

DOSSIER DE PROJET

Nicolas GUIGAY

Développeur Web Full stack

Application web et web mobile

Hello Voisins

<https://hello-voisins.com>

H

Sommaire :

| | |
|--|-----------|
| Liste des compétences du référentiel : | 3 |
| 1. Activité type 1 : « Développer la partie Front-end d'une application web ou web mobile en intégrant les recommandations de sécurité » | 3 |
| 2. Activité type 2 : « Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité » | 3 |
| Résumé du projet : | 4 |
| Environnement technique : | 5 |
| Cahier des charges : | 6 |
| 1. Description..... | 6 |
| 2. Objectifs | 6 |
| 3. Contexte technique | 7 |
| Réalisation du projet : | 8 |
| 1. La maquette | 8 |
| 2. Initialisation du projet | 9 |
| 3. Le développement Front-End | 10 |
| 4. La Base De Données | 12 |
| 5. Mise en place du chat grâce aux Websockets avec Ratchet | 14 |
| 6. Fonctionnalité la plus représentative (la gestion de contacts pour l'utilisateur) | 18 |
| 7. Recherche anglophone Geolocation API / Google Maps API..... | 21 |
| Sécurité : | 24 |
| Protections contre les attaques : | 24 |
| Sécurité de l'utilisateur : | 25 |
| Mise en ligne : | 26 |
| Configuration du serveur sur DigitalOcean..... | 26 |
| Perspectives d'améliorations : | 28 |
| Bilan : | 29 |
| 1. Compétences acquises..... | 29 |
| 2. Compétences à améliorer | 29 |

Lien du projet GitHub : (<https://github.com/Nicode611/Hello-voisins>)

Liste des compétences du référentiel :

1. Activité type 1 : « Développer la partie Front-end d'une application web ou web mobile en intégrant les recommandations de sécurité »

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateur web dynamique

L'application sera disponible sur le web et devra donc s'adapter à n'importe quelle taille d'écran, il a donc été nécessaire de maquetter l'application afin d'avoir une vision d'ensemble sur le visuel de l'application et les différents formats.

2. Activité type 2 : « Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité »

- Développer les composant d'accès aux données
- Développer la partie back-end d'une application web ou web mobile

Hello-voisins utilise majoritairement PHP afin de communiquer avec une Base de données MySQL hébergée chez AlwaysData, mais aussi afin de créer le serveur Websocket Ratchet et le gestionnaire des connexions à ce serveur. Ce qui permet à l'application d'intégrer une messagerie instantanée pour tous les utilisateurs.

Résumé du projet :

Hello-Voisins est une application web conçue pour faciliter la communication entre voisins.

L'idée de la création de cette application m'est venue lors de la préparation d'un gâteau. Il me manquait quelques ingrédients et c'est là que m'est venue l'idée :

« Il serait pratique de pouvoir discuter avec ses voisins sans avoir à toquer à chaque porte pour vérifier s'ils ont ce qu'il me faut. »

L'application permet aux utilisateurs de discuter avec leurs voisins en utilisant la localisation des utilisateurs.

Elle permet aussi la création de groupes de discussion pour des immeubles, résidences, ou pour tout autre cas d'usages, et inclus également la possibilité de voir tous les utilisateurs à proximité sur une carte, tout en respectant la confidentialité des utilisateurs qui peuvent choisir de ne pas apparaître.

Une première version de l'application est déjà en ligne à l'adresse hello-voisins.com

Environnement technique :

Après avoir terminé mon ECF, il me restait plus que 2 mois pour créer mon application et mon dossier projet.

N'ayant personne dans mon entourage qui soit dans le monde du développement web, j'ai dû travailler en totale autonomie.

Pour se faire, j'ai pu m'aider de diverses sources d'informations tels que les forums, les tutos vidéo, ou même les nouvelles technologies comme ChatGPT ou Google Bard, qui sont de réels atouts pour les développeurs.

Je travaille depuis chez moi sur un PC fixe mais également depuis la médiathèque sur un PC portable. J'ai donc eu besoin de rendre tout ce projet « portable ».

Pour se faire j'ai utilisé des services en ligne tels que Figma, Trello, Evernote.

Pour développer mon application j'ai dû être efficace et bien organisé.

Il me fallait donc absolument organiser mes pensées. Pour ça, je me suis beaucoup aidé de Trello et d'un simple tableau blanc afin d'y noter toutes les étapes qu'il me restait, celles que j'avais terminées et celles qui étaient urgentes.

Cahier des charges :

1. Description

L'application Hello-voisins vise à faciliter la communication et l'échange entre voisins. L'idée est née d'une expérience personnelle lors de la préparation d'un gâteau, où le besoin d'ingrédients a suscité l'idée de créer une plateforme favorisant la collaboration entre voisins.

2. Objectifs

- Permettre aux utilisateurs de communiquer avec leurs voisins directement via l'application.
- Utiliser la localisation pour faciliter la recherche et la communication avec les voisins à proximité.
- Ajouter des contacts
- Créer des groupes personnalisables pour une communication ciblée, tels que des groupes d'immeubles, de résidences, ou de quartiers.
- Intégrer une fonctionnalité de carte pour visualiser les utilisateurs à proximité, avec la possibilité pour les utilisateurs de choisir s'ils veulent apparaître sur la carte.

3. Contexte technique

Voici les différentes technologies que j'ai utilisé pour ce projet :

- **Environnement de travail :**

- **XAMPP** : Le logiciel XAMPP permet la création d'un serveur Apache en local et cumulé avec l'outil BrowserSync, permet la synchronisation des changements sur le code en temps réel sur le navigateur.

- **GitHub** : Github permettra de déposer le code après chaque commit afin de le sauvegarder et de pouvoir le réutiliser sur le serveur de production ou sur d'autres machines assurant ainsi une portabilité du développement.

- **Partie Front-end :**

- **HTML, CSS** : Utilisation de HTML et CSS en « vanilla » (sans framework) pour le développement de l'interface utilisateur.

- **Javascript** : Intégration de JavaScript, accompagné du framework jQuery, afin d'apporter une dynamique au site, notamment à travers l'utilisation de requêtes AJAX pour une communication asynchrone avec le serveur.

- **Partie Back-end :**

- **PHP** : Utilisation de PHP comme langage de programmation pour le back-end, assurant la gestion des communications avec le serveur, l'interaction avec la base de données, et la création du serveur WebSocket Ratchet.

- **Ratchet** : Framework PHP qui simplifie le processus de mise en place d'un serveur WebSocket en fournissant une infrastructure et des outils pour gérer la communication en temps réel. Ce dernier permet une communication bidirectionnelle entre le client et le serveur, ce qui est très utile pour la création d'une fonctionnalité de messagerie instantanée.

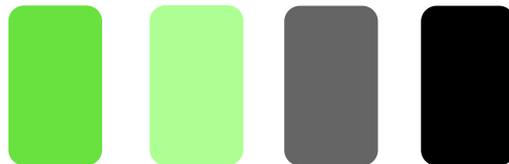
- **DigitalOcean** : DigitalOcean est, entre autres, un fournisseur de VPS offrant une grande personnalisation des machines. Cette flexibilité sera particulièrement bénéfique pour la configuration du serveur Apache en production.

