SAÉ 3 - R3.05
Tchatator

Contexte

La plateforme TripEnArvor souhaite mettre en place un mécanisme d'échange de messages au format texte brut entre les professionnels et les visiteurs/clients. Ce mécanisme aurait dû pouvoir fonctionner de façon synchrone et asynchrone, mais par souci de simplicité, on ne vous demande pas de réaliser le mode synchrone. Voici quand même une explication de chaque mode :

- synchrone : les deux correspondants sont connectés et les messages s'échangent en temps réel et en direct.
- asynchrone : les deux correspondants ne sont pas connectés en même temps et les messages sont donc émis puis reçus en décalé, quand le destinataire du message se connecte au service.

VOUS NE RÉALISEREZ DONC QUE LE MODE ASYNCHRONE

Vous allez concevoir et développer une API[1], un mécanisme qui permet de standardiser les échanges entre deux systèmes, généralement par le réseau[2], de façon structurée et sécurisée.

API

Votre API doit permettre, grâce à un protocole d'échange structuré, d'envoyer, de recevoir et de manipuler des messages émis et reçus entre clients du site Web et professionnels de la plateforme. Le détail du protocole est décrit plus loin.

Votre API doit avoir trois modes d'accès :

- Un accès professionnel pour échanger avec ses clients ou potentiels clients
- Un accès client
- Un accès administrateur pour des actions de blocage et de bannissement

[1] **A**pplication **P**rogramming **I**nterface
[2] Pas uniquement, mais de nos jours c'est généralement le cas

Terminologie

Voici les termes utilisés dans ce document et que vous devez aussi utiliser dans la documentation à produire.

SERVEUR ou **SERVICE** : le logiciel que vous allez développer, qui attend et traite les REQUÊTES.

CLIENT : le logiciel qui se connecte et interroge le SERVICE. Ne pas confondre avec le client de la plateforme. Dans ce sujet, le client et le professionnel seront tous deux des CLIENTS du SERVICE Tchatator.

PROTOCOLE : les règles et la GRAMMAIRE permettant à un CLIENT de soumettre des REQUÊTES au SERVICE.

REQUÊTE : une action faite sur le SERVICE par le biais d'un CLIENT en utilisant le PROTOCOLE.

GRAMMAIRE : la syntaxe décrivant comment doivent être formatées les REQUÊTES du PROTOCOLE.

CLÉ D'API : un code secret[3] que le CLIENT envoie au SERVICE pour autoriser son utilisation. La fourniture d'une CLÉ D'API peut donner des droits spécifiques.

PARAMÉTRAGE : un fichier texte contenant des valeurs modifiables par un administrateur, sans besoin de création d'une interface. Ces valeurs sont lues au démarrage du SERVICE.

Attendus

Protocole

Vous devez concevoir un PROTOCOLE pour interroger et faire des actions sur le SERVICE.

Vous devez documenter ce PROTOCOLE pour en expliquer les règles (sa GRAMMAIRE, cf TP sur les Sockets)

Voici un exemple qui s'inspire un peu du protocole HTTP :

- **`LOGIN:<clé_api>`** : pour s'authentifier auprès du SERVICE.
  Réponses possibles :
  - **`200/OK`** : accès autorisé
    + Un token identifiant la connexion

[3] Généralement une longue chaîne hexadécimale (32, 48, 64 chiffres hexa)

    - **403/DENIED** : accès refusé
- **MSG:<token_de_connexion>,<longueur_du_message>,<message>** : envoi d'un message
  Réponses possibles :
    - **200/OK** : message bien reçu et traité
    - **401/UNAUTH** : Client non identifié
    - **416/MISFMT** : message mal formaté
    - **426/TOOMRQ** : Trop de messages reçus

Ceci ne représente qu'un exemple sommaire dont vous pouvez vous inspirer pour créer le vôtre. Il est évidemment non exhaustif et la documentation nécessite plus de détails. Vous devez gérer toutes les erreurs possibles.

