Goulven RUELLAN Ewen RAOULT Erwan LEDU

Compte rendu de projet

Site comprenant une messagerie chiffrant et déchiffrant les messages envoyés

Sommaire

I. Introduction au sujet

A. Problématique :

Nous cherchions à créer un site fonctionnel muni d'une messagerie interne permettant d'envoyer des messages de façon sécurisée.

B. L'idée du projet :

Notre groupe, composé d'Erwan Le Dû, d'Ewen Raoult et de Goulven Ruellan, a eu cette idée car deux des personnes qui composent celui-ci ont pour objectif futur de travailler dans le secteur de la cyber sécurité. Ce projet nous apporte donc des bases ou du moins une approche plus ou moins partielle de ce domaine.

L'idée d'une messagerie interne nous est venue en discutant avec les professeurs, en effet, passer par une messagerie externe rajoute des risques de détournement des messages. Une messagerie interne permettait donc de palier à ce problème.

C. Le cahier des charges du site

Le site devra répondre à plusieurs attentes :

- Il devra permettre aux utilisateurs de communiquer avec d'autres utilisateurs
- Il devra comporter un algorithme cryptant les messages

D. La répartition du travail :

Nous avons donc réparti les différentes tâches aux membres du groupe :

- Erwan s'occupe du HTML et du CSS du site.
- Ewen s'occupe de la base de données permettant l'inscription et la connexion au site ainsi que du fonctionnement de la messagerie.
- Goulven s'occupe du chiffrage et déchiffrage des messages.

Dans un premier temps, Erwan s'occupait de créer la base de données avant d'échanger avec Ewen et de s'occuper du HTML et CSS du site. Il expliquera donc dans sa partie le fonctionnement d'une base de données.

E. Les logiciels et les langages utilisés :

Pour réaliser ce projet nous avons utilisés différents logiciels tels que Wampserver 64, phpmyadmin, sublime text 3. Ces derniers nous ont permis d'utiliser des langages tels que html, css, php et javascript.

II. Le site web et les bases de données

A. Partie d'Erwan :

Créer une base de données où stocker les informations des utilisateurs de la boîte mail sécurisée.

J'ai utilisé pour cela le logiciel Wampserver, qui permet de créer et gérer les bases de données sur un serveur local, et Sublim Text 3, logiciel de programmation.

- Premièrement, j'ai cherché ce qu'est une base de données sur le site openclassroom.

Une base de données, permet de stocker des données brutes celles-ci peuvent être de natures différentes et plus ou moins reliées entre elles.

Ces données sont toujours classées et peuvent être organisées en tables disposant de différentes colonnes.

Dans ce tableau, les colonnes sont appelées **des champs**, et les lignes sont appelées **des entrées**.

Dans notre exemple, la table est organisée avec quatres colonnes. Une pour chaques informations demandées aux créateurs de compte, le pseudonyme, le mot de passe, l'adresse mail et une dernière colonne pour l'index. L'index sert à identifier l'utilisateur au travers d'un numéro qui ne peut définir qu'un seul utilisateur.

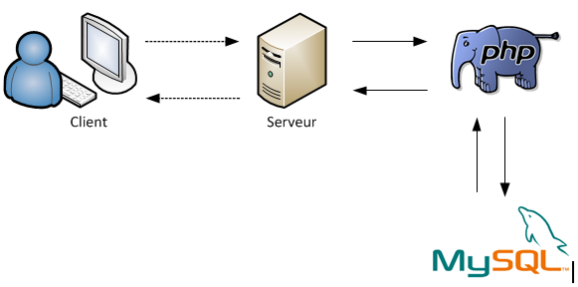
Cet index est composé d'une clé primaire et d'une auto-incrémentation.

La clé primaire permet d'identifier de manière unique une entrée dans la table et l'auto-incrémentation donne une nouvelle valeur automatiquement à la clé primaire.

Chaque colonne à un type défini à la création de la base de données comme un nombre, un texte ou encore une date respectivement appelés `INT`, `VARCHAR`, ou `TEXT` suivant la taille du texte et `DATE`.

Il y a différentes catégories qui sont classées dans ces types. Par exemple, suivant la taille des nombres qui peuvent être entrés dans la base de données, il existe différentes catégories de `INT`, tel que `TINYINT` pour les petits nombres, ou encore `BIGINT` pour les grands et d'autre pour les nombres décimaux…

- Pour ranger ces données on utilise un SGBD (**S**ystèmes de **G**estion de **B**ases de **D**onnées) qui va ranger automatiquement les informations dans les bonnes colonnes de la table. Celui que nous avons utilisé est MySQL, SQL pour Structured Query Language qui est un langage informatique servant à exploiter des bases de données relationnelles.
- On ne peut pas utiliser MySQL directement, il faut donc passer par un intermédiaire, PHP. Il fait fonctionner les langages interprétés de façon locale tel que MySQL.



- Le code PHP est seulement lisible par le serveur, c'est lui qui « génère » la page web.

- Finalement, pour utiliser une base de données l'utilisateur questionne le serveur. Pour générer la page le serveur doit donner une instruction à MySQL mais celui-ci n'utilise que PHP. Donc le serveur fait passer la question de l'utilisateur à PHP qui la relie à MySQL. Puis l'information fait le chemin inverse pour finalement que la page soit envoyée au client.
- Wampserver est très pratique pour créer une base de données. En effet, le logiciel possède une interface qui nous permet de coder graphiquement.

B- Partie d'Ewen

Dans cette partie, nous allons voir la partie PHP et gestion de la base de données du projet.

Nous voulons donc créer une messagerie sur notre site. Or, pour converser d'utilisateur à utilisateur, il faut un système de comptes et de messagerie. Nous allons donc nous baser sur la messagerie de Facebook, qui est une messagerie privée où l'on envoie des messages ou bien on en reçoit d'autre utilisateurs dans notre boîte de réception.

Ainsi, une fois sur notre site, nous devrons nous connecter ou nous inscrire pour accéder à une messagerie où nous pourrons envoyer des messages aux autres utilisateurs, mais également consulter ceux reçus via une boîte de réception.

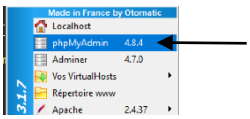
J'ai donc été regarder des tutoriels de création de formulaire d'inscription, de connexion, et de déconnexion et ai ensuite essayé de comprendre la logique derrière le code ainsi que sa signification des fonctions.

a. Création de la base de données

Avant de commencer à programmer, il faut créer une base de données où nous stockeront toutes les informations des utilisateurs qui seront utiles dans le code PHP :

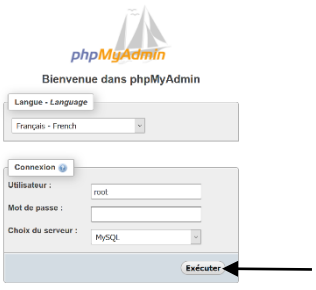
Etape 1 :

Pour accéder à notre base de données, nous allons démarrer Wamp et accéder à la section « phpMyAdmin » et faisant un clic gauche sur l'icône dans la barre des tâches



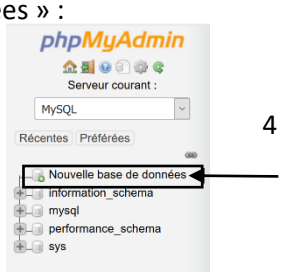
Etape 2 :

Un clic sur la section phpMyAdmin nous redirige vers cette page sur notre navigateur web :



Etape 3 :

Après avoir cliqué sur le bouton « exécuter » sur la page précédente nous accédons à la page d'accueil de phpMyAdmin sur laquelle nous allons nous focaliser sur la partie gauche, et plus précisément sur le bouton « Nouvelle base de données » :

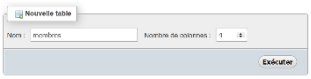


Etape 4 :

Ensuite, nous allons la nommer « espace_membres » :

Etape 5 :

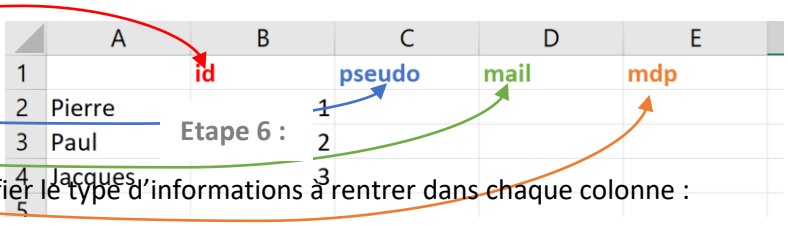
Une fois notre base de données créée, nous allons créer une **table** dans laquelle nous mettrons 4 **colonnes**



Une base de données est constituée de **tables** qui elles-mêmes, sont constituées de **colonnes**. L'organisation d'une table est semblable à celle d'un tableau Excel. Lors de la création d'une table, nous définissons le nombre de colonnes. Ici, chaque colonne correspond à une information d'un utilisateur. Les lignes sont extensibles à l'infini et leur nombre correspond au nombre d'utilisateurs inscrits sur le site.