Réalisation du projet :

1. La maquette

Pour mener à bien le projet, il faut que le design de mon application ainsi que les différents éléments présents dans mes pages soient bien définis afin que je puisse me concentrer sur une chose à la fois.

Je commence par créer le wireframe de l'application pour mettre en place les différents éléments puis je me rends sur Dribbble et Behance afin de trouver de l'inspiration. Je trouve quelques idées de styles graphique, puis je défini la palette de couleur que j'utiliserais sur le site :



Je trouve aussi quelques icones sur SVGRepo puis je me mets à créer la maquette sur Figma.

Lien vers la maquette :

(<https://www.figma.com/file/kql78BeqwTpd8hf6QkxVL/Untitled?type=design&node-id=18%3A708&mode=design&t=oNDofLisHmtGYKc-1>)

Je commence par la version mobile pour respecter la règle du « mobile-first »

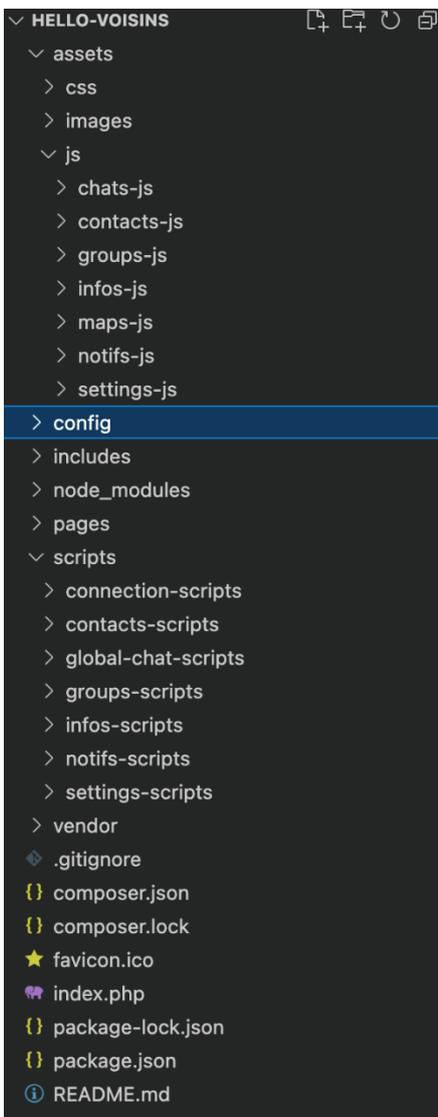
2. Initialisation du projet

Après avoir créé la maquette, je dois initialiser mon projet afin de passer au code.

J'utilise l'outil BrowserSync qui permet la synchronisation des changements sur le code en temps réel sur le navigateur, je dois donc l'importer sur mon projet avec « `npm install -g browser-sync` » .

J'organise ensuite les dossiers de mon application afin d'être bien organisé.

Voici l'arborescence des dossiers de l'application :



- **Assets** : Ce répertoire contient les fichiers statiques tels que les fichiers CSS, JavaScript et les images. Les scripts JS sont classés dans des sous-dossiers afin de repérer facilement à quel type d'action ils sont associés.
- **Config** : Contient tous les fichiers de configuration de l'application tels que le server Ratchet ou encore les infos de la BDD.
- **Includes** : Contient les fichiers autre que les scripts qui sont à inclure dans les différentes pages
- **Pages** : Contient les différentes pages de l'application.
- **Scripts** : Contient tous les scripts Back-end de l'application, ces scripts, tout comme pour les scripts JS sont classés dans des sous-dossiers.

3. Le développement Front-End

Je passe ensuite au développement de la partie visuelle de l'application.

L'application aura un menu de navigation comprenant toutes les infos de l'utilisateur connecté. Il devra être accessible depuis n'importe où sur l'application, je décide donc de placer ce menu dans le répertoire includes.

Pour le développement de la partie Front-End de mon projet, j'opte pour l'utilisation de HTML et CSS sans frameworks, car je me sens plus à l'aise avec ces langages.

Afin de rendre l'interface utilisateur dynamique il est nécessaire d'utiliser Javascript. Je l'utilise donc également en vanilla, excepté pour les requêtes AJAX que j'écris en JQuery pour les simplifier.

Voici quelques exemples de mon utilisation de Javascript qui ont permis de rendre mon site plus dynamique :

Manipulation du DOM permettant d'afficher le message envoyé dans le chat coté client :

```
function appendSentMessage(message, profileImgPath) {  
  
    var messageContainer = document.createElement('div');  
    messageContainer.className = 'sent-message-container';  
  
    var userImg = document.createElement('img');  
    userImg.className = 'self-user-img';  
    userImg.src = '../' + profileImgPath;  
    userImg.alt = '';  
  
    var messageText = document.createElement('p');  
    messageText.className = 'sent-message';  
    messageText.textContent = message;  
  
    messagesContainer.appendChild(messageContainer);  
    messageContainer.appendChild(userImg);  
    messageContainer.appendChild(messageText);  
  
    scrollToBottom();  
}
```

Cette fonction prend en paramètres 2 valeurs (le message et le lien vers l'image de profil).

Elle crée un container « sent-message-container » et définit le contenu des éléments qui seront placés dans ce container.

Le container est ensuite placé dans le chat et les éléments précédemment établis sont à leur tour placés dans le container.

Pour finir le script appelle la fonction « **scrollToBottom** » afin de faire descendre le chat pour une meilleure expérience utilisateur.

Requête AJAX pour donner suite à une action :

```
deleteBtns.forEach(deleteBtn => {
  deleteBtn.addEventListener('click', function(event) {
    var clickedBtn = event.target;
    var notifContainer = clickedBtn.closest(".notification-container");
    var confirmMessage = document.createElement("p");
    confirmMessage.textContent = "Refusé";
    const choice = 'refuses';

    $.ajax({
      type: 'POST',
      url: '../scripts/notifs-scripts/script-choice-notifs.php',
      data: {
        choice_notifs: choice
      },
      success: function(responseData) {
        // Contient les données JSON retournées par le script PHP
        responseData = JSON.parse(responseData);

        if (responseData == 'deleted') {
          notifContainer.replaceWith(confirmMessage);

          var delai = 3000;

          setTimeout(function() {
            confirmMessage.remove();
          }, delai);
        }
      }
    });
  });
});
```

Ce script décrit l'action à effectuer lors du clic sur le bouton de refus d'une notification

Tout d'abord j'itère à travers « deleteBtns » à l'aide de « forEach » pour cibler chaque élément individuellement.

Je prépare ensuite le message à retourner au client une fois le script terminé, puis je défini le choix dans ce cas « refuses » pour indiquer au script PHP que le client a refusé la notification (le script PHP s'occupera de traiter la demande en fonction du choix qui est défini).

Je crée ensuite la requête AJAX qui envoie le choix au script PHP, puis je traite la réponse retournée par le script PHP dans « **success : function(responseData)** ».

