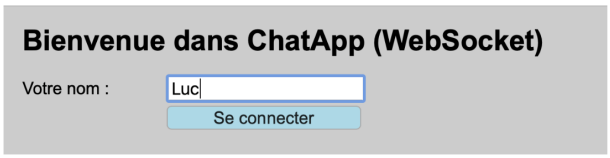
TP7 – Implémentation d'une application de messagerie instantanée

Avant-Propos

L'objectif de ce TP est de mettre en œuvre une application web de messagerie instantanée.

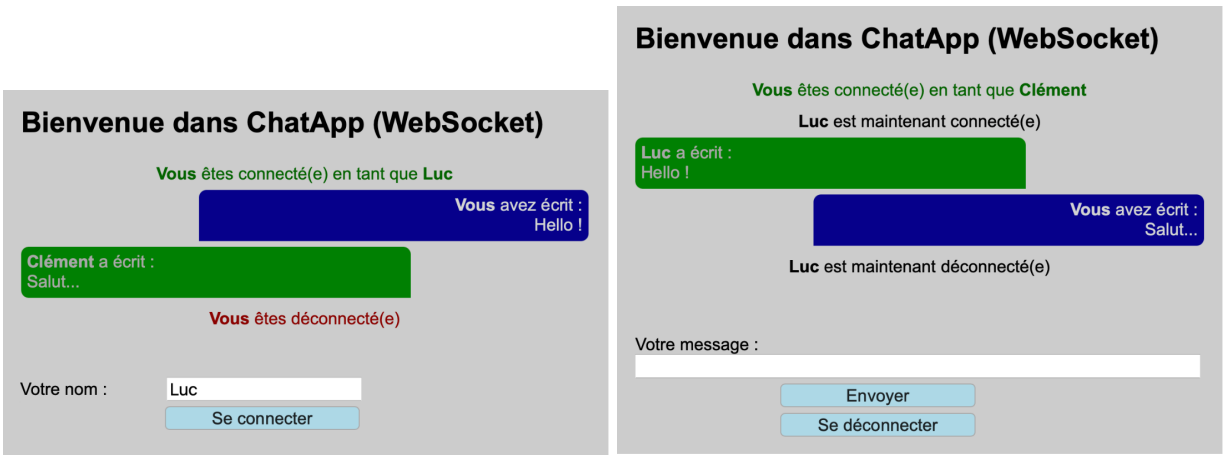
1. Un utilisateur arrive sur une fenêtre d'accueil avec un champ et un bouton pour se connecter :



2. Lorsqu'il se connecte, il est notifié de sa connexion par « **Vous** êtes connecté(e) en tant que … » en vert tandis que les autres utilisateurs le sont simultanément par « **…** est maintenant connecté(e) ». La fenêtre d'accueil laisse alors la place à une fenêtre avec un champ et un premier bouton pour envoyer un message et un second bouton pour se déconnecter.

3. Lorsqu'il envoie un message, son message précédé de « **Vous** avez écrit : » apparait dans une bulle bleue sur la droite tandis que les autres utilisateurs reçoivent simultanément le message précédé de « **…** a écrit : » dans une bulle verte sur la gauche.

4. Lorsqu'il se déconnecte, il est notifié de sa déconnexion par « **Vous** êtes déconnecté(e) » en rouge tandis que les autres le sont simultanément par « **…** est maintenant déconnecté(e) ». Par ailleurs, le champ et le bouton pour se connecter reprennent leur place sous la conversation.



L'aspect temps réel de la messagerie instantanée sera assuré par l'utilisation des **API WebSocket**.

Exercice 1 : Côté Client

Dans cette première partie, vous allez devoir implémenter la partie graphique de l'application.