Dans notre cas, nous voulons 4 colonnes pour y inscrire les informations suivantes sur chaque utilisateur :

- **Identifiant (id)**
- **Pseudonyme**
- **Adresse email**
- **Mot de passe**

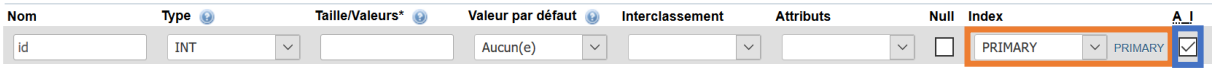


Etape 6 :

Nous allons maintenant spécifier le type d'informations à rentrer dans chaque colonne :

- La première colonne correspond à l'identifiant de l'utilisateur. Chaque utilisateur aura un identifiant unique et qui correspond au moment chronologique de son inscription. Ainsi, le premier utilisateur inscrit aura l'id 1, puis le second, l'id 2 et ainsi de suite.

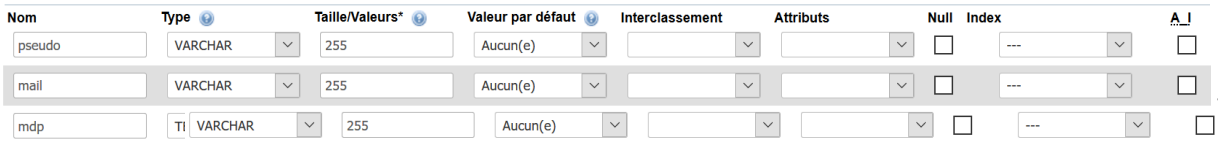
  Nous allons donc mettre le type de caractères à rentrer dans cette colonne en INT, qui correspond aux chiffres et nombres uniquement, ainsi qu'une clé et un « Auto Increment ».



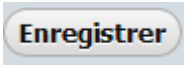
  - La clé primaire est un index qui interdit l'enregistrement des doublons ainsi que les valeurs nulles dans sa colonne
  - L'Auto Increment est utilisée avec une clé primaire afin de spécifier que la colonne à laquelle elle correspond sera incrémentée automatiquement à chaque ajout d'enregistrement dans celle-ci

- Les trois dernières colonnes correspondent respectivement au pseudonyme, à l'email et au mot de passe de l'utilisateur. Nous voulons donc que l'utilisateur rentre du texte, mais afin d'économiser du stockage, nous allons mettre une limite de caractères.

  Le type qui nous permet de rentrer des chaînes de caractères courtes (<255 caractères) est VARCHAR et sa limite maximale de caractères à rentrer dans ce type est de 255 (elle est supérieure à ce nombre dans des versions plus récentes de phpMyAdmin mais par soucis de compatibilité avec les anciennes versions, nous allons laisser 255).



Représentation d'une table sur Excel

Il ne reste plus qu'à cliquer sur 

et la table « membres » est créée !

Désormais, passons à la partie codage…

Comme dit précédemment, nous voulons d'abord faire un système de compte et d'espace membres.

Pour fonctionner, ce système aura besoin de plusieurs programmes :

- Un programme d'inscription
- Un programme de connexion
- Un programme de déconnexion
- Un programme de consultation du profil (qui fera office de page d'accueil de l'utilisateur)

De plus, la messagerie aura également besoin d'un programme d'**envoi** de messages et de **réception**.

Ces programmes seront sous le format **PHP**, mais ils contiendront des **requêtes SQL** ainsi que du **HTML** pour une interface utilisateur (bien que très basique). Ainsi, nous diviserons l'explication de nos codes en 2 parties : Une HTML et une PHP.

Le PHP (Hypertext Preprocessor) est un langage de programmation, souvent associé à une base de données

Le HTML (Hypertext Markup Language) est quant à lui un language de balisage, utilisé pour créer et représenter le contenu d'une page web

Le SQL (Structured Query Language) est un langage permettant de communiquer avec une base de données via des commandes

Nous allons donc commencer par la conception de la page d'inscription.

b. Création de la page d'inscription

La phase de conception

Afin de s'inscrire à notre site, il faudra remplir des cases, et ainsi répondre à plusieurs conditions. Notre formulaire d'inscription sera donc une concaténation de conditions. Pour clarifier ces conditions, nous insèrerons un algorigramme représentant le programme avant de programmer.

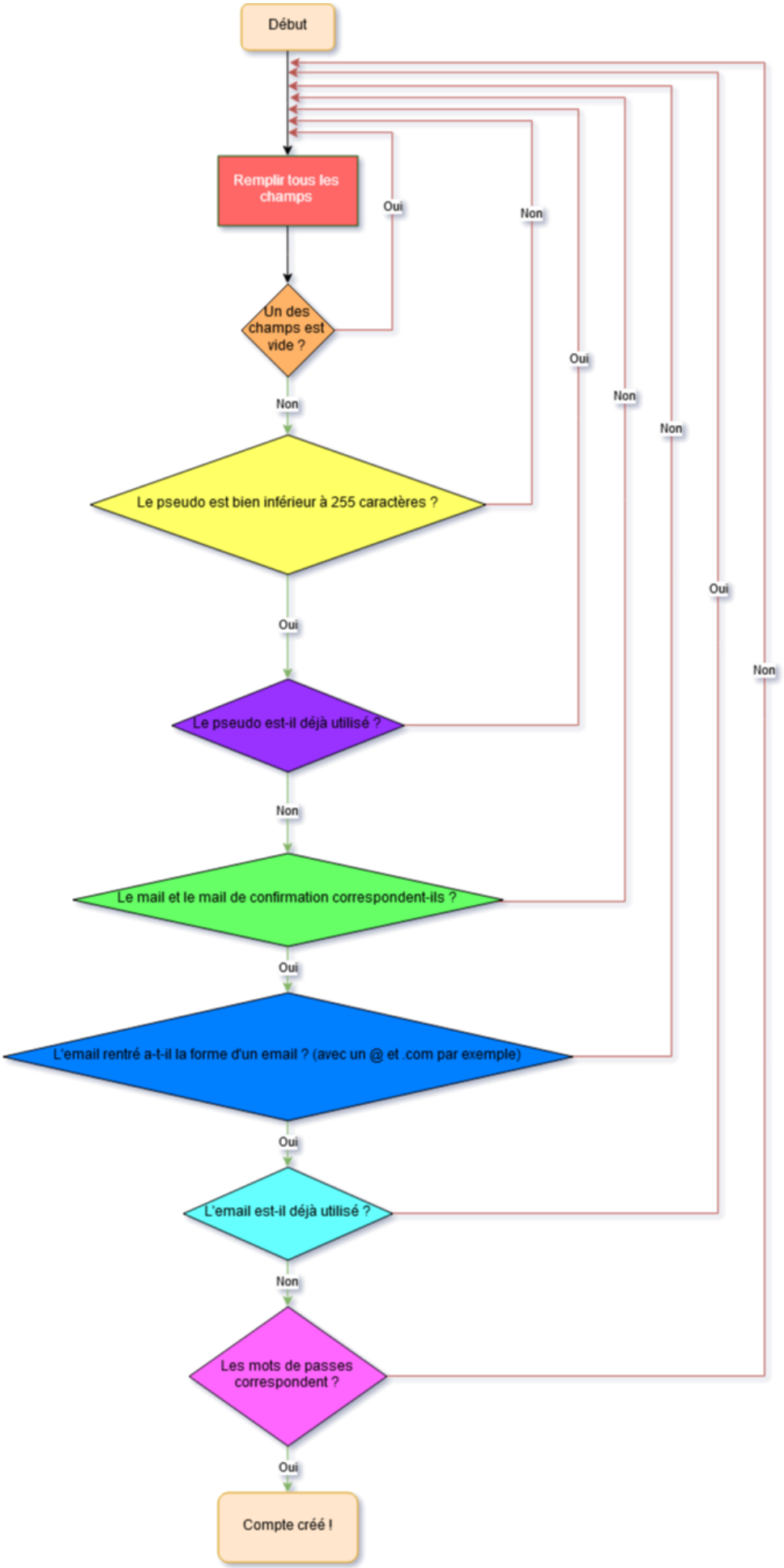
Dans la page d'inscription, il y aura 5 cases à remplir :

- Le pseudo
- L'email
- La confirmation de l'email
- Le mot de passe
- La confirmation du mot de passe

Nous allons donc soumettre les caractères rentrés dans chaque champ à plusieurs tests :

- Leur existence
- Leur forme (valeur <255, la présence d'un @ pour le mail)
- La correspondance entre les champs et leur champ de confirmation
- L'absence de doublons dans la base de données

Voici donc le programme sous forme d'un algorigramme :


Début
Remplir tous les champs
Oui
Non
Un des champs est vide ?
Oui
Non
Non
Le pseudo est bien inférieur à 255 caractères ?
Oui
Oui
Non
Le pseudo est-il déjà utilisé ?
Non
Le mail et le mail de confirmation correspondent-ils ?
Oui
L'email rentré a-t-il la forme d'un email ? (avec un @ et .com par exemple)
Oui
L'email est-il déjà utilisé ?
Non
Les mots de passes correspondent ?
Oui
Compte créé !