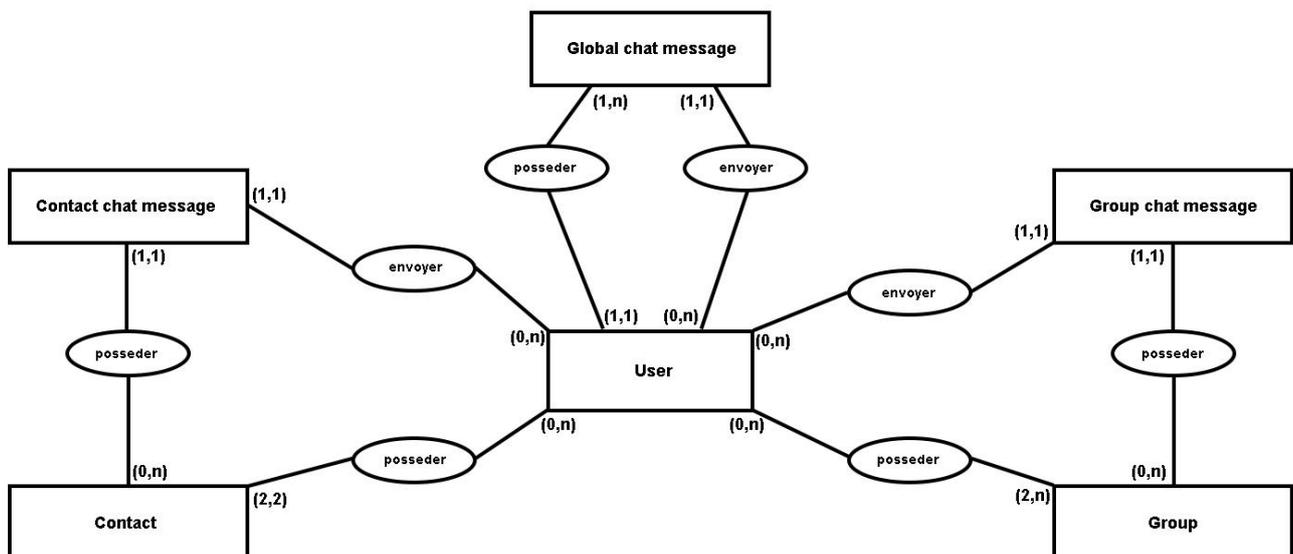
Si la réponse est « deleted » alors j'affiche le message précédemment préparé dans le conteneur de notification puis le supprime au bout de 3secondes.

4. La Base De Données

La base de données est primordiale pour mon application car elle permet de gérer plusieurs fonctionnalités tels que :

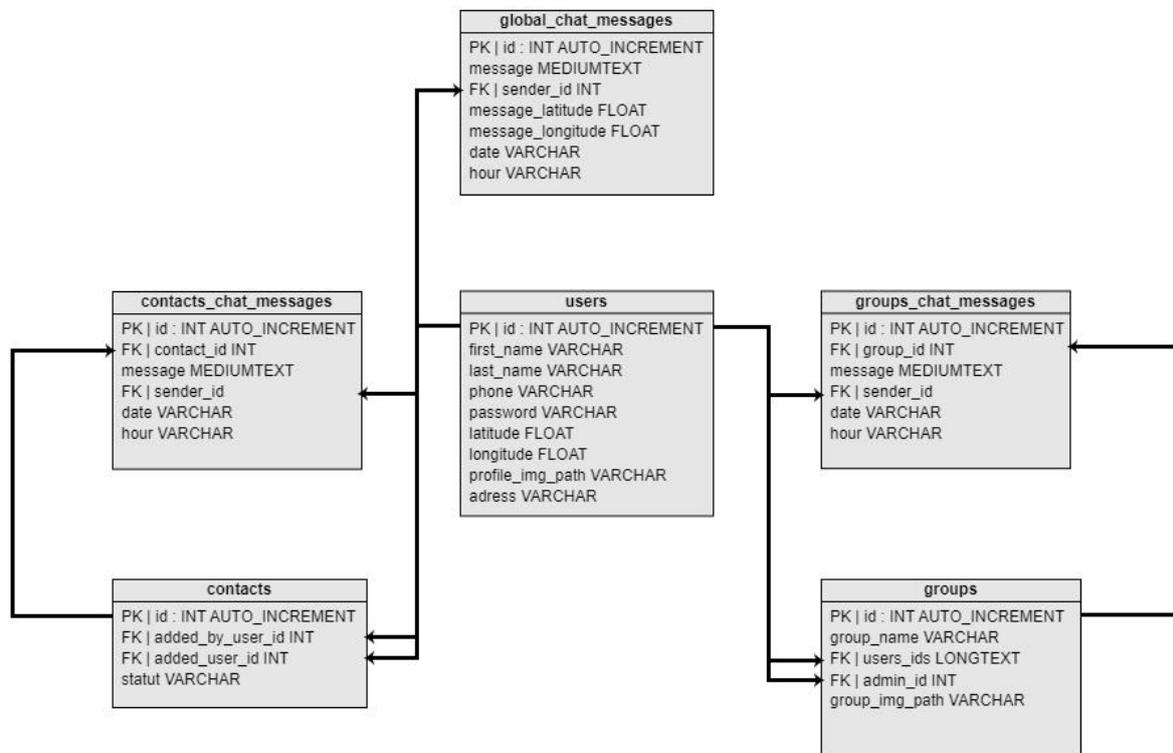
- Les utilisateurs
- Les messages des différents chats
- Les relations de contacts
- Les relations de groupes

Pour faciliter la création de celle-ci j'ai établi le Modèle Conceptuel de Données (MCD). Il permet d'établir les relations entre les différentes entités, en précisant les cardinalités :



Modèle Conceptuel de Données (MCD)

Le MCD réalisé, j'ai une vision plus précise de la structure de ma base de données, je peux à présent faire le Modèle Logique de Données (MLD) afin de voir en détail le contenu de mes tables et leurs relations.



Modèle Logique de Données (MLD)

Maintenant que toutes ces données ont été établies, je me rends sur AlwaysData et crée ma BDD grâce à PHPMyAdmin.

Afin de communiquer avec ma BDD je crée un fichier « db.php » dans le dossier « db » dans mon dossier « config », où je récupère le contenu de mes variables d'environnement contenant les infos de ma BDD (host, user, name, password) précédemment établis afin de pouvoir y accéder depuis n'importe où en important le fichier.

Tout est prêt je peux à présent interagir avec ma BDD en utilisant PHP de cette façon :

```

includeFile = "../../../config/db/db.php";
if (file_exists($includeFile)) { include($includeFile); } else { echo "Le fichier $includeFile n'a pas été trouvé."; }
$conn = new mysqli($db_host, $db_user, $db_pass, $db_name);

if ($conn->connect_error) {
    die("La connexion à la base de données a échoué : " . $conn->connect_error);
}
    
```

5. Mise en place du chat grâce aux Websockets avec Ratchet

Mon application inclue un service de messagerie instantanée.