Vous conserverez les fichiers :

- **`index.html`** : contient tous les éléments interactifs, notamment :
  - la **`<div>`** `#chat` qui héberge la conversation ;
  - la **`<div>`** `#loginscreen` qui héberge le champ texte #name et le bouton **`[Se connecter]`** ;
  - la **`<div>`** `#chatscreen` qui héberge le champ texte #message, le bouton **`[Envoyer]`** et le bouton **`[Se déconnecter]`** ;
- **`styles/chatapp.css`** : contient tous les styles, notamment :
  - la classe `.logged` de notification de connexion à l'utilisateur ;
  - la classe `.unlogged` de notification de déconnexion à l'utilisateur ;

- la classe `.event` de notification de connexion/déconnexion aux autres utilisateurs ;
- la classe `.received` des messages alignés à gauche reçu des autres utilisateurs ;
- la classe `.sent` des messages alignés à droite envoyé par l'utilisateur.

du dossier **/workspace/31-WebMVC/www** du **conteneur Docker `r401-tomcat-container`**.

Vous allez devoir finaliser le fichier :

- **`scripts/appchat.js`** qui contient la partie applicative, notamment
  La gestion des interactions utilisateur :
  - la fonction **`onLogin()`** en réponse à l'appui sur **`[Se connecter]`** ;
  - la fonction **`onLogout()`** en réponse à l'appui sur **`[Se déconnecter]`** ;
  - la fonction **`onSend()`** en réponse à l'appui sur **`[Envoyer]`** ;

  La gestion des événements de l'**API WebSocket (JavaScript)** :
  - la fonction **`whenLogged()`** en réponse à un événement open d'ouverture d'une connexion ;
  - la fonction **`whenUnlogged()`** en réponse à un événement close de clôture d'une connexion ;
  - la fonction **`whenReceived()`** en réponse à un événement message de réception de message ;

  Les variables initialisées à **`null`** et instantiées à la demande de connexion :
  - la variable **`socket`** qui contiendra l'objet **WebSocket** ;
  - la variable **`username`** qui contiendra le nom de l'utilisateur.

dans le dossier **/workspace/31-WebChat/www** du **conteneur Docker `r401-tomcat-container`**.

Le format des **messages JSON** est décrit en commentaires et dans l'exercice suivant.
Les binaires universels Tomcat sont fournis.

Pour compiler et déployer votre code, dans le **terminal Docker de `r401-tomcat-container`**, lancez :

- `deploy.sh 31-WebChat`

Pour tester votre code, dans un **navigateur**, ouvrez :

- http://localhost:8081/31-WebChat

Exercice 2 : Coté Serveur

Dans cette seconde partie, vous allez devoir implémenter la partie serveur de l'application.

Elle est constituée d'un seul fichier :

- **`MyWebSocket.java`** : contient la classe qui encapsule le **websocket** :

- la variable d'instance **`m_sessions`** représente la map des sessions connectées à la socket et qui doit être mise à jour à chaque connexion/déconnexion ;
- la méthode **`onOpen()`** en réponse à un événement **`@OnOpen`** d'ouverture d'une connexion envoie à toutes les autres sessions **un message JSON** sous la forme suivante :

  { **what** : "logged", **who** : "*{username}*" }

  où *{username}* représente le nom associé à la session utilisateur qui demande l'ouverture ;
- la méthode **`onClose()`** en réponse à un événement **`@OnClose`** de clôture d'une connexion envoie à toutes les autres sessions **un message JSON** sous la forme suivante :

  { **what** : "unlogged", **who** : "*{username}*" }

  où *{username}* représente le nom associé à la session utilisateur qui demande la clôture ;
- la méthode **`onMessage()`** en réponse à un événement **`@OnMessage`** de réception de message envoie à toutes les autres sessions **un message JSON** sous la forme suivante :

  { **what** : "message", **who** : *"{username}"*, **content** : "*{content}*" }

  où *{username}* représente le nom associé à la session utilisateur qui envoie le message et *{content}* représente le contenu du message.

dans le dossier **/workspace/32-WebChat/src** du **conteneur Docker `r401-tomcat-container`**.

Vous copierez le fichier de l'exercice précédent :

- **`scripts/appchat.js`**

dans le dossier **/workspace/31-WebMVC/www** du **conteneur Docker `r401-tomcat-container`**.

Pour compiler et déployer votre code, dans le **terminal Docker de `r401-tomcat-container`**, lancez :

- `deploy.sh 32-WebChat`

Pour tester votre code, dans un **navigateur**, ouvrez :

- http://localhost:8081/32-WebChat