



Web service

Installation

Pour installer notre projet, il suffit de cloner le repository à l'adresse suivante:

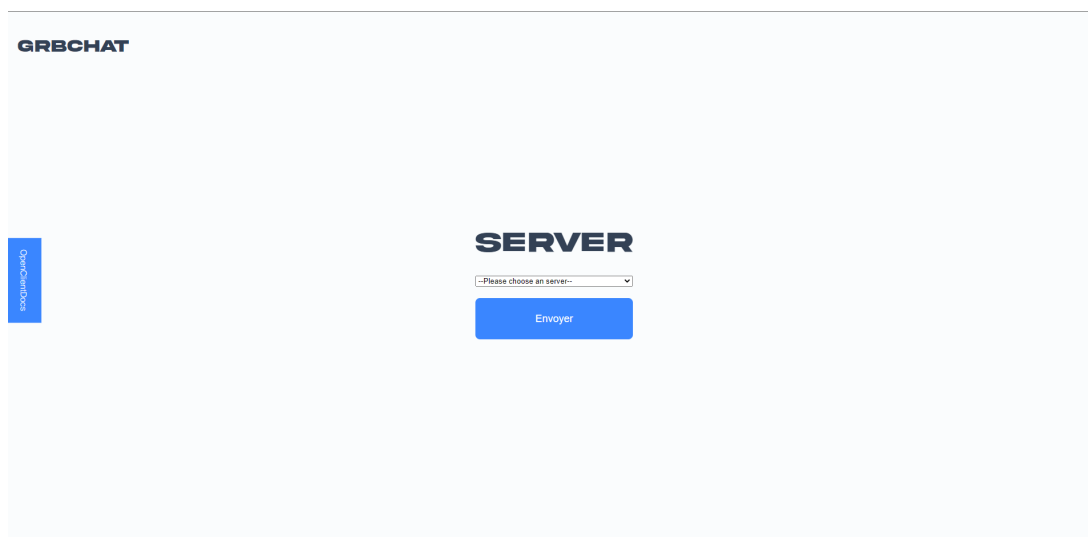
https://github.com/aspirecya/ynov_webservices

Après avoir cloner le repository, l'interface de l'outil se trouve dans le dossier **/front** et les différents serveurs se trouvent dans la racine du repo, pour les lancer, utilisez **node registry.js** pour le serveur de registre, et **node client.js** pour le serveur client

```
~/Libraries/ynov_webservice master ➤ clear  
~/Libraries/ynov_webservice master ➤ node registry.js  
Registry server running on port 1337  
—
```

```
~/Libraries/ynov_webservice master ➤ node client.js  
✓ Server 1 running on port 8005  
—
```

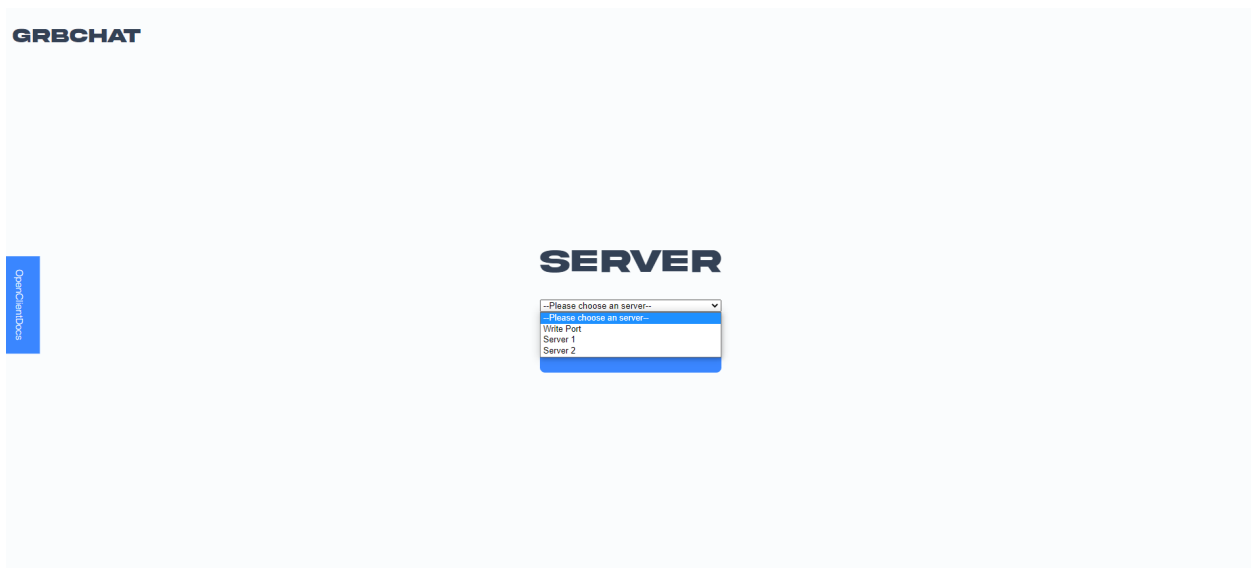
Ensuite, lancer l'installation des packages de l'interface avec la commande **npm install** – puis – **npm run dev** pour lancer l'interface.



