

Guide utilisateur

Contamain Kilian – Humbert Cédric



1. Présentation de TOMM (Tag Oriented Market Maker) :

TOMM est un logiciel permettant de mettre en relation les membres d'une communauté qui souhaitent échanger des services ou des biens en accédant aux threads qui sont différents sujet de conversation comme par exemple Rugby ou FEI. L'utilisateur a donc l'accès à tous les postes de ce thread comme par exemple dans le cas rugby où il va pouvoir consulter des postes comme « A quelle heure est l'entraînement demain ? ». Chaque utilisateur qui a été enregistré peut être souscrit à un thread et voir les postes en relation avec le sujet souscrit.

L'utilisateur peut bien sûr créer des postes sur les Threads et même créer un nouveau sujet de discussion. En théorie, l'utilisateur est aussi notifié s'il y a un nouveau poste dans le thread qu'il a souscrit.

2. Utilisation de TOMM :

Le projet est constitué de deux parties, une pour l'utilisateur une pour le serveur, il faut donc extraire chacune des deux de l'archive et les lancer séparément. Le fichier exécutable se trouve dans bin/Debug dans chacun des deux projets. Les fichiers relatifs à l'exécution du programme sont dans le même dossier.

Pour exécuter le serveur : `./serveur1 20000`

Pour exécuter le client : `./client1 localhost 20000` (à noter que par défaut c'est cette option qu'il choisira il n'y a donc pas lieu de lui préciser)

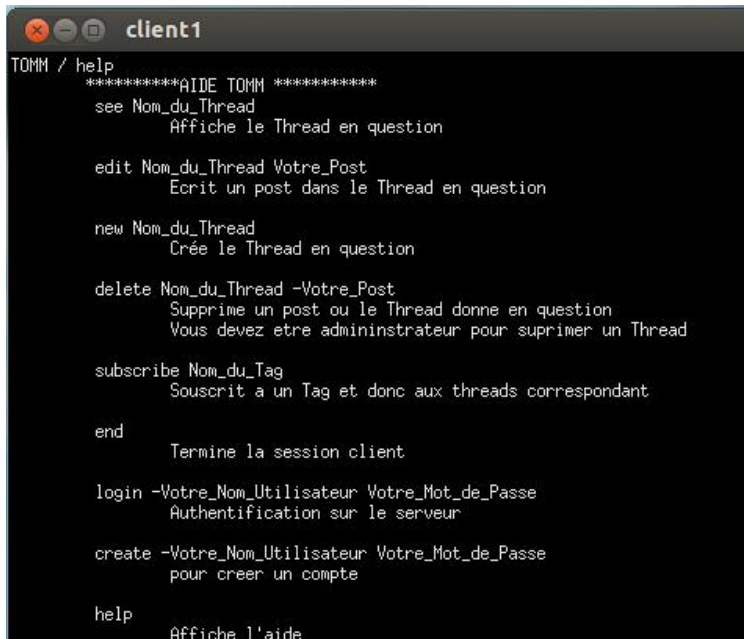
Les sources se trouvent dans la racine, respectivement serveur et client, nous avons utilisé CodeBlocks pour réaliser le projet qui est en sa totalité dans le dossier.

Le client propose une interface en ligne de commande :

« TOMM / » tel que :



On peut ensuite entrer la commande que l'on souhaite avec les paramètres nécessaires. La commande `help` permet de les résumer.



```
client1
TOMM / help
*****AIDE TOMM *****
see Nom_du_Thread
    Affiche le Thread en question

edit Nom_du_Thread Votre_Post
    Ecrit un post dans le Thread en question

new Nom_du_Thread
    Crée le Thread en question

delete Nom_du_Thread -Votre_Post
    Supprime un post ou le Thread donne en question
    Vous devez etre administrateur pour supprimer un Thread

subscribe Nom_du_Tag
    Souscrit a un Tag et donc aux threads correspondant

end
    Termine la session client

login -Votre_Nom_Utilisateur Votre_Mot_de_Passe
    Authentification sur le serveur

create -Votre_Nom_Utilisateur Votre_Mot_de_Passe
    pour creer un compte

help
    Affiche l'aide
```

Pour commencer il est conseillé de s'authentifier, trois comptes existent déjà sur le serveur, Cedric, Kilian et Admin. La commande ***login*** seule sans paramètre permet d'utiliser un fichier de log présent sur le client pour s'authentifier directement. Ce fichier est changé à chaque fois qu'un nouvel utilisateur utilise la commande ***create***.

