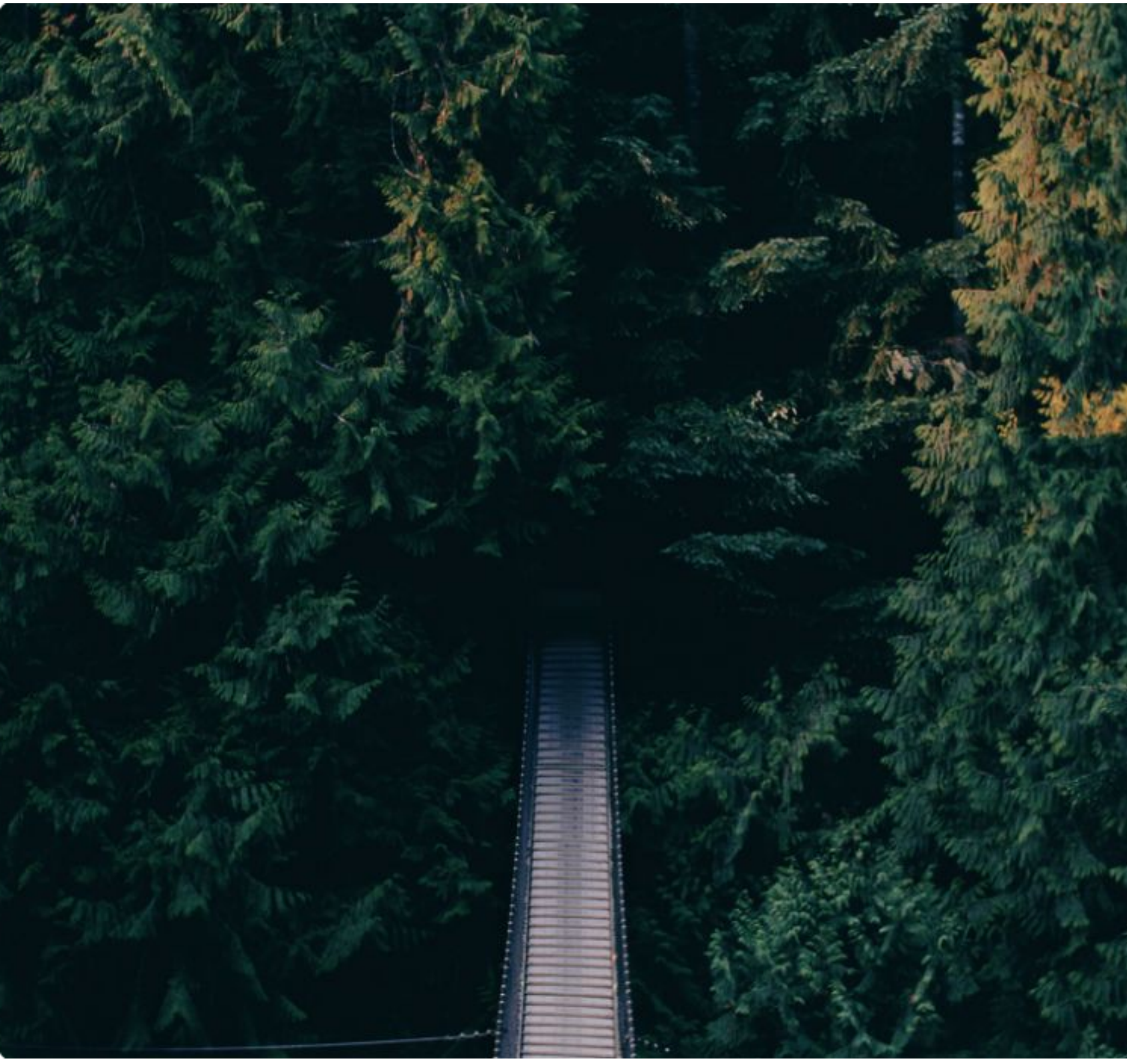
Meiresonne
Israel
53298

Projet DIN: Messagerie instantanée



Introduction

Le projet que je compte présenter est une plateforme de messagerie instantanée permettant à deux utilisateurs de de communiquer.
Malheureusement je n'ai pas eu le temps d'implémenter toutes les fonctionnalités prévues précédemment car j'ai mal géré mon temps.
Vois-ci un récapitulatif des fonctionnalisées disponibles

1. Fonctionnalités

- Affichage de la page d'accueil (avec le nom, prénom et matricule ainsi que le logo de l'ESI) grace au framework
- sources:

- template (incluent code htm et css): https://www.w3schools.com/w3css/tryit.asp?filename=tryw3css_templates_coming_soon&stacked=h

