Cours Angular

2.1 **| TD Login**

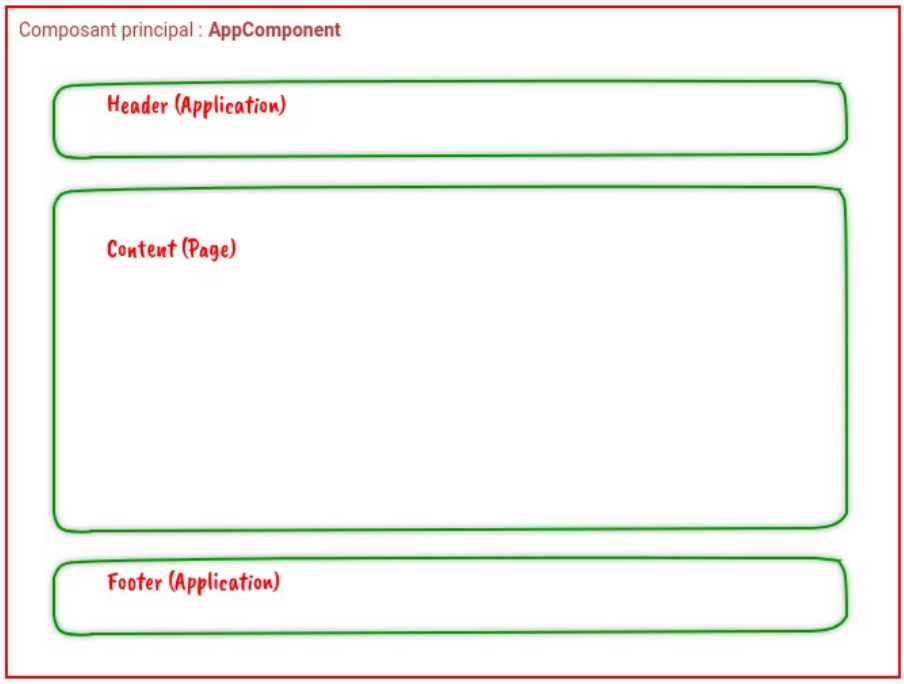
Composant, Template et Frameworks CSS, Directives

1] Structure par composants

Avec Angular, et d'ailleurs comme dans les autres frameworks front-end, une application se structure à l'aide de composants, avec ou sans template associé. On aura donc des composants axés sur l'affichage d'une page HTML et des composants dédiés à la logique métier de l'application, intégrés dans les premiers.

- ***Template usuel d'une application***

En règle générale, une application Web est constituée ainsi :



On peut décider d'utiliser des templates HTML5/CSS3 « clés en main » comme ceux trouvés sur https://html5up.net/. Par contre, oubliez les effets visuels de la librairie JQuery, cette dernière étant possible mais déconseillée avec Angular…

- ***Choix d'un framework CSS***

L'utilisation d'un template s'avère au final restrictif. Classiquement, entre tout faire soi-même et utiliser un cadre clés en main, il y a le framework CSS. Une simple recherche sur Internet permet de sortir la liste suivante des frameworks CSS les plus utilisés : **BootStrap**, **Material** (et dérivés), UIKit, **Bulma**, Foundation, Semantic, etc.

Si l'on veut utiliser un framework CSS avec Angular, **Bootstrap** (conçu par Twitter, https://ng-bootstrap.github.io/#/home) vient le premier à l'esprit, bénéficiant d'un bon support/documentation. On peut lui préférer cependant un framework basé sur le principe du *material design* de Google : **Angular Material** (https://material.angular.io/).

Le framework **Bulma** qui, à mon sens, est excellent sur les axes *performance-poids du framework* et *facilité d'utilisation-fonctionnalités* sera le premier utilisé ici.

2] Utilisation du framework Bulma

- ***Bases de l'installation***

Le projet créé va servir de base pour la démonstration de la notion de composants Angular au travers du principe d'identification dans une application Web.

T2.1 Dans la console appropriée et à l'endroit désiré (le plus «propre » serait dans un dossier spécifique aux projets Angular), créez le projet `login` et lancez-le avec les options classiques :

```
angular$ ng new login --style=css --routing=false
```

Les options indiquent respectivement l'utilisation classique des feuilles de style et la non création d'un fichier de configuration pour les routes de l'application.

