

SEG3502 – Labo 1

Introduction à Angular

L'objectif de ce laboratoire est d'introduire le développement d'applications avec le framework Angular. Le laboratoire traite de l'installation des outils nécessaires pour Angular et du développement d'une application simple de type «hello world».

Le code source du laboratoire est disponible dans le référentiel Github
<https://github.com/stephanesome/angularIntro.git>.

Installation

Node.js

Angular nécessite **Node.js**. Téléchargez et installez la dernière version à partir de <https://nodejs.org/en/>.

Vérifiez que **Node.js** est correctement installé en exécutant la commande `node -v` dans un terminal.

Le gestionnaire de packages **npm** est également nécessaire. Il devrait être installé avec **Node.js**.

Vérifiez que **npm** est correctement installé en exécutant la commande `npm -v` dans un terminal.

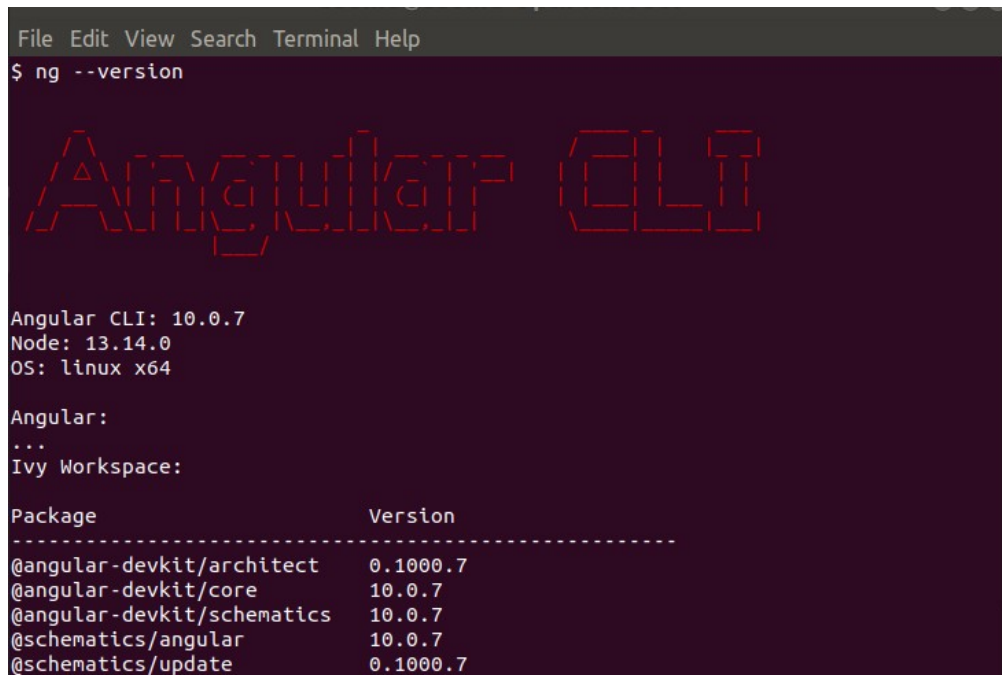
Angular CLI

[Angular CLI](#) est utilisée pour créer, tester, regrouper et déployer des projets Angular.

Installez la dernière version de Angular CLI avec la commande

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

Vérifiez l'installation en exécutant la commande `ng --version`. Vous devriez voir une sortie similaire à la suivante.



```
File Edit View Search Terminal Help
$ ng --version

Angular CLI
Angular CLI: 10.0.7
Node: 13.14.0
OS: linux x64

Angular:
...
Ivy Workspace:

Package                                  Version
-----
@angular-devkit/architect               0.1000.7
@angular-devkit/core                    10.0.7
@angular-devkit/schematics              10.0.7
@schematics/angular                    10.0.7
@schematics/update                      0.1000.7
```

Environnement de développement intégré (IDE)

Un IDE est facultatif, mais fortement recommandé pour du développement Angular. Plusieurs IDEs peuvent être utilisés, notamment [IntelliJ IDEA](#), [WebStorm](#), [Visual Studio Code](#), [Eclipse](#), [Atom](#). Assurez-vous de configurer ces outils avec les plugins appropriés pour Angular.