J'ai dû trouver une solution pour que l'utilisateur puisse communiquer avec le serveur afin d'envoyer des messages et de les recevoir en temps réel.

Je me suis d'abord tourné vers les requêtes AJAX, puis j'ai trouvé une meilleure solution : ***Un serveur WebSocket.***

Le fonctionnement d'un serveur WebSocket est beaucoup plus avantageux pour mon application que des simples requêtes AJAX.

- **Communication bidirectionnelle** : Le serveur WebSocket va permettre au serveur et au client d'envoyer des données l'un à l'autre de manière indépendante, sans attendre une demande explicite du client. Cela facilite la mise à jour en temps réel des données des deux côtés, ce qui est plus difficile à réaliser avec des requêtes AJAX qui sont principalement basées sur un modèle de demande-réponse.
- **Latence réduite** : Les connexions WebSocket réduisent la latence car la connexion reste ouverte une fois établie, évitant ainsi le coût associé à l'établissement et à la fermeture fréquente des connexions, comme c'est le cas avec les requêtes AJAX.
- **Efficacité** : Les WebSocket sont plus efficaces en termes de bande passante car le protocole WebSocket est plus léger que HTTP, éliminant le surpoids lié aux en-têtes HTTP pour chaque requête.

Le choix est fait, je dois utiliser un serveur websocket pour gérer la messagerie instantanée de mon application.

Pour l'installation du serveur avec PHP je dois utiliser la librairie Ratchet.

J'initialise donc mon projet avec Composer en lançant : **composer require cboden/ratchet**. Cette commande initialise mon projet en créant le fichier composer.json et spécifie d'utiliser la librairie Ratchet.

Je lance ensuite **composer install** pour installer toutes les librairies nécessaires au bon fonctionnement de Ratchet, puis **composer update** afin de mettre à jour ces librairies.

Une fois toutes les librairies installées, je peux enfin créer mon serveur Websocket. Dans mon fichier config je crée un fichier nommé « server.php », ce fichier contiendra mon serveur Websocket

Je dois spécifier au serveur quel port utiliser (8888), information importante car il sera nécessaire d'ouvrir ce port lors de la configuration du serveur de production.

```
<?php
require dirname(__DIR__) . '/vendor/autoload.php';

use Ratchet\Server\IoServer;
use Ratchet\Http\HttpServer;
use Ratchet\WebSocket\WsServer;
use \MyApp\Chat;

$port = 8888;

$server = IoServer::factory(
    new HttpServer(
        new WsServer(
            new Chat()
        )
    ),
    8888
);

$server->run();
```

Déclaration du serveur websocket

Une fois ce serveur initialisé, je dois aussi créer le gestionnaire des connexions à ce serveur. Je crée un nouveau fichier nommé « Chat.php » que je place également dans mon dossier « config ».

Ce fichier sera le cerveau du serveur Websocket, il va permettre d'indiquer comment gérer les différentes actions sur le serveur (connexions, déconnexions, messages, erreurs etc...).

La documentation de Ratchet fourni un fichier gérant les actions de base, mais pour mon application je dois le modifier afin de personnaliser certaines actions.

Le fichier contient les fonctions principales :

- **onOpen** : qui est appelée lorsqu'une nouvelle connexion est établie.
On l'appelle coté client via `conn.onopen = fonction(e)`
- **onMessage** : qui est appelée lorsqu'un nouveau message est reçu.
On l'appelle coté client via `conn.onmessage = fonction(e)`
- **onClose** : qui est appelée lorsqu'une connexion se ferme
On l'appelle coté client via `conn.onclose = fonction(e)`
- **onError** : qui est appelée en cas d'erreur sur une connexion
On l'appelle coté client via `conn.onerror = fonction(e)`

J'y ai également rajouté quelques fonctions utilitaires, en voici 3 :

« **sendToChannel** » qui permet d'envoyer un message sur le channel où est connecté l'utilisateur.

```
private function sendToChannel($channelName, $message) {
    if (isset($this->channels[$channelName])) {
        $messageData = json_encode($message);
        foreach ($this->channels[$channelName] as $client) {
            $success = $client->send($messageData);
            if (!$success) {
                echo "Sur le channel ". $channelName ." le message n'a pas été envoyé par : ". $client->resourceId . " Le message = " . $messageData;
            } else {
                echo "Sur le channel ". $channelName ." le message a été envoyé par : ". $client->resourceId . " Le message = " . $messageData;
            }
        }
    }
}
```

Fonction *sendToChannel*

Cette fonction prend en paramètres 2 éléments, le nom du channel, et le message qui est un tableau contenant toutes les informations nécessaires comme par exemple :

- L'id, le nom, le lien vers l'image de l'utilisateur qui a envoyé le message.
- La date du message.
- La position à laquelle il a été envoyé.

La fonction reçoit ces paramètres et encode le message en json afin de l'envoyer à tous les utilisateurs (\$client) présents sur le channel.

On vérifie également si le message a été envoyé en affichant dans la console un message lors de l'envoi ou non du message.

« **getAllConnectedUsersDataInChannel** » qui permet de récupérer tous les utilisateurs connectés dans un channel.

```
private function getAllConnectedUsersDataInChannel($channelName) {
    $connectedUsers = [];

    if (isset($this->channels[$channelName])) {
        foreach ($this->channels[$channelName] as $userConnection) {
            $userData = $this->usernames[$userConnection->resourceId];
            $connectedUsers[] = $userData;
        }
    }
    return $connectedUsers;
}
```

Fonction getAllConnectedUsersDataInChannel

La fonction récupère le nom du channel, et crée une boucle foreach afin de parcourir toutes les connexions du channel et en récupérer les données des utilisateurs stockées dans `$this->usernames[$userConnection->resourceId]` puis les retourne dans un tableau.

« **sendConnectedUsersDataToUserInChannel** » permet de récupérer le tableau des utilisateurs connectés au channel précédemment défini dans la fonction précédente et le renvoi sous forme de JSON aux utilisateurs dans le channel.

```
private function sendConnectedUsersDataToUserInChannel(ConnectionInterface $userConnection, $channelName) {
    $connectedUsers = $this->getAllConnectedUsersDataInChannel($channelName);

    $userConnection->send(json_encode(["connected_users" => $connectedUsers]));
}
```

Fonction sendConnectedUsersDataToUserInChannel

6. Fonctionnalité la plus représentative (la gestion de contacts pour l'utilisateur)

La fonctionnalité la plus représentative de l'application réside dans la gestion des contacts pour les utilisateurs. Sur l'application, les utilisateurs ont la possibilité d'ajouter d'autres utilisateurs à leur liste de contacts. Cette fonctionnalité est d'une importance capitale, car elle ouvre la porte à des conversations privées indépendamment de la distance, tout en offrant la possibilité de créer des groupes pour interagir avec plusieurs personnes simultanément.

La gestion des contacts s'opère à travers la table « contacts » de ma base de données, qui enregistre la relation entre chaque contact. Cette relation inclut l'utilisateur qui a ajouté le contact, l'utilisateur ajouté, ainsi que le statut de la relation (ajouté ou en attente).