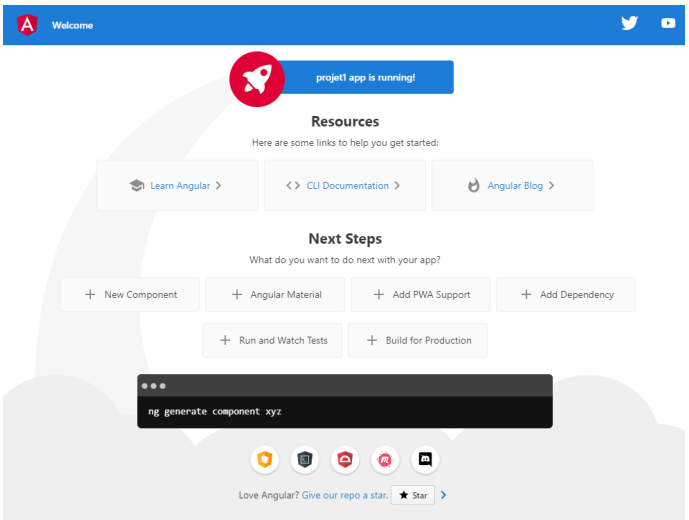
Voir la documentation pour avoir les différentes options de la commande `ng new` :
https://angular.io/cli/new.

Le routage (deuxième option) permettra plus tard à Angular de naviguer entre les différentes pages de l'application à l'aide du module `RouterModule`.

T2.2 Allez dans le répertoire du projet (oubli fréquent du débutant) et lancez l'application par le serveur intégré d'Angular :

```
angular$ cd login
login$ ng serve
```

Cette console ne devra pas être fermée ; on se retrouve avec le site Web classique de démarrage à l'adresse `localhost:4200` dans le navigateur :



T2.3 Bulma s'installe ensuite en suivant le même procédé que BootStrap (le faire par le terminal de l'IDE) :

```
angular$ npm install bulma --save
```

L'option `save` indique que le paquetage s'installe uniquement dans le projet et non globalement.

T2.4 Ajoutez une ligne dans le fichier `angular.json` en repérant bien la bonne section (la première car il y en a 2) :

```
"styles": [
  "node_modules/bulma/css/bulma.min.css",
  "src/styles.css"
```

- ***« Squelette » de l'application***

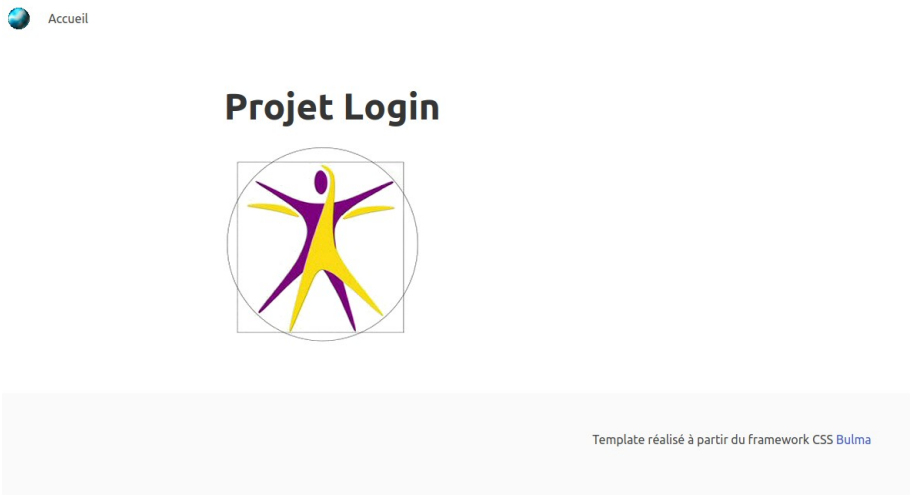
Pour la maquette de site, il vous faut deux images, l'une nommée `logo.png` au format 28x28 pixels et l'autre nommée `fond.png` au format 256x256 pixels.

T2.5 Une fois ces deux images trouvées (ou prenez l'archive donnée), créez un répertoire `images` dans le répertoire `assets` du projet et mettez-y vos deux images.

