

# TP1 - Angular

Vous utiliserez Angular pour bâtir une application monopage de messagerie simple.

## Prérequis

(Voir aussi <https://angular.io/guide/setup-local>)

Vous aurez besoin d'installer node.js - <https://nodejs.org/>.

Ensuite, vous devez installer l'outil de ligne de commande d'Angular en utilisant

```
npm install -g @angular/cli
```

Comme éditeur de code, nous vous conseillons Visual Studio Code (VSCode) - <https://code.visualstudio.com/>

Deux extensions pour Visual Studio Code sont suggérées:

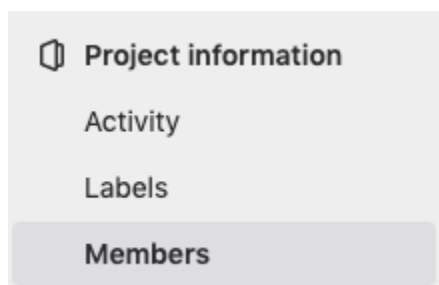
- Angular Language Service
- Prettier - Code Formatter

Selon vos préférences, vous pouvez configurer Visual Studio Code (dans les paramètres) pour que Prettier soit utilisé par défaut et qu'il s'exécute automatiquement lorsque vous sauvegardez un fichier (Preferences -> Settings -> Text Editor / Formatting / Format on save).

## Base de code

Avant de débiter, vous devrez faire un 'fork' privé du dépôt [https://gitlab.info.ugam.ca/trepanie\\_fel/tp1-base](https://gitlab.info.ugam.ca/trepanie_fel/tp1-base)

Ajoutez @trepanie\_fel comme contributeur.



Ajoutez vos noms et matricules dans le fichier *equipe.txt*.

Une fois le dépôt cloné sur votre machine, exécutez

```
npm install
```

et ensuite

```
ng serve
```

Vous devriez pouvoir vous connecter à <http://127.0.0.1:4200/> pour voir l'application.

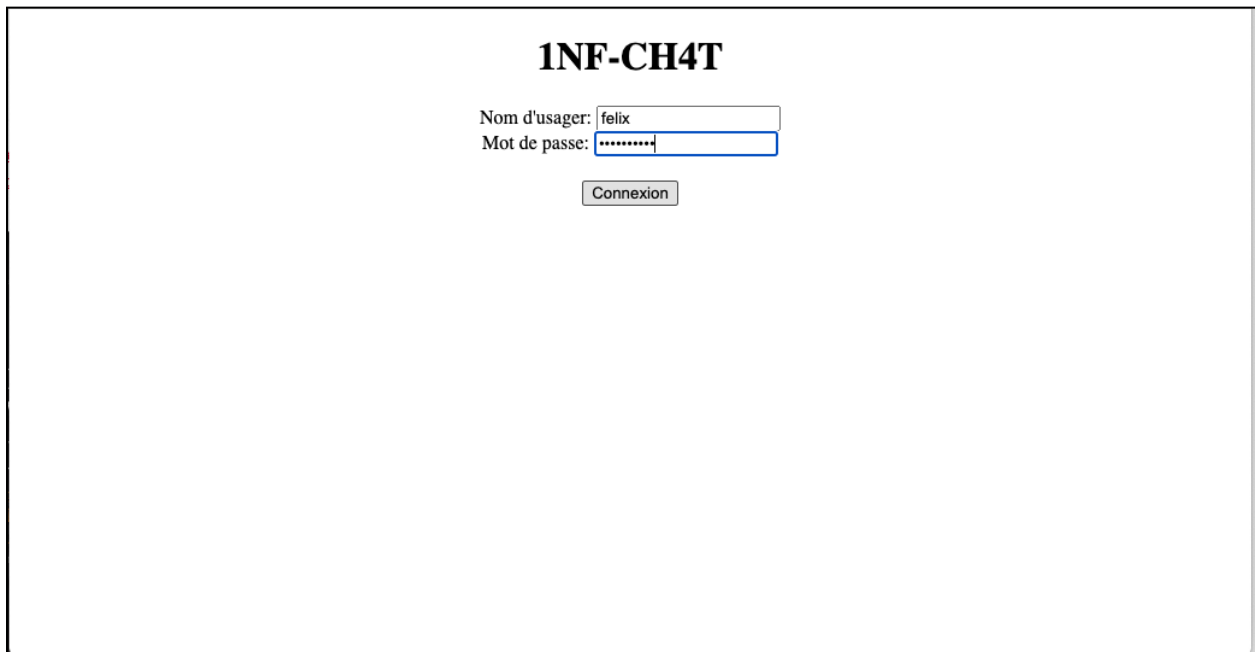
Laissez la commande `ng serve` rouler, elle mettra automatiquement à jour l'application lorsque vous effectuerez des changements.

La base de code est incomplète. Ce premier TP consiste à la compléter et l'améliorer.

