Angular.JS

date	révision
Août 2018	Création (v.38)
01/10/2018	Ajout utilisation Firefox recommandé (Chrome analyse une attaque Cross Origin)
18/07/2019	Ajout de fonctionnement général
20/09/2019	Correction <body> exemple 3.2.1
20/09/2020	v.50 Correction p.7 (type=text)
25/06/2021	Amélioration compréhension CDN
08/09/2022	Intégration NodeJS/npm et mise à jour AngularJS déprécié
17/09/2023	v.66 Ajout sur les modules, les fonctions fléchées et une API simple
25/09/2023	v.67 Ajout d'un schéma sur les bases d'Angular.js

TABLE OF CONTENTS

ANGULAR.JS

L'activité proposée ici aborde l'utilisation d'un environnement de développement asynchrone en relation avec un serveur Node.JS.

 **Durée**	 **Difficulté**	 **Taxonomie Bloom**
2 séances	❶ ❷ ❸ ④	Compréhension – Application

Le plan est le suivant :

- Explications sur le framework (et vocabulaire associé)
- Exemples d'utilisation pour préparer un projet

Note : Angular.JS est maintenant déprécié depuis 2022, il reste dans ce bloc B2 uniquement par souci pédagogique. En effet, il permet une approche extrêmement simple des SPA (Single Page Application)

Le projet sera donné dans un autre document.

1 INTRODUCTION ANGULAR.JS

Angular.JS est un framework créé par Google, qui utilise JavaScript. L'intérêt réside dans le fonctionnement avec NodeJS qui permet l'affichage de pages web dynamiques (rafraîchies automatiquement).

La philosophie initiale de ce framework était de respecter l'architecture MVC[1].

Les bonnes pratiques du "design pattern" MVC consistent à séparer les vues, les contrôleurs et les données. Pour cela, Angular.JS se place au centre des actions.

Angular.JS introduit un concept propre à lui, ce sont les directives. Il existe d'autres frameworks JS, comme BackboneJS (le plus ancien) ou ReactOS (créé par les développeurs de Facebook) ou d'autres, prometteur comme Vue.js, Angular2...

Angular.JS perdure et quelques concepts sont plus simples à découvrir avec lui. Ce support explique donc Angular.JS mais dans l'optique de faciliter l'apprentissage d'autres frameworks différents.

1.1 OBJECTIFS

À l'issue de cette activité, l'étudiant pourra :

- ☑ Comprendre le fonctionnement du framework Angular.JS
- ☑ Mettre en œuvre un projet utilisant Angular.JS
- ☑ Savoir lire un code Angular.JS
- ☑ Découvrir les concepts d'application SPA, de binding, de routage
- ☑ Découvrir les directives du framework (ng-app, ng-repeat, ng-filter, etc.)

1.2 PRÉREQUIS

Pour suivre cette activité de découverte sans être en difficulté, il est impératif :

- D'avoir une bonne maîtrise de JavaScript
- D'avoir les notions réseaux des protocoles TCP
- D'avoir les notions systèmes de cache

1 *MVC : Model – View – Controller*

2 INSTALLATION ANGULAR.JS (AVEC NODE.JS)

Angular.JS n'est qu'un framework, il est possible de l'utiliser séparément de Node.JS, mais ce dernier permet d'utiliser le gestionnaire de paquets 'npm'. En effet, il existe trois méthodes pour mettre en œuvre ce framework dans notre application :

2.1 INSTALLATION AUTONOME

La démarche serait de récupérer les fichiers et les décompresser dans le répertoire de l'application que nous voulons développer.

Le lien est https://angularjs.org/

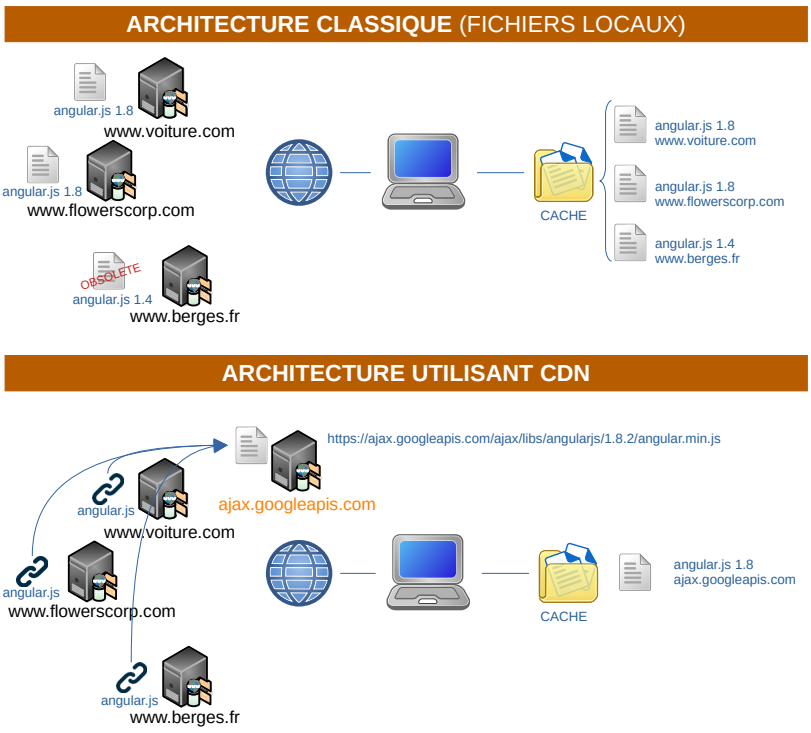
Cependant, nous n'utiliserons pas cette méthode.

2.2 INSTALLATION PAR UN CDN

CDN signifie Content Delivery Network et signifie que le contenu est disponible via le réseau.

Cette notion permet d'utiliser des librairies en ligne directement, plutôt que de les installer en local sur son serveur et courir le risque qu'elles ne soient plus à jour. Un deuxième effet positif, est qu'une librairie chargée depuis un lien est mise en cache dans le navigateur : si un autre site utilise la même librairie en provenance du même site, le navigateur utilisera celle présente dans son cache. Les objectifs sont :

- Économie de bande passante
- Amélioration des performances



Il y a toutefois un inconvénient : si le site où se trouve le framework n'est pas disponible, le site client s'affichera avec des parties manquantes.

2.3 INSTALLATION VIA NPM

Le dernier moyen utilise Node.JS et s'appelle **npm** : c'est un manager de paquets que nous utiliserons pendant nos séances.

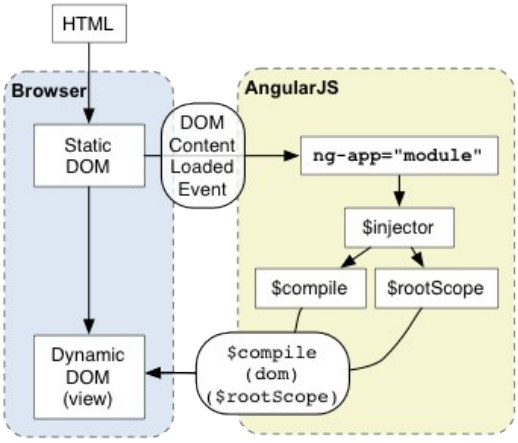
3 UTILISATION ANGULAR.JS

Angular.JS utilise un *bootstrap*[2] au chargement, pour lire les différents fichiers et préparer la structure. Les fichiers HTML peuvent contenir des balises Angular.JS qui seraient ignorées par le navigateur. L'idée est de faciliter les accès au DOM[3] du navigateur.

