Documentation

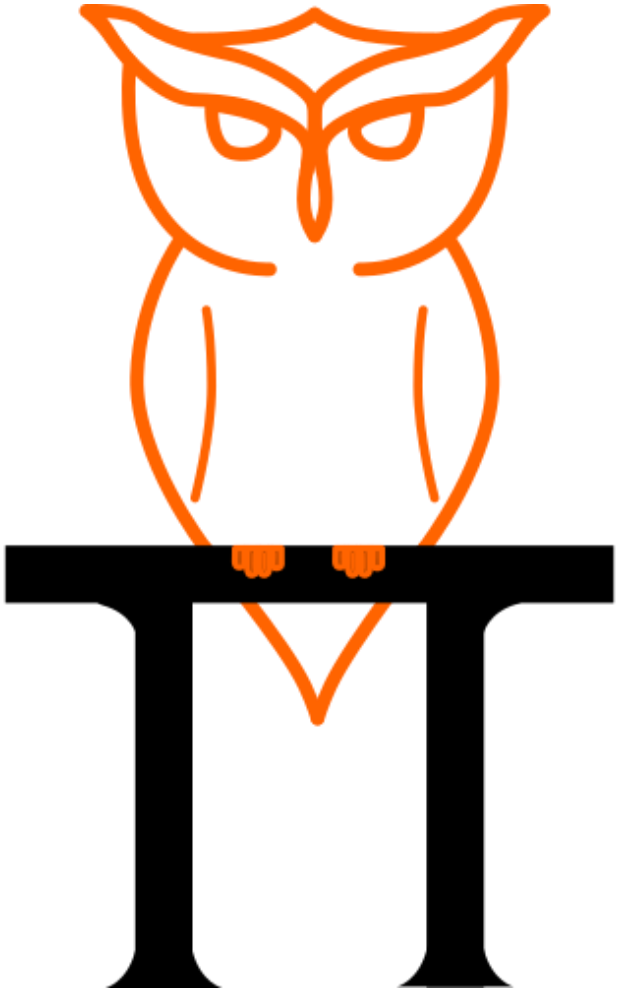
- ## Spécifications techniques :

(1) Choix du logo

Nous avons choisi pour notre logo de représenter un **hibou perché sur un Pi**.

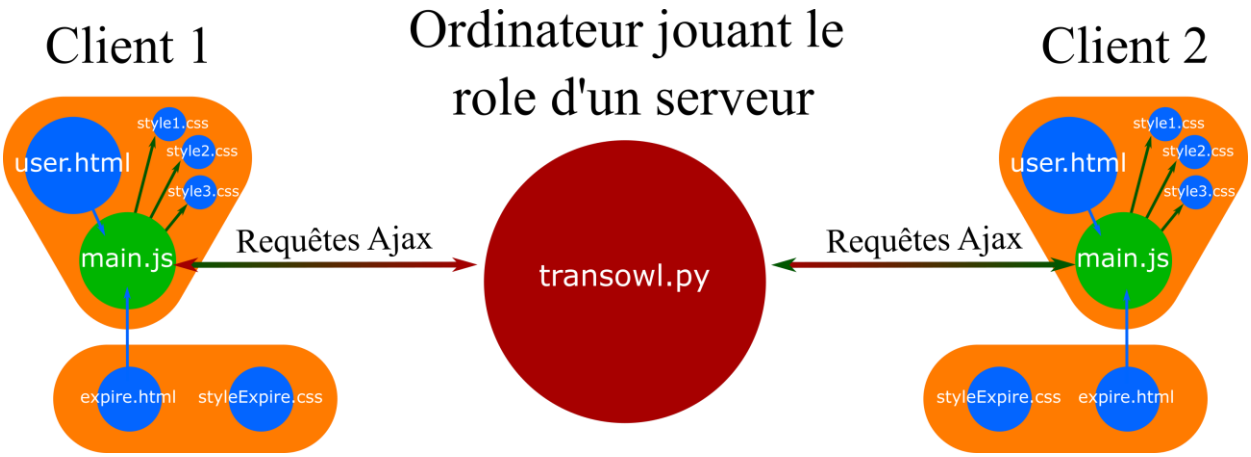
Le nombre Pi avait évidemment sa place sur notre logo, TransCrypt reposant sur ses particularités. S'il a eu et a toujours différentes représentations, nous avons préféré celle se rapprochant le plus d'un perchoir pour également y mettre en avant un hibou.

Le hibou a divers symboliques à travers les âges. Il est tout d'abord associé à la **connaissance**, ce qui est précisément le but d'une communication. Son côté "surnaturel", sans compter sa capacité à voler, à s'élever, en ferait ensuite un représentant adéquat de la **transcendance** du nombre pi. Enfin, en particulier dans les récentes fictions, le hibou est considéré comme un **animal messager**.



(2) Architecture

TransOwl a l'architecture suivante :



(3) Langages et bibliothèques utilisés

Pour développer TransCrypt et mettre en place TransOwl, nous avons utilisé plusieurs langages, étudiés en cours de spécialité NSI. Nous nous sommes ainsi servis du **python** (*TransCrypt initial, backend*), du **javascript** *(frontend & TransCrypt dans la messagerie*), de l'**HTML** (*frontend*) et du **CSS** (*frontend*). Nous les avons choisis car ils nous ont semblés les langages les plus adaptés à chaque partie du projet.

Nous avons également exploité des bibliothèques et outils comme **jQuery** (*pour **AJAX***), **Flask** (*micro-framework*), **Math** (*calculs mathématiques*).

(4) Matériel

Notre projet n'étant pas physique, il ne nécessite et n'a nécessité que **peu de matériel**, rendant sa **mise en place d'autant plus facile**. Nous n'avons donc eu besoin pour ce projet que d'ordinateurs et d'une connexion internet. Concernant la vidéo, elle a été tournée grâce à un téléphone - téléphone servant aussi dans la démonstration. Les plans ont été réalisés par nous-mêmes ou avec l'aide d'autres camarades lorsque ce n'était pas possible.

(5) Format de stockage des données

Les données ne sont pas conservées sur le serveur. Toutefois, elles peuvent prendre différentes formes au fil de la procédure d'envoi. Le message est d'abord une **chaîne de caractères** (String ou str). Au cours du chiffrement/déchiffrement, il est ensuite enregistré sous les formats **hexadecimal** (hex) et **entier** (BigInt pour ne pas faire face aux limites du javascript sur les entiers), avant d'être reconverti en chaîne de caractères.

- **Spécifications fonctionnelles :**

(6) Fonctionnement de l'algorithme de chiffrement TransCrypt

Le fonctionnement de TransCrypt repose sur des concepts simples.

Scénario/protocole : Une personne A souhaite envoyer un message clair M_1p chiffré à la personne B qui lui répond.