## Objectif

À la fin de cet exercice, vous devriez avoir l'application décrite ci-dessous.

L'application contient deux pages. La première vous permet de vous connecter à la messagerie. Comme l'authentification des usagers n'est pas encore implémentée, cette première version acceptera toutes combinaisons de nom d'utilisateur et de mot de passe. Une fois la connexion complétée, l'utilisateur est redirigé vers la page de messagerie.



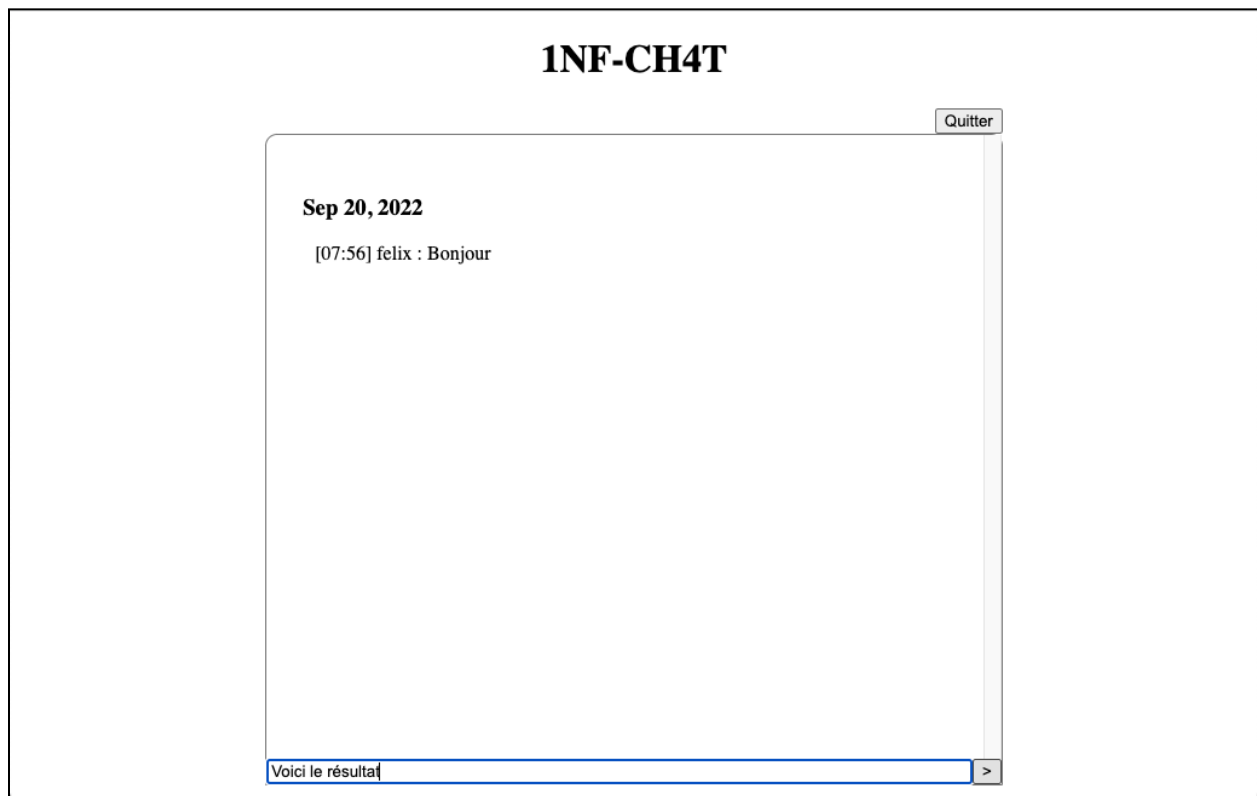
**1NF-CH4T**

Nom d'utilisateur:

Mot de passe:

*Page de connexion*

Sur la page de messagerie, il est possible d'envoyer des messages qui sont automatiquement affichés. Le bouton *Quitter* permet de se déconnecter et de revenir à la page de connexion.



*Page de messagerie*

Les messages sont stockés en mémoire pour cette première version. Donc en rafraîchissant la page, la messagerie redevient vide.

Par contre, le nom de l'utilisateur est stocké dans le localStorage donc il est possible de continuer à envoyer des messages avec le même nom d'utilisateur sans se reconnecter même après avoir rechargé la page.

Pour compléter l'application, vous devrez suivre les étapes décrites dans les sections suivantes.

Prenez le temps de vous familiariser un peu avec la base de code avant de faire les modifications.

## Partie 1 - Page de connexion

### Étape A - LoginFormComponent - onLogin

Lorsque l'utilisateur clique sur le bouton **connexion**, la méthode `onLogin` est appelée.

Écrire le code de la méthode `onLogin` du composant `LoginFormComponent`. Ce composant de présentation doit émettre un événement en utilisant le `EventEmitter login` pour permettre au `LoginPageComponent` (composant intelligent) de gérer la connexion de l'utilisateur.