En intégrant le script angular.js dans une page web, celui-ci s'exécute et prend la main.

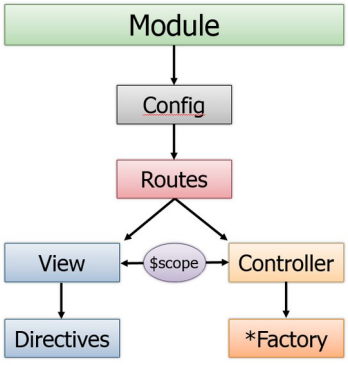
Dès lors, il devient possible d'utiliser les balises propres à Angular.JS pour exécuter du code JavaScript.

Bien entendu, tout cela se fait côté client, c'est-à-dire, sans devoir appeler une page web dynamique : mais il est impératif que JavaScript puisse être exécuté sur le navigateur (il existe des préférences ou des extensions pour bloquer l'exécution de scripts JavaScript).



La création d'une application Angular.JS nommée implique la création d'un module correspondant. Lorsque un **module** est créé (ng-app), il devient possible d'utiliser des **directives** (ng-repeat, ng-src, ng-model, ng-controller…) pour générer des **contrôleurs**, des **vues**.

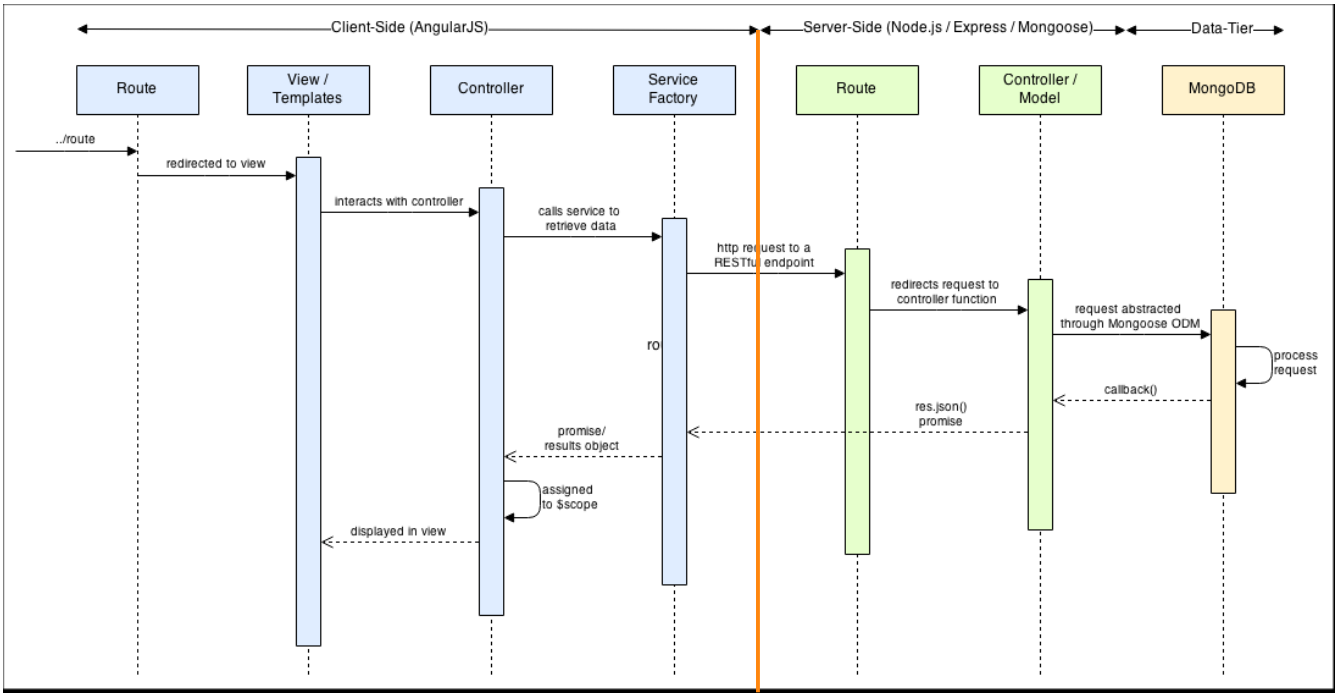
Une partie des directives permettent de créer des **routes** : une URL porte un nom, et ce nom est associé à une vue. Le mécanisme d'association URL ↔ Vue s'appelle une route. Il peut y avoir plusieurs vues associées à une URL, en fonction du terminal (téléphone, poste de travail, etc.)



- **Angular.JS ne nécessite pas Node.JS**. Il peut interagir avec d'autres serveurs et s'exécute côté client.
- **Angular.JS contient une version de jQuery**. Il est donc préférable d'utiliser les concepts d'Angular.JS plutôt que d'intégrer de l'Angular.JS dans une application jQuery.
- **Angular.JS applique le modèle MVC** (Modèle – Vue – Contrôleur). Il est normal d'avoir plusieurs fichiers et devoir lier (bind) les éléments entre eux.
- **Angular.JS propose une variable globale** $scope.
- **Angular.JS utilise des instructions commençant par ng-**. Il est également possible (grâce aux directives) de créer ses propres préfixes, comme *my-*.
- **Angular.JS est un framework de type SPA (Single Page Application)**. Il inclut le concept de contrôleur, de vue, de route, etc.

Afin de bien séparer les choses, Angular.JS est un code qui s'exécute côté navigateur :

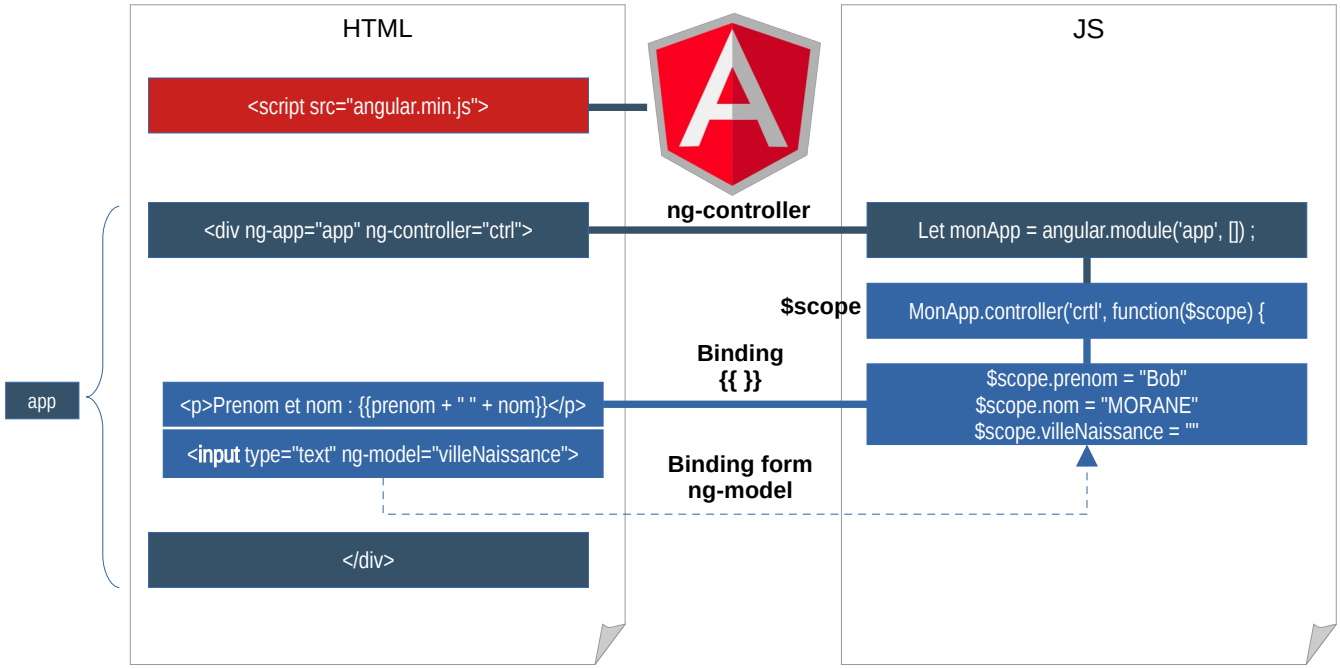
2 *Système d'initialisation, ne pas confondre avec le framework CSS du même nom.*
3 *DOM : Document Object Browser*



Cela ne signifie pas que le code se trouve du côté client, plutôt que le serveur renvoie des pages Angular.JS et qu'il traitera le résultat avec Node.JS et express.JS (comme cela est présenté dans le schéma).

