Documentation pour le projet 2

Membre de l’équipe :

* Simon-Pierre Fortin
* Michèle Mouafo
* Alexandre Jolicoeur
* Lindsay Radford

# Architecture du projet

Le projet présente deux composantes principales :

* Le serveur, qui comprend les éléments suivants :
  + Un fichier serveur.js pour le code du serveur même
  + Un fichier courriels.json qui est une base de données des courriels de chaque utilisateur
* Le client, qui comprend les éléments suivants :
  + Un fichier index.html comprenant la page Web chargé à l’exécution du serveur
  + Un fichier script.js comprenant le code à exécuter pour la page Web
  + Un fichier style.css pour ajouter du style à la page Web
* Plusieurs autres fichiers sont compris, mais il s’agit de librairies externes

## Architecture du fichier courriels.json

Voici la structure du fichier :

{

"-----BEGIN PUBLIC KEY----- (Clé publique ici) -----END PUBLIC KEY-----": {

"user": "(Nom de l’utilisateur ici)",

"emails": [(Array de courriels chiffrés ici)]

},

"-----BEGIN PUBLIC KEY----- (Clé publique ici) -----END PUBLIC KEY-----": {

"user": "(Nom de l’utilisateur ici)",

"emails": [(Array de courriels chiffrés ici)]

},

Et ainsi de suite

}

La clé publique permet de sélectionner le bon utilisateur, ensuite on utilise la clé « emails » de l’objet pour obtenir le array des courriels chiffré. Cet array sera envoyé au client selon la clé publique transmise.

## Architecture du fichier serveur.js

Les principaux modules utilisés sont les suivants :

* *Express* pour la création du serveur Web
* *Net* pour la communication entre serveur
* *Json file object* pour gérer le fichier courriels.json

**Express** est utilisé pour réagir aux requêtes POST et GET envoyées par le client.

**Réponse à une requête GET** :

On s’attend à recevoir une clé publique du client. On utilise ensuite cette clé pour obtenir le array de courriels chiffrés associé à cette clé, ce qu’on retourne au client.

**Réponse à une requête POST :**

On s’attend à recevoir une clé publique et un message chiffré du client. On utilise la clé publique pour déterminer à qui s’adresse le courriel. Ensuite, on utilise la méthode « unshift » sur l’objet JSON obtenu en ouvrant le fichier courriels.json pour ajouter le courriel chiffré au array du destinataire.

**Net** est utilisé pour effectuer la communication entre serveurs. On utilise ici des websockets pour effectuer la communication. Un serveur créer un serveur permettant de recevoir des connexions.

Chaque fois qu’un serveur distant se connecte, un échange de l’objet JSON contenant les courriels est effectué. Les serveurs effectuent une comparaison pour savoir si chaque serveur a bien tous les courriels de chacun, si ce n’est pas le cas, il utilise la méthode « unshift » pour les ajouter à son objet JSON.

Chaque serveur est également doté de la fonction suivante :

function connect\_to\_distant (port\_distant, adresse\_distant) {…}

Elle permet de créer des connexions à un serveur distant en précisant le port et l’adresse IP du serveur distant. Chaque fois qu’un serveur se connecte à un serveur distant, le même échange d’objet JSON est effectué pour vérifier si les courriels de chaque serveur sont bien à jour.

**Json file object** est utilisé pour gérer le fichier courriels.json. On utilise la fonction suivante :

json\_file\_object({value:{}, file:"data/courriels.json", saveEverySecs:5, forceNew:false})

On associe la variable « my\_obj » à cette fonction. Elle permet d’ouvrir le fichier courriels.json et de placer son contenu dans la variable « my\_obj ». Cette fonction vérifiera la variable aux 5 secondes et mettra à jour le fichier courriels.json selon ce qui se trouve dans cette variable.

## Architecture du fichier script.js

Ce fichier comprend presque toutes les fonctionnalités comprises dans le projet 1. Quelques fonctions ont été supprimées afin de simplifier le site. Nous avons ajouté une fonction qui simule le log in à la page principale du site Web. Cette page permet de déterminer qui utilise le site. On pourra ainsi lui retourner les courriels qui lui sont destiné. À la sélection de l’utilisateur, on lui détermine ses clés publique et privée.