## Étape B - LoginPageComponent - onLogin

Le composant `LoginPageComponent` reçoit maintenant les informations nécessaires pour la connexion de l'utilisateur. Il doit utiliser le `LoginService` pour effectuer la connexion.

Pour avoir accès au `LoginService` dans le composant, injecter le dans le constructeur du `LoginPageComponent`. Écrire le code de la méthode `onLogin` du `LoginPageComponent`.

## Étape C - LoginService - login

Lors de sa construction, ce service lit le nom d'utilisateur à partir du `localStorage`. Ceci permet à l'utilisateur de rafraîchir la page sans devoir se reconnecter.

Le service est aussi responsable d'effectuer la connexion et de stocker le nom d'utilisateur dans le `localStorage` au moment de la connexion.

Écrire le code de la méthode `login` pour stocker le nom d'utilisateur dans le `localStorage` et mettre à jour le nom de l'utilisateur connecté en utilisant la méthode `next` sur le `BehaviorSubject username`.

En réalité, il y aurait un appel serveur pour valider les informations de l'utilisateur, mais pour ce TP, nous supposons que les informations sont toujours valides.

## Étape D - LoginService - logout

Écrire le code de la méthode `logout`.

## Étape E - Nouvelle Route

Une fois la connexion effectuée, l'utilisateur devrait être redirigé vers la page de messagerie.

Ajouter une nouvelle route dans `AppRoutingModule` pour que le chemin d'accès <http://127.0.0.1:4200/chat> affiche le `ChatPageComponent`.

*Note: Les routes sont évaluées dans l'ordre dans lequel elles sont définies.*

## Étape F - Redirection

Déterminer l'endroit approprié dans le code pour rediriger l'utilisateur, à la fin du processus de connexion, vers la page de messagerie (`/chat`) en utilisant le `Router` d'Angular et sa méthode `navigate`.

## Partie 2 - Messagerie

L'utilisateur devrait maintenant pouvoir voir la page de messagerie après sa connexion.

### Étape A - `MessageService` - `postMessage`

Le `ChatPageComponent` utilise la méthode `postMessage` lorsqu'un nouveau message est envoyé.

Écrire le code de la méthode `postMessage` du `MessageService` pour qu'elle ajoute le nouveau message et émette un nouveau tableau de messages.

*Note: On peut obtenir la valeur courante d'un `BehaviorSubject` en utilisant son attribut `value`.*

### Étape B - Afficher les messages

Le HTML du `ChatPageComponent` utilise l'attribut `messages` pour afficher les messages. Cet attribut est initialisé en tant que tableau vide et n'est jamais mis à jour pour le moment.

Utiliser l'Observable `messages$` pour mettre à jour l'attribut `messages` lorsqu'un nouveau tableau de messages est émis.

### Étape C - Quitter

Écrire le code pour la méthode `onQuit` du `ChatPageComponent` pour effectuer une déconnexion (logout) et rediriger l'utilisateur vers la page de connexion

## Partie 3 - Améliorations

Le composant `ChatPageComponent` gère l'affichage et fait appel aux services. Essayons de simplifier ce composant en créant deux composants de présentation.

## Étape A - Création d'un composant qui affiche les messages

Extraire un composant de présentation dont le rôle est seulement d'afficher les messages.

*Note: Utiliser le CLI d'Angular pour générer le nouveau composant (e.g. `ng g c chat/messages`).*

## Étape B - Création d'un composant qui permet de publier un message

Extraire un composant de présentation dont le rôle est seulement de publier un nouveau message.

*Note: Utiliser le CLI d'Angular pour générer le nouveau composant (e.g. `ng g c chat/new-message-form`).*

## Étape C - Une dernière chose...

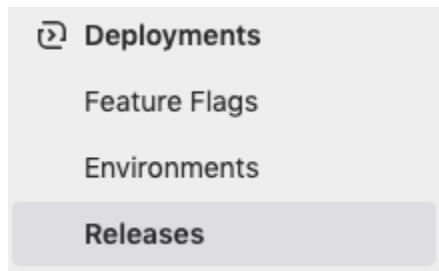
Tester le cas où le nombre de messages dépasse la taille du composant d'affichage de messages.

Essayer de faire en sorte que le dernier message soit visible automatiquement après sa publication (sans avoir à utiliser la barre de défilement).

## Remise

La date de remise pour ce TP est le 7 octobre à 21h.

Avant cette date limite, vous devrez créer une nouvelle distribution (release) avec le nom *tp1* le tag *tp1*.



## Pondération

Pour la partie 1, chaque étape complétée donne 1 pt pour un total de 6 pts.

Pour la partie 2, chaque étape complétée donne 1 pt pour un total de 3 pts.

Pour la partie 3, chaque étape complétée donne 2 pts pour un total de 6 pts.

La qualité du code (lisibilité, structure, efficacité) vaut 5 pts pour un total de 20 points.