Le détail des actions du PROTOCOLE est défini plus loin.

Format des messages

Rien n'est imposé ni suggéré pour le format des messages échangés.

Il faut que votre API soit capable de répondre aux attentes décrites dans le sujet avec notamment :

- Un identifiant unique pour chaque message
- Un horodatage des messages
- Un horodatage de la dernière modification d'un message
- Un émetteur et un destinataire
- Un sens : reçu ou émis
- Un statut de suppression d'un message
- Un format texte brut : aucun attribut typographique n'est attendu, ni la prise en charge d'URL, de tags HTML, etc.
- Une taille maximale d'un message (paramétrable, qu'on fixera à 1000 caractères).
- Une gestion des erreurs par une réponse appropriée comme par exemple pour le dépassement de la taille maximale d'un message.

CLÉ D'API

Ci-après **des parties de texte ~~ont été rayées~~ car on ne vous demande pas de les réaliser**. Nous les avons gardées ici pour préserver du contexte avec le cadre théorique et simulé de la SAÉ. A la place des pages Web qui sont citées, vous développerez un CLIENT en C qui fera les actions attendues (login, envoi d'un message, affichage des messages non lus, de l'historique etc.) et présentera un menu pour faire ces actions. Même si ça demande un peu de codage en amont, ça simplifiera aussi vos tests ensuite.

Il y a trois types d'utilisateurs et donc trois types de CLÉS D'API :

- Le client de la plateforme : ~~il se connectera depuis une page Web du site et pourra échanger avec le professionnel depuis cette page~~. Sa CLÉ D'API est associée à son compte utilisateur "client". Ses droits sont restreints aux seuls envois et modifications de messages.
- Le professionnel : ~~il se connectera aussi depuis une page Web du site, dans son espace gestionnaire de ses offres~~. Il a aussi une unique CLÉ D'API mais elle lui confère des droits plus étendus, notamment celui de (dé)bloquer les messages d'un client vers lui. Il ne peut évidemment pas bannir un client de la plateforme.
- L'administrateur de la plateforme : ~~il se connectera depuis une page dédiée aux administrateurs~~. Possède les droits supplémentaires de bannir ou bloquer l'envoi de message par un utilisateur client ou professionnel.

Pour les deux premiers types de CLÉS D'API, ils doivent être générés (et re-générables) automatiquement depuis la page de profil de l'utilisateur. Elles sont stockées en BDD.

Pour la CLÉ D'API d'administration, vous la créerez manuellement, et vous la stockerez dans le fichier de paramétrage.

Sécurité

Afin de réduire les risques de SPAM / FLOOD de l'API, vous devez mettre en place un système qui limite le nombre de REQUÊTES faites sur un intervalle de temps donné, et que vous devez définir par deux paramètres : un nombre maxi (qu'on fixera à 12) par minute et un autre par heure (qu'on fixera à 90).

Tout message reçu en dépassement doit faire l'objet d'une réponse (du PROTOCOLE) adaptée.

Il faut prévoir deux modes de blocage, par le professionnel, des messages d'un client donné vers lui (c'est détaillé dans le PROTOCOLE plus loin).

Paramétrage

Le fichier de paramétrage (ports, durées de BAN, nombre maxi de messages, taille maxi etc.) doit être auto-documenté[4]. Il est lu au démarrage du SERVICE et doit pouvoir être relu à réception d'un signal que vous choisirez judicieusement. La relecture "en live" doit permettre la prise en compte des nouvelles valeurs sauf pour la file d'attente du socket (voir plus loin).

Dans le sujet, partout où il est fait référence à un paramètre, il faut comprendre qu'il s'agit d'une information qui doit être lue dans ce fichier de paramétrage.

[4] C'est-à-dire dans le fichier lui-même. On utilise souvent le **#** ou le **;** comme caractère pour mettre en commentaire ce qui le suit.