3.1 Les commandes Angular.JS

L'avantage d'un framework est de faciliter le travail du développeur. Angular.JS, apporte un découplage fort entre le code HTML et le code JS, tout en permettant une excellente interactivité.



3.2 ANGULAR.JS

Pour comprendre la philosophie d'Angular.JS, nous allons étudier un premier exemple avec **Angular.JS**, le fameux "Hello World !".

Vous pouvez encore télécharger Angular.JS sur https://angularjs.org/ ou bien (comme dans l'exemple ci-dessous) utilisez la version CDN[4].

3.2.1 Hello Word !

Voici un tout premier exemple, enregistrez le fichier sous le nom "Hello World.html" :

```
<html>
  <head>
    <meta charset="UTF-8">
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.8.3/angular.min.js">
    </script>
  </head>
  <body>
    <div ng-app>
      Saisir votre prénom : <input type="text" ng-model="name">
      <br>
      Bonjour {{name}} !
    </div>
  </body>
</html>
```

Double-cliquez sur le fichier (il n'y a pas besoin de serveur), le résultat est le suivant :

Saisir votre prénom : Davi
Bonjour Davi !

3.2.2 Concept de binding

Le système est dynamique : vous tapez dans le champ texte, et le résultat s'affiche dans la variable `{{name}}`. C'est possible grâce au mécanisme de **binding** (lien) :

Dans Angular.JS, une super variable `$scope` contient les déclarations et les liens associés. Dans notre cas :

Dans le fichier JavaScript :

```
$scope.name = "Chuck Norris"
```

Dans le fichier HTML :

```
ng-model="name"   →     {{name}}
```

💬 Note : Ce mécanisme sera différent dans Angular2.

Angular.JS n'est cependant pas fait pour remplacer JavaScript mais bien pour étendre ses capacités. Nous allons maintenant apprendre les commandes utiles pour déclarer une application.

4 *CDN : Content Delivery Network*

3.3 UTILISATION NPM POUR ANGULAR.JS

L'exemple suivant que vous devez réaliser, n'utilisera pas la version CDN. Il faut donc installer NPM (Node Package Manager). En effet, pour des raisons historiques, NPM est lié à l'installation de Node.JS.

3.3.1 Installation NPM et Node.JS

Le document de cours B2-DEV 304 installation NodeJS indique la démarche à suivre pour installer Node.JS.

3.3.2 Installation AngularJS

Pour installer le paquetage `angular`, il faut d'abord se rendre dans le répertoire de travail. Dans mon cas, `E:\David\WorkSpaces\NodeJS Projets\MyFirstApp`

```
npm init
```

Cette commande pose quelques questions pour préparer le projet, voici les champs importants :

- "name" : saisir le nom de votre application (laisser par défaut)
- "description" : saisir ce que fera l'application ("c'est mon premier programme")
- "author" : saisir votre nom

Cette commande prépare le projet localement : à chaque nouveau projet, il faut recommencer.

La commande suivante, permet d'installer notre framework localement :

```
npm install angular
```

Le résultat devrait ressembler à ceci

```
npm WARN deprecated angular@1.8.3: For the actively supported Angular, see https://www.npmjs.com/package/@angular/core. AngularJS support has officially ended. For extended AngularJS support options, see https://goo.gle/angularjs-path-forward.
added 1 package, and audited 2 packages in 1s
1 moderate severity vulnerability

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
```

Et le répertoire de travail doit abriter un fichier package.json et un dossier contenant les modules (ici, `angular`).

Toute la démarche précédente permet enfin d'avoir un dossier de travail avec notre environnement.

Comme l'installation d'Angular.JS a été faite par npm, ce dernier créé un répertoire **`node_modules`** et y placera les modules demandés, comme dans notre cas, **`angular`**.

Nous verrons plus tard l'ajout d'un module (spécifique à Angular.JS) qui s'appelle **`angular-route`**.

3.3.3 Déclaration d'une application

La démarche sera la suivante (ne codez rien pour le moment) :

Il faudra d'abord créer une page web `index.html` à la racine de notre projet, intégrant Angular.JS et déclarant le nom de notre application. Elle ressemblera à ceci :

```
<html>
  <head>
    <script src="../node_modules/angular/angular.js"></script>
  </head>
  <body ng-app="maDemo"></body>
</html>
```

Nom de la zone d'application

Appel du framework

Le fichier HTML ci-dessus appelle le framework JavaScript puis déclare une application Angular.JS "maDemo". Une application doit contenir un module, il faut donc créer un **script JavaScript** (soit à l'intérieur du fichier HTML, soit dans un fichier à part qu'il faut intégrer également grâce à **<script src... ></script>**

La création d'un module se fait comme ceci :

Nom module

```
var demoApp = angular.module('maDemo', []);
```

Modules optionnels (routage, animation...)

La déclaration d'un contrôleur dans le module s'écrit en utilisant une fonction anonyme :

```
demoApp.controller('MonCtrl', function($scope) { … } );
```

Nom contrôleur

Le contrôleur récupère la variable `$scope` et peut donc interagir avec son contenu. C'est aussi simple que cela : un contrôleur, une vue.

Cette approche est un 'design pattern' connu sous le nom de MVC (qui sera abordé plus loin dans l'année).

3.3.4 Exemples

Pour apprendre, il faut pratiquer et tester. Je vous recommande fortement de tester les codes suivants et de les modifier pour faire vos propres essais. Vous pouvez les copier/coller, mais il faudra par la suite, en rédiger vous-même et vérifier si la syntaxe et bonne.

3.3.4.1 Exemple 1 : application hello World !

Il faut maintenant utiliser Angular.JS : pour cela, la structure est généralement un fichier HTML et un fichier JavaScript. Ng-app permet de débuter la section Angular.JS. Dans ce bloc, on peut créer des contrôleurs (des fonctions/vues). Voici le fichier HTML (*index.html*) qui va utiliser :

index.html

```
<!doctype html>
<html lang="fr">
	<head>
		<meta charset="UTF-8">
		<title>My First Angular</title>
		<script src="../node_modules/angular/angular.js"></script>
		<script src="exemple_World.js"></script>
	</head>
	<body ng-app="ExempleApp1">
		<h1 ng-controller="ControleurNumero1">Hello {{maVariable}} !</h1>
	</body>
</html>
```

et voici le premier script utilisant Angular.JS, *exemple_World.js* :

exemple_World.js

```
var MyApp = angular.module("ExempleApp1", []);
MyApp.controller("ControleurNumero1", function($scope) {
	$scope.maVariable = "World";
});
```

Explications :

- La première ligne déclare une application Angular.JS, appelée "ExempleApp1" (pouvant recevoir une liste d'argument)
- La deuxième ligne créée le contrôleur "ControleurNumero1" et associe un comportement de fonction. La variable *$scope* est une variable globale, comme pour *$_POST* en PHP.
- La troisième ligne affecte "World" à la variable *maVariable*. La variable est alors disponible vers HTML.