Typescript

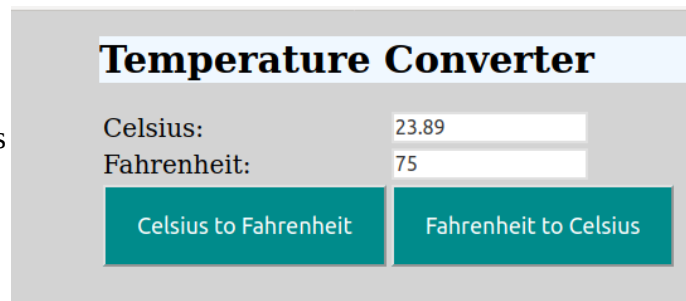
Angular utilise **Typescript**, un sur-ensemble open-source de ES6 développé par Microsoft. **TypeScript** intègre les éléments introduits dans ES6 telles que les classes, les modules, les fonctions Fat Arrow (https://www.w3schools.com/js/js_es6.asp). Il introduit des éléments supplémentaires, notamment une typage fort et des décorateurs. Les types suivants sont supportés: *String*, *Number*, *Boolean*, *Array*, *Enums*, *Any*, *Void*. *Any* est un type par défaut lorsque le type d'une variable est omis, tandis que *Void* est utilisé comme type retourné pour les fonctions sans valeur de retour.

Consultez <https://www.typescriptlang.org/docs> pour des détails concernant **Typescript**.

Exemple de convertisseur de température

Nous allons implémenter une application Angular simple pour la conversion de température Celsius - Fahrenheit.

L'application permettra à la fois de convertir de Celsius en Fahrenheit et de Fahrenheit en Celsius. Dans chaque cas, l'utilisateur doit entrer une valeur à convertir dans un champ de texte et cliquer sur un bouton pour la conversion. L'application affichera alors la valeur convertie dans l'autre champ.



The screenshot shows a web application titled "Temperature Converter". It has two input fields: "Celsius:" with the value "23.89" and "Fahrenheit:" with the value "75". Below the inputs are two buttons: "Celsius to Fahrenheit" and "Fahrenheit to Celsius".

Créer une Application

Entrez la commande `ng new lab1-temp-converter`, répondez N (non) à l'ajout du routage et conservez

CSS comme
format de
style.

```
File Edit View Search Terminal Help
$ ng new lab1-temp-converter
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE lab1-temp-converter/README.md (1035 bytes)
CREATE lab1-temp-converter/.editorconfig (274 bytes)
CREATE lab1-temp-converter/.gitignore (631 bytes)
CREATE lab1-temp-converter/angular.json (3670 bytes)
CREATE lab1-temp-converter/package.json (1271 bytes)
CREATE lab1-temp-converter/tsconfig.base.json (458 bytes)
CREATE lab1-temp-converter/tsconfig.json (426 bytes)
CREATE lab1-temp-converter/tslint.json (3184 bytes)
CREATE lab1-temp-converter/.browserslistrc (853 bytes)
CREATE lab1-temp-converter/karma.conf.js (1031 bytes)
CREATE lab1-temp-converter/tsconfig.app.json (292 bytes)
CREATE lab1-temp-converter/tsconfig.spec.json (338 bytes)
CREATE lab1-temp-converter/src/favicon.ico (948 bytes)
CREATE lab1-temp-converter/src/index.html (303 bytes)
CREATE lab1-temp-converter/src/main.ts (372 bytes)
```

Angular CLI générera la structure de répertoires et les fichiers de configuration nécessaires pour l'application Angular.