La phase de réalisation

Nous allons créer ici la partie HTML du code :

```
<html>
    <head>
        <title>Inscription</title>
        <meta charset="utf-8">
    </head>
    <body>
        <div align="center">
            <h1>Inscription</h1>
            <br /><br /><br />
            <form method="POST" action="">
```

Comprends l'ensemble des caractères du code qui sont compris dans le « répertoire universel des caractères codés »

Aligne toutes les sous-sections au centre de la page

On saute 3 lignes entre le titre de la page et le reste

L'action ne redirige vers aucune page car c'est cette page va également servir de traitement

Les « method » sont utilisées pour envoyer les données d'un formulaire HTML (ici, formulaire d'inscription). Il existe 2 types de méthodes :

- GET, qui est utilisé pour les formulaires n'ayant pas de répercussions sur les bases de données
- POST, pour les formulaires ayant des répercussions (modification d'une base de données, par exemple)

Ici, le formulaire va créer des entrées dans la base de données « espace_membres », nous allons donc mettre la méthode « POST ».

Nous allons ensuite organiser notre formulaire sous forme d'une table, nous allons donc commencer par utiliser `<table>`, puis appliquer la même écriture à chaque ligne du tableau :

Les éléments « label for » et « id » permettent une « connexion » entre les 2 cellules : lorsque l'on clique sur « Mot de passe : », on est directement redirigé dans le champ où il faut l'entrer.

Début de ligne
Début de cellule
Fin de cellule
Fin de ligne

```
<tr>
    <td align="right">
        <label for="mdp">Mot de passe :</label>
    </td>
    <td>
        <input type="password" placeholder="mot de passe" id="mdp" name="mdp" />
    </td>
</tr>
```

L'élément « name » permettra de reconnaître la valeur entrée dans le champ, c'est-à-dire ici, le mot de passe entré. Il sera utile dans la partie PHP.

On rajoute un bouton pour confirmer l'inscription :

```
<tr>
    <td></td>
    <td>
        <br />
        <input type="submit" name="forminscription" value="Inscription">
    </td>
</tr>
```

Et visuellement, notre code ressemble à ça :

Mais notre code n'est pas encore fonctionnel, pour cela, il faut programmer la partie PHP, c'est-à-dire essayer de traduire toutes les conditions de l'algorigramme sous la forme d'un code PHP :

Dans un premier temps, nous allons définir les variables utilisées dans la suite du code :

```php
<?php

$bdd = new PDO('mysql:host=127.0.0.1;dbname=espace_membres', 'root', '');

if(isset($_POST['forminscription']))
{
    $pseudo = htmlspecialchars($_POST['pseudo']);
    $mail = htmlspecialchars($_POST['mail']);
    $mail2 = htmlspecialchars($_POST['mail2']);
    $mdp = sha1($_POST['mdp']);
    $mdp2 = sha1($_POST['mdp2']);
```

- Dans un premier temps, on définit une variable pour se connecter à MySQL :

```php
$bdd = new PDO('mysql:host=127.0.0.1;dbname=espace_
```

Cette variable (du nom de « bdd ») sert à se connecter à MySQL, chez l'hôte 127.0.0.1 (l'hôte local ou « localhost »), à la base de données « espace_membres », avec le nom d'utilisateur 'root' et aucun mot de passe (afin d'avoir toutes les autorisations de modifications et création).

La variable qui nous permet de nous connecter à notre serveur SQL est "PDO", c'est une extension PHP qui permet d'accéder à n'importe quel type de base de données.

Une fois cette variable définie, nous allons commencer la série de conditions :

- Tout d'abord, nous allons vérifier si, après que l'utilisateur ait cliqué sur le bouton d'envoi, le formulaire ai bien été envoyé.

```php
if(isset($_POST['forminscription']))
```

isset détermine si une variable est définie et différente de « NULL » (c'est-à-dire qu'elle n'existe pas ou est nulle)

La variable $_POST fait référence au POST du formulaire HTML, et ici, plus précisément, à l'élément ayant pour nom « forminscription »

La signification de cette ligne est donc :

Si l'élément « forminscription » n'est pas nul, faire…

Autrement dit, si le bouton a été cliqué, faire…

- Ensuite, on définit les variables :

```php
$pseudo = htmlspecialchars($_POST['pseudo']);
$mail = htmlspecialchars($_POST['mail']);
$mail2 = htmlspecialchars($_POST['mail2']);
$mdp = sha1($_POST['mdp']);
$mdp2 = sha1($_POST['mdp2']);
```

La fonction htmlspecialchars converti les caractères spéciaux en entités HTML pour qu'ils soient lisibles. En effet, Certains caractères ont des significations spéciales en HTML, et doivent être remplacés par des entités HTML pour conserver leurs significations.

La fonction sha1 est une fonction de hachage, elle va ici sécuriser les mots de passes pour qu'ils ne soient pas lus par des individus malveillants. Il en existe plusieurs. Ici, nous avons arbitrairement pris le « sha1 », mais d'autres fonctions de hachages sont plus sûres aujourd'hui et nous aurions pu utiliser d'autres fonctions.

Les variables correspondront aux valeurs entrées dans les cases du formulaire par l'utilisateur, mais converties pour accepter les caractères spéciaux et êtres sécurisés pour ensuite être stockées dans la base de données ou les comparer à des valeurs déjà existantes dans cette dernière.

- Nous devons ensuite vérifier si tous les champs sont remplis, nous allons donc vérifier si chaque champ n'est pas vide :

```php
if(!empty($_POST['pseudo']) AND
    !empty($_POST['mail']) AND !
    empty($_POST['mail2']) AND !
    empty($_POST['mdp']) AND !
    empty($_POST['mdp2']))
```

Le !empty signifie " not empty ", n'est pas vide. Donc cette condition vérifie si aucun des champs n'est vide.

Si la condition est remplie, nous pouvons passer à la condition suivante, sinon, une erreur s'affiche :

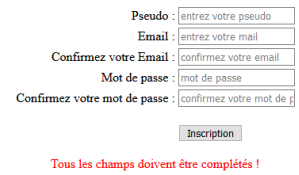
```php
else
{
    $erreur = "Tous les champs doivent être complétés !";
}
```

On fait ici appel à une variable « erreur » qu'on a défini plus bas :

```php
if(isset($erreur))
{
    echo '<font color="red">'.$erreur."</font>";
}
?>
```

echo affiche une valeur à l'écran de l'utilisateur, en l'occurrence la phrase stockée dans la variable erreur, avec une couleur de police d'écriture rouge

Ainsi, si tous les champs ne sont pas remplis, cette erreur va s'afficher à l'écran de l'utilisateur :



- En revanche, si la condition est remplie, il faut passer à la condition suivante, c'est-à-dire vérifier si le pseudo fait moins de 255 caractères :

```
15          $pseudolength = strlen($pseudo);
16          if($pseudolength <= 255)
17          {
```

Pour cela, on définit une nouvelle variable, $pseudolength, qui correspond au nombre de caractères du pseudo,

On vérifie ensuite si cette variable est inférieure ou égale à 255

Si la variable est supérieure à 255, l'erreur suivante s'affiche :

```
else
{
    $erreur = "Votre pseudo est trop long";
}
```

. En revanche, si elle est inférieure ou égale à 255, on peut passer à la condition suivante

- Nous allons vérifier si le pseudo est déjà utilisé, c'est-à-dire s'il est déjà présent dans la base de données. Pour cela, nous allons effectuer une requête préparée :

```
$reqpseudo =$bdd->prepare("SELECT * FROM membres WHERE pseudo = ?");
$reqpseudo->execute(array($pseudo));
$pseudoexist = $reqpseudo->rowCount();
if($pseudoexist == 0)
```

On se connecte à la base de données via la variable

On prépare une requête SQL

On exécute la requête précédemment stockée dans la variable $reqpseudo

execute n'accepte en argument qu'un tableau de données, on met donc nos données dans un tableau

rowCount retourne le nombre de lignes affectées par la requête et les stocke dans $pseudoexist

La variable $reqpseudo sera donc pour préparer une requête où l'on sélectionne tous les éléments de la table membres où les pseudos ont une valeur inconnue (que l'on définira dans la ligne suivante)

On exécute la requête avec à la place du « ? », la variable $pseudo

Ce qui reviens à sélectionner tous les pseudos dans la base de données qui sont identiques à la variable $pseudo et à les stocker dans $reqpseudo

On compte ensuite combien de lignes sont stockées dans $reqpseudo grâce à la fonction rowCount et on stocke ce nombre dans la variable $pseudoexist.