1- A obtient le delta Δ_{1c} du déplacement correspondant à la position du début du bandeau dans les décimales de Pi, sauvegardé.
2- A calcule un nouveau delta Δ_{2c} pour M_{2c} de B et applique un ou exclusif XOR $\Delta_{2c}= b_0 * b_2 \otimes b_1 * b_4$ des premiers octets de M_{1p} et le sauvegarde
3- A transforme le message M_{1p} de base binaire en base hexadécimale
4- A obtient la longueur L_{1p} en octets du message M_{1p}
5- A obtient par BBP un bandeau de π de longueur L_{1p} débutant position Δ_{1c}
6- A applique un ou exclusif XOR $M_{1c} = M_{1p} \otimes \pi[\Delta_{1c}, \Delta_{1c} + L_{1p}]$
7- A communique M_{1c} à B
8- B obtient le delta Δ_{1c} du déplacement correspondant à la position du début du bandeau dans les décimales de Pi, sauvegardé.
9- B obtient la longueur $L_{1c} = L_{1p}$ en octets du message M_{1c}
10-B obtient par BBP un bandeau de π de longueur L_{1c} débutant position Δ_{1c}
11-B applique un ou exclusif XOR $M_{1p} = M_{1c} \otimes \pi[\Delta_{1c}, \Delta_{1c} + L_{1c}]$
12-B transforme le message M_{1p} de base hexadécimale en base binaire
13-B calcule un nouveau delta Δ_{2c} et applique un ou exclusif XOR $\Delta_{3c}= b_0 * b_2 \otimes b_1 * b_4$ des premiers octets de M_{2p} et le sauvegarde
14-B obtient le delta Δ_{1c} du déplacement correspondant à la position du début du bandeau dans les décimales de Pi, sauvegardé.
15-B calcule un nouveau delta Δ_{2c} pour M_{3c} de A et applique un ou exclusif XOR $\Delta_{2c}= b_0 * b_2 \otimes b_1 * b_4$ des premiers octets de M_{2p} et le sauvegarde
16-B transforme le message M_{2p} de base binaire en base hexadécimale
17-B obtient la longueur L_{2p} en octets du message M_{2p}
18-B obtient par BBP un bandeau de π de longueur L_{2p} débutant position Δ_{2c}
19-B applique un ou exclusif XOR $M_{2c} = M_{2p} \otimes \pi[\Delta_{2c}, \Delta_{2c} + L_{2p}]$
20-B communique M_{2c} à A
21-A obtient le delta Δ_{2c} du déplacement correspondant à la position du début du bandeau dans les décimales de Pi, sauvegardé.
22-A obtient la longueur $L_{2c} = L_{2p}$ en octets du message M_{2c}
23-A obtient par BBP un bandeau de π de longueur L_{2c} débutant position Δ_{2c}
24-A applique un ou exclusif XOR $M_{2p} = M_{2c} \otimes \pi[\Delta_{2c}, \Delta_{2c} + L_{2c}]$
25-A transforme le message M_{2p} de base hexadécimale en base binaire
26-A calcule un nouveau delta Δ_{3c} et applique un ou exclusif XOR $\Delta_{3c}= b_0 * b_2 \otimes b_1 * b_4$ des premiers octets de M_{2p} et le sauvegarde

NOTA BENE : une preuve de concept (ou *Proof of concept POC*) dans le langage Python version 3 est fournie.

(7) Implémentation en javascript

- **Chiffrement**

```
92  function crypt(clef, msg){
93      /*
94      Applique transcrypt à un msg avec une clef donnée
95       */
96      return xor_two_str(msg,pi_hex(clef, msg.length)); // renvoie le message chiffré
97  }
```

```
function pi_hex(n, prec){
    /*
    calcul avec BBP : on obtient un bandeau de pi de longueur prec débutant à la position n, en hexadécimal
     */
    n=Math.abs(Math.floor(n));
    prec=Math.abs(Math.floor(prec));
    n-=1;
    let a=[4n,2n,1n,1n]; //coef de séries dans BBP
    let j=[1n,4n,5n,6n];
    let D = BigInt(pi_dn(n, prec));
    //n = BigInt(n);
    let x = (a[0]*pi_ser(j[0], n, prec)
        - a[1]*pi_ser(j[1], n, prec)
        - a[2]*pi_ser(j[2], n, prec)
        - a[3]*pi_ser(j[3], n, prec)
        ) & (16n**D - 1n);

    prec = BigInt(prec);
    x = x / 16n**(D - prec);
    x = x.toString(16);
    return x;
}


function pi_dn(n,prec){
    /*
    Calcul de D qui va assurer la précision des décimales du calcul de pi jusqu'au rang n+prec+1
     */
    return Math.floor(Number(Math.log(n + prec + 1)/Math.log(16) + prec + 3));
}
```

```
function pi_ser(j, n, prec){
    /*
    Calcule la somme des 16^(n-k)/(8k + j) pour k allant de 0 à D
    où D est une valeur de sécurité assure une certaine précision des décimales jusqu'au rang n + prec

    On travaille avec des bigint puisqu'on se retrouve avec des nombres d'au moins prec chiffres hex, soit 4prec chiffres binaires.
    La limite des entiers normalement utilisés en javascript est 2^53 - 1, (attribut publique MAX_SAFE_INTEGER de la classe Number)
     */

    let D = BigInt(pi_dn(n, prec));
    let D4 = 4n*D;//Pour faire des "bitshifts" de base 16;
    n = BigInt(n);

    let d = BigInt(j); //denominateur
    let s = 0n; // somme

    // Comme on s'intéresse uniquement aux décimales à partir du rang n, on peut calculer sous modulo les premiers membres de la somm
    for (var k = 0n; k < n+1n ; k++){
        s += (((16n**(n-k))%d)<<D4)/d; //
        d += 8n;
    }

    let t = 0n;
    let e = D4 - 4n;
    d = 8n*k + BigInt(j);

    //On continue de sommer jusqu'à ce que les membres de la somme atteignent le nombre voulu
    while(true){
        let dt = (1n<<e)/d;
        if (!dt){
            break;
        }
        t += dt;
        e -= 4n;
        d += 8n;
    }
    return (s + t) ;
}
```

```
function xor_two_str(a,b){
    /*
    Applique l'opérateur ou exclusif entre deux châines de caractères
    */
    let xored = "";
    for (let i = 0; i < a.length;i++){
        let u = (a.charCodeAt(i)^b.charCodeAt(i%b.length)).toString(16); //xor puis conversion en hex
        if (u.length < 2){ u = "0"+u }
        xored += u;
    }
    return xored;
}
```

- **Déchiffrement**

Le déchiffrement utilise les mêmes fonctions que le chiffrement, avec, en prime, celles ci-dessous :

```javascript
function hex_to_ascii(str1) {
    /*
    Convertit une chaîne de caractères hexadécimale en caractères ascii
     */
    let hex  = str1.toString();
    let str = '';
    for (let n = 0; n < hex.length; n += 2) {
        str += String.fromCharCode(parseInt(hex.substr(n, 2), 16));
    }
    return str;
}


function decrypt(clef, msg){
    /*
    Inverse de crypt.
    On ne peut réutiliser crypt du fait de l'implémentation du xor en javascript.
     */
    return hex_to_ascii(xor_two_str(hex_to_ascii(msg), pi_hex(clef, msg.length)));
}
```

(8) Prérequis et ouverture de TransOwl

Comment lancer le serveur :
$ signifiera que ce qui suit est une commande à entrer dans un terminal.
Nous supposerons que pip est déjà installé sur la machine qui servira de serveur.

(optionnel) **Préférablement créer un environnement python :**
D'abord, nous conseillons de créer un environnement pour les modules de python qui seront installés, bien que cela ne soit pas absolument nécessaire, pour éviter de possibles bugs de compatibilité avec d'autres modules installés sur l'ordinateur.

Pour cela d'abord installer venv :
Dans cmd : $py –m pip install virtualenv
Ou, dans BASH: $python3 –m pip install virtualenv

Ensuite pour créer l'environnement :
Cmd : $py -m venv [nom dossier]
BASH : $python3 –m venv [nom dossier]

Pour le lancer il suffira d'entrer dans la console de commande :
Cmd : $[nom dossier]\Scripts\activate.bat
BASH : $source [nom dossier]\bin\activate

1) Installer les bibliothèques :

Cmd : $py –m pip install -r requirements.txt
BASH : $python3 –m pip install –r requirements.txt

Alternativement (déconseillé) :
Cmd : $py -m pip install flask
BASH : $python3 –m pip install flask

2) Configurer le serveur :

D'abord, obtenir l'adresse de la machine sur le réseau local :
Sur windows, via la commande $ipconfig, et sous la majorité des installation linux $hostname –I

Modifier ensuite la dernière ligne de transowl.py, remplaçant 10.0.0.1 par l'adresse obtenue précédemment. En enlevant l'option host=, le serveur tournera par défaut en localhost (127.0.0.1) et ne sera accessible que depuis la machine qui héberge le serveur.

3) Exécuter transowl.py

Pour cela, télécharger le fichier zip contenant transowl.py, et, dans cmd : se placer dans le dossier contenant le programme (ex : *C:/Users/adalovelace/Downloads/transowl/transowl.py*), puis indiquer $python transowl.py.

