées un peu plus loin.

Messagerie

Model

Contact

Description

La classe « Contact » permet de construire et de gérer un contact. Un contact est un correspondant d'un utilisateur de la messagerie.

Elle hérite de la classe « ContactDto » (cf : Commun, commun, ContactDto). Pour éviter la redondance car un contact a des attributs supplémentaires par rapport à un ContactDto.

En plus des attributs de la classe ContactDto, nous avons jugé important d'y ajouter le statut de connexion afin de pouvoir gérer les connexion et déconnexion soudaine du contact.

ConnexionStatus

Description

L'énumération « ConnexionStatus » contient les différents statuts que peut prendre un contact. Les différents statuts sont :

- TRY, pour une tentative de connexion ;

-JUSTCONNECTED, pour une connexion ;

-JUSTDISCONNECTED, pour une déconnexion.

Cette classe permet notamment une meilleure gestion des évènements extérieures via le patron « observateur observer »

ContactList

Description

La classe « ContactList » permet de créer et gérer une liste de contacts d'un utilisateur.

Nous avons implémenté cette classe en implémentant l'interface Collection car nous avions besoin des certaines méthodes supplémentaires pour la recherche de contact dans la liste selon certains critères. (Recherche par IP, par nom, …)

Observer

Observable

Description

Cette interface est l'une des deux interfaces qui permet d'implémenter le patron « observateur-observé ».

Observer

Description

Cette interface est l'une des deux interfaces qui permet d'implémenter le patron « observateur-observé ».

Network

Communication

Description

La classe « Communication » permet de créer et gérer une communication entre deux utilisateurs de la messagerie de manière sécurisée. Nous l'avons implémenté de telles sortes qu'à la création d'une communication, une paire de clé asymétriques soit généré des deux côtés et échangé. Puis une fois cela fait, elle permettra de générer une clé dites de session et l'enverra au client. Elle sert aussi à commencer une communication dites « client à client » afin de pouvoir débuter une discussion.

Elle permet également d'envoyer des messages, des fichiers « Json » et des fichiers binaires et ce, de manière sécurisée entre deux utilisateurs de la messagerie via son attribut newKey.

Sécurité informatique utilisé

Une communication utilise plusieurs clés permettant d'assurer une communication sécurisée. L'attribut « key » contient toutes les clés nécessaires. Une paire de clés asymétriques pour crypter et décrypter les premiers échanges.

Cette paire de clés permet d'envoyer une clé symétrique qui sera utilisé entre les utilisateurs de la messagerie pour crypter et décrypter les messages échangés, reçus et envoyés avec d'autres utilisateurs et/ou le serveur.

La paire de clés est créée par l'algorithme RSA et un chiffre aléatoire de 1024 bits et de l'utilisation de l'algorithme SHA1PRNG généré de manière aléatoire.

Cette paire de clés permet un échange initial d'une clé symétrique, ce qui créera une clé session.

La clé symétrique est générée via l'utilisation de l'algorithme AES de taille 128 bits.

Selon que l'on soit serveur ou le client dans la messagerie, c'est-à-dire que lorsqu'on envoie (serveur) ou que l'on reçoit (client) pour la première fois, les étapes doivent se dérouler dans un ordre précis. Le choix d'implémentation a été fait ainsi suite à de nombreux bugs sous-jacents. Nous avions par exemple un blocage des ressources des deux côtés pour l'attente de la clé. C'est pour cela que le serveur ou le créateur d'une discussion est celui qui génère et envoie la clé en premier.

PollingConnection

Description

La classe « PollingConnection » permet d'initialiser et de maintenir une communication entre deux utilisateurs de la messagerie. La classe gère le client/serveur et le client.

Fréquemment, les clients doivent vérifier que leurs contacts soient toujours connectés.

Pour cela, nous avons eu recours à la classe Timer afin de ne pas utiliser inutilement les ressources du pc.

Elle permet par ailleurs, lors de la connexion d'un client, d'avertir tous ses contacts de sa connexion mais aussi d'avertir le serveur lors de la déconnexion soudaine d'un contact.