Si le nombre de pseudo stockés dans la variable $pseudoexist est de 0, c'est-à-dire si aucun pseudo dans la base de données ne correspond à celui que l'utilisateur a rentré, alors on passe à la condition suivante, sinon, cette erreur s'affiche :

```
else
{
    $erreur = "Votre pseudo est déjà utilisé";
}
```

- La condition suivante consiste juste à vérifie que le mail entré correspond au mail de confirmation entré. On a donc : `if($mail == $mail2)
{`, si cette condition est vraie, on passe à la suivante, sinon, cette erreur s'affiche :

```
else
{
    $erreur = "les emails ne correspondent pas";
}
```

- La condition suivante consiste à valider la forme de l'écriture de l'adresse email, c'est-à-dire de la forme exemple@exemple.com(ou.fr) : `if(filter_var($mail, FILTER_VALIDATE_EMAIL))`

On utilise la fonction filter_var pour filter une variable avec un filtre spécifique. Ici, on s'en sert pour vérifier si la variable $mail a la bonne syntaxe. Si oui, on continue les conditions, si non, cette erreur s'affiche :

```
else
{
    $erreur = "Votre adresse email n'est pas valide";
}
```

- La prochaine condition consiste à vérifier si l'email est déjà présent dans la base de données. La syntaxe est exactement la même que pour vérifier les doublons de pseudo, il faut juste remplacer toutes les entrées « pseudo » par « mail » :

```
$reqmail = $bdd->prepare("SELECT * FROM membres WHERE mail = ?");
$reqmail->execute(array($mail));
$mailexist = $reqmail->rowCount();
if($mailexist ==0)
```

Si cette condition n'est pas remplie, cette erreur s'affiche `$erreur = "Adresse mail déjà utilisée";`, sinon, on passe à la condition suivante.

- Cette condition est également la même que pour vérifier la correspondance entre la valeur entrée dans la case mail et la case confirmation de mail, mais cette fois-ci avec les cases mot de passe et confirmation de mot de passe : `if($mdp == $mdp2)`. Si cette condition n'est pas remplie, cette erreur s'affiche : `$erreur ="les mots de passe ne correspondent pas";`, dans le cas contraire, toutes les conditions sont enfin réunies pour créer le compte !

On doit donc ajouter les valeurs entrées par l'utilisateur dans la base de données, pour cela, nous allons faire une requête préparée :

```
$insertmbr = $bdd->prepare("INSERT INTO membres(pseudo, mail, mdp) VALUES(?, ?, ?)");
$insertmbr->execute(array($pseudo, $mail, $mdp));
```

On se connecte à la base de données et on prépare la requête qui consiste à insérer dans la table « membres », sous forme de (pseudo, mail, mot de passe), des valeurs inconnues qui seront définies au moment de l'exécution de la requête.

On exécute ensuite la requête en remplaçant les valeurs inconnues par $pseudo, $mail et $mdp respectivement.

Enfin, on annonce la bonne nouvelle à l'utilisateur en lui mettant à disposition un lien le redirigeant vers la page de connexion (on affiche le message sur l'écran via la variable erreur mais cela n'a aucune influence si ce n'est que le message est affiché en rouge).

```
$erreur = "Votre compte a bien été créé ! <a href=\"connexion.php\">Me connecter</a>";
```

C. Création de la page d'inscription

- Le HTML :

Cette fois-ci, le HTML est plus court, pas besoin de faire de tableau puisque nous ne voulons afficher que 3 éléments sur la page :

- L'email
- Le mot de passe
- Le bouton de connexion

Ainsi, au lieu d'utiliser un tableau, nous allons créer chaque élément sous forme d'input (élément utilisé pour interagir avec l'utilisateur) :

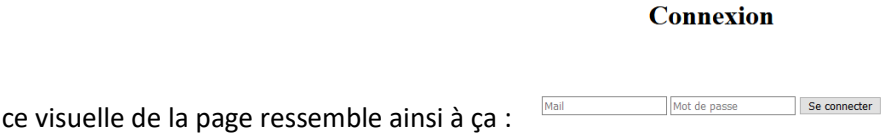
```
<input type="email" name="mailconnect" placeholder="Mail"/>
<input type="password" name="mdpconnect" placeholder="Mot de passe"/>
<input type="submit" name="formconnexion" value="Se connecter" />
```

Le type d'input pour l'email sera « email » pour vérifier si sa syntaxe est correcte,

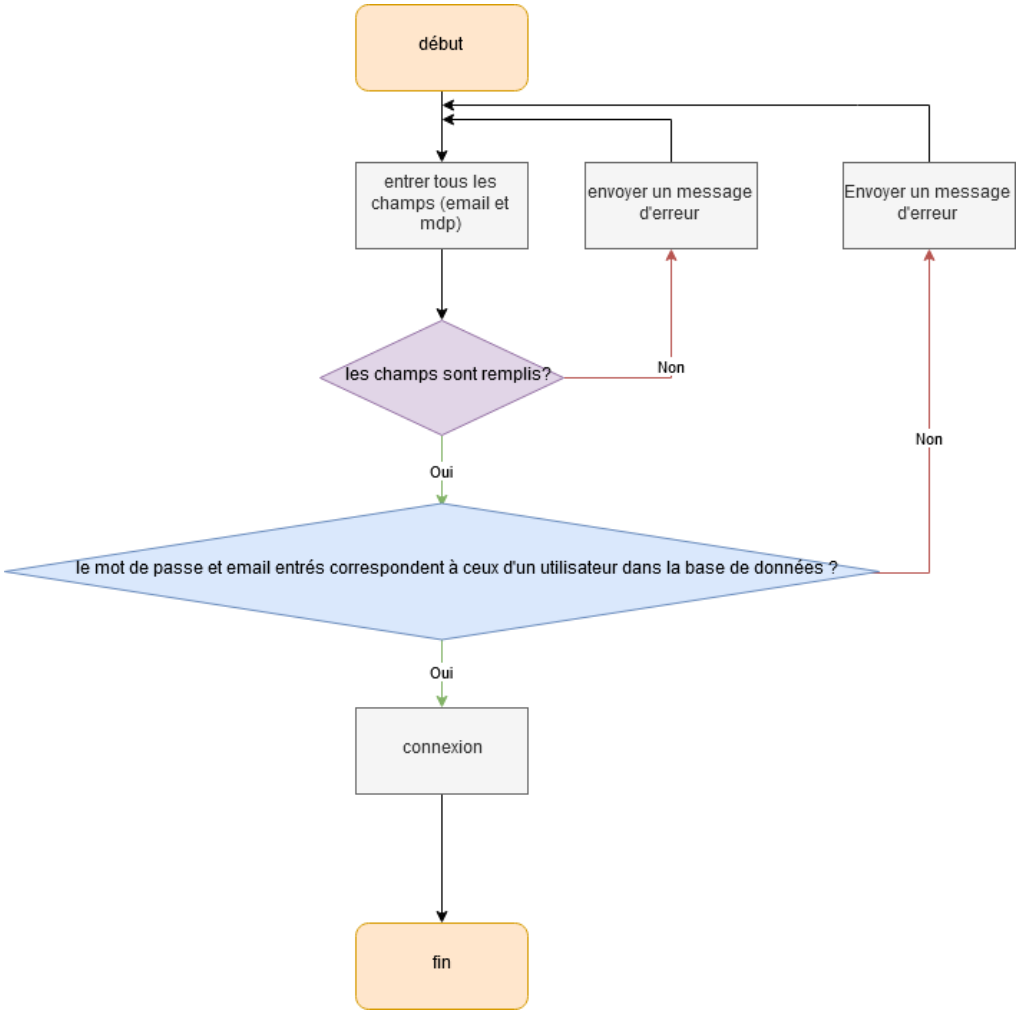
Pour le mot de passe, le type d'input sera « password » afin de cacher les caractères,

Pour le bouton, le type d'input sera « submit » afin que l'input prenne la forme d'un bouton.

L'interface visuelle de la page ressemble ainsi à ça :



- Le PHP : Voici un algorigramme représentant les conditions requises pour que l'utilisateur puisse se connecter :



Tout d'abord, comme pour la page d'inscription, nous définissons une variable qui nous permettra de nous connecter à la base de données. Mais ici, on rajoute la fonction session_start() au début du code. Cette fonction va permettre de démarrer une nouvelle session ou reprendre une session existante, ainsi que récupérer les variables de sessions que nous utiliserons dans le code.