Un message de la sorte devrait apparaître (ici en localhost):

```
C:\Users\                \Downloads\transowl (1)\v0.3>python transowl.py
 * Serving Flask app 'transowl' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 207-028-482
```

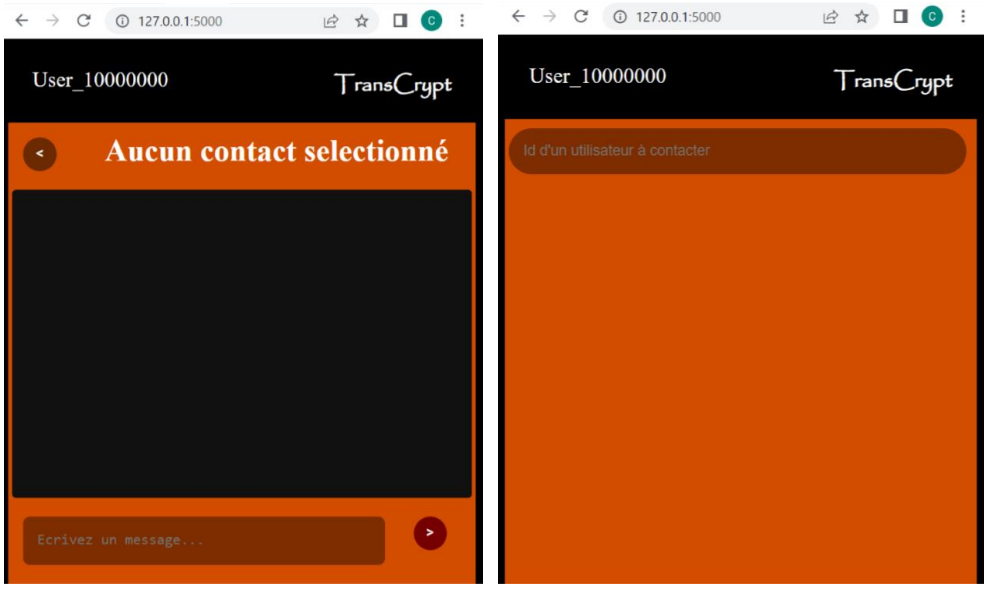
Les autres appareils du LAN pourront accéder à la messagerie en renseignant cette adresse sur la barre de recherche de leur navigateur. Nous recommandons d'utiliser Chrome, qui est le navigateur qui reste compatible avec le plus de fonctionnalités du css.

(9) Utilisation de TransOwl

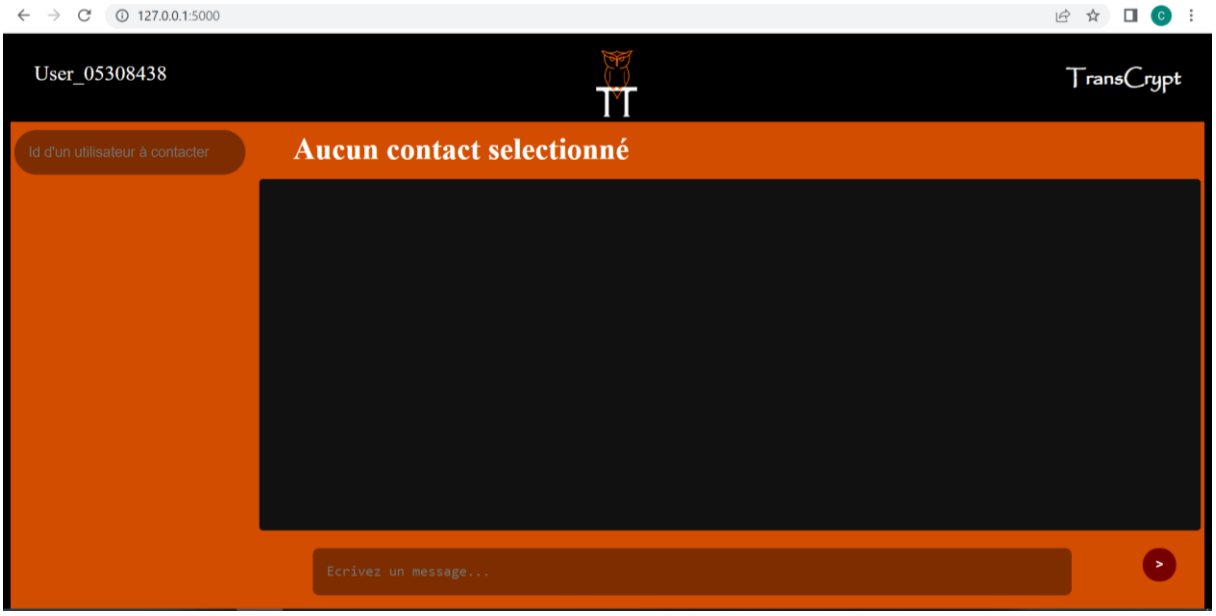
1- En lançant transowl.py, ses différentes fonctions seront exécutées (voir (10)) et une adresse serveur d'exécution sera indiquée dans l'invite de commandes (à changer en fonction des besoins). Il suffit alors de recopier celle-ci dans votre barre de recherche pour accéder à la page.
2- Au chargement de la page, la fonction initialisation() de main.js est appelée, adaptant le css aux dimensions de la page et attribuant un identifiant et un code secret à l'utilisateur.

```
function initialisation () {
    /*
    Fonction appelée lors du chargement de la page.
    Elle adapte le css aux dimensions de la page, et fait une requête au serveur pour obtenir un identifiant et un code
    'secret', afin que le serveur puisse vérifier, lors des échanges futurs, que l'identité du client n'est pas usurpée.
    Aucun paramètre n'est utilisé.
     */

    document.getElementById("contenuContact").style.maxHeight = window.innerHeight - 100 + "px";
    document.getElementById("contenuMessage").style.maxHeight = window.innerHeight - 257 + "px";
    document.getElementById("contenuMessage").style.minHeight = window.innerHeight - 257 + "px";


    $.ajax({
    type: "POST",
    url: "/getid",
    contentType: "application/json",
    dataType: 'json',
    success: function(result) { //un dictionnaire de la forme {"userid": , "usercode": }
            userid = result["userid"];
            usercode = result["usercode"];
            document.getElementById("nameUser").innerHTML = "User_"+userid;
        }
    });
}
```

3- Arrivée sur la liste de contacts
Sur la version mobile vous accéderez d'abord à un écran d'accueil, il faudra cliquer sur le bouton en haut à gauche pour accéder à la barre de recherche et la liste des contacts :



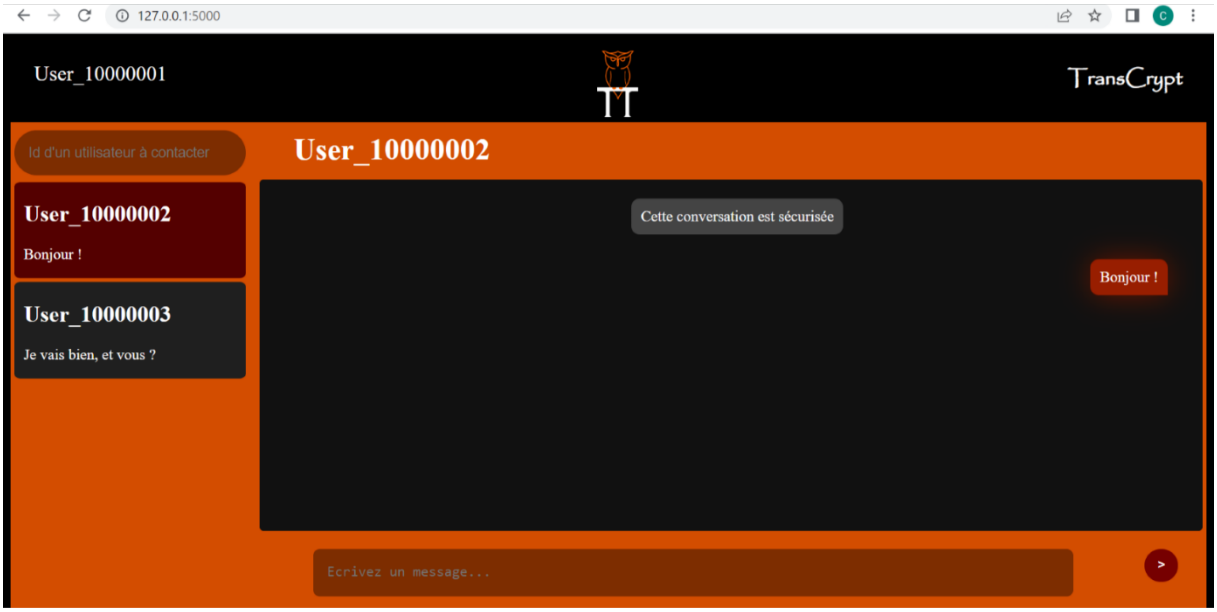
En utilisant la version pour ordinateur, vous aurez également sous les yeux une page de garde :



