ANGULARJS

Eléments de base

Les bases d'AngularJS

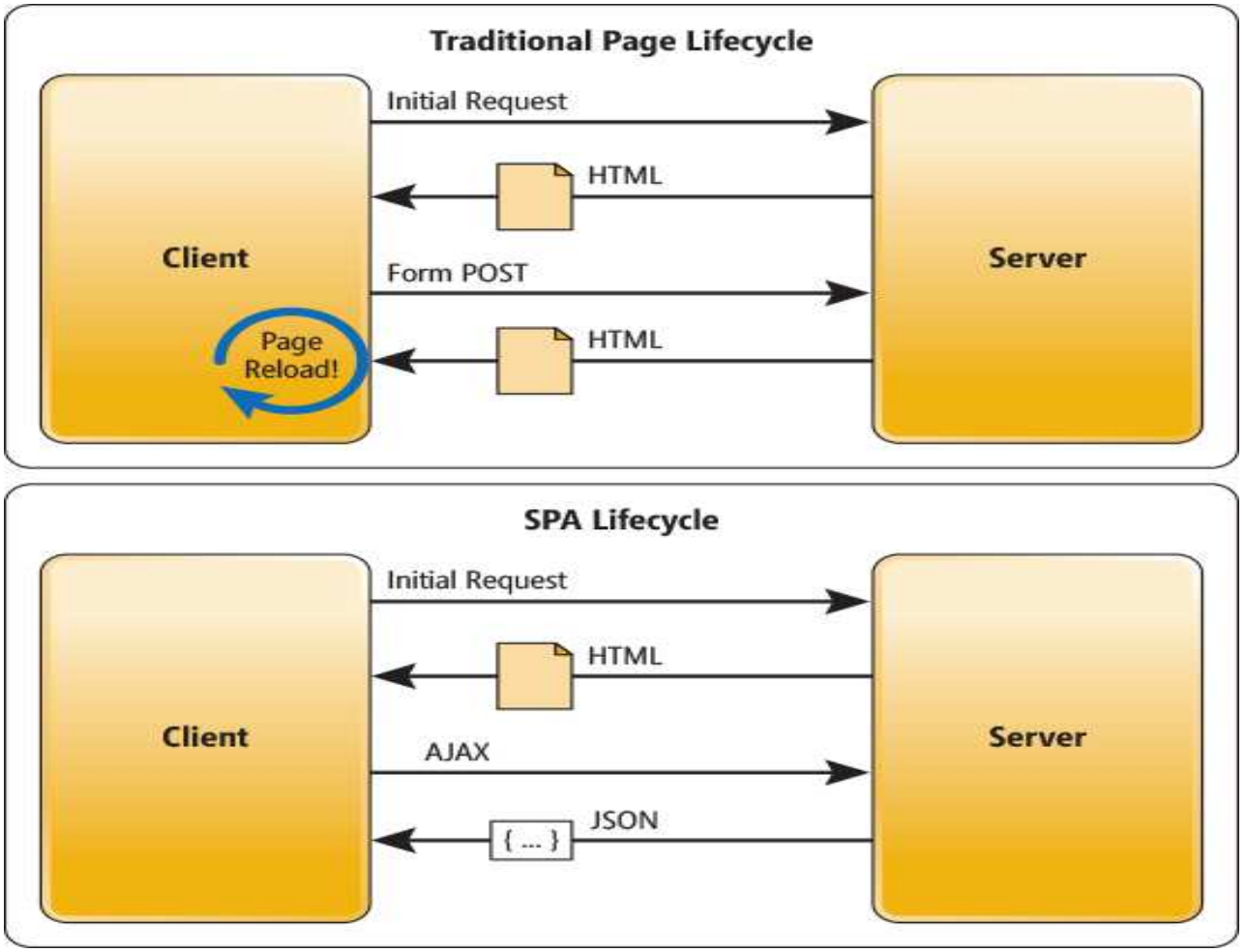
Introduction

- AngularJS est né en 2009 dans les locaux de Google.
- Angular est construit autour des concepts clés suivants:
  - Architecture MVVM (Modèle-Vue-Vue-Modéle)
  - Two-way data binding
  - Injection de Dépendances
  - Manipulation du DOM au moyen de directives
  - **SPA** (Single Page Application)