> Une variable de session PHP est une variable stockée sur le serveur. C'est une variable temporaire qui a une durée limitée et est détruite à la déconnexion (fermeture du navigateur). Les variables de session sont **partagées** par toutes les pages PHP d'une session (accès depuis un même navigateur). Elles permettent donc le passage d'information entre pages.

Une session permettra de faire circuler différentes variables (comme les mots de passes) à travers les différentes pages du site.

```
2   session_start();
3
4   $bdd = new PDO('mysql:host=127.0.0.1;dbname=espace_membres', 'root', '');
```

Également comme pour la page d'inscription, après avoir vérifié que le bouton ait été cliqué, nous allons définir les variables utiles dans la suite du code :

```
6   if(isset($_POST['formconnexion']))
7   {
8       $mailconnect = htmlspecialchars($_POST['mailconnect']);
9       $mdpconnect = sha1($_POST['mdpconnect']);
```

Ces variables seront ici $mailconnect pour l'adresse mail, et $mdpconnect pour le mot de passe

- Encore à l'instar de la page d'inscription, nous vérifions si tous les champs sont remplis via la fonction !empty : `if(!empty($mailconnect) AND !empty($mdpconnect))`. Si la condition est validée, on passe à la deuxième, sinon, ce message d'erreur s'affiche :
```
else
{
    $erreur = "Tous les champs doivent être complétés !";
}
```
.
- La prochaine condition consiste à vérifier si les champs rentrés correspondent à des champs correspondant à un seul utilisateur dans la base de données. Nous allons donc créer une requête préparée :

```
$requser = $bdd->prepare("SELECT * FROM membres WHERE mail = ? AND mdp = ?");
$requser->execute(array($mailconnect, $mdpconnect));
$userexist = $requser->rowCount();
if($userexist == 1)
```

On prépare puis exécute donc la requête en sélectionnant toutes les lignes où le mail correspond à celui entré par l'utilisateur ET où le mot de passe correspond à celui entré par l'utilisateur.

On stocke le nombre de ces lignes dans la variable $userexist

Si le nombre de ligne est égal à un, c'est-à-dire si ce mot de passe et cet email correspondent à un seul utilisateur dans la base de données, alors l'utilisateur se connecte, sinon, ce message d'erreur s'affiche : `$erreur = "Mauvais mail ou mot de passe !";`

Ensuite, il faut stocker les informations de session dans la variable $_SESSION.

Pour cela, nous allons récupérer les informations stockées dans $requser et les mettre dans $userinfo via la fonction fetch() (qui permet de parcourir le résultat d'une requête ligne par ligne).

Puis on attribue les id, pseudo et mail contenus dans $userinfo à la variable $_SESSION

```php
$userinfo = $requser->fetch();
$_SESSION['id'] = $userinfo['id'];
$_SESSION['pseudo'] = $userinfo['pseudo'];
$_SESSION['mail'] = $userinfo['mail'];
```

Pour finir, on redirige l'utilisateur vers la page de profil qui lui correspond, c'est-à-dire celle qui correspond à son id, que nous venons de récupérer dans la variable de session. On redirige donc l'utilisateur vers la page profil.php avec l'id de l'utilisateur qui sera noté dans l'URL de la page.

```php
header("Location: profil.php?id=".$_SESSION['id']);
```

C. Création de la page de profil :

La page de profil n'a pas besoin de formulaire, elle fera juste office de page d'accueil avec des redirection vers d'autre pages comme la page de déconnexion, ou la page d'envoi de messages.

- Le HTML :

Tout d'abord, on créé une page personnalisée pour chaque utilisateur en faisant appel au PHP, plus précisément aux variables de session, dans le HTML. En effet, on peut afficher des informations spécifiques à l'utilisateur via la fonction PHP echo suivie de la variable contenant l'information à afficher, ainsi, on a :

```php
<body>
    <div align="center">
        <h1>Profil de <?php echo $userinfo['pseudo']; ?></h1>
        <br /><br />
        Pseudo : <?php echo $userinfo['pseudo']; ?>
        <br />
        Mail : <?php echo $userinfo['mail']; ?>
        <br />
```

- Le PHP :

On commence notre PHP avec la fonction session_start() afin d'utiliser les variables de session, puis on défini la variable qui nous permettra de nous connecter à la base de données, comme dans les pages précédentes :

```php
<?php
session_start();

$bdd = new PDO('mysql:host=127.0.0.1;dbname=espace_membres', 'root', '');
```

Ensuite, on va sécuriser notre URL en vérifiant qu'il y a bien un id dans l'URL et qu'il est supérieur à 0 :

```php
if(isset($_GET['id']) AND $_GET['id'] > 0)
```

On peut également vérifier que l'id entré est bien un nombre, pour cela, on créé une requête préparée :

```php
$getid = intval($_GET['id']);
$requser = $bdd->prepare('SELECT * FROM membres WHERE id = ?');
$requser->execute(array($getid));
```

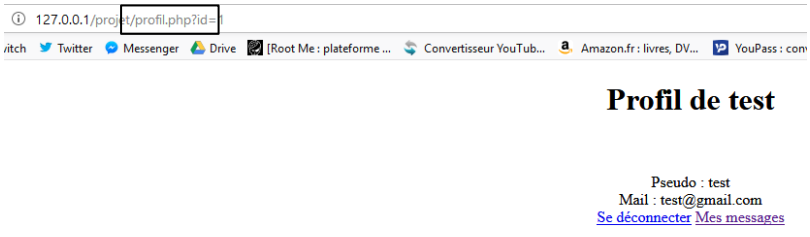
On va convertir l'id en nombre grâce à intval, si c'est déjà un nombre, cela ne changera rien, mais s'il est composé de lettres, il va être converti en nombre afin d'éviter les erreurs.

On va ensuite aller chercher les informations de l'utilisateur dans $requser puis les mettre dans $userinfo : `$userinfo = $requser->fetch();`.

Nous allons revenir dans la partie HTML pour voir comment afficher des éléments propres à l'utilisateur. Nous utiliserons cette fonction pour afficher les liens vers la page de déconnexion et de réception.

```php
if($userinfo['id'] == $_SESSION['id'])
{
?>
<a href="deconnexion.php">Se déconnecter</a>
<a href="reception.php">Mes messages</a>
<?php
}
?>
```

On vérifie si l'id marqué dans l'URL correspond à l'id de session, si oui, les liens pour se déconnecter et consulter ses messages s'affichent. La page ressemble donc finalement à ça :



d. Page de déconnexion

Cette page est très courte et ne comporte que du php.

Comme toutes les pages où l'on utilise les variables de session, on commence la page par la fonction session_start(), puis on vide les variables de session, on finit la session via session_destroy() et enfin on redirige l'utilisateur vers la page de connexion :

```php
<?php
session_start();
$_SESSION = array();
session_destroy();
header("Location: connexion.php");
?>
```

On ne met rien à l'intérieur du tableau pour montrer que la variable de session est vide.

e. Création de la page d'envoi

Nous voulons donc une page d'envoi avec 3 éléments :

- Un élément qui permet de choisir l'utilisateur auquel envoyer le message
- Un champ pour rentrer le message
- Un bouton d'envoi

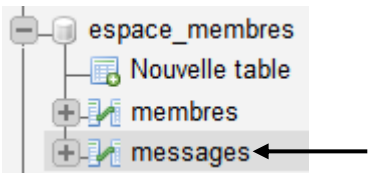
Nous allons créer ces éléments sous HTML

- Le HTML :
  - Nous allons créer l'élément qui permet de choisir un utilisateur sous forme d'une liste qui regroupe tous les utilisateurs du site, nous allons donc utiliser la balise `<select>` et nous la nommerons `name="destinataire">`
  - Pour le champ de texte, nous allons nous créer l'élément `<textarea placeholder="Votre message" name="message"></textarea>`
  - Enfin, pour le bouton d'envoi : `<input type="submit" value="Envoyer" name="envoi_message" />`

Ce qui, à l'écran, donne cela :

Maintenant, nous allons faire fonctionner cette page, mais avant de commencer le php, il faut créer une table dans notre base de données afin d'y stocker les messages, l'identifiant de l'envoyer et celui du destinataire.

- La création de la table :
  Nous créons donc une nouvelle table « messages » dans la base de données espace_membres :



Comme pour la table « membres », nous allons attribuer un id à chaque message, qui sera l'index primaire de la table avec un auto increment :

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
☐	1	**id**	int(11)			Non	*Aucun(e)*		AUTO_INCREMENT	Modifier Supprimer Plus
☐	2	**id_expediteur**	int(11)			Non	*Aucun(e)*			Modifier Supprimer Plus
☐	3	**id_destinataire**	int(11)			Non	*Aucun(e)*			Modifier Supprimer Plus
☐	4	**message**	text	latin1_swedish_ci		Non	*Aucun(e)*			Modifier Supprimer Plus

On récupère également l'id de l'expéditeur et l'id du destinataire ainsi quel le message sous type de texte.