4- Choisir son interlocuteur

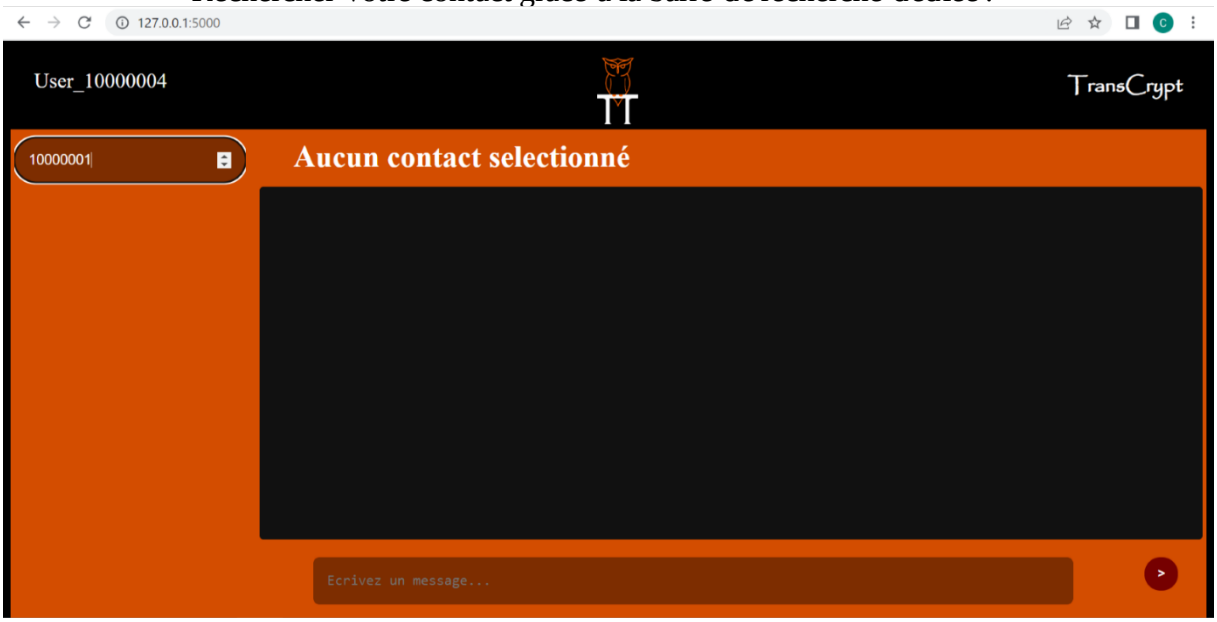
Deux options s'offrent à vous :

- Cliquer directement sur votre contact si celui-ci est enregistré :



Au clic sur ce contact, la fonction afficherConversationContact(e) de main.js (décrite à la page suivante) est appelée.

- Rechercher votre contact grâce à la barre de recherche dédiée :



Les fonctions suivantes de main.js s'exécutent alors tour à tour suivant les cas.

```
function chercherNouveauContact(e) {
    /*Fonction appelée quand le client saisit un caractère dans le champ
            dédié à la recherche de nouveaux contacts.
    Elle permet de regarder si l'utilisateur que le client recherche
            est déjà dans sa liste de contact. S'il ne l'est pas, la
            fonction doesUserBe est appelée lorsque le client presse
            la touche Entrer.
    Le paramètre "e" est la touche que le client presse dans le champ
            dédié à la recherche de nouveaux contacts. */

    let sentId = document.getElementById("idNouveauContact").value;
    let bool = false;

        let listeDiscussions = document.querySelectorAll(".convContainers");

        listeDiscussions.forEach(function(userId) {
            if (userId.getAttribute("id") == sentId) {
                bool = true;
            }
        });
    if(e.keyCode==13 && bool==false && sentId!=""){
        doesUserBe(parseInt(sentId));
    }
}
```

Si le client recherché n'est pas dans la liste de contacts, une requête au serveur est adressée pour savoir si ce client est connecté au serveur :

```
function doesUserBe(id){
    /*
    Fonction appelée quand le client saisit un caractère dans le champ dédié à la recherche de nouveaux contacts, et que
    le client dont l'identifiant a été saisi, n'est pas dans les contacts du client.
    Elle permet de faire une requête au serveur afin de savoir si l'identifiant renseigné est associé a un client
    connecté au serveur.
    Le paramètre "id" est l'identifiant à rechercher.
    */

    $.ajax({
        type:"POST",
        url : "/tobeornottobe",
        data : JSON.stringify({"id":id.toString()}),
        contentType : 'application/json',
        dataType : 'json', //le serveur renvoie un objet de la forme {"exists":} exists = 0 ou 1
        success : function(result){
            if (result["exists"] == 1){
                newConv(id);
            } else {
                alert("L'utilisateur " +  id.toString() + " n'existe pas");
            }
        }
    });
}
```

Si l'identifiant renseigné est associé à un client connecté au serveur, une nouvelle discussion est alors créée et l'identifiant ajouté à la liste de contacts :