Vous pouvez exécuter l'application générée en entrant la commande `ng serve` dans le répertoire racine du projet.

```
File Edit View Search Terminal Help
$ cd lab1-temp-converter/
$ ng serve
Compiling @angular/compiler/testing : es2015 as esm2015
Compiling @angular/core : es2015 as esm2015
Compiling @angular/animations : es2015 as esm2015
Compiling @angular/common : es2015 as esm2015
Compiling @angular/animations/browser : es2015 as esm2015
Compiling @angular/core/testing : es2015 as esm2015
Compiling @angular/common/http : es2015 as esm2015
Compiling @angular/platform-browser : es2015 as esm2015
Compiling @angular/animations/browser/testing : es2015 as esm2015
Compiling @angular/platform-browser-dynamic : es2015 as esm2015
Compiling @angular/platform-browser/testing : es2015 as esm2015
Compiling @angular/platform-browser/animations : es2015 as esm2015
Compiling @angular/forms : es2015 as esm2015
Compiling @angular/common/testing : es2015 as esm2015
Compiling @angular/router : es2015 as esm2015
Compiling @angular/common/http/testing : es2015 as esm2015
Compiling @angular/platform-browser-dynamic/testing : es2015 as esm2015
Compiling @angular/router/testing : es2015 as esm2015

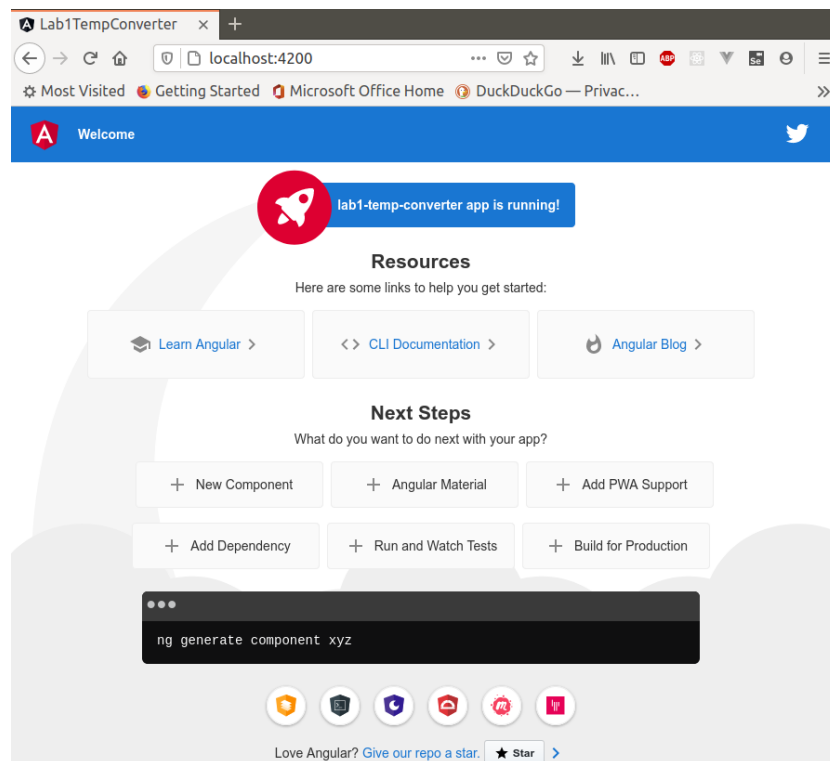
chunk {main} main.js, main.js.map (main) 57.8 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 12.4 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 2.42 MB [initial] [rendered]
Date: 2020-08-27T00:37:30.750Z - Hash: b002f672de3376fb4ebc - Time: 10163ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
: Compiled successfully.
```

Angular CLI intègre un serveur de développement et déploie l'application à l'URL

<http://localhost:4200>.

Accédez à l'URL

<http://localhost:4200> pour voir l'application.



L'application générée fournit des liens vers des tutoriels et de la documentation sur Angular.

Ouvrir l'application dans l'IDE

J'utiliserai IntelliJ IDEA pour développer l'application. Bien que nous puissions développer nos applications à l'aide d'un simple traitement de texte, l'utilisation d'un IDE avec des fonctionnalités telles que la complétion de code ou la vérification des erreurs rend le travail plus productif.