Le processus d'ajout de contact implique des actions tant en Back-end qu'en Front-end, et je vais maintenant détailler l'ensemble du processus, mettant en lumière chaque étape de manière exhaustive.

1. L'ajout de contacts :

Afin de trouver de nouveaux contacts à ajouter, les utilisateurs peuvent se rendre dans le chat de proximité et y voir les utilisateurs connectés.



Lors du clic sur un utilisateur, le script « show-user.js » s'exécute. Ce script récupère l'identifiant associé à l'image de l'utilisateur en question et envoie une requête AJAX au script « script-select-users-to-show.php ». Ce dernier se charge d'interroger la base de données pour récupérer les informations de l'utilisateur ainsi que la relation de contact dans la table « contacts », si elle existe.

Une fois ces informations récupérées, le script PHP renvoie la réponse au format JSON à la requête AJAX, fournissant ainsi toutes les données nécessaires. Par la suite, le script « show-user.js » continue en affichant une fenêtre modale qui présente les informations de l'utilisateur concerné.

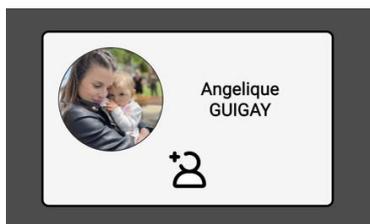
Si la relation de contact existe et est en statut « added », on affiche un message afin de notifier que le contact est déjà ajouté.



Si la relation de contact existe et est en statut « waiting », on affiche un message afin de notifier que la demande de contact est déjà effectuée.



Si la relation de contact n'existe pas, on affiche un bouton d'ajout de contact



L'utilisateur n'a plus qu'à cliquer sur le bouton d'ajout de contact afin de créer la relation de contact et de la passer en « waiting ».

2. La validation ou le refus via les notifications :

À chaque rechargement de page, le script « show-notifs.php » est déclenché par le biais d'une requête AJAX. Ce script a pour objectif d'actualiser le contenu du conteneur de notifications en vérifiant la présence de nouvelles demandes d'ajout de contacts. Pour ce faire, il effectue une requête SQL qui recherche toutes les relations de contact ayant l'utilisateur actuel (celui dont l'ID est stocké dans la variable de session) comme destinataire et ayant le statut « waiting ».

Voici la requête SQL :

```
"SELECT * FROM contacts WHERE added_user_id = $selfId AND statut = 'waiting';"
```

Si le résultat n'est pas vide on ajoute la notification dans le conteneur sous cette forme :



L'utilisateur a alors le choix d'accepter ou de refuser.

S'il accepte : la relation passe en statut « added », la notification est supprimée, un message de validation s'affiche et le contact est ajouté à la liste de contacts.

S'il refuse : La relation est supprimée, la notification est supprimée et un message de validation s'affiche.

7. Recherche anglophone Geolocation API / Google Maps API

L'application utilise la localisation de l'utilisateur pour gérer diverses fonctionnalités, telles que la carte affichant l'emplacement de tous les utilisateurs ou le chat de proximité. N'étant pas familier avec de telles technologies au départ, j'ai entrepris des recherches sur plusieurs forums.

Après des investigations, j'ai fini par trouver un excellent article en anglais sur le site medium.com (<https://medium.com/risan/track-users-location-and-display-it-on-google-maps-41d1f850786e>) traitant précisément de tout ce dont j'avais besoin. Cet article m'a fourni l'aide nécessaire, notamment sur l'utilisation de l'API Google Maps pour créer une carte et placer des marqueurs de position aux emplacements des utilisateurs consentants.

J'ai pu suivre les explications de cet article pas à pas afin de créer ma propre carte.

La première étape consistait à créer un projet sur Google Cloud Console et à activer l'API "Maps JavaScript API". Ensuite, j'ai généré une clé API que j'ai intégrée dans mon application sous la forme suivante :

```
<script src="https://maps.googleapis.com/maps/api/js?key=YOUR API KEY&callback=initMap" async defer></script>
```

En suivant les étapes détaillées dans l'article, j'ai pu initialiser ma propre carte sans difficulté.

L'article explique également comment récupérer la position d'un utilisateur et la stocker sous forme de latitude et longitude grâce à l'API de géolocalisation, en utilisant la méthode « `getCurrentPosition()` ».

J'ai donc suivi ces instructions, dont voici un extrait :

Extrait du site en anglais :

Using the `getCurrentPosition` Method :

The following code checks whether the Geolocation API is supported by the browser. We simply check if the geolocation property exists within the navigator object.

```
if ('geolocation' in navigator) {  
    // Geolocation API is supported  
} else {  
    // Geolocation API is not supported  
    alert('Geolocation is not supported by your browser.');
```

If the Geolocation API is supported, we can then safely use the `getCurrentPosition()` method to retrieve user's location.

```
navigator.geolocation.getCurrentPosition(success, error, [options])
```

This method accepts three parameters:

success: A callback function that will be invoked once the user's location is retrieved successfully. The callback receives a `Position` object that holds the user's location.

error: A callback function that will be invoked if user's location is failed to retrieve. The callback accepts a `PositionError` object.

options: Is an optional `PositionOptions` object that can be used to configure the retrieval process.

Upon successful retrieval, we simply print the user's coordinate. And if it's failed, we show the error message using `alert`.

Open our page in the browser. It will ask your permission to get your current location. Click Allow to give it permission and proceed.

Our page ask for the permission to get the user's location.

 Geolocation API is only available in HTTPS

Note that this Geolocation API is only available in secure contexts. So you'll need to access your page through HTTPS protocol.

Traduction en français :

En utilisant la méthode `getCurrentPosition()` :

Le code suivant vérifie si l'API GeoLocation est prise en charge par le navigateur. Nous vérifions simplement si la propriété de géolocalisation existe dans l'objet navigateur.

```
if ('geolocation' in navigator) {  
    // Geolocation API is supported  
} else {  
    // Geolocation API is not supported  
    alert('Geolocation is not supported by your browser.');
```

Si l'API GeoLocation est prise en charge, nous pouvons alors utiliser en toute sécurité la méthode `getCurrentPosition()` pour récupérer l'emplacement de l'utilisateur.

```
navigator.geolocation.getCurrentPosition(success, error, [options])
```

Cette méthode accepte trois paramètres :

success : une fonction de callback qui sera invoquée une fois que l'emplacement de l'utilisateur aura été récupéré avec succès. Le callback reçoit un objet position qui contient l'emplacement de l'utilisateur.

error : une fonction de callback qui sera invoquée si la localisation de l'utilisateur ne parvient pas à être récupérée. Le callback accepte un objet `PositionError`.

options : est un objet `PositionOptions` optionnel qui peut être utilisé pour configurer le processus de récupération.

Une fois la récupération réussie, on enregistre simplement les coordonnées de l'utilisateur. Et en cas d'échec, on affiche le message d'erreur en utilisant une alerte.

Ouvrez la page dans le navigateur. Il vous demandera la permission d'obtenir votre position actuelle. Cliquez sur Autoriser pour lui donner l'autorisation et continuer.

La page demande l'autorisation d'obtenir la localisation de l'utilisateur.