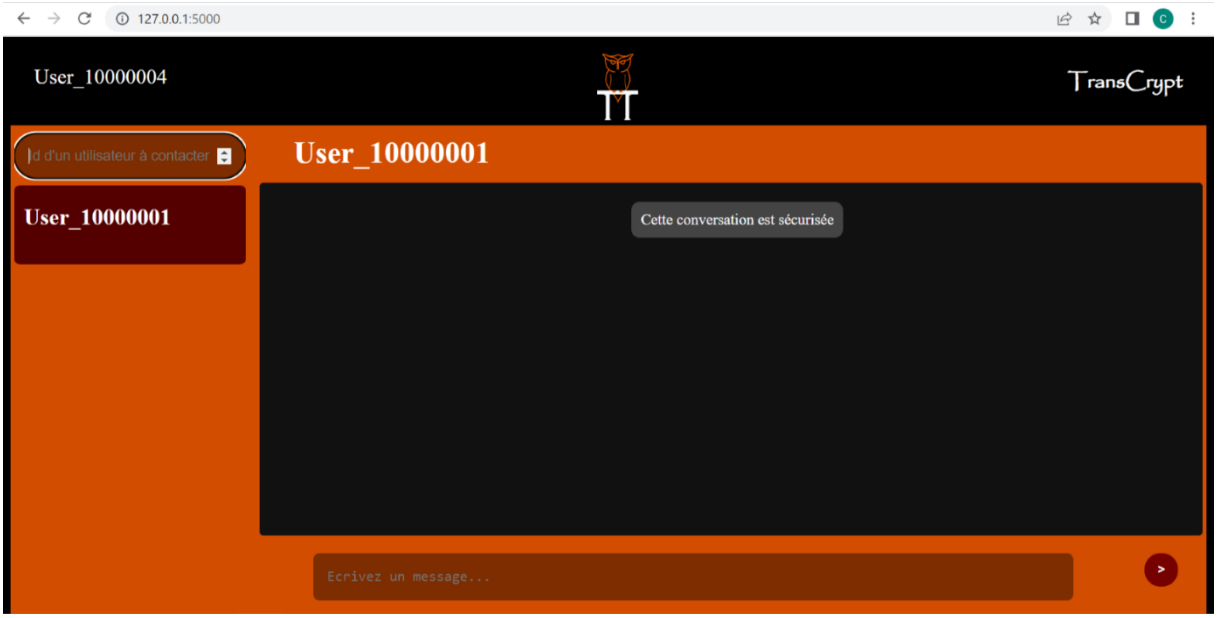
```
function newConv (id) {
    /*
    Fonction appelée lorsque le client ouvre une nouvelle conversation avec un autre utilisateur, et lorsque le client
    reçoit un message d'un utilisateur avec qui il n'a jamais conversé.
    Elle permet de créer un espace de discussion avec le nouveau contact, et avec le message introducteur (qui rappelle
    que l'échange est chiffré). Elle appelle aussi les fonctions afficherContact et afficherConversationContact.
    Le paramètre "nom" est l'identifiant du nouveau contact.
    */

    document.getElementById("idNouveauContact").value = "";
    let messageIntro = document.createElement("div");
    messageIntro.setAttribute("class", "messageIntro");
    messageIntro.appendChild(document.createTextNode("Cette conversation est sécurisée"));
    let convContainer = document.createElement("div");
    convContainer.setAttribute("id", id);
    convContainer.setAttribute("class", "convContainers");
    convContainer.appendChild(messageIntro);
    document.getElementById("contenuMessage").appendChild(convContainer);
    afficherContact(id, "");
    let sentIdUser = id+"case";
    afficherConversationContact(sentIdUser);
}
```

```
function afficherContact (nom, lastMessage){
    /*
    Fonction appelée lorsque le client ouvre une nouvelle conversation avec un autre utilisateur, et lorsque le client
    reçoit un message d'un utilisateur avec qui il n'a jamais conversé.
    Elle permet de créer dans la liste des contacts, un nouveau bouton cliquable associé à l'utilisateur "nom".
    Le paramètre "nom" est l'identifiant du contact que représente le bouton et "lastMessage" est le dernier message
    échangé avec ce contact.
     */

    let boxContact = document.createElement("div");
    let nameContact = document.createElement("h1");
    let latestWord = document.createElement("p");

    boxContact.setAttribute("id", nom + "case");
    boxContact.setAttribute("onclick", "afficherConversationContact(this.id)");
    boxContact.setAttribute("class", "caseContact");
    nameContact.setAttribute("class", "nomContact");
    latestWord.setAttribute("class", "derniersMots");
    latestWord.setAttribute("id", "derniersMots"+nom);

    nameContact.appendChild(document.createTextNode("User_"+nom));
    latestWord.appendChild(document.createTextNode(lastMessage));

    boxContact.appendChild(nameContact);
    boxContact.appendChild(latestWord);

    document.getElementById("contenuContact").appendChild(boxContact);

    document.getElementById("messageAEnvoyer").dataCible = nom;
}
```

Par clic ou recherche, dès lors que le destinataire indiqué existe et est connecté au serveur, la page de conversation avec lui s'ouvre et il devient destinataire du prochain message :

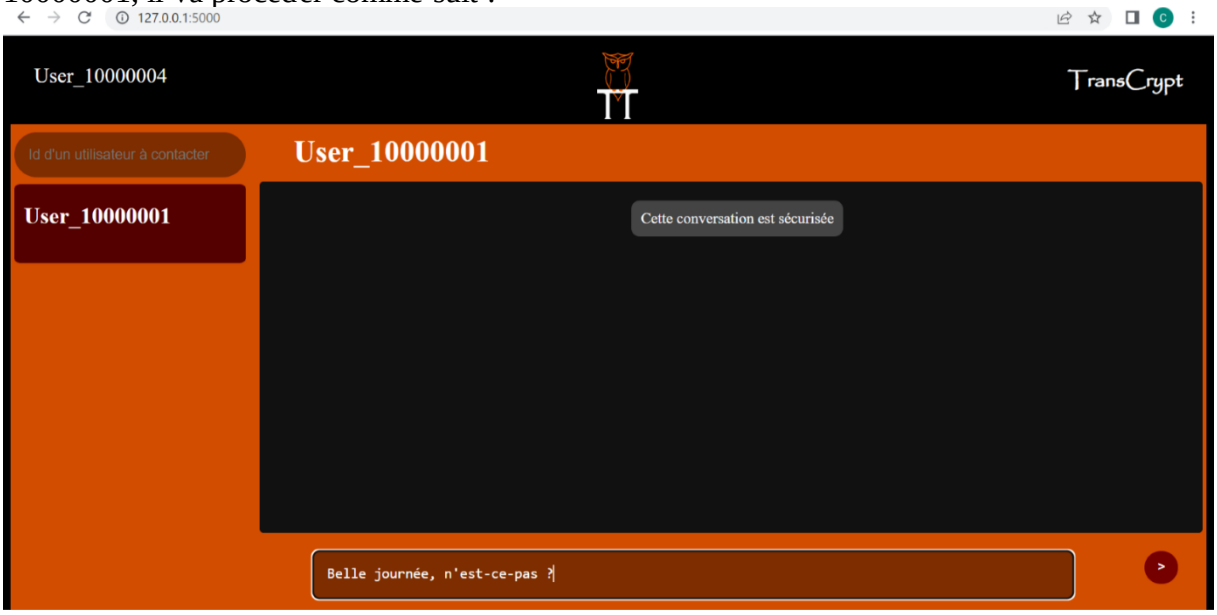
```
function afficherConversationContact (e) {
    /*
    Fonction appelée lorsque l'on clique sur le bouton associé à un  contact, lorsque le client ouvre une nouvelle
    conversation avec un autre utilisateur et lorsque le client reçoit un message.
    Elle permet de modifier le css d'un bouton associé à un contact, lorsqu'il est cliqué, d'afficher la conversatio
    avec ce contact, et de définir ce contact comme cible du prochain message (première ligne de la fonction).
    Le paramètre "e" est l'id du bouton associé au contact d'identifiant e.slice(0, -4)
     */

    document.getElementById("messageAEnvoyer").dataCible = e.slice(0, -4);

    let listeContacts = document.querySelectorAll(".caseContact");
    listeContacts.forEach(function(userId) {
            document.getElementById(userId.getAttribute("id")).style.backgroundColor = "#1F1F1F";
    });
    document.getElementById(e).style.backgroundColor = "#540000";

    let listeDiscussions = document.querySelectorAll(".convContainers");
    listeDiscussions.forEach(function(userId) {
            document.getElementById(userId.getAttribute("id")).style.display = "none";
    });
    document.getElementById(e.slice(0, -4)).style.display = "initial";

    document.getElementById("barreContact").innerHTML = "User_"+e.slice(0, -4);

    document.getElementById("contenuMessage").scrollTo(0, document.getElementById("contenuMessage").scrollHeight);

    if (window.innerWidth <= 630){
        document.getElementById("contenuContact").style.display = "none";
        document.getElementById("main").style.display = "initial";
    }
}
```

Ce qui donne :



5- Ecrire son message

L'utilisateur peut, une fois arrivé sur la conversation, écrire son message dans l'espace prévu à cet effet. Par exemple, si l'utilisateur 10000004 souhaite envoyer un message à l'utilisateur 10000001, il va procéder comme suit :



Les fonctions suivantes de main.js sont appelées :

```
function maybeSend(e) {
    /*
    Fonction appelée dès que le client saisit un nouveau caractère dans la barre dédiée à la composition du message.
    Elle permet d'échanger avec l'autre utilisateur une clef grâce à la fonction sendKey et d'envoyer le message si la
    touche saisie est "Entrée". Cette fonction permet aussi d'adapter la mise en page de la page en fonction de la
    taille du message, grâce à la fonction nouvelleTaille.
    Le paramètre "e" est la touche que le client presse dans le champ dédié à la recherche de nouveaux contacts.
     */

    let message = document.getElementById("messageAEnvoyer").value;
    if(e.keyCode==13 && lastInputLetterCode != 13 && document.getElementById("messageAEnvoyer").dataCible != ""){
        let idCible = document.getElementById("messageAEnvoyer").dataCible;
        let contenuDeMessage = document.getElementById(idCible);
        let enfantContenuDeMessage = contenuDeMessage.children;
        if (enfantContenuDeMessage.length == 1){
              sendKey();
        }
        sendToServeur();
    }
    let taille = document.getElementById("messageAEnvoyer").scrollHeight;
    if(taille < 200){
        document.getElementById("envoyerMessage").style.height = taille + 30 + "px";
        nouvelleTaille();
    }
    if (lastInputLetterCode = 13){
        clean(e);
    }
    lastInputLetterCode = e.keyCode;
}
```

La fonction précédente maybeSend(e) appelle la fonction sendKey() ci-dessous, juste avant l'envoi du premier message d'une conversation, qui génère et envoie la clef de communication/chiffrement à l'autre utilisateur.