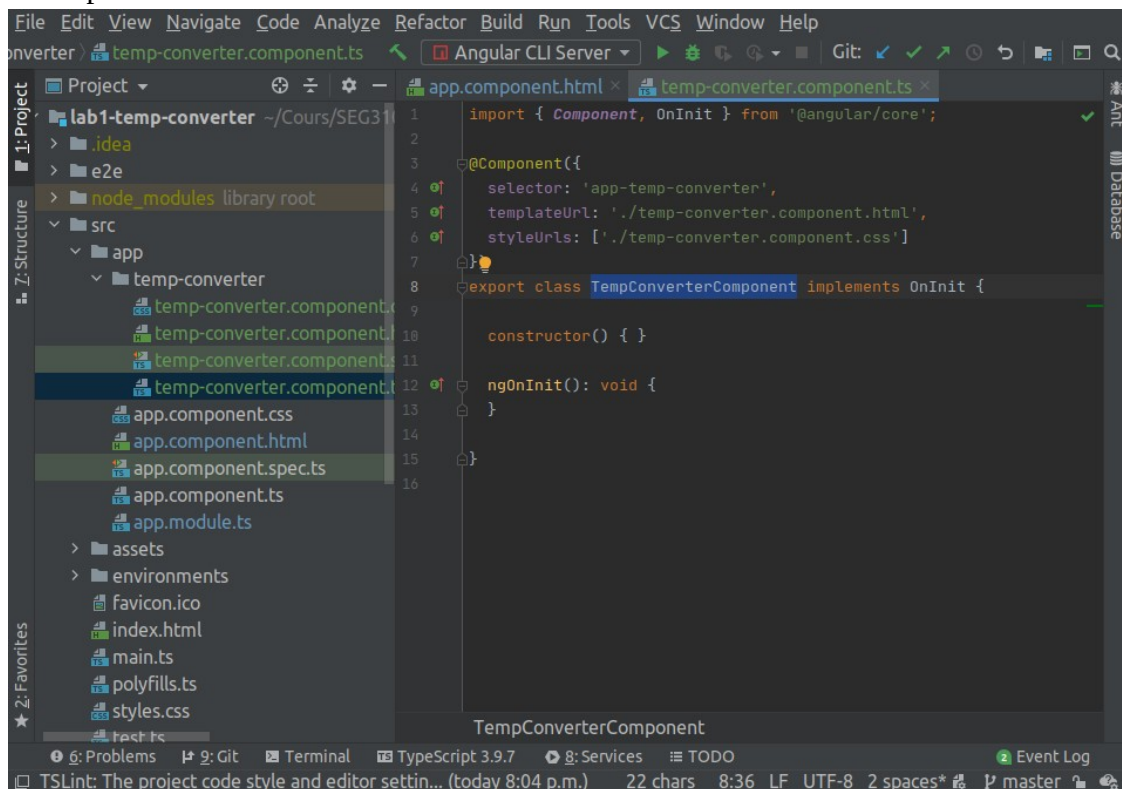
- **File -> Open** puis, parcourez pour sélectionner le répertoire racine du projet.

Créer un composant de conversion de température

- Créer un composant avec la commande `ng generate component temp-converter`.

```
File Edit View Search Terminal Help
$ ng generate component temp-converter
CREATE src/app/temp-converter/temp-converter.component.css (0 bytes)
CREATE src/app/temp-converter/temp-converter.component.html (29 bytes)
CREATE src/app/temp-converter/temp-converter.component.spec.ts (678 bytes)
CREATE src/app/temp-converter/temp-converter.component.ts (306 bytes)
UPDATE src/app/app.module.ts (426 bytes)
$
```

La commande génère le composant `TempConverter` que nous utiliserons pour la conversion de température.



Configuration du composant Temperature Converter

- Configurer le modèle HTML du composant `TempConverter`. Modifiez `src/app/temp-converter/temp-converter.component.html` pour qu'il soit comme suit.

```
1. <table>
2.   <tr>
3.     <td>
4.       <label for="celsius">Celsius:</label>
5.     </td>
6.     <td>
7.       <input name="celsius" id="celsius" value="{{celsiusValue | number: '1.0-2'}}" #celsius>
8.     </td>
9.   </tr>
10.  <tr>
11.    <td>
12.      <label for="fahrenheit">Fahrenheit:</label>
13.    </td>
14.    <td>
15.      <input name="fahrenheit" id="fahrenheit" value="{{fahrenheitValue | number: '1.0-2'}}" #fahrenheit>
16.    </td>
17.  </tr>
18.  <tr>
19.    <td>
20.      <button (click)="convertCelsius(celsius.value)">Celsius to Fahrenheit</button>
21.    </td>
22.    <td>
23.      <button (click)="convertFahrenheit(fahrenheit.value)">Fahrenheit to Celsius</button>
24.    </td>
25.  </tr>
26. </table>
```

Les valeurs des éléments `input` aux lignes 7 et 15 sont liées aux propriétés de la classe de composant à l'aide d'une interpolation de chaîne (string interpolation). Nous utilisons le pipe `number` pour afficher les valeurs avec 2 décimales maximum.