 L'API de géolocalisation est uniquement disponible en HTTPS

Notez que cette API de géolocalisation n'est disponible que dans des contextes sécurisés. Vous devrez donc accéder à votre page via le protocole HTTPS.

Sécurité :

Au cours du développement de mon application, la sécurité a été primordiale. J'ai mis en place plusieurs mesures pour prévenir les attaques potentielles et assurer la sécurité des données, en mettant l'accent sur la confidentialité, l'intégrité des données.

Protections contre les attaques :

Injections SQL : Les requêtes SQL qui injectent du contenu directement écrit par l'utilisateur, sont conçues de manière sécurisée, en utilisant des méthodes paramétrées, pour éviter tout risque d'injection SQL. La validation des entrées utilisateur est aussi faite du côté du formulaire afin de garantir un format de donnée correct.

Faille XSS : Des mécanismes de sécurité, tels que l'échappement des caractères spéciaux avec « htmlspecialchars » et la validation des données, sont mis en œuvre pour prévenir les attaques XSS. Les entrées utilisateur sont filtrées et échappées pour éviter l'exécution de scripts malveillants.

Variables d'environnement : Les infos sensibles de la BDD sont stockées dans des variables d'environnement sur le serveur afin d'éviter que ces données sensibles ne puissent fuir.

Vérifications des formats de fichiers pour les images : Les téléchargements d'images sont restreints aux formats autorisés, et chaque fichier est inspecté pour s'assurer qu'il respecte les normes définies. De plus les images sont compressées lors du téléchargement sur le serveur. Cela garantit l'intégrité du système en évitant les risques potentiels liés aux fichiers malveillants.

Sécurité de l'utilisateur :

Mots de passe hachés : Les mots de passe des utilisateurs sont stockés de manière sécurisée dans la BDD, en utilisant la méthode « password_hash() » qui permet de hacher le mot de passe. Cela garantit que même en cas d'accès non autorisé, les informations sensibles restent inaccessibles.

Transparence envers l'utilisateur : Lorsque l'application requiert l'utilisation de la localisation de l'utilisateur, une demande d'autorisation lui est systématiquement présentée. Même après avoir donné son accord initial, l'utilisateur conserve le contrôle total sur sa confidentialité. À tout moment, il peut choisir de ne pas apparaître sur la carte en cochant simplement une option dans les paramètres de l'application. Cette fonctionnalité vise à respecter pleinement la vie privée de chacun.

Mise en ligne :

Configuration du serveur sur DigitalOcean

Lors du déploiement de l'application, j'ai rencontré divers obstacles. En effet, l'utilisation d'un serveur WebSocket au sein de l'application exige une configuration assez précise, ce que des hébergeurs d'applications tels que Heroku, auxquels j'avais l'habitude de recourir pour le déploiement de mes applications PHP, ne permettent pas.

Face à cette contrainte, j'ai dû rechercher une alternative offrant une personnalisation étendue du serveur. Après quelques recherches, j'ai découvert l'existence des VPS (Virtual Private Server), permettant une configuration presque illimitée du serveur. La plupart des fournisseurs de VPS ne proposent pas de solutions entièrement gratuites, mais le GitHub Student Pack fourni par mon établissement de formation possède une offre de 200\$ à l'année sur la plateforme DigitalOcean, qui héberge des VPS appelés « Droplets ».

Grâce à cette opportunité, j'ai pu héberger mon application sur un VPS en me connectant à la console du serveur via SSH, puis en important mon projet Git dans le répertoire `var/www/html/Hello-voisins`.

Voici quelques modifications que j'ai pu ensuite faire sur ce serveur :

Nom de domaine :

Tout d'abord afin de me connecter à ce serveur à distance sans être obligé d'utiliser l'adresse ip, je suis parti acheter le nom de domaine hello-voisins.com sur domains.squarespace.com. Puis j'ai connecté mon nom de domaine à mon serveur via les DNS et j'arrivais à accéder à mon serveur via mon nom de domaine.

Certificat SSL :

Afin de rendre mon site sécurisé j'ai activé le certificat SSL sur mon application en installant CertBot grâce à `sudo apt install certbot`.

Ensuite pour obtenir le certificat SSL il m'a juste fallu lancer la commande `sudo certbot --apache -d hello-voisins.com`

Pare feu :

Mon serveur WebSocket utilise le port 8888, donc j'ai dû ouvrir ce port et spécifier dans le pare-feu de permettre le trafic sur ce port.

Après avoir minutieusement configuré tous les paramètres requis, je croyais que mon application était prête à accueillir le serveur WebSocket. Ainsi, j'ai tenté la connexion au serveur WebSocket en utilisant la syntaxe suivante : « `var conn = new WebSocket('wss://hello-voisins.com')` », tout en ayant au préalable lancé le serveur via la commande « `php config/server.php` ». Malheureusement, la connexion refusait de s'établir.

Après plusieurs jours de recherche approfondie, ponctués de nombreux tests et une perplexité grandissante quant à l'origine du problème, j'ai finalement découvert une discussion sur StackOverflow abordant le **module Apache mod_proxy_wstunnel**.

Le protocole WebSocket a été conçu pour permettre une communication bidirectionnelle en temps réel entre un client, tel qu'un navigateur web, et un serveur. Cependant, certains serveurs intermédiaires peuvent ne pas prendre en charge directement ce protocole. C'est ici que le module `mod_proxy_wstunnel` d'Apache entre en jeu. Il offre la possibilité d'établir un « tunnel » entre le client WebSocket et le serveur WebSocket sous-jacent. Ce tunnel facilite le passage des données WebSocket à travers le serveur intermédiaire, agissant essentiellement comme un intermédiaire transparent, favorisant ainsi une communication fluide via WebSocket.

La résolution de mon problème a été relativement simple : il m'a suffi de suivre la documentation officielle d'Apache concernant ce module (https://httpd.apache.org/docs/2.4/mod/mod_proxy_wstunnel.html) pour le paramétrer correctement, et tout est désormais opérationnel.

Perspectives d'améliorations :

Le projet, bien que dans ses premières étapes de développement, présente déjà un socle solide. Après l'examen, je prévois de poursuivre le développement de l'application, ayant déjà accompli l'essentiel du travail.

Je suis conscient des perspectives d'amélioration qui s'offrent à l'application. Bien que réalisables, la contrainte de temps actuelle limite la mise en œuvre de ces améliorations.

Parmi les pistes d'amélioration étudiées, il est envisagé d'agrandir la base de données pour convenir à la croissance future de l'application. De plus, l'idée d'afficher des messages plus anciens lors du défilement vers le haut dans le chat, au-delà de la limite actuelle de 20 messages, serait un ajout utile.

Une autre amélioration utile serait l'implémentation d'un système d'envoi de mails pour la récupération de mot de passe en cas d'oubli via un service SMTP, ainsi que la mise en place de notifications de navigateur en utilisant l'API Notification pour informer les utilisateurs des nouveaux messages ou des événements importants dans l'application.