Nous allons ensuite créer un programme php qui stocke les messages dans cette table :

- Le PHP :

Avant tout, nous allons créer une variable qui liste les pseudos des utilisateurs du site, que nous incorporeront dans la liste des destinataires :

Nous allons donc créer la requête SQL :

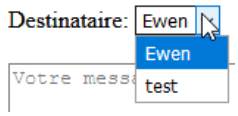
```php
$destinataires = $bdd->query('SELECT pseudo FROM membres ORDER BY pseudo');
```

Cette requête va stocker dans la variable $destinataires, tous les pseudos des utilisateurs présents dans la table « membres », en les classant par ordre alphabétique.

Ensuite, dans notre HTML, nous allons faire une boucle php pour afficher tous les destinataires :

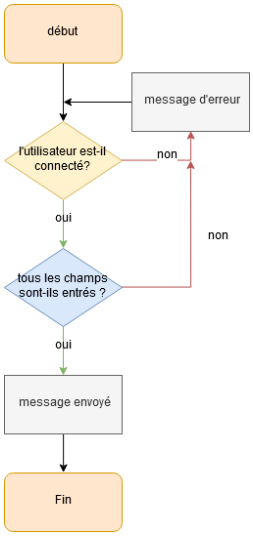
```php
<?php while($d = $destinataires->fetch()) { ?>
<option><?php echo $d['pseudo'] ?></option>
<?php } ?>
```

Ce qui donne cela :



Maintenant, nous allons nous intéresser aux conditions requises pour l'envoi d'un message :

Nous allons représenter les conditions par un algorigramme :



Nous allons commencer par les éléments communs à tous nos programmes comportant des variables de session ainsi que des formulaires :

Tout d'abord, on commence notre programme avec le session_start() et la variable de connexion à la base de données.

```
session_start();
$bdd = new PDO('mysql:host=127.0.0.1;dbname=espace_membres', 'root', '');
```

Nous allons également créer une variable d'erreur :

```
<?php if(isset($error)) { echo '<span style="color:red">'.$error.'</span>'; } ?>
```

Ensuite, nous allons vérifier que l'utilisateur est bien connecté :

```
if(isset($_SESSION['id']) AND !empty($_SESSION['id'])) {
```

Enfin, nous allons vérifier que le bouton d'envoi du message a bien été cliqué, donc que le message a bien été envoyé :

```
if(isset($_POST['envoi_message'])) {
```

Nous allons maintenant vérifier que d'une part, toutes les variables existent, et que les champs auxquels elles correspondent ne sont pas vide :

```
if(isset($_POST['destinataire'],$_POST['message']) AND !empty($_POST['destinataire']) AND !empty($_POST['message'])) {
```

Si ces conditions ne sont pas réunies, nous avons cette erreur :

```
$error = "Veuillez compléter tous les champs";
```

Sinon, toutes les conditions sont réunies, mais avant d'envoyer le message dans la base de données, nous allons sécuriser les variables afin qu'il n'y ait pas d'erreur dans la base de données :

```
$destinataire = htmlspecialchars($_POST['destinataire']);
$message = htmlspecialchars($_POST['message']);
```

Ensuite, nous voulons récupérer l'id du destinataire à partir de son pseudo (d'où l'intérêt de ne pas accepter les doublons de pseudo dans la base de données).

Pour cela, nous allons d'abord récupérer l'id du membre du site donc le pseudo correspond à valeur stockée dans la variable $destinataire :

```php
$id_destinataire = $bdd->prepare('SELECT id FROM membres WHERE pseudo = ?');
$id_destinataire->execute(array($destinataire));
```

On va ensuite mettre les valeurs récupérées sous la forme d'un table via fetch() , puis nous allons récupérer l'id dans ce tableau :

```php
$id_destinataire = $id_destinataire->fetch();
$id_destinataire = $id_destinataire['id'];
```

Et enfin, nous allons insérer les données récupérées dans la table « messages » en informant l'utilisateur que son message a été envoyé :

```php
$ins = $bdd->prepare('INSERT INTO messages(id_expediteur,id_destinataire,message) VALUES (?,?,?)');
$ins->execute(array($_SESSION['id'],$id_destinataire,$message));
$error = "Votre message a bien été envoyé !";
```

f. Création de la boîte de réception :

La boîte de réception va se présenter sous la forme d'une liste de tous les messages.

- Le PHP :

Tout d'abord, nous allons démarrer la session, créer la variable de connexion à la base de données et vérifier si l'id de session est bien le bon :

```php
session_start();
$bdd = new PDO('mysql:host=127.0.0.1;dbname=espace_membres', 'root', '');

if(isset($_SESSION['id']) AND !empty($_SESSION['id'])) {
```

Nous allons ensuite récupérer tous les messages dont l'id_destinataire correspond à notre id, c'est-à-dire les messages qui nous sont destinés :

```php
$msg= $bdd->prepare('SELECT * FROM messages WHERE id_destinataire = ?');
$msg->execute(array($_SESSION['id']));
```

Ensuite, dans la partie HTML, nous allons faire une boucle permettant d'afficher les messages ainsi que leur expéditeur :

- Nous allons d'abord afficher le pseudo de l'expéditeur du message en récupérant son pseudo :

```php
while($m = $msg->fetch()) {
    $p_exp = $bdd->prepare('SELECT pseudo FROM membres WHERE id = ?');
    $p_exp->execute(array($m['id_expediteur']));
    $p_exp = $p_exp->fetch();
    $p_exp = $p_exp['pseudo'];
```

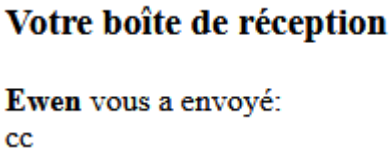
- Et enfin, nous allons afficher le pseudo de l'expéditeur ainsi que le message envoyé :

```
<b><?=  $p_exp ?></b> vous a envoyé: <br />
<?= nl2br($m['message']) ?><br />
----------------------------------------<br />
```

Insère un retour à la ligne HTML à chaque nouvelle ligne

< ?= est la même chose que < ?php echo

Ainsi, si nous avons reçu un message, notre écran affiche cela :



Conclusion de la partie : Nous avons donc bien réussi à créer un système de création de compte, de déconnexion et connexion, ainsi qu'une messagerie, il reste maintenant à la sécuriser.

III. Le chiffrement des données et le déchiffrement

A. Le chiffrement

Notre première idée concernant le chiffrement des données a été de concevoir un programme qui chiffrait celles-ci grâce à une clé de chiffrement différente pour chaque message. J'ai donc commencé à réaliser une clé de chiffrement de manière aléatoire, et d'une taille suffisante pour permettre assez de sûreté. Cependant, après des recherches pour mettre en œuvre cette idée dans son ensemble, nous y avons renoncé car cela ne nous convenait pas. Nous nous sommes alors dirigés, après concertation du groupe, vers un programme permettant de changer aléatoirement des caractères du message afin de le brouiller.

J'ai donc tout d'abord essayé de compter dans un mot le nombre de lettre qu'il y a dans celui-ci en ne m'intéressant qu'à un seul caractère défini à l'avance. Une fois ceci fait, j'ai pu essayer de les échanger avec une autre lettre elle aussi au début de valeur finie, inchangeable.