T2.6 Effacez totalement le contenu template du composant principal (`app.component.html`) pour y mettre ce code :

```
<nav class="navbar" role="navigation" aria-label="main navigation">
  <div class="navbar-brand">
    <a class="navbar-item" href="#">
      <img src="assets/images/logo.png" />
    </a>
  </div>

  <div id="navbarBasic" class="navbar-menu">
    <div class="navbar-start">
      <a class="navbar-item">Accueil</a>
    </div>
  </div>
</nav>

<section class="section">
  <div class="container">
    <div class="column">
      <h1 class="title is-1">Projet Login</h1>
      <img src="assets/images/fond.png">
    </div>
  </div>
</section>

<footer class="footer">
  <div class="container">
    <div class="content has-text-centered">
      <p>
        Template réalisé à partir du framework CSS <a href="https://bulma.io">Bulma</a>
      </p>
    </div>
  </div>
</footer>
```

Voici le visuel résultant (**attention :** il est nécessaire de relancer le serveur) :



Allez voir la documentation officielle pour plus d'explications :https://bulma.io/documentation/.

Sinon, on a bien une page divisée en trois éléments : le `header` (`nav`), le corps de la page (`section`) et le bas (`footer`).

- ***Constructions des modules***

Nous allons maintenant créer deux composants supplémentaires pour transformer notre composant principal en une sorte de « layout » plus conforme au schéma du template vu au début. Comprenez bien que ces deux composants auront pour but le template de page et non la logique métier.

> ***Note :*** *On peut prendre avec profit le terminal de l'IDE car on se situe directement dans le dossier du projet, mais uniquement sous Linux ; des restrictions l'interdit sous Windows ; vous êtes obligé de passer par la console NodeJS.*

Afin de suivre une structure ordonnée, la création supplémentaire de sous-répertoire dans `app` est utile pour bien distinguer les différentes traitements.

T2.7 Créez le sous-répertoire `components` dans `app`.

T2.8 Toujours dans un terminal et avec la commande CLI, créez les deux composants :

```
angular$ ng generate component components/header
angular$ ng generate component components/footer
```

Cette commande créée tous les fichiers des composants chacun dans un dossier séparé : le template, les styles, etc. et met à jour le fichier du module général `app.module.ts` pour les déclarations et importations.

T2.9 Remplacez respectivement le contenu du template de chacun des deux modules (il n'y a qu'une seule ligne actuellement) par toute la balise `nav` et son contenu pour le composant `header` et toute la balise `footer` et son contenu pour le composant `footer`.

Pour retrouver l'affichage complet de la page, il faut indiquer à Angular où intégrer les deux composants et modififer le « layout ».

T2.10 Ajoutez les balises des composants dans le template principal `app.component.html` (préfixées de `app` car n'oubliez-pas, les composants se trouvent dans ce répertoire) :

```
<app-header></app-header>

<section class="section">
  <div class="container">
    <div class="column">
      <h1 class="title is-1">Projet Login</h1>
      <img src="assets/images/fond.png" />
    </div>
  </div>
</section>

<app-footer></app-footer>
```

Le résultat visuel doit être le même… A vous de comprendre quel est l'intérêt d'un tel découpage !

3] Directives

- ***Structure conditionnelle***

Jusqu'à présent, l'application ne comporte qu'une seule page : celle de l'accueil. Classiquement une application Web digne de ce nom comporte une authentification utilisateur et c'est ce que nous allons initier maintenant et compléter dans un prochain TD.

Cette noveauté porte sur le principe de la directive structurelle `ngIf`.

Une directive structurelle avec Angular permet de modifier le DOM au niveau du template. Pour la directive structurelle `ngIf`, n'hésitez pas à consulter la documentation officielle : https://angular.io/api/common/NgIf.

- ***Adaptations de la structure***

T3.1 Dans premier temps, ajoutez au template du `header`, juste avant la balise fermant `nav`, le code (en gras) pour indiquer l'absence d'authentification :

```
…
  <div class="navbar-end">
    <div class="navbar-item">
      <div class="buttons">
        <p>Non connecté</p>
      </div>
    </div>
  </div>
</nav>
```

T3.2 Remplacez ensuite les lignes entre les balises des composants `header` et `footer` par ceci dans `app.component.html` :

```
<app-header></app-header>

<div *ngIf="getConnect(); then connect else noconnect"></div>
  <ng-template #connect>
    <section class="section">
      <div class="container">
        <div class="column">
          <h1 class="title is-1">Projet Login</h1>
          <img src="assets/images/fond.png">
        </div>
      </div>
    </section>
  </ng-template>
<ng-template #noconnect>
  <h1>Identification</h1>
</ng-template>

<app-footer></app-footer>
```

Explications : Nous faisons appel à la directive structurelle notée `*ngIf` qui, en fonction du résultat booléen de la fonction `getConnect()` affichera soit le template `connect`, soit `noconnect`.