En outre, la création d'une fiche de profil individualisée pour chaque utilisateur est envisagée. Cette fonctionnalité permettrait une plus grande personnalisation de l'expérience utilisateur, contribuant ainsi à rendre l'application plus engageante et conviviale.

Bilan :

1. Compétences acquises

Ce projet a été une expérience extrêmement enrichissante qui a considérablement élargi ma compréhension du domaine du développement web. Il m'a permis d'explorer des technologies dont j'ignorais même l'existence, et j'ai pu mettre en pratique rapidement les connaissances nouvellement acquises. Un exemple concret en est l'implémentation d'un serveur WebSocket avec Ratchet.

La mise en ligne de mon application sur un serveur externe, grâce à la configuration du VPS fourni par DigitalOcean, a également été une étape cruciale de mon apprentissage. Cette expérience m'a offert une vision concrète du déploiement d'applications dans un environnement réel.

Par ailleurs, ce projet a été l'occasion pour moi de perfectionner ma maîtrise de JavaScript et PHP.

Ce projet m'a également enseigné l'art de travailler rapidement et efficacement, tout en maintenant une organisation rigoureuse. La nécessité de respecter un délai qui semblait initialement insurmontable a été un défi stimulant. Cette contrainte de temps m'a poussé à développer des compétences en gestion du temps et à adopter une approche méthodique pour atteindre mes objectifs.

2. Compétences à améliorer

Malgré ma satisfaction d'avoir concrétisé ce projet avec succès, je demeure conscient que des opportunités d'amélioration subsistent. Je suis déterminé à ne pas me reposer sur mes acquis et à poursuivre activement le développement de mes compétences. En particulier, je souhaite approfondir ma compréhension de JavaScript. Mon objectif est de l'utiliser de manière plus étendue, notamment en tant que langage Back-end grâce à Node.js, ce qui ouvrirait de nouvelles perspectives dans mes projets futurs.

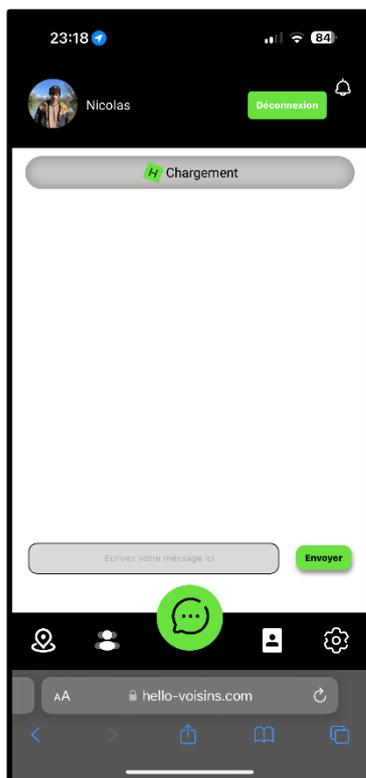
Annexes :



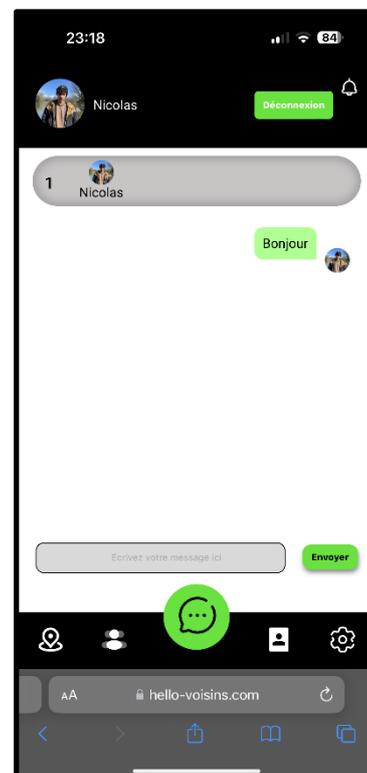
Page de connexion



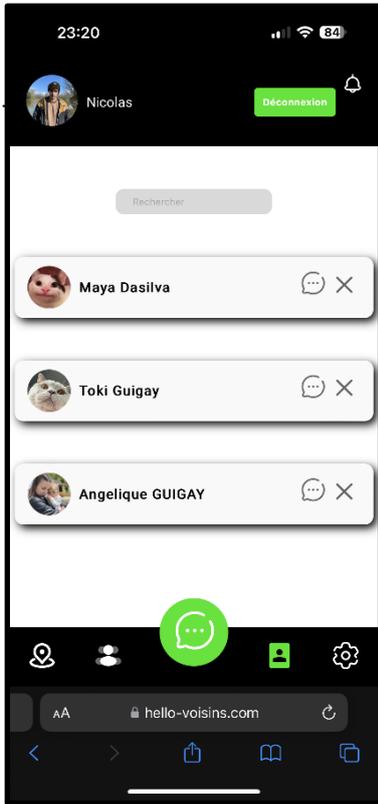
Page de création de compte



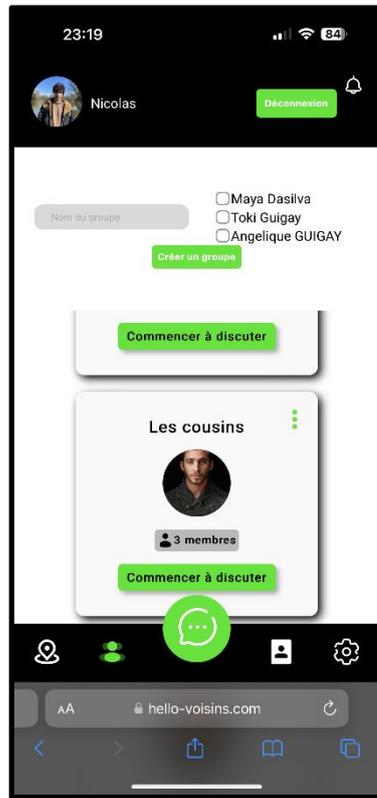
*Page du chat de proximité
(chargement)*