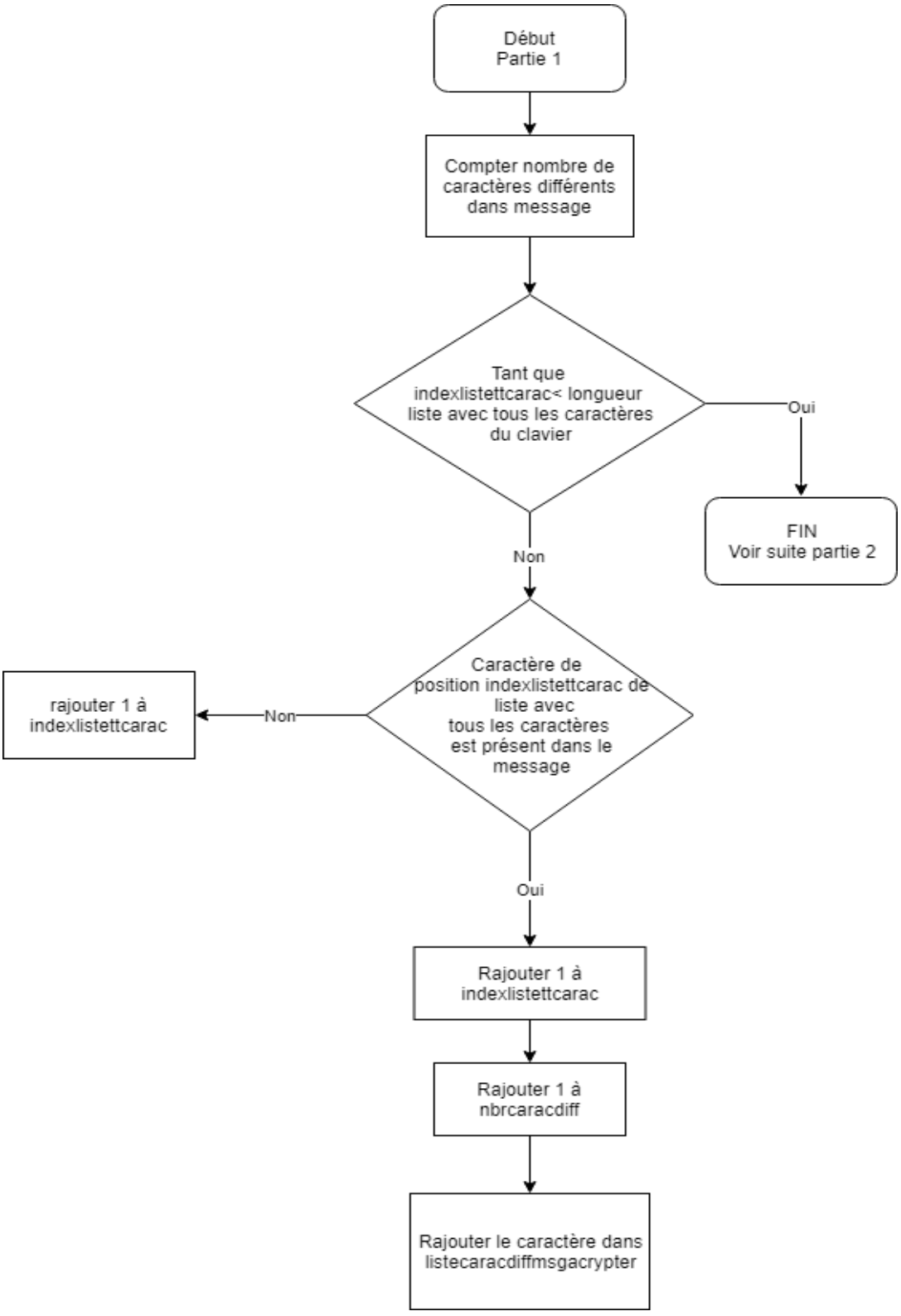
Une fois cette première approche du problème effectuée, il a donc fallu concevoir petit à petit un programme permettant de modifier un message entier tout en étant sûr de sa fiabilité, d'où la nécessité d'y ajouter un caractère aléatoire. Il a aussi fallu créer des variables aléatoires pour choisir aléatoirement les caractères à remplacer, le nombre de fois que nous les remplaçons ainsi que la position de chacun d'entre eux. Prenons l'exemple d'un mot comprenant 4 fois la lettre « o », grâce à ce programme, n'importe lequel de ces « o » peut être remplacé aussi bien le 1er que le 4ème. Pour finir nous avons dû rendre aléatoire le choix du caractère qui remplace celui choisi. Ainsi, si l'on reprend le même exemple que précédemment, un des « o » pourra être remplacé par un « A » et un autre par un « # ».

Le fonctionnement global du programme a été explicité par 3 algorigrammes pour en faciliter la compréhension. Ceux-ci font appels à différentes variables portant pour certaines d'entre elles le même nom que dans le programme d'origine. De ce fait, pour permettre une lecture plus claire des algorigrammes, un tableau regroupant toutes les variables utilisées dans ceux-ci ainsi que leur intérêt est inclus en annexe. Les variables sont classés par ordre d'apparition.

En annexe est aussi disponible un tableau regroupant les différentes fonctions utilisées dans le programme en lui-même. Des captures d'écran de celui-ci sont aussi disponibles via un lien renvoyant vers Google drive.

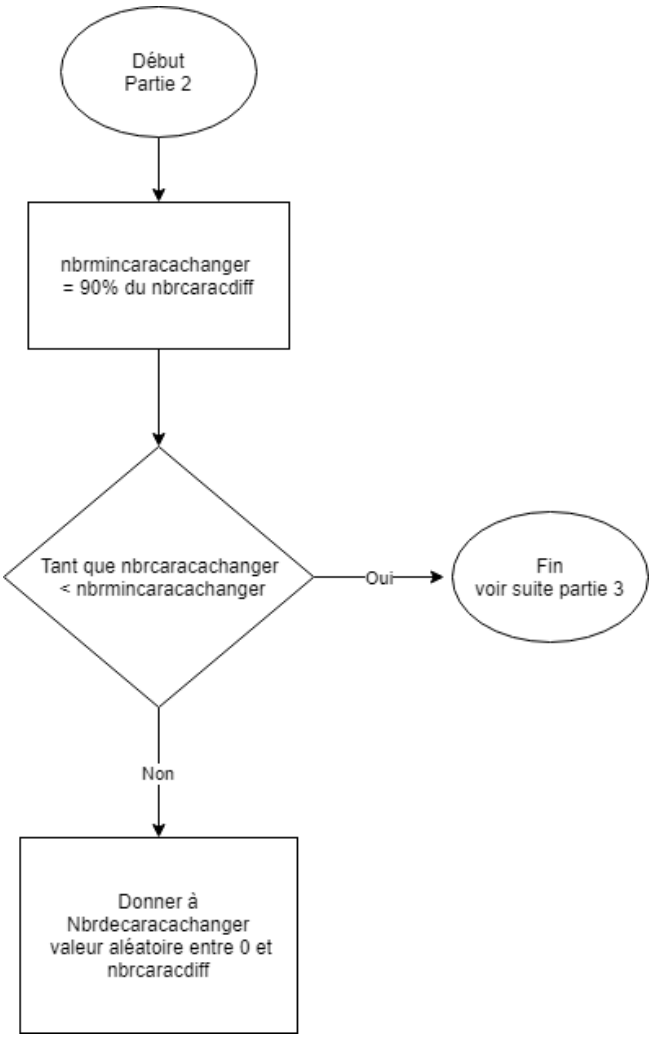
Voici maintenant les 3 algorigrammes :

Algorigramme de la première partie du programme :



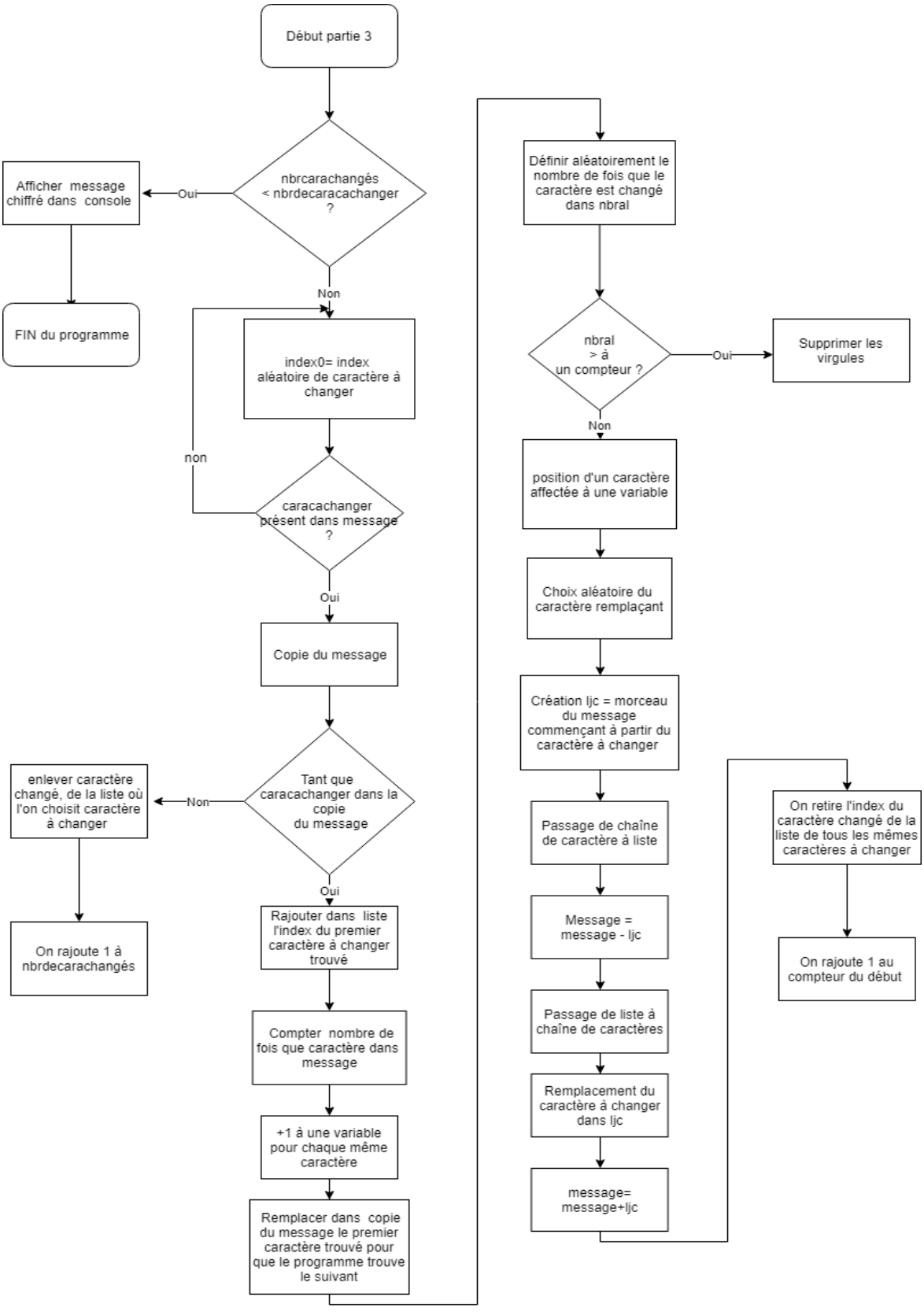
Cette partie du programme permet donc de connaitre le nombre de caractères différents qu'il y a dans le message grâce à la variable nbrcaracdiff mais aussi d'avoir ces caractères. Ceci nous permettra par la suite de les changer plus facilement.