En exécutant le fichier index.html vous devriez obtenir un résultat similaire :

Hello World !

3.3.4.2 Exemple 2 : le binding

Le binding signifie créer un lien. Ce lien est fait grâce à la balise Angular.JS **ng-model**. Dans le programme

suivant, le lien sera fait entre le champ de saisie et notre variable dans notre Hello World !

exemple_Binding.html

```
<!doctype html>
<html lang="fr">
	<head>
		<meta charset="UTF-8">
		<title>My First Angular</title>
		<script src="../node_modules/angular/angular.js"></script>
		<script src="exemple_World.js"></script>
	</head>
	<body ng-app="ExempleApp1">
		<div>
			<h1>Hello {{maVariable}} !</h1>
			<p> saisissez un prénom </p>
			<input type="text" ng-model="maVariable"><br>
		</div>
		<div>
			<hr />
			<h1>Bye {{"Bye "+maVariable+" !"}}</h1>
		</div>
	</body>
</html>
```

La structure est modifiée afin que la portée du contrôleur soit étendue a tout le bloc .

Le fonctionnement de ce programme est simple : le lien entre le contenu de la balise <input text> et la variable maVariable est fait grâce à l'instruction **ng-model**. La variable JS est alimentée par le formulaire HTML. Voici le résultat.

Remarquez la possibilité de faire de la concaténation entre les {{ et }} : il est également possible d'effectuer des opérations comme {{(maVariable*1)+2000}}. Mettre entre parenthèse et multiplier par 1 (ou bien soustraire 0) force Angular.JS à traiter notre variable comme un nombre.

Essayez d'ajouter le code suivant :

```
<h2>Calcul (si possible) : {{(maVariable*1)+2000}}.</h2>
```

3.3.4.3 Exemple 3 : les listes

Il s'agit ici d'utiliser l'instruction **ng-repeat** du framework Angular.JS, ainsi que votre premier contrôleur (directive **ng-controller**). La directive ng-repeat agit comme une boucle foreach.

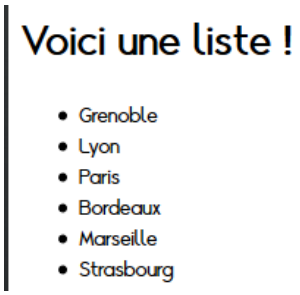
exemple_Liste.html

```
<!doctype html>
<html lang="fr">
	<head>
		<meta charset="UTF-8">
		<title>Gestion de listes</title>
		<script src="../node_modules/angular/angular.js"></script>
		<script src="exemple_Liste.js"></script>
	</head>
	<body ng-app="ExempleApp2" ng-controller="ControleurListe">
		<h1>Voici une liste !</h1>
		<ul>
			<li ng-repeat="ville in listeVilles">{{ville}}</li>
		</ul>
	</body>
</html>
```

exemple_Liste.js

```
var MyApp = angular.module("ExempleApp2", []);
MyApp.controller("ControleurListe", function($scope) {
	$scope.listeVilles = ["Grenoble", "Lyon", "Paris", "Bordeaux", "Marseille",
"Strasbourg"];
});
```

Vous devriez avoir un affichage similaire à ceci :



Attention, une erreur classique est de ne pas terminer correctement le contrôleur : comme il ressemble à une fonction, il est fréquent de terminer } au lieu de }) ; ce qui entraîne une erreur. Si votre code ne fonctionne pas, n'oubliez pas d'activer le débogueur du navigateur, [CTRL]+[Shift]+[i]

La variable $scope représente le modèle (les données) de l'application. Le scope permet au contrôleur et à la vue (ng-model et {{var}}) de partager leurs valeurs en temps réel. Le scope est organisé de manière hiérarchique (comme la structure DOM en JavaScript).

3.3.4.4 Exemple 4 : les filtres

Angular.JS permet de gérer des filtres[5] d'affichage de données. L'exemple qui suit affiche une liste. Pour une fois, il n'y a qu'un seul fichier, mais la directive `ng-init` n'est pas recommandée en production.

exemple_filtres.html

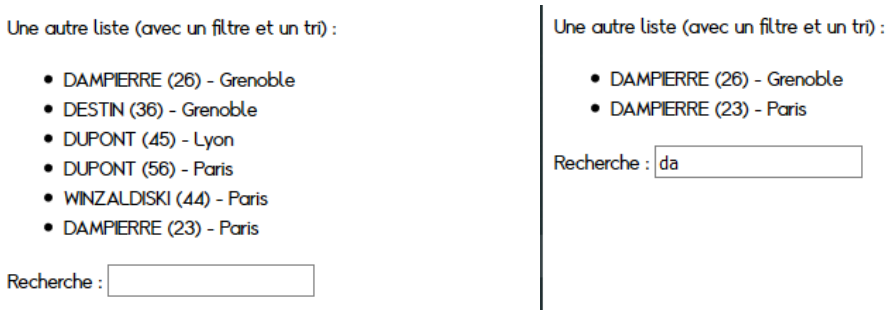
```
<!doctype html>
<html lang="fr">
	<head>
		<meta charset="UTF-8">
		<title>Gestion de listes triées</title>
		<script src="../node_modules/angular/angular.js"></script>
	</head>
	<body ng-app="">
		<p>Une autre liste (avec un filtre et un tri) :</p>
		<div ng-init="personnes=[{nom:'DUPONT', ville:'Paris', age:56},
			{nom:'DUPONT', ville:'Lyon', age:45},
			{nom:'DAMPIERRE', ville:'Grenoble', age:26},
			{nom:'DESTIN', ville:'Grenoble', age:36},
			{nom:'WINZALDISKI', ville:'Paris', age:44},
			{nom:'DAMPIERRE', ville:'Paris', age:23}
			]">
			<ul>
				<li ng-repeat="per in personnes | filter:txtCherche |
					orderBy:'ville'">
					{{per.nom}} ({{per.age}}) - {{per.ville}}
				</li>
			</ul>
			Recherche : <input type="text" ng-model="txtCherche" />
		</div>
	</body>
</html>
```

En associant le contenu du champ `ng-model="txtCherche"` avec la fonctionnalité `filter:txtCherche` disponible dans la commande `ng-repeat`, on obtient un filtre dynamique qui change l'état de la liste à chaque modification du champ de recherche.

L'aspect dynamique est un point essentiel de ce framework. Le codeur n'a plus besoin d'utiliser les événements pour lancer une fonction (par exemple 'onChange'). Grâce à la notion de binding, Angular.JS détecte le changement du changement et le répercute sur les zones concernées.

5 https://docs.angularjs.org/api/ng/filter

L'affichage devrait donner ceci :



Notez bien le format des données dans la liste : ce tableau qui contient des objets est le format utilisé par JavaScript et est appelé JSON (JavaScript Object Notation).

La recherche se fait sur l'ensemble des champs, mais il est possible de limiter la recherche uniquement à ville : il faut écrire `ng-model="txtCherche.ville"` !