Receiver

Description

La classe « Receiver » permet de réceptionner les messages entrants du contact avec qui l'utilisateur courant discute. Elle permet par la suite de mettre à jour la vue via la méthode update du patern Observer.

Pour faire cela sans bloquer le reste de l'application, nous avons eu recours au thread.

Cela nous a permis de ne pas bloquer la vue et de pouvoir continuer à récolter les messages envoyés par son contact et d'envoyer ses propres messages à son contact.

Server

Description

La classe « Server » permet à l'utilisateur de se connecter au serveur et d'attendre les messages de ce dernier.

Pour implémenter ça, nous avons choisi d'utiliser le port 2009. Ce port nous sert pour les discussions avec le serveur.

De plus, pour éviter que cette classe ne soit bloquant, nous avons eu recours une nouvelle fois au thread.

View

Client

Description

La classe « Client » permet de gérer un utilisateur qui se connecte au serveur, il devient alors un client.

Il gère tout ce qui en rapport avec un client. Connexion, lancement de thread, lancement des services pour que le client reçoive des infos de ses contacts…

Discussion

Description

La classe « Discussion » permet d'initialiser et gérer une discussion entre deux utilisateurs et ce, sur le port 2009.

Nous avons par ailleurs utilisé un port différent de celui utilisé par le serveur étant donné que ce dernier tournait sur la même machine où le client tournait.

Cette classe hérite de communication.

Cet héritage a été fait car pour nous, une discussion est une forme de communication.

Messagerie

Description

La classe « Messagerie » est la classe principale (Main) du projet Messagerie.

Messagerie_Server

MessagerieDB

BlockedDB

Description

La classe « BlockedDB » permet de gérer les changements dans la liste de contacts bloqués d'un utilisateur se trouvant dans la base de données. Cette classe permet d'ajouter et de supprimer un/des contact(s) bloqué(s) dans la liste présente dans la table de la base de données.

ContactDB

Description

La classe « ContactDB » permet de gérer les changements dans la liste de contact d'un utilisateur. Elle permet d'ajouter et supprimer un contact, de mettre à jour, dans la table de la base de données.

DBManager

Description

La classe « DBManager » permet de gérer la connexion à la base de données et de valider ou annuler des transactions vers celles-ci.

SequenceDB

Description

La classe « SequenceDB » permet de gérer et de maintenir une cohérence et une synchronisation dans l'attribution des identifiants pour les « ContactDB », « UsersDB » et « BlockedDb » car JAVADB ne sait pas s'en occuper seul.

UsersDB

Description

La classe « UsersDB » permet de gérer les utilisateurs dans la base de données. Cela part du fait de retrouver un utilisateur dans la base de données, en ajouter un, en supprimer un ou de mettre à jour le statut d'un utilisateur.

Bl

AdminFacade

Description

La classe « AdminFacade » joue le rôle d'intermédiaire pour traiter les actions sélectionnées par l'utilisateur pour la gestion de sa liste de contacts, contacts bloqués, ...

BlockedBl

Description

La classe « BlockedBl » correspond à la partie « traitement des données métier » dans l'architecture 3 tiers pour la gestion des blocages.

ContactBl

Description

La classe « ContactBl » correspond à la partie « traitement des données métier » dans l'architecture 3 tiers pour la gestion des contacts.

Façade

Description

La classe « Façade » joue le rôle d'intermédiaire pour connaitre le statut de certains contacts, ou certains logins selon l'utilisateur qui le demande. Elle permet également de récupérer les listes des contacts en ligne et la liste de contact dont les demandes sont en attentes pour un certain utilisateur. Elle permet donc de contrôler l'accès à la BD.

UsersBl

Description

La classe « UsersBl » correspond à la partie « traitement des données métier » dans l'architecture 3 tiers pour la gestion des contacts.

Dto

BlockedDto

Description

La classe « BlockedDto » permet le transfert d'un contact à bloquer sous la forme d'un objet.

UsersDto

Description

La classe « UsersDto » permet le transfert d'un utilisateur et de tous ses attributs sous la forme d'un objet.