Nous avons choisi de conserver une unique clef par conversation plutôt que d'en envoyer une nouvelle à chaque message. En effet le principal désavantage de la première option (la vulnérabilité à une analyse fréquentielle) est amoindri par la nature courte et éphémère des conversations sur la messagerie. D'autre part, le protocole de chiffrement se fait dans le code javascript, qui est envoyé à chaque utilisateur se connectant au serveur. Un potentiel attaquant connaitrait donc la façon dont sont chiffrées les données. Ainsi, dans le cas où la première clef de la conversation serait craquée, il n'aurait qu'à "descendre" successivement de clef en clef pour obtenir la dernière en date (la nouvelle clef étant obtenue à partir de la précédente dans l'algorithme TransCrypt), ce qui fait finalement que l'utilisation d'une clef régulièrement mise à jour, est presqu'aussi sécurisée qu'utiliser une seule clef pour l'ensemble d'une discussion (ceci allège les échanges client-serveur et permet un déploiement à grande échelle plus facile).

```
function sendKey(){
    /*
    Fonction appelée juste avant l'envoi du premier message d'une conversation.
    Elle permet de générer et d'envoyer la clef de communication à l'autre utilisateur.
    Aucun paramètre n'est utilisé.
     */

    let key = Math.floor(Math.random() * 100000000); //10^8
    let cible = document.getElementById("messageAEnvoyer").dataCible;
    $.ajax( {
        type: "POST",
        url: "/send",
        data: JSON.stringify({"userid":userid , "usercode":usercode , "to":cible , "msg":key, "key":"true"})
        contentType: "application/json",
        dataType: 'json',
        async: false,
        success: function (){
            keys[cible] = key;
        },
        error: function (xhr, status, error){
              if (xhr.status == 403){
                  window.location.href = "/expire"
              }
              if (xhr.status == 400){
                  alert("Erreur 400 : L'utilisateur demandé n'existe pas");
              }
        }
    });
}
```

A chaque caractère entré -donc à chaque appel de maybeSend(e)- la fonction nouvelleTaille() ci-dessous est appelée. Elle adapte la mise en page de la page en fonction de la taille du message :

```
function nouvelleTaille () {
    /*
    Fonction appelée dès que la fenêtre change de taille ou que le client saisi un nouveau caractère lors de la
    composition de son message.
    Elle adapte la taille de la conversation et de la liste des contacts,à la taille de la fenêtre et du message en
    cours de rédaction. Elle assure aussi la transition entre les différents modes d'affichage (écran large et écran fin)
    Aucun paramètre n'est utilisé.
    */


    document.getElementById("contenuContact").style.maxHeight = window.innerHeight - 100 + "px";
    let tailleInput = document.getElementById("messageAEnvoyer").scrollHeight;
    let tailleSpan = document.getElementById("envoyerMessage").style.height;

    if (tailleInput>200){
        document.getElementById("contenuMessage").style.maxHeight = window.innerHeight - parseInt(tailleSpan.slice(0, -1)) -177 + "px";
        document.getElementById("contenuMessage").style.minHeight = window.innerHeight - parseInt(tailleSpan.slice(0, -1)) -177 + "px";
    }
    else {
        document.getElementById("contenuMessage").style.maxHeight = window.innerHeight - tailleInput - 208 + "px";
        document.getElementById("contenuMessage").style.minHeight = window.innerHeight - tailleInput - 208 + "px";
    }

    if (window.innerWidth > 630){
        document.getElementById("main").style.display = "initial";
        document.getElementById("contenuContact").style.display = "initial";
    }
    else if (document.getElementById("main").style.display != "none"
        && document.getElementById("contenuContact").style.display != "none"){
        document.getElementById("main").style.display = "initial";
        document.getElementById("contenuContact").style.display = "none";

    }
}
```

6- Envoyer son message

Pour envoyer un message, il faut presser la touche “Entrée” ou appuyer sur le bouton “Envoi” symbolisé par une flèche à côté de la zone de texte.
Les fonctions suivantes de main.js sont alors appelées :

```
function boutonSend (){
    if (lastInputLetterCode != 13){
        sendToServeur();
        lastInputLetterCode = 13;
    }
}
function sendToServeur(){
    /*Fonction appelée lorsque le client presse la touche Entrer ou appuie
            sur le bouton d’envoi du message.
    Elle permet d’envoyer le message et son destinataire au serveur, et
        d’appeler sendInJs si l’opération est effectuée avec succès
    Aucun paramètre n'est utilisé. */
    let clair = document.getElementById("messageAEnvoyer").value;
    let cible = document.getElementById("messageAEnvoyer").dataCible;
    let key = keys[cible];
    let message = crypt(Math.floor(key/10000), crypt(key%10000, clair));

    $.ajax( {
    type: "POST",
    url: "/send",
    data: JSON.stringify({"userid":userid , "usercode":usercode , "to":cible , "msg":message, "key":"false"}),
    contentType: "application/json",
    dataType: 'json',
    success: function (){
        sendInJs(cible, clair);
    },
    error: function (xhr, status, error){
        if (xhr.status == 403){
            window.location.href = "/expire"
        }
        if (xhr.status == 400){
            alert("Erreur 400 : L'utilisateur demandé n'existe pas");
        }
    }
});
}
```

Remarquons que c’est cette fonction sendToServeur() qui va appeler la fonction crypt chiffrant le message (chiffrement décrit en (3)).