3.3.4.5 Exemple 5 : Affichage masqué ou forcé

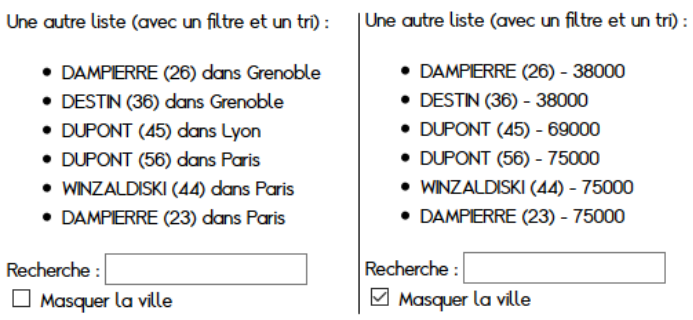
Angular.JS propose deux directives pour masquer ou afficher des éléments. En modifiant le code précédent, nous pouvons afficher la ville ou bien le code postal.

exemple_masque.html

```
<!doctype html>
<html lang="fr">
      <head>
            <meta charset="UTF-8">
            <title>Gestion de listes triées</title>
            <script src="../node_modules/angular/angular.js"></script>
      </head>
      <body ng-app="">
            <p>Une autre liste (avec un filtre et un tri) :</p>
            <div ng-init="personnes=[
              {nom:'DUPONT', ville:'Paris', cp:75000, age:56},
              {nom:'DUPONT', ville:'Lyon', cp:69000, age:45},
              {nom:'DAMPIERRE', ville:'Grenoble', cp:38000, age:26},
              {nom:'DESTIN', ville:'Grenoble', cp:38000, age:36},
              {nom:'WINZALDISKI', ville:'Paris', cp:75000, age:44},
              {nom:'DAMPIERRE', ville:'Paris', cp:75000, age:23}
            ]">
            <ul>
                  <li ng-repeat="per in personnes | filter:txtCherche |
orderBy:'ville'">
                        {{per.nom}} ({{per.age}})
                        <span ng-hide="masqueVille"> dans {{per.ville}} </span>
                        <span ng-show="masqueVille">- {{per.cp}} </span>
                  </li>
            </ul>
            Recherche : <input type="text" ng-model="txtCherche.ville" /><br>
            <input type="checkbox" ng-model="masqueVille"> Masquer la ville
            </div>
      </body>
</html>
```

Cet exemple permet de montrer comment Angular.JS récupère la valeur booléenne de la case à cocher pour choisir comment afficher une valeur. Ici, une simple balise est utilisée, mais vous pouvez transformer le code en un tableau avec des balises <td> et <td ng-hide="masqueVille"> pour la colonne Ville.

Notez qu'il est possible d'inverser la valeur en écrivant `<td ng-hide="!masqueVille">`.

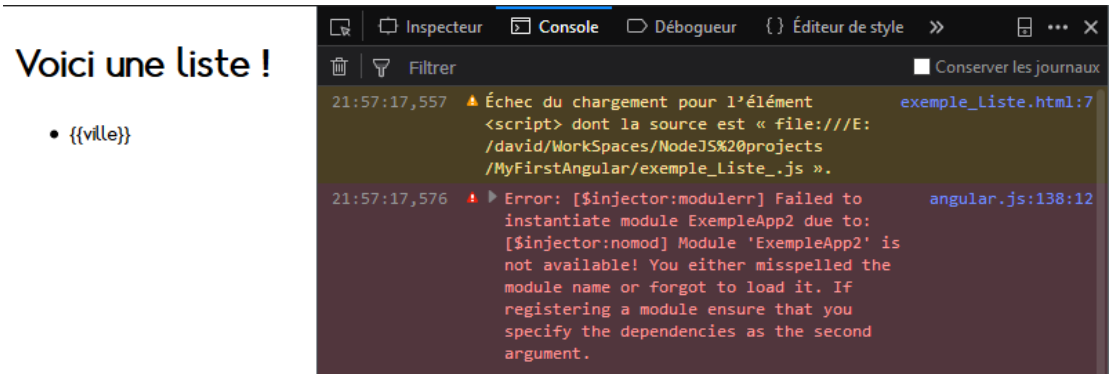


3.3.5 Petite Pause : les erreurs possibles

Pour avoir une description d'erreur plus parlante, il est recommandé de ne pas utiliser la bibliothèque `angular.min.js` mais `angular.js`, qui autorise des messages plus explicites pendant le développement :

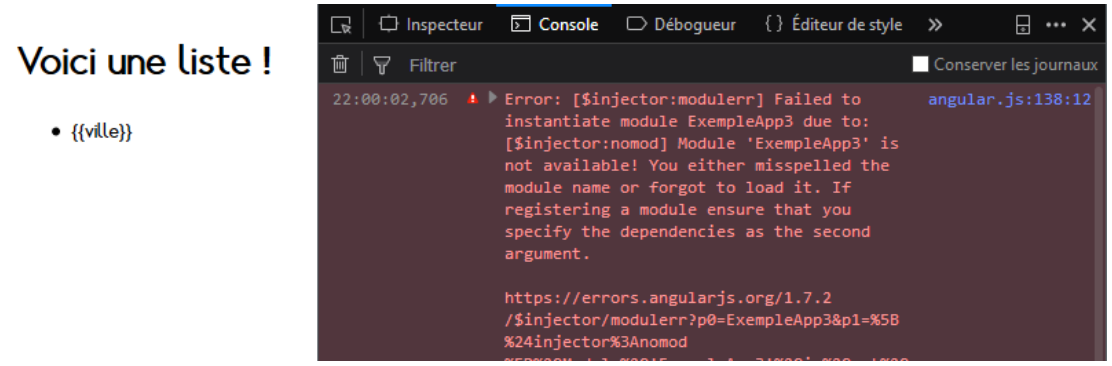
3.3.5.1 fichier inexistant

Même si la couleur indique plutôt un avertissement, l'échec d'accès au fichier sera bloquant :



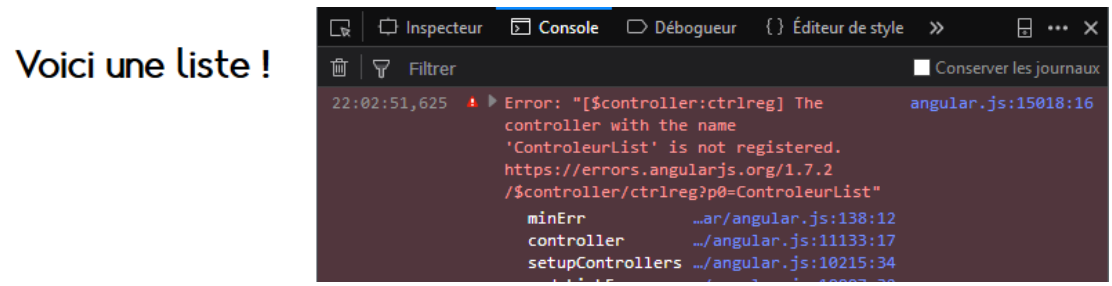
3.3.5.2 Mauvaise application

Le cas suivant indique un problème d'application. "Module 'ExempleApp3' is not available..."



3.3.5.3 Problème de contrôleur

Cette fois, le contrôleur appelé n'existe pas dans l'application, "the controller with the name..."



3.4 LE ROUTAGE ANGULAR.JS

Je rappelle qu'Angular.JS est un framework de type **SPA (Single Page Application)** : cela signifie que pour l'utilisateur, la navigation paraît rapide, car il n'y a qu'un téléchargement vers le serveur pour récupérer toutes les pages.