Les éléments `input` sont respectivement associés aux variables de référence de modèle `#celsius` et `#fahrenheit`. Ces variables de référence de modèle sont utilisées pour transmettre les valeurs d'entrée aux gestionnaires d'événements de clic de boutons aux lignes 20 et 23.

- Configurez la classe `TempConverterComponent`. Editez `src/app/temp-converter/temp-converter.component.ts` afin qu'elle soit comme suit.

```
1. import { Component } from '@angular/core';
2.
3. @Component({
4.   selector: 'app-temp-converter',
5.   templateUrl: './temp-converter.component.html',
```

```

6.   styleUrls: ['./temp-converter.component.css']
7. })
8. export class TempConverterComponent {
9.   celsiusValue = 0;
10.  fahrenheitValue = 0;
11.  constructor() { }
12.
13.  convertCelsius(value: string): void {
14.    this.celsiusValue = Number(value);
15.    this.fahrenheitValue = ((this.celsiusValue * 9) / 5) + 32;
16.  }
17.
18.  convertFahrenheit(value: string): void {
19.    this.fahrenheitValue = Number(value);
20.    this.celsiusValue = ((this.fahrenheitValue - 32) * 5) / 9;
21.  }
22. }

```

Le décorateur spécifie les métadonnées pour le composant. La propriété `selector` de la ligne 4 définit la balise du composant, à utiliser dans les modèles HTML pour inclure le composant.

Les propriétés `templateUrl` à la ligne 5 et `styleUrls` à la ligne 6 spécifient les fichiers pour le modèle HTML du composant et les feuilles de style.

La classe `TempConverterComponent` définit deux propriétés `celsiusValue` et `fahrenheitValue` qui transmettent des valeurs au modèle HTML par interpolation de chaîne.

Les fonctions `convertCelsius` aux lignes 13-16 et `convertFahrenheit` aux lignes 18-21 sont les gestionnaires d'événements de clic de bouton. Ces fonctions effectuent la conversion requise et définissent les propriétés `celsiusValue` et `fahrenheitValue`.

- Ajouter du style.

Éditez `src/styles.css` de sorte que ce soit comme suit.

```

1. body {
2.   background-color: lightgray;
3.   margin-left: 70px;
4.   margin-top: 20px;
5. }

```

`src/styles.css` spécifie les styles qui s'appliquent à l'ensemble de l'application.

Éditez `src/app/app.component.css` comme suit pour le style du composant `Root`.

```

1. h1 {
2.   font-size: large;
3.   font-family: serif;
4.   font-weight: bold;
5. }

```

Finalement, éditez `src/app/temp-converter/temp-converter.component.css` comme suit pour le style du composant `TemperatureConverter`.

```
1. input {
2.   border-style: double;
3.   width: 150px;
4. }
5. button {
6.   background-color: darkcyan;
7.   padding: 20px 25px;
8.   font-size: 18px;
9.   color: white;
10. }
11.
12. label {
13.   font-size: 20px;
14. }
```

Tests

Angular CLI génère des composants avec des tests associés.

