Kévin MAUGE

kmauge@etud.u-pem.fr

Pape NDIAYE

pndiaye@etud.u-pem.fr

Master Informatique

M1-S2

Groupe : 2

Projet

Programmation réseau

Matou - Service de Chat

*Manuel développeur*

**- 01/05/2016 -**

**Présentation générale du projet**

Objectifs

L'objectif du projet est d'implanter un service de chat dans une architecture centralisée (client/serveur) et où le serveur est implanté en "non-bloquant".

**Fonctionnement**

Comportement du serveur

Lorsqu'il est lancé, le serveur surveille un port de connexion et accepte toutes les connexions pendantes sur ce port. Si la limite du maximum de connecté est atteinte, il ferma la connexion aussitôt.

Lorsqu'un client envoie une requête au serveur, le serveur lit la requête argument par argument.

Si un client ne respecte pas le protocole, le serveur ferme immédiatement la connexion avec ce client. Ce cas ne doit jamais se produire avec un client réglementaire.

Les erreurs d'utilisation comme l'ouverture d'une connexion privée avec un utilisateur déconnecté ne sont pas considérées comme des erreurs de protocole. Dans ce cas, le serveur envoie un message d'erreur au client.

Comportement du client

Si un pseudo a été indiqué à la fin de la ligne de commande, le client va essayer de se connecter au serveur avec ce pseudo. En cas d'echec, le client s'arrête immédiatement.

Si aucun pseudo n'a été indiqué à la fin de la ligne de commande, l'utilisateur devra indiquer un pseudo manuellement sur le terminal afin que le client essaye de se connecter au serveur avec ce pseudo. En cas d'échec, le client attendra un nouveau pseudo jusqu'à recevoir un pseudo valide.

Une fois connecté et authentifié, le client est identifié de manière unique par ce pseudo durant toute sa session.

Le pseudo d'un client est libéré au moment de sa déconnexion.

**Architecture**

Packages

Le service dispose de plusieurs packages

* fr.upem.matou.client : package nécessaire au client
* fr.upem.matou.client.network : contient l'implémentation réseau du client
* fr.upem.matou.client.ui : contient l'implémentation de l'interface graphique du client
* fr.upem.matou.server : package nécessaire au serveur
* fr.upem.matou.server.network : contient l'implémentation réseau du serveur
* fr.upem.matou.shared : package nécessaire au client et au serveur
* fr.upem.matou.shared.logger : contient le logger
* fr.upem.matou.shared.network : contient la base du protocol de communication réseau
* fr.upem.matou.shared.utils : regroupe des classes et méthodes utilitaires

Responsabilité de chaque classe

Voici la liste des différentes classes du service ainsi que leurs propriétés respectives :

* package "client"
  + ClientMatou : C'est le point d'entrée du client.
* package "client.network"
  + ClientCommunication : Classe composée de méthodes statiques utilisées par le client pour garantir le respect du protocol de communication.
  + ClientCore :
  + ClientInstance
  + ClientSession :
  + ClientEvent :
  + Message
  + SourceConnectionData
  + DestinationConnectionData
* package "client.ui"
  + UserInterface
  + ShellInterface
  + ShellCommand
* package "server"
  + ServerMatou
* package "server.network"
  + ServerCommunication : Classe composée de méthodes statiques utilisées par le serveur pour garantir le respect du protocol de communication.
  + ServerCore
  + ServerDataBase
  + ServerSession :
  + SelectorLogger : C'est le logger du sélecteur.
* package "shared.network"
  + NetworkCommunication : Classe composée de méthodes statiques utilisées par le client et le serveur pour garantir le respect du protocol de communication. C'est dans cette classe que sont définies la plupart des règles à respecter.
  + NetworkProtocol : Enumération des différents types de requête. Chaque type de requête est identifié par un entier unique.
  + ErrorType : Enumération des différentes erreurs d'utilisation que le serveur peut envoyer au client.
  + Username : Représente un nom d'utilisateur. Un nom d'utilisateur est une chaîne de caractère identifiée par sa représentation en minuscule.
* package "shared.logger"
  + Logger : C'est le logger d'évenements du service. Il n'y a qu'une instance de cet objet à l'exécution, il est manipulé avec des méthodes statiques.
  + Colorator : Classe qui regroupe des méthodes statiques permettant de formater des chaines de caractère afin qu'elles s'affichent en couleur sur le terminal.
* package "shared.utils"
  + ByteBuffers : Classe qui regroupe des méthodes statiques sur les objets de type "ByteBuffer".
  + Configuration : Classe qui regroupe des méthodes statiques pour lire un fichier de configuration.

**Choix de développement**

Format du logger

Gestion des erreurs

Buffers

Fichier de configuration

Autres

**Conclusion**

Objectifs réalisés

Notre programme répond aux objectifs demandés par le sujet et respecte les consignes imposées.

Points forts

* abra
* kadabra

Points faibles

* abra
* kadabra

Difficultés rencontrés

Voici les principales difficultés que nous avons rencontré durant ce projet :

* abra
* kadabra

Dysfonctionnements possibles :

Nous avons effectué plusieurs tests sur notre programme. Voici les fonctionnements anormaux que nous avons remarqué et que nous n'avons pas pu pleinement corrigé :

* abra
* kadabra

Perspectives d'améliorations

Voici quelques perspectives d'améliorations et d'optimisations possibles pour le projet. Il s'agit généralement d'idées que nous avions eu et que nous n'avons pas eu le temps de mettre en place ou bien d'améliorations qui nous sont venus trop tard dans la réalisation du projet :

* abra
* kadabra

**Annexes**

Ressources extérieures

Pour concevoir ce projet, nous avons utilisé les outils suivants :

* PlantUML/PlantText/Graphviz : Pour générer un diagramme UML graphique à partir d'un fichier formaté.
  + <http://plantuml.com/>
  + <http://www.planttext.com/planttext>
  + <http://www.graphviz.org/>
* Sources extérieures pour l'architecture du serveur :
  + abra
  + kadabra