Mais alors, comment une application JavaScript peut utiliser des liens et en même temps, ne rien charger à distance ?

Sans rentrer dans les détails techniques, il existe un moyen de dire au navigateur qu'il ne doit pas changer de page mais se déplacer dans la page : il s'agit d'une balise HML appelée "ancre" et dont le symbole est #. L'usage des ancres (# dans une URL) permet à AngularJS de traiter le lien URL, sans envoyer de requête au serveur.

3.4.1 Explications

Le routage est une des options les plus intéressantes dans tous les frameworks. Cependant, il faut charger d'autres modules et sources.

L'avantage de mettre en place des routes, est d'afficher une URL explicite, plutôt qu'une URL complexe.

- URL complexe : http://monsite/index.php?id=17&profil=aR3ft&mode=vt
- URL explicite : http://monsite/user/adm/edit

Peu importe si l'affichage de l'URL correspond à l'arborescence des dossiers, c'est la logique qui compte.

Pour cela, il faut établir une table de routage[6]. Par exemple :

URL	Page réel
/consultation	/web/patient/consultation.html
/operation	/web/chu/service/operer.html
/accueil	/index.html

Le module **ngRoute** permet de créer les routes grâce au composant **$routeProvider** : c'est le fournisseur (provider).

Ce module permet également de lire les URL demandées par le navigateur, grâce au composant **$route** : c'est un service (nous l'appellerons rarement).

Il reste l'utilisation de la directive `ng-view` pour insérer le **template** de la page dans le "layout" général. C'est une directive, qui permet d'utiliser le même format pour l'ensemble des pages qui seront à l'intérieur du modèle (nous verrons cela plus loin dans l'exemple).

3.4.2 Installation ngRoute

Comme pour Angular.JS, nous allons profiter de l'installateur npm, dans le répertoire principal du projet, exécutez cette commande :

```
npm install --save angular-route
```

6 *Finalement, il n'y a pas que les SISR qui font des tables de routage*

Dans le code HTML, il faudra alors ajouter le script angular-route.js qui sera dans le répertoire angular-route :

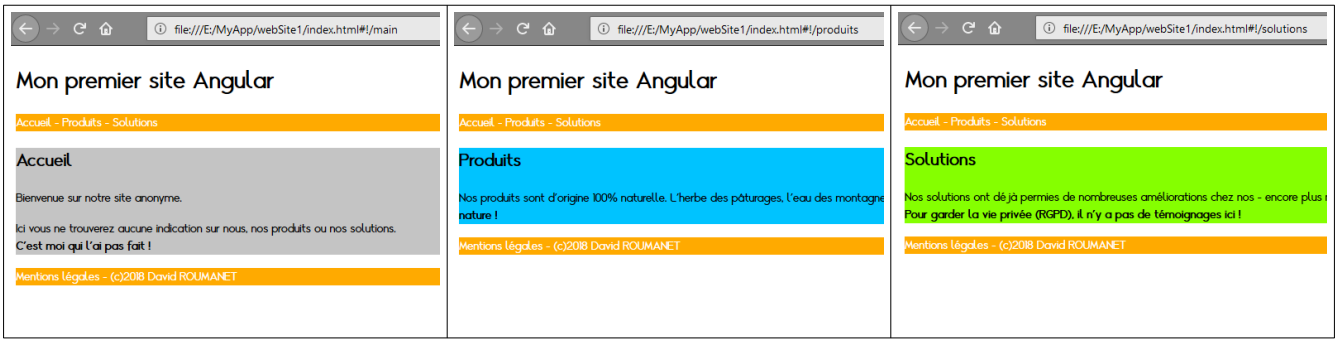
```
<script src="node_modules/angular-route/angular-route.js"></script>
```

3.4.3 Exemple de route

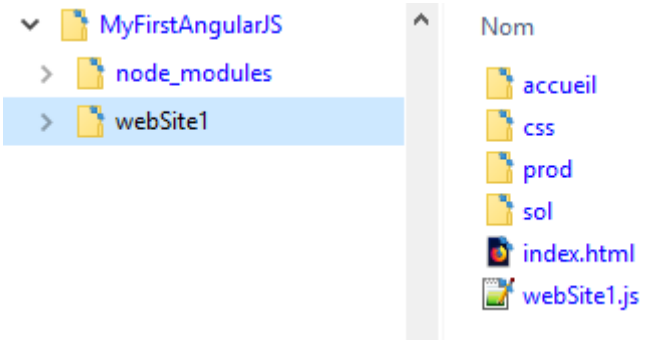
Nous allons créer une véritable application (dossier monApp), avec plusieurs pages, chacune étant dans un dossier différent :

```
├──node_modules
│    ├──angular
│    │      angular-csp.css
│    │      angular.js
│    │      angular.min.js
│    └──angular-route
│           angular-route.js
│           angular-route.min.js
└──webSite1
     │   index.html
     │   webSite1.js
     ├──accueil
     │      main.html
     ├──css
     │      monStyle.css
     ├──prod
     │      produits.html
     └──sol
            soluces.html
```

La page ressemblera à ceci, trois sous pages :



Pour créer cette application, nous allons utiliser l'arborescence suivante :



Le répertoire css ne contient qu'un fichier CSS pour l'ensemble des vues.

Accueil, prod et sol sont respectivement les vues pour l'accueil, les produits et les solutions (nous ajouterons un menu tarif qui ne sera pas développé pour montrer la route par défaut).

L'application doit être placée dans un serveur WEB.

Voici le contenu des différents fichiers.

Index.html

```
<!doctype html>
<html lang="fr">
	<head>
		<meta charset="UTF-8">
		<title>Accueil du site</title>
		<script src="../node_modules/angular/angular.js"></script>
		<script src="../node_modules/angular-route/angular-route.js"></script>
		<script src="webSite1.js"></script>
		<link rel="stylesheet" href="css/monStyle.css">
	</head>
	<body ng-app="webSite1">
		<header>
			<h1>Mon premier site Angular</h1>
		</header>

		<div class="menu">
			<a href="#!main">Accueil</a> -
			<a href="#!produits">Produits</a> -
			<a href="#!solutions">Solutions</a> -
			<a href="#!tarifs">Tarifs</a>
		</div>
		<ng-view>ici...</ng-view>
		<footer class="menu">
			Mentions légales - (c)2018 <a
href="mailto:david.roumanet@free.fr">David ROUMANET</a>
		</footer>
	</body>
</html>
```

Le menu pointe vers des liens de type ancre : cette astuce permet de ne pas obliger le navigateur à appeler un serveur et de rester sur la page… ce qui laisse Angular.JS faire le travail.

⚠ L'ordre des scripts est important !

website1.js (le routeur)

```
var app = angular.module("webSite1",["ngRoute"]); // appli avec module ngRoute

app.config(['$routeProvider', function($routeProvider) {
	$routeProvider
	.when("/main", {
		templateUrl : "accueil/main.html"
	})
	.when("/produits", {
		templateUrl : "prod/produits.html"
	})
	.when("/solutions", {
		templateUrl : "sol/soluces.html"
	})
	.otherwise({
		redirectTo : "/main"
	});
}]);
```

Le routeur permet de décorréler les répertoires et fichiers de l'arborescence dans l'URL, par exemple :

localhost/main affichera la vue dans le répertoire accueil/main.html

L'instruction `otherwise` permet de faire une route par défaut (pour les vues qui n'existent pas).