Algorigramme de la seconde partie du programme :



Cette partie permet de définir la valeur de la variable nbrdecaracachanger. Ainsi on saura combien de caractère on aura à changer au cours de ce programme

Algorigramme de la dernière partie du programme :



Cette dernière partie du programme permet donc de changer certains caractères du mot, un certain nombre de fois, le tout étant déterminé aléatoirement. Pour ce faire, on a dû créer plusieurs boucles « tant que » et « si ».

Durant la réalisation de ce programme j'ai été confronté à plusieurs problèmes. Le premier problème a été celui de l'essence même des listes en JavaScript. En effet, lorsque je mettais dans une liste les index de chaque caractère différent, la liste était créée de telle sorte que tous les index étaient mis à la suite sans séparation, de plus si l'index était supérieur à dix, il était compté comme deux index distincts (l'index 1 Et l'index 0). J'ai essayé de rajouter des virgules entre chaque index mais de cette sorte, cela créait un grand nombre de virgule entre chaque index. Je me suis rendu compte que ce n'était pas la meilleure solution.

Après de nombreuses recherches, je me suis donc dirigé vers l'utilisation de tables. Celle-ci permettant de séparer correctement les index sans dissocier ceux supérieur à 10. Cependant l'utilisation de certaines fonctions comme la fonction splice n'est pas applicable pour les tables, elle n'est utilisable que sur les listes, pour régler ce problème il a donc fallu, après avoir utilisé au début du programme une table, modifier son type en liste pour pouvoir utiliser cette fonction.

Le passage d'un type de variable à un autre a eu pour conséquences d'intégrer des virgules à la chaîne de caractères, Il a donc fallu en tenir compte et créer plusieurs lignes de codes permettant de les supprimer. C'est à ce moment que s'est posée la question de la présence des virgules au sein du message. Pour résoudre cette dernière, nous avons décidé que nous allions afficher un message aux utilisateurs pour leur indiquer que leurs messages devaient être dépourvus de toute virgule.

En résumé, par un processus rendu aléatoire, ce programme permet de remplacer les caractères de telle sorte que le message de base ne soit pas lisible. Cependant, ce programme, bien que fonctionnant très bien seul, n'est pas relié au site même. Il faudra donc lier le JavaScript à cette page web et récupérer le contenu du message que l'on souhaite envoyer. En effet, dans le programme précédemment réalisé, il n'est question que d'un message déjà créé et de valeur finie.

B. Le déchiffrement

Par manque de temps, nous n'avons pas pu réaliser le déchiffrement du message. Cependant plusieurs idées nous sont venues à l'esprit pour pouvoir réussir ceci.

Nous avons pensé à lier le message envoyé aux informations présentes sur la base de données du site à propos destinataire du message, ainsi, toutes les variables aléatoires y seraient stockées de façon à ce que l'on pourrait les réutiliser pour déchiffrer le message. Cependant, nous ne savons pas comment faire ceci. De plus, si nous le faisons de cette sorte, plusieurs problèmes se posent : Il faut avoir la place de stocker toutes ces données dans les informations relatives au destinataire ; Comment résoudre le fait que deux messages puissent être envoyés au même moment à la même personne, ceci poserait le souci du déchiffrement car il ne serait pas aisé de savoir quelles variables correspondent au premier message et lesquelles correspondent au deuxième. Il faudrait donc faire en sorte de résoudre ces problèmes.

IV. Conclusion

Nous avons donc réussi à créer un site avec une messagerie et grâce auquel nous pouvons nous connecter. Cependant, nous n'avons pas eu le temps de relier le chiffrement fait en JavaScript au reste du programme, le déchiffrage n'a lui non plus pas eu le temps d'être réalisé. De ce fait, le site n'est pas complètement fini. La présentation du site pourrait elle aussi être améliorée.

Même si nous n'avons pas pu le finir dans son intégralité, ce projet nous a tous permis d'améliorer et de développer notre rigueur, notre esprit de recherche, ainsi que nos capacités à travailler au sein d'un groupe.

V. Annexes

A. Liens et compléments

Nom de la variable	Intérêt de cette variable
indexlistettcarac	Variable qui va nous servir à connaître les différents types de caractères qu'il y a dans le message.
Liste avec tous les caractères du clavier	Liste comprenant la plupart des caractères présents sur le clavier.
nbrcaracdiff	Variable nous servant à connaitre le nombre de fois qu'il y a un caractère donné dans un mot.
listecaracdiffmsgacrypter	Liste comprenant l'intégralité des caractères présents dans le message .
nbrmincaracachanger	Variable permettant de définir un nombre minimum de caractères à changer au sein du mot.
nbrcaracachanger	Variable définissant le nombre réel de caractères qui sont à changer.
Nbrcarachangés	Variable nécessaire à une grande partie du message. Elle permet de compter le nombre de caractères qui ont été changés.
Index0	Index définissant la position dans une liste d'un caractère qui devra être changé parmi tous les caractères présents dans le message. Elle est changée aléatoirement.
Caracachanger	Variable qui définit le caractère qu'il faut changer.
Nbral	Variable aléatoire permettant de définir le nombre de fois qu'un même caractère doit être changé.
Compteur	Variable permettant de répéter le changement du caractère à changer par un autre, lui aussi choisi aléatoirement.
Ljc	Variable qui va permettre d'effectuer le changement de caractère.

Tableau regroupant les différentes fonctions qui ont pu être utilisées tout au long de ce programme ainsi qu'une courte explication pour pouvoir mieux comprendre le programme

Fonction à travers un exemple	Intérêt à travers un exemple
Math.floor(Math.random()*5)	Permet de choisir un nombre aléatoire entre 0 inclus et 5 exclus.
Math.ceil(Math.random()*5)	Permet de choisir un nombre aléatoire entre 0 exclus et 5 inclus.
Message.indexOf(5)	Cette fonction permet de retourner l'index du premier 5 trouvé au sein de la variable message, si ill n'y en a aucun dans le message, la fonction retourne la valeur -1.
Message.length	Permet de savoir la longueur de ma variable Message
While/ If/ else	Boucles tant que/ si/sinon qui permettent d'automatiser le code en y introduisant des conditions
Console.log()	Cette fonction est utilisée à la toute fin du programme mais en réalité tout au long de celui-ci pour permettre d'afficher ce que l'on veut (entre autre ici les résultats ressortis pour vérifier qu'il n'y a pas d'erreur) dans la console du site web, accessible en faisant f12 une fois arrivé sur une page web classique.
Message.replace(5 ;4)	Permet de remplacer dans la variable Message le premier 5 trouvé par un 4.
Message.substring(5)	Permet d'extraire une partie de la variable Message(de l'index 5 jusque la fin de la variable
Message.splice(5)	Permet de retirer tout ce qui suit l'index 5
Message=Array.from(message) Message =message.toString()	Fonctions permettant respectivement de passer d'une chaîne de caractères à une table et inversement.

Lien google drive permettant d'accéder aux captures d'écrans du programme de chiffrement :

https://docs.google.com/document/d/1V3kpm3vkcsC9xT1YESn8qgXlL-2A2fszVIdCzzbciT4/edit?usp=sharing

B. Sources

Sites :

php.net
sql.sh
primfx.com
openclassrooms.com (forum et cours)
xul.fr
developer.mozilla.org
lephpfacile.com
stackoverflow.com
tutorialspoint.com
jkorpela.fi
phpdebutant.org
commentcamarche.net (faq)