Langages et outils

Vous devez fournir le code source ainsi que la documentation nécessaire à la compilation de votre SERVICE sur une machine Linux (exclusivement). Vous devez disposer, sur votre serveur dédié, de tout ce qui est nécessaire, librairies comprises. Si ce n'est pas le cas, vous demanderez de l'aide.

Langage C & Socket

Le Tchatator, écrit en langage C, doit mettre en œuvre une programmation de sockets.

Options

Le SERVICE doit être lancé en ligne de commande et peut[5] disposer d'options de lancement. Vous pouvez choisir la technique que vous souhaitez pour le passage des options, mais sachez qu'il existe une fonction prévue pour ça : **`getopt()`**.

Documentation

Vous devez produire une documentation suffisamment détaillée pour pouvoir compiler et exécuter votre SERVICE. Ceci doit pouvoir être fait par un développeur, sans nécessiter de connaissances particulières autres que de savoir lancer des commandes dans un Terminal.

Actions du PROTOCOLE

Votre PROTOCOLE doit permettre de faire les actions suivantes et vous devez proposer une GRAMMAIRE précise et sans ambiguïté.

Vous devez penser les réponses de votre API pour en faciliter leur utilisation informatiques et pas uniquement leur simple affichage à l'écran. Même si pour le moment il ne vous est pas demandé de les exploiter dans un logiciel client tel qu'un navigateur Web, il faut toujours avoir cette finalité à l'esprit, on ne sait jamais… 😇

Identification

Pour pouvoir faire une action, le client ou le professionnel doit d'abord montrer patte blanche.

[5] doit ?

Pour cette API, vous devez coder l'utilisation de CLÉS D'API permettant d'identifier le compte de l'utilisateur. Cette identification doit être faite avant que d'autres REQUÊTES puissent être envoyées. Votre PROTOCOLE doit fournir des réponses en conséquence.

Envoi d'un message

Tous les messages sont stockés en BDD mais si le destinataire est présent, le message doit aussi lui être délivré immédiatement et marqué "lu".

Réception des messages non lus

Un CLIENT peut recevoir des messages en son absence (déconnecté).

Il faut donc pouvoir collecter tous les messages non réceptionnés en les triant par date de réception (du plus ancien au plus récent) et sans limite du nombre de ces messages.

Réception d'un historique de messages (lus et émis)

Il est nécessaire d'avoir accès à l'historique des anciens messages en plus des messages non lus.

La réception de ces messages (lus ou émis) doit se faire par bloc de messages jusqu'à concurrence d'un nombre maximal, paramétrable et qu'on fixera à 20.

Il faut aussi pouvoir demander la réception d'un bloc de messages qui précèdent un message dont on spécifie l'identifiant unique. Ceci pourra se faire autant de fois que souhaité, jusqu'au 1er message de l'historique.

Modification ou suppression d'un message

Évidemment la modification ou la suppression d'un message ne peut se faire que sur un message émis.

La modification d'un message doit être identifiable par un "modifié il y a <durée>".

La durée doit être exprimée ainsi :

- "quelques secondes" : si < 1 min
- "quelques minutes" : si < 15min
- "il y a 15, 30, 45 min" : si < 1H, par arrondi au ¼ d'heure
- "il y a <nombre d'heures>" : si < 24H
- "il y a <nombre de jours>" : au-delà de 24H.

Les messages supprimés ne sont pas réellement supprimés en BDD mais ne sont plus délivrés par l'API.

Statut d'un CLIENT

Il doit être possible de connaître le statut de connexion (en ligne, hors ligne) d'un CLIENT du SERVICE

Blocage d'un CLIENT

Permet à un professionnel ou un administrateur de bloquer les messages d'un CLIENT du SERVICE pendant une durée limitée, paramétrable et fixée à 24H.

Opéré par le professionnel, les messages sont uniquement ceux dont il est le destinataire et provenant du client ciblé.