Placez le fichier suivant dans le répertoire CSS :

monStyle.css

```
.menu {
	color: White;
	background: orange;
}
.menu a:link {
	color: white;
	text-decoration: none;
}
.menu a:visited {
	color: white;
	text-decoration: none;
}
.menu a:hover {
	color: black;
	background: yellow;
}
.produits {
	background: DeepSkyBlue;
}
.solutions {
	background: Chartreuse;
}
.main {
	background: Silver;
}
```

Les fichiers des vues en HTML :

accueil/main.html

```
<section class="main">
      <h1>Accueil</h1>
      <p>Bienvenue sur notre site anonyme.</p>
      <p>Ici vous ne trouverez aucune indication sur nous, nos produits ou nos
solutions.
            <br />
            <b>C'est moi qui l'ai pas fait !</b>
      </p>
</section>
```

prod/produits.html

```
<section class="produits">
      <h1>Produits</h1>
      <p>Nos produits sont d'origine 100% naturelle. L'herbe des
            pâturages, l'eau des montagnes filtrée pendant des milliers
      d'années. <b>Bref, le top de la nature !</b></p>
</section>
```

sol/soluces.html

```
<section class="solutions">
      <h1>Solutions</h1>
      <p>Nos solutions ont déjà permies de nombreuses améliorations
      chez nos - encore plus nombreux - clients. <br />
      <b>Pour garder la vie privée (RGPD), il n'y a pas de témoignages ici
!</b></p>
</section>
```

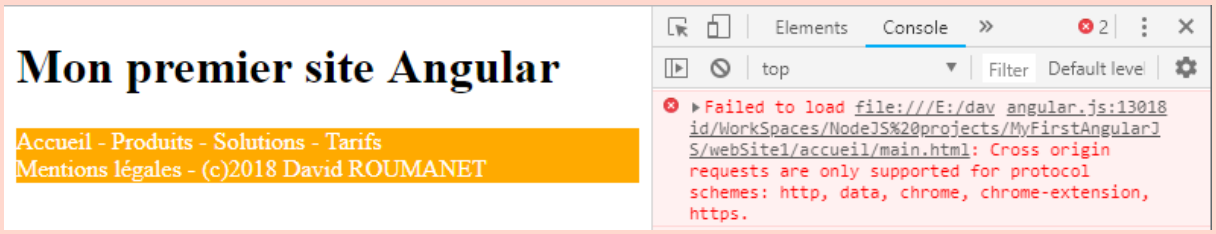
Si vous avez respecté l'arborescence, vous pouvez lancer le fichier index.html et tester le résultat.

3.4.4 Les contraintes d'Angular.JS

Il existe deux contraintes importantes pour pouvoir utiliser Angular.JS

3.4.4.1 sécurité Cross-Script

Désormais, les navigateurs luttent contre les insertions de codes ou pages provenant d'autres sites. Malheureusement, si vous travaillez directement en Angular.JS sur votre disque-dur, vous risquez d'avoir le message d'erreur suivant :



La solution est de lancer un serveur web (WAMP/XAMPP/UwAMP…) et de placer votre application Angular.JS à l'intérieur du répertoire public (généralement /www mais cela dépend de votre serveur).

3.4.4.2 Les fonctions anonymes vs les fonctions fléchées

Un autre point important :

Les fonctions fléchées remplacent progressivement les fonctions anonymes. Cependant, elles ne sont pas strictement identiques :

- le mot clé this des fonctions anonymes est relatif à l'objet qui les contiennent
- le mot clé this des fonctions fléchées prend la valeur du this du contexte parenthèse

Cette différence associée à l'architecture d'Angular.JS empêchent de pouvoir remplacer les fonctions anonymes dans les appels aux fonctions d'Angular.JS, en particulier, pour les contrôleurs.

3.5 LES MODULES ET LES CONTRÔLEURS

3.5.1 Les modules

Revenons sur la notion de contrôleur dans Angular.JS :

```
var MyApp = angular.module("ExempleApp2", []);
MyApp.controller("ControleurListe", function($scope) {
      $scope.listeVilles = ["Grenoble", "Lyon", "Paris", "Bordeaux", "Marseille",
"Strasbourg"];
});
```

Nous avons également vu le module dans le code du routeur :

```
var app = angular.module("webSite1",["ngRoute"])
```

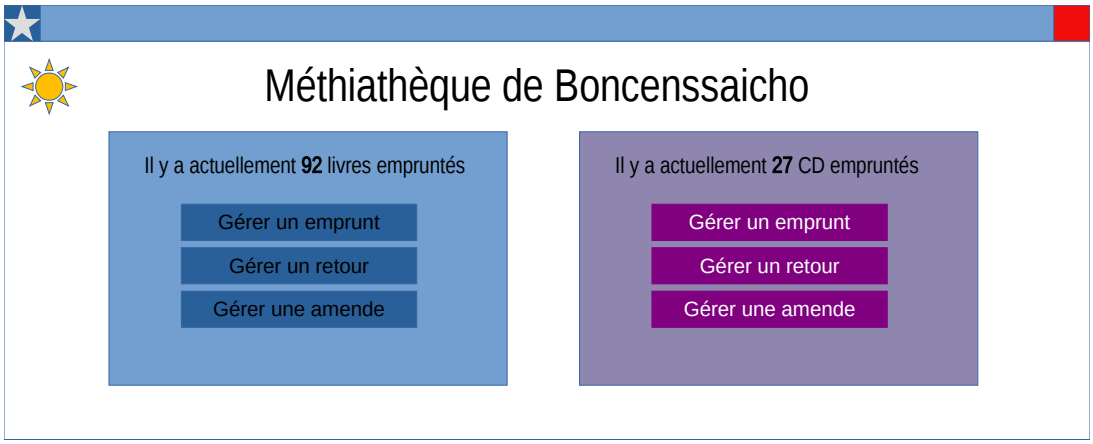
La plupart des frameworks vont lier l'application avec un ou plusieurs contrôleurs.

Attention, il n'est pas possible d'avoir plusieurs applications dans la même page, mais bien des contrôleurs appartenant à la même application.

Pour séparer correctement les contrôleurs, on peut utiliser les modules ()

Le contrôleur est une partie de code qui ne traite qu'un seul type de données ou d'objets métiers.

Par exemple, dans une médiathèque, les employés peuvent gérer les livres et les CD. Ce ne sont pas les mêmes classes d'objets, mais il s'agit de la même application.



Afin d'éviter les interactions entre les contrôleurs (risque d'usage de variables ou fonctions portant le même nom), il existe une instruction qui rend l'application modulaire : `angular.module()`

Ainsi, la variable `$scope` devient contextuelle : chaque contrôleur aura sa propre variable `$scope` indépendante, selon le concept **d'encapsulation** vu en programmation objet.

L'intérêt est de pouvoir y stocker tous les objets et toutes les fonctions dont nous avons besoin, à l'intérieur de ce contrôleur.

On peut alors créer un module qui inclut plusieurs modules.

