Étude d'un premier Framework

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.

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

- Comprendre le fonctionnement du framework Angular.JS
- Mettre en œuvre un projet utilisant Angular.JS
- Savoir lire un code Angular.JS

Le plan est le suivant :

- Explications sur le framework (et vocabulaire associé)
- Exemples d'utilisation

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.

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

Angular.JS perdure et quelques concepts sont plus simples à découvrir qu'avec Angular.

Ce support explore donc Angular.JS mais **dans l'optique de faciliter l'apprentissage d'Angular qui est très différent**.

2 INSTALLATION ANGULAR.JS (AVEC NODE.JS)

En réalité, comme Angular.JS n'est qu'un framework, il est possible de l'utiliser séparément de Node.JS.

2.1 INSTALLATION AUTONOME

La démarche serait de récupérer les fichiers et les décompresser dans le répertoire de l'application que nous voulons développer. Il est également possible d'utiliser le lien **CDN** pour avoir une version directement sur le web.

La démarche ici, est de comprendre le développement Front-End (Angular.JS) et back-end (NodeJS).

2.2 NPM : NODEJS PACKAGE MANAGER (télécharger et installer nodejs)

npm est le gestionnaire de paquets officiel de **Node.js**. Depuis la version 0.6.3 de Node.js, npm fait partie de l'environnement et est donc automatiquement installé par défaut.

L'utilisation de npm se fait en ligne de commande. La première commande à tester est :

```
npm --version
```

Cette commande affichera la version en cours (dans mon cas, v6.14.4).

Nous verrons comment utiliser npm ultérieurement: Angular.JS est inclus avec, mais nous ne l'utiliserons pas.

1 *MVC : Model – View – Controller*

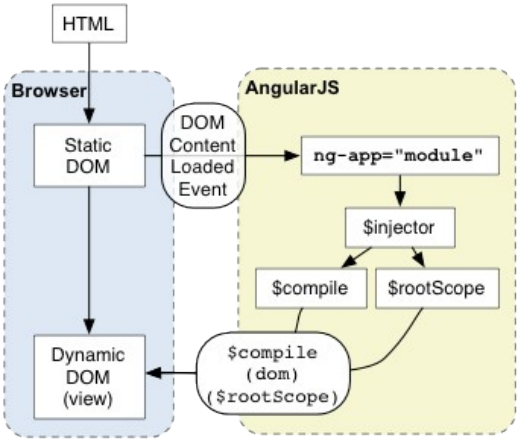
3 UTILISATION ANGULAR.JS

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.

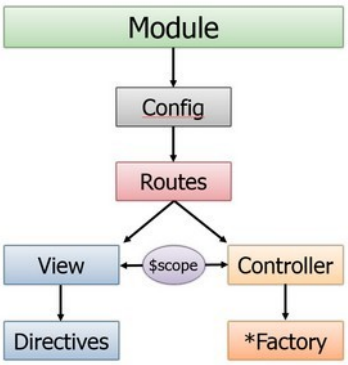
Dès lors, il devient possible d'utiliser les **balises propres à Angular.JS** pour exécuter du code JavaScript.

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.)