Une base de données est préenregistrée et chargée/sauvegardée à chaque début/fin de session Serveur.

Pour commencer l'utilisation initialisez un nouveau Thread avec ***new*** puis ajoutez lui des posts avec ***edit*** enfin vous pourrez visualiser la totalité du Thread avec ***see***. Pour quitter il faut faire ***end*** la session se fermera automatiquement.

3. Fonctionnement Contraintes :

Les commandes ***delete*** et ***subscribe*** ne sont pas implémentées, la base de données préenregistrée ne veut plus s'afficher car un caractère vient se glisser en fin du premier argument passé à la fonction `ce`, uniquement pour les Threads qui se trouvent dans le dossier `data` initial ; les autres s'afficheront normalement.

La seule contrainte qui existe pour les posts est le nombre de caractères qui ne doit pas excéder `TAILLE_POS` soit 140 caractères.

Le premier argument qui est en général le nom du Thread doit tenir un seul bloc sans espace.

Le transfert du client au serveur et vice-versa s'effectue par le biais de fichiers générés dans le répertoire « Temp » de bin/Debug, ils sont automatiquement générés et supprimés par le client et le serveur.

Le projet a été créé sur CodeBlocks, un fichier .cdb est donc aussi présent sur la machine nous n'avons pas donc eut à faire de Makefile. A noter cependant que le projet qui est envoyé est susceptible de compiler dans n'importe quel répertoire Linux.

Exemple :

```
Commande See demandee
-----
Titre : Voiture_Paris
Identifiant : 1 et nombre de Posts : 4
Date de début :
Le 2014/11/23 à 22:22:46
    AAAA/MM/JJ    HH:MM:SS
Date de fin :
Le 2014/11/23 à 22:22:46
    AAAA/MM/JJ    HH:MM:SS
-----
Post n° 1 dans le Thread n° 1 :
Posté par : Fake1
Le 2014/11/23 à 22:22:46
    AAAA/MM/JJ    HH:MM:SS
'Bonjour,Je voudrais aller a Paris ce Week-end'
-----
Post n° 2 dans le Thread n° 1 :
Posté par : Fake2
Le 2014/11/23 à 22:22:46
    AAAA/MM/JJ    HH:MM:SS
'Salut Fake, je vais justement à Paris en R5 vendredi'
-----
Post n° 3 dans le Thread n° 1 :
Posté par : Fake1
Le 2014/11/23 à 22:22:46
    AAAA/MM/JJ    HH:MM:SS
'Merci Fake, On pars apres ls cours ?'
-----
Post n° 4 dans le Thread n° 1 :
Posté par : Fake2
Le 2014/11/23 à 22:22:46
    AAAA/MM/JJ    HH:MM:SS
'Ca marche !'
```

On a ici la totalité d'un Thread, la commande **see** a été identifiée et ce que nous voyons ici c'est le contenu du fichier « Temp/Answer » qui a été envoyé du serveur et chargé dans une liste chaînée pour être affiché.

Le serveur lui aussi affiche des contenus similaires.

Pour ce qui est du fonctionnement un peu plus concret du programme, pour chaque requête le client envoie une chaîne de six caractères disant qu'il va soumettre cette requête, le serveur l'écoute se dispose à enregistrer le fichier qui va lui être envoyé contenant un message. Le serveur s'exécute alors au mieux pour répondre à la requête du client et lui renvoi sous forme la aussi de fichier. Le client prend alors connaissance du contenu du fichier et affiche à l'utilisateur si sa requête a abouti.