Equestre.messagerieserver

Action

Description

La classe « action » permet de gérer l'action sélectionnée d'un utilisateur.
L'utilisateur peut réaliser plusieurs actions :

S'inscrire s'il n'est pas encore inscrit ou soit de créer un second compte avec un autre identifiant ;

Se connecter s'il est au préalablement inscrit ;

Envoyer une demande d'ami à un autre utilisateur pour l'ajouter en tant que contact ;

Bloquer un contact ;

Débloquer un contact ;

Supprimer un contact ;

Sélectionner un contact avec qui discuter ;

Se déconnecter.

Block

Description

La classe « Block » permet de bloquer ou de débloquer un contact pour un utilisateur.

Clients

Description

La classe « Clients » permet au serveur de gérer les utilisateurs qui souhaitent se connecter ou s'inscrire au système, dès lors qu'il devient un client. Cela selon l'architecture Client/Serveur.

Communication

Description
La classe « communication » permet de gérer la communication entre deux utilisateurs du programme. La communication se fait par « socket ». Elle permet d'envoyer et recevoir des messages.

Sécurité informatique utilisé
Si les clés, paire de clés asymétriques et clé symétrique, ne sont pas encore générées, elles le seront par la classe « communication ».

Connexion

Description
La classe « Connexion » permet à un utilisateur de se connecter au système s'il est préalablement inscrit.

Sécurité informatique utilisé
Le mot de passe est haché à l'aide de l'algorithme SHA-256. Avant que le mot de passe soit haché, il y a eu l'ajout du sel c'est-à-dire qu'on ajoute un groupe de bytes aléatoires au mot de passe de chaque utilisateur. Ce hachage permet de vérifier si le même résultat pour le hash du mot de passe est le même que dans la base de données pour ce login.

Demand

Description
La classe « Demand » permet à un utilisateur de gérer les demandes, envoyées ou reçues. Cela en les envoyant, les acceptants ou les refusant.

Inscription

Description
La classe « Inscription » permet de gérer l'inscription d'un utilisateur.

Sécurité informatique utilisé
L'utilisateur doit rentrer un mot de passe comprenant minimum 8 caractères dans lequel doivent figurer un chiffre, un caractère minuscule, un caractère majuscule.

Le stockage du mot de passe sur la base de données est sécurisé car on ne stock pas le mot de passe tel quel en claire mais le résultat de l'algorithme de hachage SHA-256 du mot de passe auquel on a ajouté le sel.

NewMain

Description

La classe « NewMain » est la classe principale du projet Messagerie_Server, elle permet de lancer le programme.

PollingDeconnexion

Description

La classe « PollingDeconnexion » permet la connexion au « socket » et de vérifier, suite à la demande d'un client, que son contact est toujours connecté. Elle permet également d'avertir les personnes ayant comme contact la personne qui vient de se déconnecter, qu'elle s'est déconnectée de manière soudaine.

Commun

Ce projet nous permet d'éviter la redondance de classe et de permettre à l'application une meilleure évolution.

On utilise le jar de ce projet pour faire compiler les deux autres.

Commun

Key

Description

La classe « Key » permet de générer et de gérer la paire de clés asymétriques, la clé symétrique et clé publique du client.

Sécurité informatique utilisé

La paire de clés asymétriques est générée par l'algorithme RSA et un chiffre aléatoire de 1024 bits, 128 bytes, généré par l'algorithme SHA1PRNG de manière sécurisée.

La clé symétrique est générée par l'algorithme AES et est de taille 128 bits.

ContactDto

Description

La classe « ContactDto » permet de transférer un contact ainsi que tous ses attributs sous la forme d'un objet.

EntityDto

Description

La classe abstraite « EntityDto » est hérité par toutes les classes qui gèrent le transfert de données sous forme d'objet.

StatusRequest

Description

L'énumération « StatusRequest » permet de connaitre le statut d'une demande d'amis : refusée, en attente, acceptée ou envoyée.