Pour les fonctions pertinentes au projet 2, on a utilisé Ajax et JQuery.

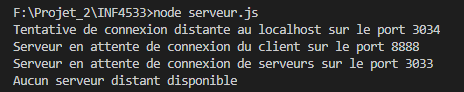
Pour la requête GET, le client envoie sa clé publique au URL /getLetters comme paramètre et le serveur lui envoie un array de courriels chiffrés. On utilise ensuite la clé privée pour déchiffrer les courriels à l’aide de fonction « decrypt\_message (message, clé privée) ». Les courriels sont traités de façon différente selon la fonction du client qui effectue la requête GET.

Pour la requête POST, le client utilise la fonction « encrypt\_message (message, clé publique du destinataire) » pour chiffrer le message, puis il envoie un colis dans le format « {"dest": clef\_dest, "msg": nouveau\_courriel\_crypted} » à l’URL « /addLetters ».

# Fonctionnement du projet

Exécution unique locale

Pour exécuter le serveur en instance seule localement, vous devez naviguer jusqu’au répertoire du serveur dans une fenêtre d’invite de commande ou Bash, puis faire les étapes suivantes :

* Exécuter la commande « node serveur.js ».
* Les indications de connexion suivantes s’afficheront à la fenêtre : 
* Dans un navigateur, inscrivez dans la boîte d’adresse « [http://localhost:8888](http://localhost:8888/) ».
* Vous pouvez modifier le port du client dans le code, vous n’aurez qu’à modifier le port dans l’adresse également.
* Une fois la page ouverte en navigateur, vous devriez arriver à la page principale.
* Choisissez votre utilisateur et consultez ses courriels ou écrivez-en à d’autres personnes comprises dans le carnet.
* Pour l’instant, les adresses de destinataire sont limitées aux personnes se trouvant dans le carnet.
* Vous remarquerez que les courriels des boites de réception de chacun se mettra à jour avec les nouveaux courriels reçus chaque fois que vous irez les consulter par la suite.

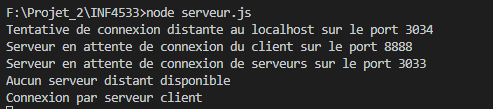
Exécution pour communication entre serveurs

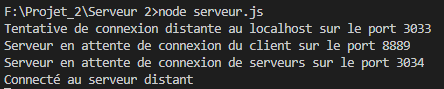
Pour tester la communication entre serveurs, vous devez cloner le répertoire du projet et modifier les ports de la deuxième instance pour avoir une bonne communication entre les serveurs.

Modifiez les ports aux lignes 16 et 102 du fichier serveur.js du deuxième serveur pour qu’elle soit différentes du premier. Ensuite indiquez les bons ports à la ligne 48 du fichier de chaque serveur pour avoir une communication fonctionnelle.

Si tout se passe bien, vous verrez les messages suivants s’afficher en console.

Pour le serveur qui reçoit la connexion :



Pour le serveur qui se connecte au serveur distant :

La communication entre serveur s’effectuera à la connexion et les listes de courriels seront comparées, puis mises à jour selon les listes de chaque serveur.

# Aspects à améliorer ou à ajouter au projet

Nous aurions souhaité ajouter davantage de fonctionnalité au projet, mais par manque de temps, nous nous sommes limités aux fonctionnalités de base indiquées dans le présent document.

Voici une liste de fonctionnalités ou d’amélioration souhaitées :

* Rafraichissement de la boîte de réception à l’arrivée d’un nouveau courriel sans cliquer à nouveau sur le bouton « Boite de réception ». Même chose pour la recherche dans les courriels.
* Ajout d’un carnet d’adresse sur le serveur comprenant des clés privées chiffrées à l’aide d’une passphrase, ce qui permettrait d’ajouter de nouveaux utilisateurs au site de façon ponctuelle.
* Créer un serveur « arbitre », ce qui permettrait d’éviter d’aller modifier manuellement les adresses et ports de connexion aux autres serveurs distants. Les connexions aux autres serveurs seraient automatisées de cette façon.
* Créer une vraie authentification à l’ouverture de la page Web avec nom d’utilisateur et mot de passe pour éviter que les utilisateurs puissent aller cliquer sur les boutons de la barre de gauche et pour qu’ils ne puissent pas aller lire les courriels des autres.