`src/app/temp-converter/temp-converter.component.spec.ts` contient la configuration nécessaire pour tester l'unité du composant de convertisseur de température avec le cadre de test Jasmine. Ajoutez les cas de test suivants.

```
1. it('should convert 0 celsius to 32 fahrenheit', () => {
2.   const tempval = '0';
3.   component.convertCelsius(tempval);
4.   expect(component.fahrenheitValue).toBeCloseTo(32);
5. });
6.
7. it('should convert -100 fahrenheit to -73.33 celsius', () => {
8.   const tempval = '-100';
9.   component.convertFahrenheit(tempval);
10.  expect(component.celsiusValue).toBeCloseTo(-73.33);
11. });
```

Configurer le composant Root

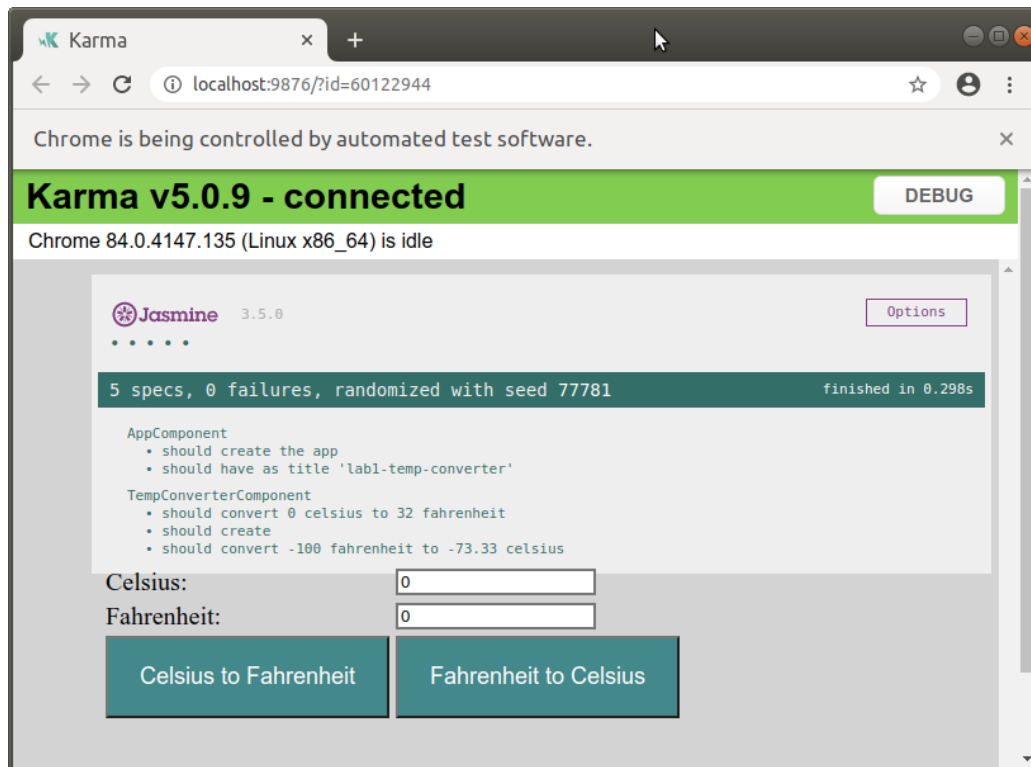
- Supprimez le contenu généré dans le modèle HTML AppComponent du composant root (`/src/app/app.component.html`) et remplacez-le par ce qui suit.

```
1. <div>
2.   <h1>Temperature Converter</h1>
3.   <app-temp-converter></app-temp-converter>
4. </div>
```


Le modèle HTML du composant Root comprend la balise de sélection du composant `TemperatureConverter`. Celainstanciera et rendra le composant où les sélecteurs sont spécifiés.

Exécution des tests

Nous exécutons tous les tests avec la commande `ng test`. Cela inclut les tests unitaires Jasmine ainsi que les tests de bout en bout pilotés par le navigateur.



Exécution de l'Application

Entrez la commande `ng serve` pour construire et démarrer l'application. Puis, naviguez à l'URL <http://localhost:4200>.

Exercice

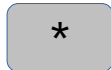
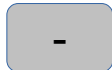
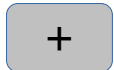
L'exercice suivant est le livrable pour le laboratoire. Complétez et enregistrez le code dans Github Classroom avant la date limite. Seul le travail de ce exercice sera évalué.

Implémentez une application Angular pour une calculatrice de base. L'application doit afficher une interface similaire à la suivante

Premier nombre:

Second nombre:

Resultat:



L'utilisateur pourra alors entrer deux nombres et sélectionner une opération. Suite à la soumission, l'application doit effectuer l'opération requise et afficher le résultat.