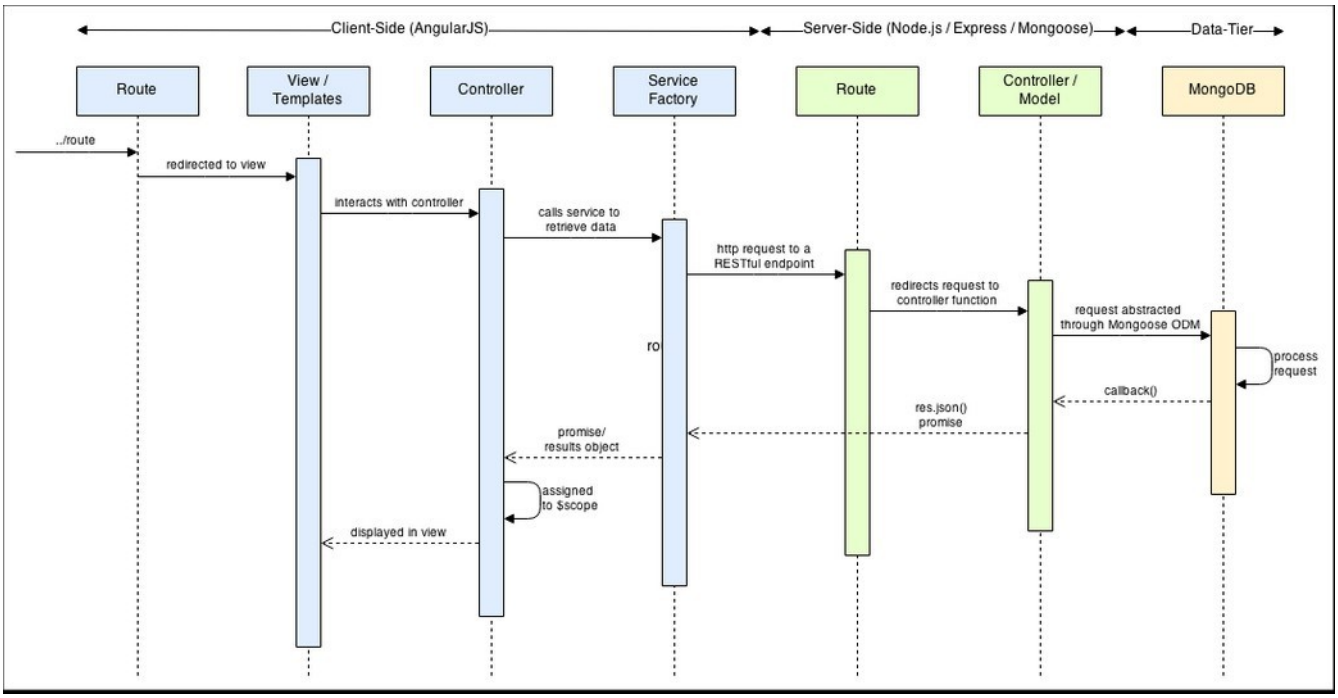


- **Angular.JS ne nécessite pas Node.JS**. Il peut interagir avec d'autres serveurs et s'exécute côté client.
- **Angular.JS contient une version de jQuery**. Il est donc préférable d'utiliser les concepts d'Angular.JS plutôt que d'intégrer de l'Angular.JS dans une application jQuery.
- **Angular.JS 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

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

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** :



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.

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, 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].

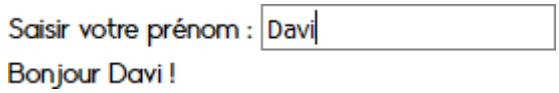
2 *Système d'initialisation, ne pas confondre avec le framework CSS du même nom.*
3 *DOM : Document Object Browser*
4 *CDN : Content Delivery Network*

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://ajax.googleapis.com/ajax/libs/angularjs/1.8.0/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 :



Notez le système 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 supervariable **$scope** contient les déclarations et les liens associés. Dans notre cas :

ng-model="name" → {{name}}

Ce mécanisme sera différent dans Angular.

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.

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

3.3 UTILISATION NPM POUR ANGULAR.JS (nécessite d'installer node)

Pour installer le paquetage angular, il faut d'abord se rendre dans votre répertoire de travail. Exemple 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 notice created a lockfile as package-lock.json. You should commit this
file. npm WARN myfirstangular@1.0.0 No repository field.

+ angular@1.7.2
added 1 package in 13.876s
```

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**.

3.3.1 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

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

ceci :

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 :

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

Nom module

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

C'est aussi simple que cela : un contrôleur, une vue.

3.3.2 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.

3.3.2.1 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*.

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

Hello World !

3.3.2.2 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**. 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>
```

3.3.2.3 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.

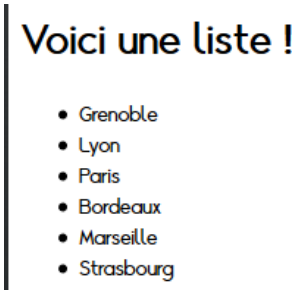
exemple_Liste.html

```
<!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 déboqueur 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).

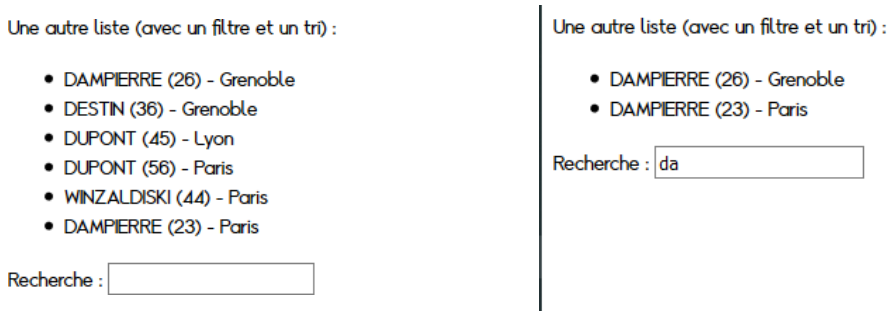
3.3.2.4 Exemple 4 : les filtres

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>
```

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).*

Pour ne chercher que sur la ville par exemple, il faut écrire ng-model="txtCherche.ville" !

