# Réseau 2016 - 2017

## Identification du groupe

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Membre | Application | Langage |
| #1 | LECOINTRE Thomas | Serveur | Java |
| #2 | EVERAERT Flavien | Client | Java |

## Protocole de transport

UDP pour chaque requête.

## Port utilisé pour communiquer

1901

## Les échanges entre le client et le serveur

### Connexion / Déconnexion

Le client envoie CONNECT avec le nom de l’utilisateur au serveur. Le serveur répond avec CONNECT\_KO s’il y a trop de clients connectés, CONNECT\_OK sinon. A la déconnexion le client envoie DISCONNECT avec son nom. Si le serveur doit signaler un problème, il renvoie DISCONNECT\_KO avec un message expliquant que la déconnexion n’est pas possible.

### Création de la chambre

Le client envoie CREATE avec le nom de la chambre et le mode de modération (libre, modéré ou autre…). Le serveur répond avec CREATE\_KO si la chambre existe déjà ou s’il y a trop de chambres sur le serveur, etc…, CREATE\_OK sinon. Le serveur va enregistrer la chambre avec le nom de la chambre et le nom du modérateur si la chambre est modérée.

### Charger les chambres existantes

Le client envoie LOAD avec le nom de l’utilisateur. Le serveur peut répondre LOAD\_KO en cas d’erreur sur le serveur, sinon il renvoie LOAD\_OK avec une liste de toutes les chambres existantes, les modes de chaque chambre et un booléen indiquant si le client est abonné à la chambre ou non.

### S’abonner à une chambre / se désabonner d’une chambre

Si le client souhaite s’abonner à une chambre, il envoie SUBSCRIBE avec son nom et le nom de la chambre en question. Si le client veut se désabonner d’une chambre il envoie UNSUBSCRIBE avec son nom et le nom de la chambre.

Le serveur renvoie SUBSCRIBE\_KO si le client est déjà abonné à la chambre ou si le serveur ne peut pas satisfaire la demande d’abonnement tout simplement. Le serveur renvoie SUBSCRIBE\_OK pour indiquer que l’abonnement est validé.

Le serveur renvoie UNSUBSCRIBE\_KO si le client n’est pas déjà abonné à la chambre. Le serveur renvoie UNSUBSCRIBE\_OK.

### Valider un message

Le modérateur envoie VALIDATE avec le nom de la chambre, le nom de l’utilisateur et le message à valider. Le serveur envoie à tous les clients abonnés à la chambre une requête NEW\_MESSAGE avec le nom de la chambre, le nom du client dont le message a été validé et le message en question. La fenêtre 4 est mise à jour. Le serveur renvoie VALIDATE\_KO si le serveur n’arrive pas à identifier le message à valider.

### Invalider un message

Le modérateur informe le serveur que le message est invalidé avec la requête INVALIDATE suivie du nom de la chambre, le nom du client dont le message a été rejeté et le message en question. Le serveur renvoie INVALIDATE\_OK indiquant au modérateur que le message a bien été invalidé. Le serveur renvoie INVALIDATE\_KO si le message à invalider ne peut pas être identifié.

### Envoyer un message

Le client envoie NEW\_MESSAGE avec le nom de la chambre, son nom et le message. Le serveur renvoie NEW\_MESSAGE\_KO s’il ne peut pas accepter le nouveau message ou si le client n’est pas déjà abonné à la chambre. Sinon tous les clients abonnés reçoivent NEW\_MESSAGE avec le nom de la chambre qui vient de recevoir un nouveau message, le nom de l’auteur du message et le message en question. Si la chambre est modérée, le serveur transmettra d’abord le message au modérateur via une requête NEW\_MESSAGE. Le modérateur renverra soit VALIDATE soit INVALIDATE et le serveur mettra à jour l’affichage des abonnés si nécessaire.

## Messages client vers serveur et organisation de ces messages

### Message CONNECT

CONNECT string

### Message DISCONNECT

DISCONNECT string

### Message CREATE

CREATE string int\_32

### Message LOAD

LOAD string

### Message SUBSCRIBE

SUBSCRIBE string string

### Message UNSUBSCRIBE

UNSUBSCRIBE string string

### Message VALIDATE

VALIDATE Message

### Message INVALIDATE

INVALIDATE Message

### Message NEW\_MESSAGE

NEW\_MESSAGE Message

## Messages serveur vers client et organisation de ces messages

Généralement les messages X\_KO renvoie un String contenant l’erreur qu’a rencontré le serveur

### Message CONNECT\_OK

CONNECT\_OK

### Message CONNECT\_KO

CONNECT\_KO string

### Message DISCONNECT\_KO

DISCONNECT\_KO string

### Message CREATE\_OK

CREATE\_OK

### Message CREATE\_KO

CREATE\_KO string

### Message LOAD\_OK

LOAD\_OK User

### Message LOAD\_KO

LOAD\_KO string

### Message SUBSCRIBE\_OK

SUBSCRIBE\_OK

### Message SUBSCRIBE\_KO

SUBSCRIBE\_KO string

### Message UNSUBSCRIBE\_OK

UNSUBCRIBE\_OK

### Message UNSUBSCRIBE\_KO

UNSUBSCRIBE\_KO string

### Message NEW\_MESSAGE

NEW\_MESSAGE Message

### Message NEW\_MESSAGE\_KO

NEW\_MESSAGE\_KO string

### Message VALIDATE\_KO

VALIDATE\_KO string

### Message INVALIDATE\_OK

INVALIDATE\_OK

### Message INVALIDATE\_KO

INVALIDATE\_KO string

## Organisation physique des types échangés

### Types de base

Ce sont les types qui devraient exister dans tous les langages de programmation.

Discriminant : long long

Booléen : char ‘T’, ‘F’

Byte : octet

Caractère : utf-8

Enuméré : char[]

Entier court : int\_16 en big-endian

Entier long : int\_32 en big-endian

Flottant : IEEE 754

Chaîne : struct { int\_32 length ; char[] contents } ;

Tableau : struct { int\_32 length ; byte[] contents } ;

### Types spécifiques

User : struct {

Char[] name ;

Room[] subscribedRooms ;

} ;

Room : struct {

Char[] name ;

Bool mode ;

Message[] messages ;

User moderator ;

} ;

Message : struct {

Room room ;

Char[] author ;

Chaîne[] contents ;

} ;