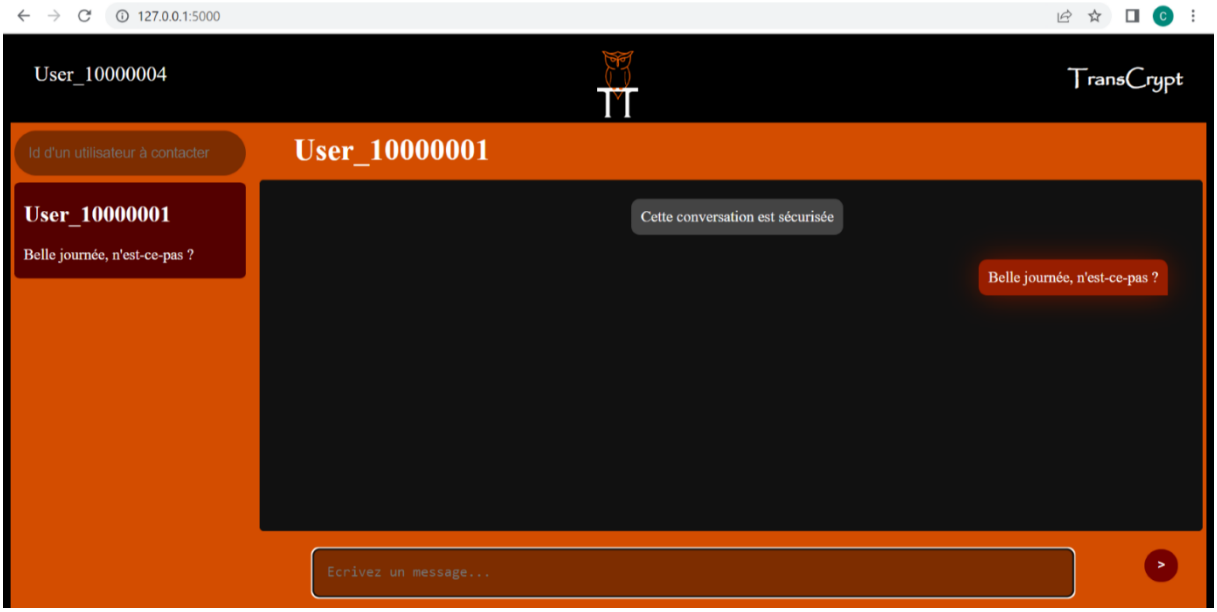
Si l’envoi du message au serveur se déroule bien, la fonction sendInJs(cible) suivante va être appelée, affichant le message (par l’appel d’afficherMessage()).

```
function sendInJs(cible, message) {
    /*
    Fonction appelée si l'envoi du message au serveur se déroule bien.
    Elle permet de vider la barre de saisie du message qui a été envoyé, d'y adapter la mise en page grâce à la fonction
    nouvelleTaille, et d'afficher ce message grâce à la fonction afficherMessage.
    Le paramètre "cible" est l'identifiant du destinataire du message et "message" est la string du message envoyé.
    */

    afficherMessage(1, parseInt(cible), message);

    document.getElementById("messageAEnvoyer").value="";
    nouvelleTaille();
}
function afficherMessage (me, others, valeur){
    /*
    Fonction appelée lorsque le client reçoit ou envoie un message.
    Elle permet d'afficher les messages envoyés et reçus et d'adapter le bouton associé à l'utilisateur avec lequel se
    passe l'échange. Lorsque le client reçoit le premier message d'un autre utilisateur, la fonction appelle la
    fonction newConv.
    Le paramètre "me" prend la valeur 1 lorsque l'auteur du message est le client.
    Le paramètre "others" est l'identifiant de l'utilisateur avec lequel le client interagit.
    Le paramètre "valeur" est le contenu du message.
    */

    let booleen = false;
    let listeDiscussions = document.querySelectorAll(".caseContact");
    listeDiscussions.forEach(function(userId) {
        let idDiscussion = userId.getAttribute("id");
        if (idDiscussion.slice(0, -4) == others) {
            booleen = true;
        }
    });

    if (booleen == false) {
        newConv(others);
    }

    let myMessage = document.createElement("p");
    let myContainer = document.createElement("article");

    if (me==1){
        myMessage.setAttribute("class", "fromUser");
        myContainer.setAttribute("class", "rightContainer");
    }
    else {
        myMessage.setAttribute("class", "toUser");
        myContainer.setAttribute("class", "leftContainer");
    }

    myMessage.appendChild(document.createTextNode(valeur));

    myContainer.appendChild(myMessage);

    document.getElementById(others.toString()).appendChild(myContainer);

    document.getElementById("derniersMots"+others).innerHTML = valeur;

    afficherConversationContact(others+"case");

    document.getElementById("contenuMessage").scrollTo(0, document.getElementById("contenuMessage").scrollHeight);
}
```

La barre de saisie est ensuite réinitialisée grâce à la fonction suivante :

```
function clean(e){
    /*
    Fonction appelée dès qu'une touche du clavier est relâchée, dans la barre de saisie du message,
    ou lorsque la dernière touche pressée est la touche "Entrée".
    Elle permet de réinitialiser la barre d'envoi et évite les problèmes de mise en page dans le cas où l'utilisateur
    maintiendrait la touche "Entrée" enfoncée.
    Le paramètre "e" est la touche que le client presse dans le champ dédié à la recherche de nouveaux contacts.
     */
    if(e.keyCode==13){
        document.getElementById("messageAEnvoyer").value="";
        document.getElementById("envoyerMessage").style.height = "80px";
        document.getElementById("contenuMessage").style.maxHeight = window.innerHeight - 257;
        nouvelleTaille();
    }
}
```

Nous obtenons finalement :



7- Recevoir un message

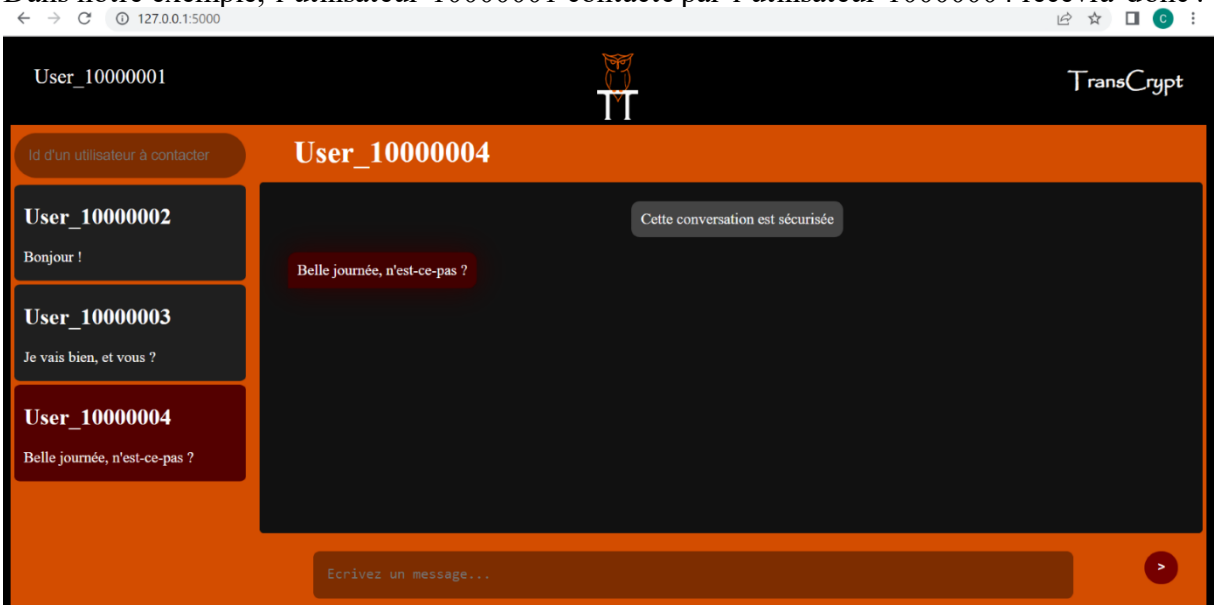
Afin de recevoir un message, la fonction getMessage() suivante est appelée environ toutes les secondes.

```
function getMessage(){
    /*
    Fonction appelée toute les secondes (cet intervalle pourrait être bien plus court mais en vue d'un déploiement à
    grande échelle, nous avons essayé de réduire le nombre de requetés au serveur).
    Elle permet de faire une requête au serveur afin de vérifier si un message a été envoyé au client. Si le message est
    une clef de communication, celle-ci est stockée du coté du client (et supprimée sur le serveur), sinon la fonction
    afficherMessage est appelée.
    Aucun paramètre n'est utilisé.
    */

    $.ajax({
        type: "POST",
        url: "/check",
        data:JSON.stringify({"userid": userid , "usercode": usercode} ),
        contentType: "application/json",
        dataType: 'json',
        success: function(result) {
            for (let i = 0 ; i < result.length ; i++){
                if (result[i]["key"]=="true"){
                    keys[result[i]["id"]] = result[i]["msg"];
                }
                else {
                    let cible = result[i]["id"]
                    let key = keys[cible];
                    let cryp = result[i]["msg"];
                    let message = decrypt(key%10000, decrypt(Math.floor(key/10000), cryp));
                    afficherMessage(0,parseInt(cible),message);

                }
            }
        },
        error: function () {
            window.location.href = "/expire";
        }
    });
}
```