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

3.3.2.5 Exemple 5 : Affichage masqué ou forcé

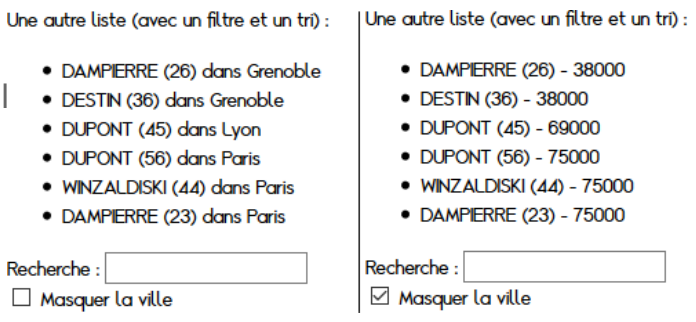
Angular.JS propose deux directives pour masquer ou afficher des éléments. En modifiant le code précédent, nous pouvons afficher la ville ou bien le code postal.

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 val en écrivant <td ng-hide=" **!** masqueVille">.

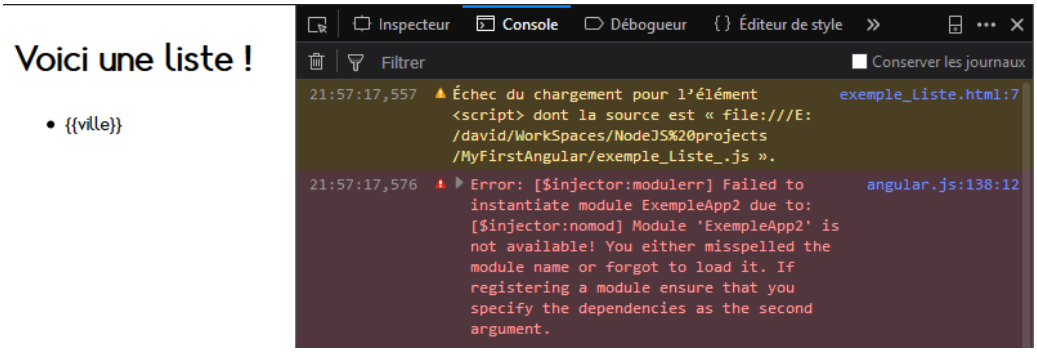


3.3.3 Petite Pause : les erreurs possibles

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 :

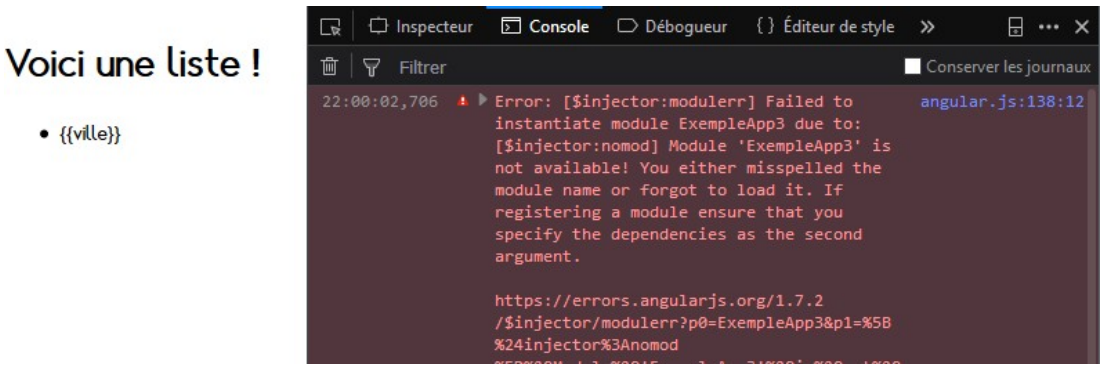
3.3.3.1 fichier inexistant

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



3.3.3.2 Mauvaise application

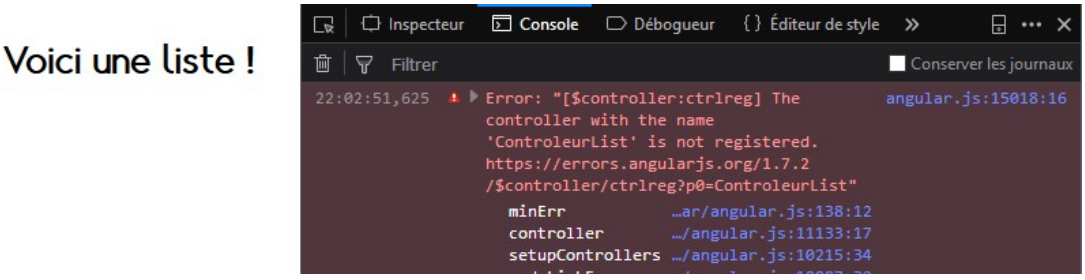
Le cas suivant indique un problème d'application. "Module 'ExempleApp3' is not



available..."

3.3.3.3 Problème de contrôleur

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



the name..."

A suivre....