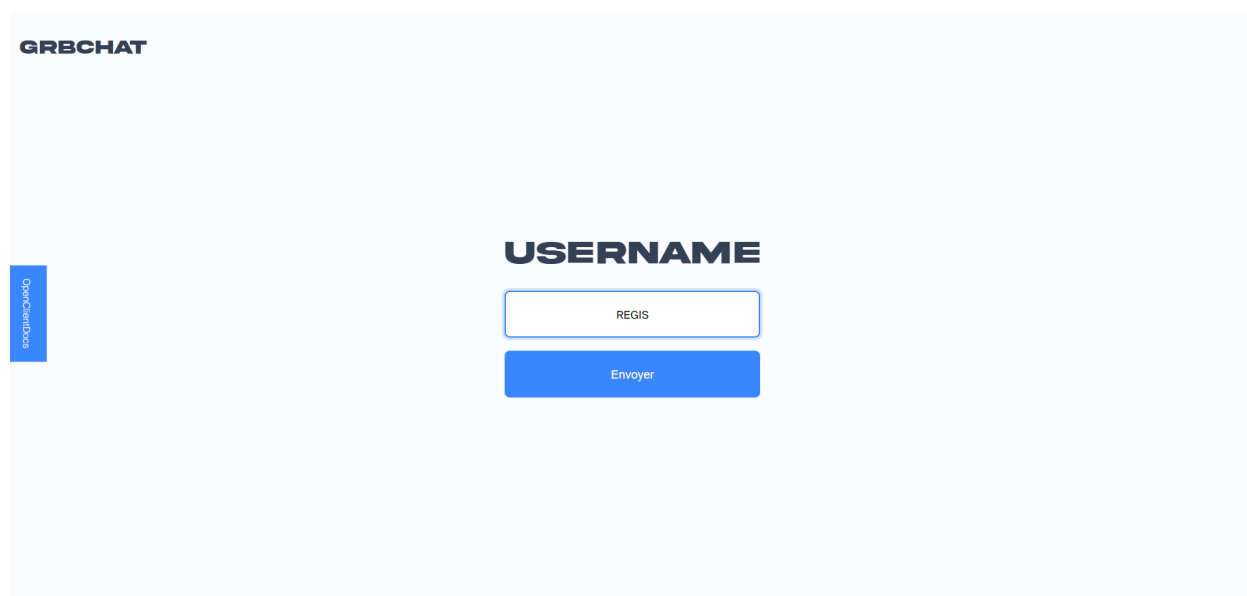
Notre application de chat à pour avantage d'être très simple d'utilisation et respectant les normes de normalisation

Pour commencer notre application vous demande sur quel serveur clients vous voulez vous connecter nous avons 3 options

1. écrire le port (si on veut utiliser un serveur client non enregistré)
2. server1 (l'application utilisera le serveur client port 8005)
3. server2 (l'application utilisera le serveur client 8692)