Opéré par un administrateur, il concerne tous les messages du client à tout destinataire.

Passé le délai, le blocage est automatiquement levé.

Les réponses du PROTOCOLE doivent être adaptées à cette situation.

Un blocage peut être levé à tout moment par celui qui l'a activé ou par un administrateur.

Bannissement d'un CLIENT

Le bannissement est similaire au blocage mais est définitif. Il est toujours possible de le lever dans les mêmes conditions que pour le blocage.

Blocage et bannissement ne concernent que le Tchataror, l'accès au site Web n'est pas impacté.

BDD

Votre SERVICE doit accéder à la BDD de la plateforme TripEnArvor. Vous devez donc utiliser la librairie officielle MariaDB ou PostgreSQL :

- https://www.postgresql.org/docs/current/libpq.html
- https://mariadb.com/docs/server/connect/programming-languages/c/

Logs

Votre SERVICE doit produire des logs détaillés dans un fichier texte dont le chemin est paramétrable.

Vos logs doivent être exhaustifs afin de pouvoir suivre précisément ce que fait, ce que reçoit, ce que répond, votre SERVICE, comment il a été lancé, avec quelles options, etc.

Chaque ligne de log doit être préfixée des informations suivantes :

- Date+heure
- Identité du CLIENT (si connue) ou vide
- IP du CLIENT[6]

Service

Votre SERVICE est mono-processus et accepte une seule connexion à la fois. Il doit toutefois gérer une file d'attente[7] paramétrable.

Votre SERVICE doit afficher une aide en ligne par l'utilisation d'une option **`--help`** et, dans ce cas, s'arrêter immédiatement. Inspirez-vous de ce qu'un **`gcc --help`** affiche, par exemple.

Le SERVICE écoute sur toutes les interfaces et sur un port paramétrable.

Si l'option **`--verbose`** est spécifiée, vous devez aussi produire les logs détaillés à l'écran en plus du fichier de logs

Livrables

Voici la liste des livrables attendus, à déposer sur Moodle :

- Un lien vers une vidéo démontrant de façon claire et détaillée tous les éléments du PROTOCOLE (via Telnet par exemple). L'écran capturé ne doit pas dépasser 1024x768. Il est obligatoire d'avoir un commentaire vocal et le son doit être audible. Vous pouvez ajouter un sous-titrage si vous voulez
- Le code source C du SERVICE et des autres éventuels outils complémentaires + une documentation pour compiler et pour exécuter en ligne de commande ;
- Un document PDF détaillant le PROTOCOLE
- Le code source C (et sa documentation PDF) d'un logiciel CLIENT permettant de tester le SERVICE
- Une documentation PDF de tous les cas d'utilisation par l'exemple (pratique)
- Un PDF de répartition (en %age) du travail individuel par tâche. Au minimum ceci, à détailler/compléter si utile :
    - Conception du PROTOCOLE
    - Codage et tests
    - Documentation.

[6] Voir le paramètre 2 du **`accept()`**
[7] Voir le paramètre 2 du **`listen()`**

🏆 Notation et Évaluation

Vous serez notés sur l'ensemble des livrables et vous aurez une évaluation supplémentaire et individuelle qui viendra pondérer cette note au sein de l'équipe.

Cette évaluation durera 15 min en salle de DS.

Elle portera sur votre connaissance du travail réalisé. Vous devrez être capable d'identifier le travail de votre équipe et de répondre à des questions sur le mécanisme mis en place par votre équipe et sur son mode de fonctionnement.

Précisions

En **R4.01 - Architecture logicielle**, nous étudierons une autre technique de développement d'API qui s'appelle les WebServices mais ce n'est pas ce mode de fonctionnement qui est attendu dans la SAÉ.

Enfin, notez que les éventuelles mises à jour de ce document vous seront signifiées et seront matérialisées par un surlignement des parties modifiées ou ajoutées de cette façon depuis la version précédente.