Le statut « envoyé », « SENDED », permet de différencier celui qui a envoyé la demande et celui qui la réceptionne. Celui qui réceptionne peut refuser ou accepter tandis que celui qui envoie peut juste attendre.

Base de données

La base de données est composée des tables contact, sequences, users et blocked.

Table Contact

```
create table "TUNCER_CHRIS".CONTACT

(

        ID BIGINT not null primary key,

        USERS_ID BIGINT,

        USERS_FRIEND_ID BIGINT not null,

        BLOCKED BOOLEAN default false not null,

        STATUS VARCHAR(256)

)
```

Table SEQUENCES

```
create table "TUNCER_CHRIS".SEQUENCES

(

        ID VARCHAR(50) not null primary key,

        SVALUE NUMERIC(10) not null

)
```

Table USERS

```
create table "TUNCER_CHRIS"."USERS"

(

ID BIGINT not null primary key,

LOGIN VARCHAR(256),

PASSWORD VARCHAR(256),

IP VARCHAR(256),

ENLIGNE BOOLEAN default false not null,

SALT VARCHAR(256) FOR BIT DATA,

ESSAI SMALLINT,

TIME BIGINT

);
```

Table BLOCKED

```
create table "TUNCER_CHRIS".BLOCKED

(

        ID BIGINT not null primary key,

        ID_CONTACT_1 BIGINT,

        ID_CONTACT_2 BIGINT

)
```

Sécurités informatique supplémentaire

Base de données

Le mot de passe utilisateur n'est pas stocké dans la base de données, ce qui est stocké est le hash du mot de passe utilisateur. Le sel utilisé est différent pour chaque utilisateur et n'est pas secret. Il est stocké aussi dans la base de données.

Connexion Serveur

Le nombre de tentatives de connexion est limité à 3 essais, après cela un blocage de 3h est activé. Ce n'est qu'après les 3h que 3 autres essais sont à nouveau disponibles. Cela permet de diminuer le risque de la réussite d'une attaque par dictionnaire.

Un utilisateur inscrit ne peut se connecter qu'à une et une seule machine avec un identifiant, si un utilisateur est connecté à une machine, il ne saura pas se connecter sur une autre machine avec le même identifiant tant que l'autre ne se déconnecte pas de la première machine.

Organisation du travail

Pour l'organisation du travail, nous avons découpé le projet en 4 parties.

Premièrement, mettre en place les outils pour pouvoir travailler en binômes. Tels que par exemple la création de projet sur git.

Deuxièmement, on a fait l'analyse UML du projet, avons sorti les problèmes majeurs puis discuter de leurs solutions.

Une fois les solutions choisies, nous avons codé le projet sans tenir compte réellement de la sécurité. Nous voulions que cela marche au premier abord.

Et enfin, nous avons mis en place toutes les sécurités possibles pour que l'application soit le plus sécurisé possible.

Evidemment, cette séparation s'est faite de manière naturelle car lorsqu'un ne pouvait pas coder, l'autre prenait le relais. La reprise du code et sa compréhension était en effet facilité par l'étape numéro deux.

Conclusion

Le projet messagerie permet de fournir un programme de messagerie instantanée sécurisée. La communication entre les différentes entités se passe par l'utilisation de « socket » et ce, sur des ports bien définis.

Il y a trois types d'entités : le serveur, qui permettra la mise en communication initiale de deux contacts ; le client pur qui discute avec un autre client ; et le client/serveur qui lui héberge la communication entre lui et un autre client, ce qui permet d'avoir une messagerie client/client.

La sécurité de la messagerie se fait grâce au cryptage des messages échangés pour empêcher une quelconque utilisation du contenu de ces messages. Elle se fait également par la sécurisation en stockant le hachage du mot de passe salé dans la base de données, en limitant le nombre de tentative de connexion à 3 essais par 3h, en limitant une personne à se connecter sur une seule machine et en limitant l'utilisation du même identifiant à 1 et uniquement 1.

Pour la sécurisation de la messagerie, il faut sécuriser les transferts des messages mais également certaines données sensibles, limitées ou restreindre certains accès et action de l'utilisateur.

Bibliographie