Le code suivant montre un exemple d'utilisation de deux contrôleurs séparés dans une même page.

index_controllers.html

```
<!doctype html>
<html lang="fr">
<head>
      <meta charset="UTF-8">
      <title>Accueil du site</title>
      <script src="../node_modules/angular/angular.js"></script>
      <script src="controllers.js"></script>
      <link rel="stylesheet" href="css/monStyle.css">
</head>
<body ng-app="app">
<div ng-app='app'>
    <div ng-controller='CtrlArticles'>
        <h1>Votre commande</h1>
        <div ng-repeat='item in items'>  <span>{{item.title}}</span>
            <input type="text" ng-model="item.quantity"><span>{{item.price |
                        currency:"€"}}</span> | <span>{{item.price *
                        item.quantity | currency:"€"}}</span>
        </div>
        <!-- la variable d'un autre module n'est pas accessible -->
        <p>variable d'un autre module [{{client.nom}}] (inaccessible)</p>
    </div>

    <hr/>

    <div ng-controller='CtrlTexte'>
        <h1>Recherche client</h1>
        Nom du client : <input ng-model='text.message'>
        <h5>Bienvenue {{client.nom}}, Vous disposez d'une remise de 5%</h5>
    </div>
</div>
```

Les modules permettent d'encapsuler les données

Le module qui utilise les **deux autres modules**

controllers.js

```
let commande = angular.module('commande', []);
let remise = angular.module('remise', []);
let app = angular.module('app', ['commande', 'remise']);

remise.controller('CtrlTexte', function ($scope) {
    $scope.client = {nom: 'Bob Morane'};
});

commande.controller('CtrlArticles', function ($scope) {
    $scope.items = [
      {title: 'Crayon', quantity: 8, price: 4.2},
      {title: 'Stylo', quantity: 2, price: 5.2},
      {title: 'Gomme', quantity: 3, price: 3.14}];
});
```

Le fait d'associer le contrôleur à un module est donc une méthode sécurisante de travail.

Le résultat est le suivant :

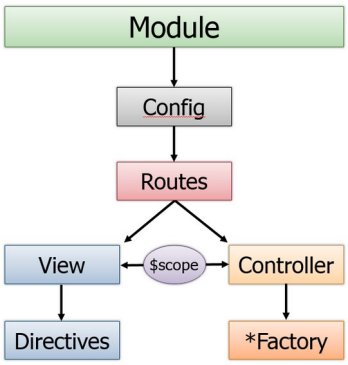
- L'application utilise deux modules indépendants
  - Le module commande contient les produits et les prix. Ce module ne peut accéder aux variables des autres modules.
  - Le module remise permet de récupérer le nom du client et de lui proposer une remise.

3.5.2 Les contrôleurs

Il existe une possibilité de partager une variable entre deux modules, en utilisant une instruction factory(). Cependant, Angular.JS étant abandonné par ses développeurs, nous n'irons pas plus loin dans son utilisation.

4 LES NOTIONS À RETENIR

Dans le début du cours, ce schéma permettait d'illustrer l'architecture d'Angular.JS.



À la fin de cette découverte, il doit être compréhensible par chacun d'entre-vous :

- Les **modules** permettent de créer des zones de codes indépendantes, sécurisées et faciles à coder.
- Les **routes** permettent de naviguer dans l'application, indépendamment de l'architecture des fichiers et des dossiers physiques.
- Les **vues** sont l'affichage de l'application (dépendent du routage)
- Les **contrôleurs** sont associés aux modules et permettent de définir les actions à réaliser.
- Les frameworks gèrent les **liaisons** (binding) entre les variables JavaScript et le contenu du DOM dans les pages HTML
- La variable **$scope** agit comme une variable globale contextuelle, qui contient l'ensemble des objets nécessaires aux contrôleurs et aux modules.

Angular.JS permet de créer une application de type **SPA (Single Page Application)** ayant l'avantage de charger l'application au démarrage (un peu plus long) et de n'échanger avec un serveur, que les données (beaucoup plus léger que de recharger chaque page). Cela constitue un avantage sur les applications en PHP.

5 ANNEXES

5.1 SOURCES

https://weblogs.asp.net/dwahlin/video-tutorial-angularjs-fundamentals-in-60-ish-minutes

https://docs.angularjs.org/api

https://www.youtube.com/playlist?list=PL6n9fhu94yhWKHkcL7RJmmXyxkuFB3KSl

Routage avec Angular.JS
http://www.tutoriel-angularjs.fr/tutoriel/2-utilisation-complete-d-angularjs/1-le-routage#la-directive-ngView

Information sur le CROSS Scripting (CORS) :
https://medium.com/@tusharrungta18/the-above-image-is-almost-a-horror-movie-43a9df5afd9

Multiples contrôleurs
https://stackoverflow.com/questions/20561116/how-to-have-more-than-one-controllers-in-angularjs-correctly

récupérer une information sur l'élément cliqué :

https://www.tutorialsteacher.com/angularjs/angularjs-events

Vidéo d'un étudiant BTS SIO pour un projet de liste

https://www.youtube.com/watch?v=3aeg1MzlNOg

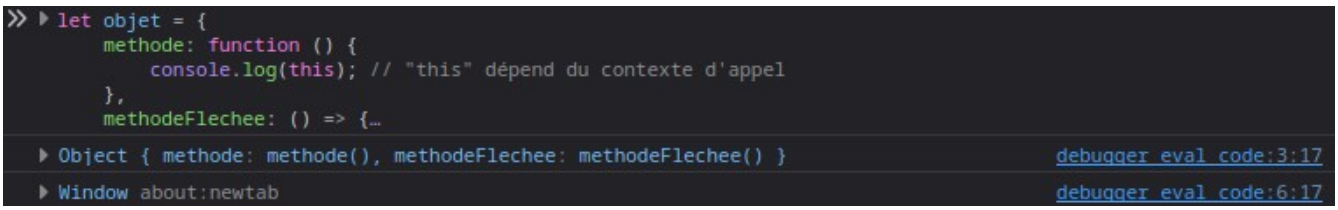
5.2 DÉTAIL FONCTION FLÉCHÉE ET FONCTION ANONYME

Testez ce code en mode REPL dans votre navigateur (CTRL+Shift+i puis collez-le) :

```
let objet = {
    methodeAnonyme: function () {
        console.log(this); // "this" dépend du contexte d'appel
    },
    methodeFlechee: () => {
        console.log(this); // "this" est capturé du contexte de création
    }
};

objet.methodeAnonyme(); // "this" dans methode fait référence à "objet"
objet.methodeFlechee(); // "this" dans methodeFlechee fait référence à la portée globale
```

Résultat, les deux fonctions ne renvoient pas le même environnement 'this' :



5.3 UTILISATION D'UNE API (POUR ALLER PLUS LOIN)

Le code suivant montre (en avance) comment utiliser un framework pour appeler des services en lignes. Si vous êtes curieux, testez-le…

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
    <meta charset="utf-8">
    <title>Prévision Météo Grenoble</title>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></
script>
</head>

<body ng-controller="WeatherController">
    <h1>Prévision Météo pour Grenoble</h1>

    <div ng-if="loading">Chargement en cours...</div>

    <div ng-if="!loading && weatherData">
        <h2>{{ weatherData.city_info.name }}</h2>
        <p>Actuellement : {{ weatherData.current_condition.condition }}</p>
        <p>Température : {{ weatherData.current_condition.tmp }}°C</p>
        <p>Humidité : {{ weatherData.current_condition.humidity }}%</p>
    </div>

    <div ng-if="!loading && !weatherData">
        Impossible de récupérer les données météo.
    </div>

    <script>
        var app = angular.module('myApp', []);

        app.controller('WeatherController', function ($scope, $http) {
            $scope.loading = true;

            // Fait une requête GET à l'API de prévision météo
            $http.get('https://prevision-meteo.ch/services/json/grenoble')
                .then(function (response) {
                    $scope.weatherData = response.data;
                    $scope.loading = false;
                })
                .catch(function (error) {
                    console.error('Erreur récupération des données :', error);
                    $scope.loading = false;
                });
        });
    </script>
</body>
</html>
```