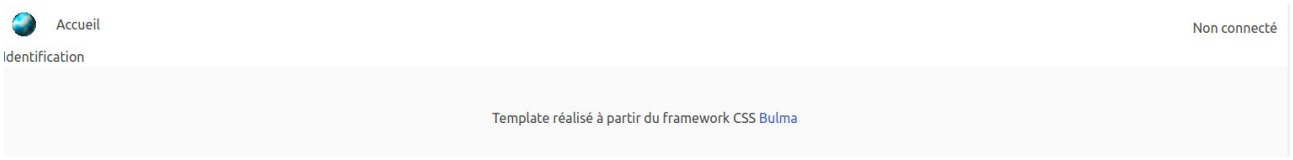
Les directives commencent toutes par un astérisque.

La fonction `getConnect()` n'existant pas, il faut la créer dans `app.component.ts`, conteneur de la logique métier (ou contrôleur). Elle va retourner vrai ou faux en fonction de l'existence du contenu d'une variable nom qui nous servira pour l'affichage de la personne connectée.

T3.3 Intégrez le code dans la partie `export` :

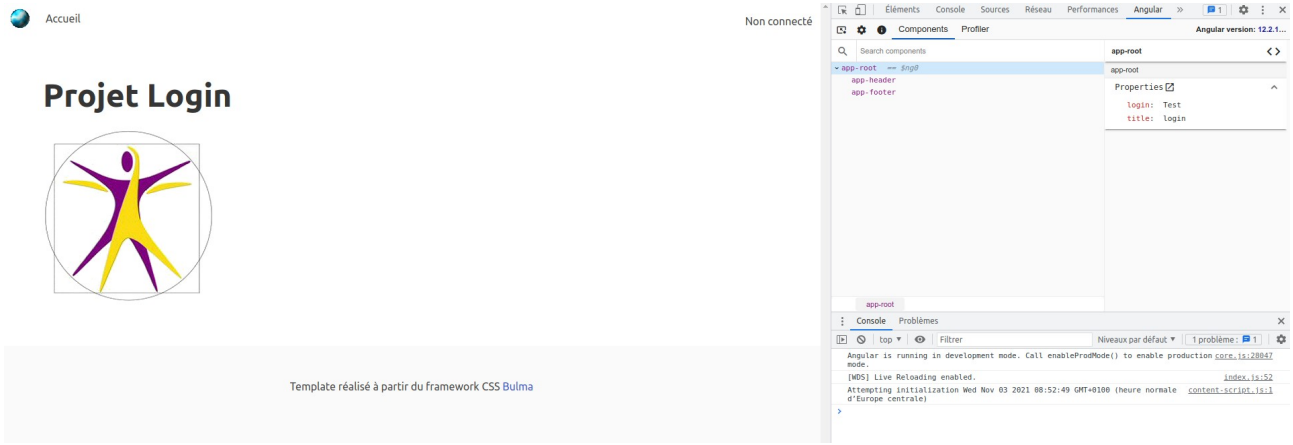
```
…
export class AppComponent {

  title = 'login';

  // Déclaration de la variable (typée !) pour l'utilisateur
  login:string = '';

  // Fonction retournant un booléen et non le nom
  getConnect() {
    return this.login != '';
  }
}
```

Cette fois-ci, la page donne :



Exercice A l'aide de l'extension **Augury** du navigateur Chrome, changez la valeur de la variable `login` dans les propriétés de `app-root` de façon à faire accroire une identification.

On note que cette identification fausse ne change pas le fait que l'indication « Non connecté » reste car le composant métier n'existe pas encore :


Accueil
Non connecté
Projet Login
Template réalisé à partir du framework CSS Bulma
Éléments
Console
Sources
Réseau
Performances
Angular
Components
Profiler
Angular version: 12.2.1...
Search components
app-root
app-header
app-footer
Properties
login: Test
title: login
Console
Problèmes
Filtrer
Niveaux par défaut
1 problème :
Angular is running in development mode. Call enableProdMode() to enable production mode.
[WDS] Live Reloading enabled.
Attempting initialization Wed Nov 03 2021 08:52:49 GMT+0100 (heure normale d'Europe centrale)