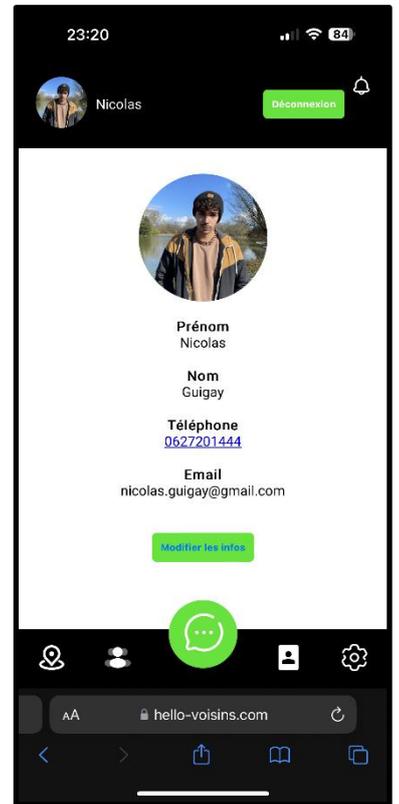
Page du chat de proximité



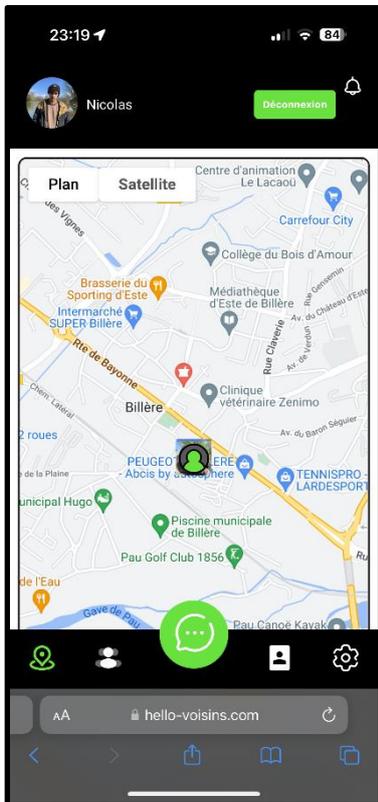
Page des contacts



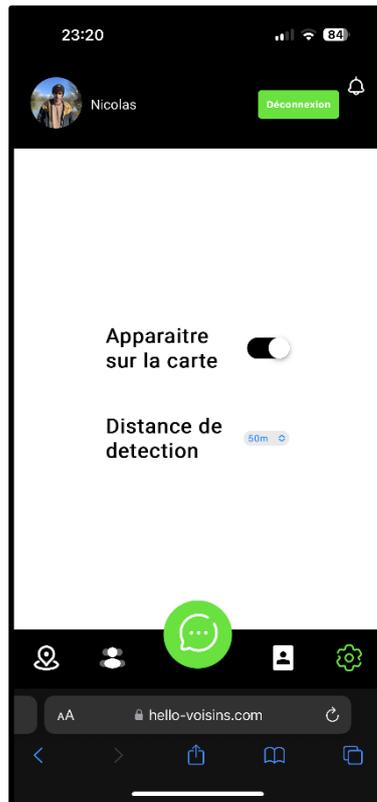
Page des groupes



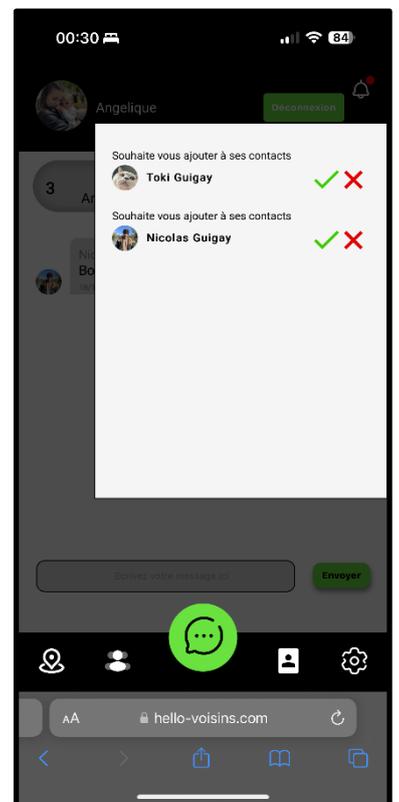
Page du profil personnel



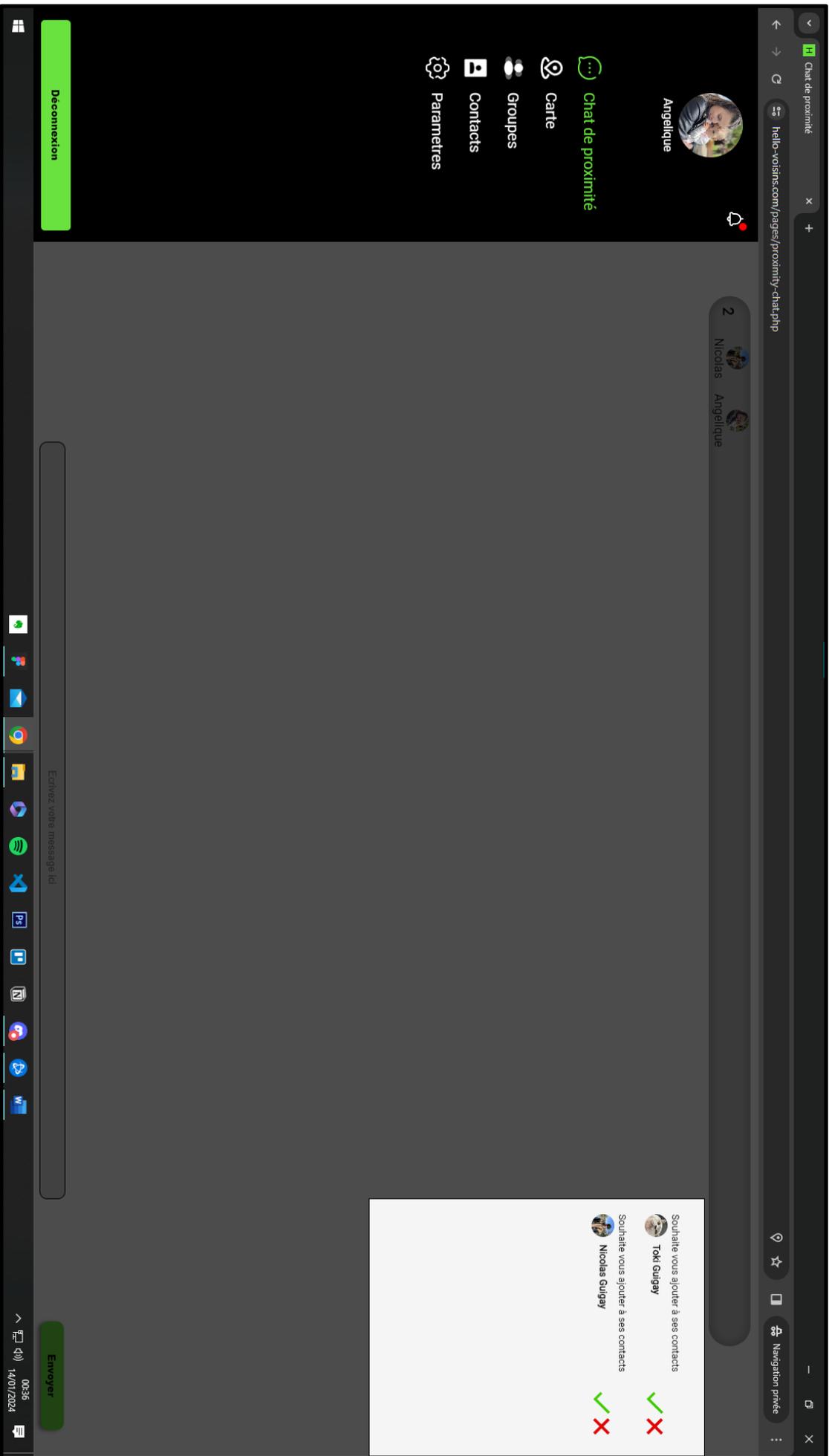
Page de la carte des utilisateurs



Page des paramètres



Notifications



Application sur Desktop