Remarquons que c'est cette fonction qui appelle la fonction decrypt permettant de déchiffrer un message (déchiffrement décrit en (3)).
Lorsqu'un message autre qu'une clef de communication est reçu, la fonction afficherMessage suivante (déjà utilisée pour l'envoi) est appelée :

```
function afficherMessage (me, others, valeur){
    /*
    Fonction appelée lorsque le client reçoit ou envoie un message.
    Elle permet d'afficher les messages envoyés et reçus et d'adapter le bouton associé à l'utilisateur avec lequel se
    passe l'échange. Lorsque le client reçoit le premier message d'un autre utilisateur, la fonction appelle la
    fonction newConv.
    Le paramètre "me" prend la valeur 1 lorsque l'auteur du message est le client.
    Le paramètre "others" est l'identifiant de l'utilisateur avec lequel le client interagit.
    Le paramètre "valeur" est le contenu du message.
    */

    let booleen = false;
    let listeDiscussions = document.querySelectorAll(".caseContact");
    listeDiscussions.forEach(function(userId) {
        let idDiscussion = userId.getAttribute("id");
        if (idDiscussion.slice(0, -4) == others) {
            booleen = true;
        }
    });

    if (booleen == false) {
        newConv(others);
    }

    let myMessage = document.createElement("p");
    let myContainer = document.createElement("article");

    if (me==1){
        myMessage.setAttribute("class", "fromUser");
        myContainer.setAttribute("class", "rightContainer");
    }
    else {
        myMessage.setAttribute("class", "toUser");
        myContainer.setAttribute("class", "leftContainer");
    }

    myMessage.appendChild(document.createTextNode(valeur));

    myContainer.appendChild(myMessage);

    document.getElementById(others.toString()).appendChild(myContainer);

    document.getElementById("derniersMots"+others).innerHTML = valeur;

    afficherConversationContact(others+"case");

    document.getElementById("contenuMessage").scrollTo(0, document.getElementById("contenuMessage").scrollHeight);
}
```

Dans notre exemple, l'utilisateur 10000001 contacté par l'utilisateur 10000004 recevra donc :



8- Retourner à la liste des contacts

Pour retourner à la liste des contacts, il suffit d'appuyer sur le bouton en haut à gauche symbolisé par une flèche (dans les petites fenêtres uniquement). A sa pression, la fonction suivante est appelée :

```
198
199 function changeArea(){
200     /*
201     Fonction appelée lorsque le bouton d'id "boutonRetour" est cliqué.
202     Elle permet de retourner à la liste des contacts, lorsque la messagerie est utilisée sur un écran ou une fenêtre mince.
203     Aucun paramètre n'est utilisé.
204     */
205
206     document.getElementById("main").style.display = "none";
207     document.getElementById("contenuContact").style.display = "initial";t
208 }
209
```

9- Quitter la page

Pour quitter la messagerie, il suffit de quitter la page. La fonction quit() suivante est appelée

```
585 function quit(){
586     /*
587     Fonction appelée quand l'utilisateur quitte la page, pour que le serveur marque l'id comme non utilisée
588     */
589
590     $.ajax( {
591         type: "POST",
592         url: "/quit",
593         data: JSON.stringify({"userid":userid , "usercode":usercode}),
594         contentType: "application/json",
595         dataType: 'json',
596     });
597 }
```

(10) Description de transowl.py

transowl.py a plusieurs fonctions.

Pour s'exécuter, celles-ci nécessitent l'initialisation et les importations suivantes :

```
1  from flask import (
2      Flask, request, url_for, render_template, jsonify
3      )
4  #import os
5  from random import randint
6  from secrets import token_hex
7
8  ### Init de l'app
9  app = Flask(__name__)
10
```

Dans l'ordre attendu par le guide d'utilisation, les fonctions suivantes sont appelées :

- Au chargement de la page, donne un identifiant et le code associé au client :

```
74 @app.route("/getid", methods=["GET", "POST"])
75 def getid():
76     """
77     Appelée quand le client charge la page
78     :return: renvoie l'id spécifique du client ainsi que le code associé
79     """
80     global codes
81
82     if request.method == "POST" :
83         r = 10_000_000
84         # r =  randint(0, 100_000_000)
85         while codes.get(str(r).zfill(8)) != None :
86             r = randint(0, 100_000_000)
87         code = token_hex(8)
88         codes[str(r).zfill(8)] = code
89         newmessages[str(r).zfill(8)] = []
90
91         return jsonify({"userid":str(r).zfill(8), "usercode":code})
```

- Vérification de l'identité de l'utilisateur (renvoie vrai ou faux en fonction de la validité de la correspondance) :

```
def verif(identification, code):
    """
    :param identification: l'id spécifique à l'utilisateur client
    :param code: le code spécifique
    :return: Vrai ou faux dépendant de si ce code correspond bien à celui de l'utilisateur
    """
    return (codes.get(identification) == code)
```

- Si l'identité de l'utilisateur n'est pas valide, renvoi d'un lien vers une page d'expiration, qui permettra la reconnexion :

```
@app.route("/expire", methods=["GET", "POST"])
def expired():
    """
    Appelée quand verif renvoie False
    :return: une page d'expiration, qui permet de se réconnecter à la page principale
    """
    return render_template("expire.html")
```

- Au chargement de l'adresse, renvoi de la page html principale :

```
@app.route("/")
@app.route("/home")
def home():
    """
    Appelée quand le client va sur l'adresse / ou /home
    :return: renvoie la page html principale à l'utilisateur
    """
    return render_template("conv.html")
```

- Lors de l'ajout d'un contact, pour vérifier l'existence de celui-ci :

```
@app.route("/tobeornottobe", methods = ["GET", "POST"])
def ishe():
    """
    Appelée quand un client rajoute un contact
    :return: 1 si l'utilisateur demandé existe 0 sinon
    """

    if request.method == "POST":
        user = request.get_json()["id"].zfill(8)
        resp = {"exists":1}
        if codes.get(user) == None :
            resp["exists"] = 0
        return resp
```

- A chaque envoi (clé ou message) :

```
def addtobuffer(content, src, to, key):
    """

    :param content: le message ou la clef
    :param src: l'id de l'utilisateur qui envoie
    :param to: l'id de l'utilisateur qui reçoit
    :param key: vrai ou faux dépendant de si c'est l'envoi de la clef ou d'un message
    :return: rien.

    """
    if key == "true":
        newmessages[to].append({"id": src, "msg": content, "key": "true"})
    else:
        newmessages[to].append({"id": src, "msg": content, "key": "false"})
    print(newmessages)
```

- A chaque envoi de message :

```
@app.route("/send", methods=["GET", "POST"])
def send():
    """
    Appelée quand l'utilisateur envoie un message
    :return: la valeur renvoyée n'a pas une grande utilité,
    """
    if request.method == "POST" :
        msgData = request.get_json()

        if verif(msgData["userid"], msgData["usercode"]):
            try :
                addtobuffer(msgData["msg"], msgData["userid"].zfill(8), msgData["to"].zfill(8),
msgData["key"])
                return {}, 200

            except KeyError : return "ID d'utilisateur invalide", 400

        else : return "Session expiree", 403
```

- Environ toutes les secondes (0,8s pour donner une impression d'instantané, sans surcharger le serveur) pour chaque client, afin de vérifier si un message a été reçu :

```
@app.route("/check", methods=["GET", "POST"])
def check():
    """
    Appelé périodiquement par le client (toutes les secondes) pour vérifier si de nouveaux
messages ont été envoyés
    :return: une liste des messages reçus depuis le dernière appel
    """
    if request.method == "POST":
        rqData = request.get_json()

        if verif(rqData["userid"], rqData["usercode"]):
            buffed = readbuffer(rqData["userid"])
            return jsonify(buffed)

        else : return "Session expiree", 403
```

- check() se sert de la fonction readbuffer() suivante, pour lire les nouveaux messages et supprimer ceux déjà lus :

```python
def readbuffer(id_check):
    """
    Lit les nouveaux messages et supprime ceux déjà lus.
    :param id_check: l'id de la personne pour lquelle on lit le buffer:
    :return: la liste des messages envoyées à l'utilsateur id_check

    """
    a = newmessages[id_check]
    newmessages[id_check] = []
    return a
```

- Enfin, quand le client quitte sa session, pour libérer son id :

```python
@app.route("/quit", methods=["GET", "POST"])
def quit():
    """
    Appelée quand le client quitte sa session
    Libère son id
    :return: sans importance
    """
    if request.method == "POST":
        rqData = request.get_json()
        if verif(rqData["userid"], rqData["usercode"]):
            codes.pop(rqData["userid"])
            newmessages.pop(rqData["userid"])
            print(codes)
        return {}
```