2. Description de la base de donnée

Table Users:

- Regroupe tous les utilisateur inscrit sur la plateforme (les inscrits et les administrateurs)
- La colonne `permission` permet de de distinguer les utilisateurs("users") des administrateurs("admin")
- Le mot de passe:
- Il est crypté en combinant les fonctions de ashage "sha1()" puis "password_hash()":

```php
/** crypt the password passed in parm
 * @return: the hashcode of the password passed in param
 */
function cryptPass($password)
{
    return password_hash(sha1($password), PASSWORD_BCRYPT);
}
```

- le mot de passe n'est jamais décrypté, lors de la vérification du mot de passe de l'user, la fonction ci-dessous vérifie juste si le mot de passe entré correspond au hachage stocké dans la colonne Users.password:

```php
/** vérify if the hashcode of password entered by the user  is the same that the
 *  one witch is stored in the DB
 * @return: true if $password correspond to the hash code stored in DB else false*/
function verifyCrypt($password, $passHash)
{
    return password_verify(sha1($password), $passHash);
}
```

- ainsi le mot de passe n'est jamais connu du système mais juste par l'utilisateur

Table Users_Professions:

- Cette table regroupe les différentes professions que peuvent pratiquer un utilisateur
- Ainsi un utilisateur peut indiquer autant de profession qu'il désire

Table Professions:

- Cette table contient les différentes professions disponibles pour les utilisateurs

Table Contacts:

- Cette table regroupe les différentes contacts que possède un utilisateur
- la colonne contactStatus permet d'indiquer la relation entre deux utilisateur:

- "know": l'utilisateur **pseudo**_ connais l'utilisateur **contact**
- "unknow": l'utilisateur **pseudo**_ ne connais pas l'utilisateur **contact**
- "blocked": l'utilisateur **pseudo**_ a bloqué l'utilisateur **contact**

- Cette organisation de table permet a un utilisateur d'en connaitre un autre et lui envoyer des message sans que ce dernier ne le connaisse à condition d'avoir son pseudo

- si le second utilisateur ne désire pas parler avec un autre ou s'il ne le connais pas il a la possibilité de le bloquer
- sinon si il connais celui qui lui écrit il peut le marquer comme connu 'know' et ainsi l'ajouter à ses contact
- ainsi deux utilisateurs qui se connaissent l'un l'autre génère deux ligne

II Table Discussions:

- Cette table regroupe toutes les discussions crée ainsi que leurs informations et qui sont toujours détenue (non supprimée) par au moins un participant de la discussion

Table Participants:

- Cette table permet qu'elle sont les participant de chaque discussion
- ainsi une discussion peut accueillir minimum deux utilisateurs

Table Messages:

- Cette table permet de savoir à quelle discussion appartient chaque message et qui est son expéditeur
- Les message peuvent être du text(msgType = "text") ou être de une image (msgType = "image")
- les message sont crypté et la l'attribut **Messages.msgPublicK** contient la clef **privé** utilisé pour crypté le message (⚠️ j'ai fait une erreur dans le nomage de cette attribut car c'est effectivement la clef privé de l'utilisateur qui est stocké et non la public)
- Cryptage des messages:

1. lorsque un utilisateur de connecte à son compte, une clef privé et une clef public sont généré et stocké dans sa session ($_SESSION) grace aux fonctio PHP dédié:

- générer une paire de clef privé/public: `**openssl_pkey_new()**`
- extraire la clef privé: `**openssl_pkey_export()**`
- extraire la clef public: `**openssl_pkey_get_details()**`

2. Les message envoyé par chaque utilisateur sont crypté avec la clef public à l'aide de la fonction ci-dessous et stocké dans la table Messages avec le clef privé qui l'a crypté:

- crypter le message avec la clef public: `**openssl_public_encrypt()**`

3. Quand le code javascript du destinataire rafraichira la discussion grace à une requête Ajax en method POST, ce dernier recevra tous ses nouveaux message. Ansi le system va récupérer les messages et les décrypter avec la clef privé qui les a crypté et l'envoyer à son destinataire (on peut envoyer le message en textuel car les requête de type POST sont déjà sécurisé contre les MITM):

- décrypter le message avec la clef privé: `**openssl_private_decrypt()**`

3. Code d'accès

Désolé je n'ai pas assez développé mon code pour implémenter les codes d'accès.

4. Conclusion

Mon projet est très loin d'être finis malheureusement en raison notamment de ma mauvaise gestion du temps où j'ai perdu beaucoup de temps à faire des schéma au lieu de coder.
Je tiens à m'excuser du temps que je vous prend pour corriger un projet quelque peu vide.