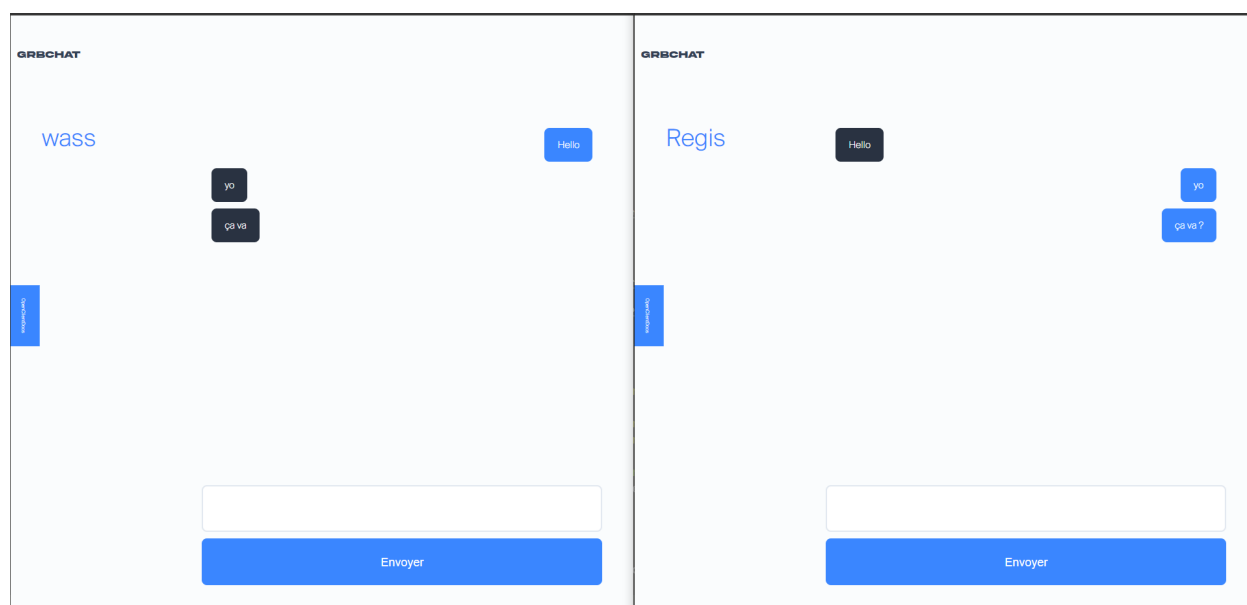


Après l'avoir renseigné le serveur client que vous voulez utiliser, vous pouvez enregistrer votre pseudo pour commencer le chat.



The screenshot shows the GRBCHAT registration interface. At the top left is the logo "GRBCHAT". On the left side, there is a vertical blue button labeled "OpenClientDocs". In the center, the word "USERNAME" is displayed in bold. Below it is a text input field containing the text "REGIS". Underneath the input field is a blue button labeled "Envoyer".

Pour pouvoir tester si le chat fonctionne vous pouvez ouvrir une nouvelle fenêtre du front et renseigner un autre serveur client et vous envoyez un message



The image shows two side-by-side screenshots of the GRBCHAT chat interface. Both windows have the "GRBCHAT" logo at the top left and a vertical blue button labeled "OpenClientDocs" on the left side. The left window shows a chat log with the message "wass" in blue, followed by two dark grey messages "yo" and "ça va". At the bottom, there is a text input field and a blue button labeled "Envoyer". The right window shows a chat log with the message "Regis" in blue, followed by two dark grey messages "Hello" and "yo", and then a blue message "ça va ?". At the bottom, there is a text input field and a blue button labeled "Envoyer".

SERVEUR DE REGISTRE

Pour la conception du serveur de registre on a créé des mutators pour chaque information stockée dans le registre.

Nous n'avons qu'un tableau de users pour savoir si un utilisateur est connecté ou pas nous avons donc les fonctions suivant:

1. sendRegistryUser (pour récupérer un ou tous les users actuellement connectés)
2. storeRegistryUser (pour enregistrer un utilisateur et le mettre dans le tableau)
3. deleteRegistryUser (pour supprimer un utilisateur dans le tableau)

Sachant que tous les utilisateurs ne sont pas persistants nous n'avons pas eu le besoin d'avoir une fonction qui supprime tous les utilisateurs dans le tableau

Pour savoir si les différents clients des utilisateurs sont encore en marche, il a été décidé d'utiliser une fonction de ping qui envoie une requête toutes les minutes aux différents clients des utilisateurs connectés dans le tableau et si aucune réponse n'est reçue de la part du client, celui-ci est déconnecté et supprimé du registre.

SERVEURS CLIENTS

Nos serveurs clients ont les mêmes fonctionnalités :

1. ils appellent le registre pour enregistrer supprimer et récupérer les utilisateurs
2. ils renvoient une réponse au serveur de registre, lorsqu'ils sont une requête /ping
3. et enfin ils ont des mutators pour enregistrer et supprimer les messages en fonction du nom d'utilisateurs qui est associé

INTERFACE

Pour l'interface clients nous avons choisi d'utiliser Nextjs car nous avons déjà différents composants nous permettant de faire une interface de chat et dans un contexte professionnel nous apporte les avantages d'une interface flexible ainsi que l'optimisation d'image de seo etc...

Notre interface clients à pour avantage de demander au début sur quel serveur client l'utilisateur veut se connecter comme cela sur une machine locale avec l'interface nous pouvons interagir entre les différents serveurs clients sans faire de manipulation supplémentaire.

Notre interface n'a que 4 requête possible au serveur client:

Login pour enregistrer l'utilisateur .

GetUsers pour afficher les utilisateurs connecté avec qui nous pouvons échanger

GetMessage/[nom de l'utilisateur] pour récupérer les messages qu'un utilisateur nous a envoyés.

Logout pour déconnecter l'utilisateur.