Introduction

- Précision préliminaire pour les développeurs JavaScript "old school": avec Angular, il n'y a, en général, pas de manipulation directe du DOM.
- Avec jQuery, Prototype et autres librairies JavaScript, on doit presque toujours sélectionner un élément (via l'API DOM) pour pouvoir l'utiliser.
- Avec AngularJS on peut ajouter, supprimer et modifier la page HTML sans faire aucun appel au DOM: plus besoin de $(), getElementByID(), ...
- **Note:** les accros à jQuery ont toutefois accès, depuis Angular, à jQLite (un jQuery Lite) mais c'est souvent déconseillé.

SPA: Single Page Application



SPA: Single Page Application

- AngularJS permet de développer des applications Web de type SPA.
- Une SPA (Single Page Application) est une application web accessible via une page web unique.
- Le but est d'éviter le chargement d'une nouvelle page à chaque action demandée et d'améliorer ainsi l'expérience utilisateur (meilleure fluidité).

SPA: Single Page Application

- La différence entre une SPA et un site web classique réside dans leur structure et dans la relation qu'ils établissent entre le navigateur et le serveur:
  - Une SPA est donc composée d’une seule page. Le rôle du browser (front-end) est beaucoup plus
  - important : toute la logique applicative y est déportée.
  - Le serveur (back-end) est "seulement" responsable de la fourniture des ressources à l’application et surtout de l'exposition des données.

Pourquoi le SPA ?

- Les frameworks JS comme AngularJS et EmberJS participent à la popularité des SPA.
- Les SPA s'appuyant sur de tels frameworks ont en général comme avantage d'être:
  - testables (unitairement et fonctionnellement)
  - fluides (pas de rechargement d'url etc)
  - bien organisées
  - maintenables et évolutives
  - ...

SPA: les pièges à éviter

- Néanmoins il y a plusieurs points à prendre en compte lorsqu'on réalise une SPA pour éviter des problèmes potentiels:
  1. La taille du DOM (trop gros il peut ralentir fortement le browser).
  2. Les traitements (trop lourds il peuvent ralentir le browser).
  3. L'empreinte mémoire de l'application (qui doit être la plus réduite possible).
- Solutions:
  1. Essayer de ne charger que les éléments nécessaires à l'utilisateur du site et décharger les autres.
  2. Ecrire et tester correctement son code.
  3. Utiliser une solution basée sur un "virtual DOM" (React.JS est un framework JavaScript pionnier en la matière).

SPA: les pièges à éviter

- Autre sujet qui peut être un problème avec les SPA: le référencement (problématique du "SEO" - Search Engine Optimization).
- Les moteurs de recherche ont du mal à "crawler« des sites dynamiquement gérés par du JavaScript.
- Des solutions existent pour fournir spécifiquement à Google une version correspondante aux "snapshots« HTML générés par les applications SPA.
- Ces solutions sont accessibles soit en mode SAAS (payantes et hébergées) soit en mode Open- Source à héberger soi-même.

Two-way data binding

- Une des fonctionnalités phares d'AngularJs est le **two-way data binding** qui permet de notifier tous les éléments qui font référence à une variable, de son changement.
- Ainsi sans coder explicitement du JavaScript, on peut mettre à jour en temps réel un titre qui fait référence à un champ texte.
- Exemple :

```
<script src="lib/angular/angular.js"  ...
<input type="text" ng-model="yourName">
<h1>Hello {{yourName}} !</h1>
```

Premier pas avec ANGULARJS (TP 1)

- Créer une page web simple (index.html) affichant les éléments suivants :
  - Un titre (H1) : « *Premier pas avec AngularJs*»
  - Un formulaire ayant :
    - Texte « Saisir votre nom » suivi d'une zone de saisi de texte « nom »
    - Texte « Saisir votre ville » suivi d'une zone de saisi de texte « ville »
  - Un texte dynamique : « Bonjour $nom, votre ville est $ville », avec $nom et $ville sont les valeur saisies

Premier pas avec ANGULARJS (TP)

- Les étapes à suivre :
  1. Référencer la librairie JS :
     - <script src="lib/angular/angular.js" > </script>
  2. Ajouter un qui regroupe tous les éléments de la page.
  3. Utiliser le **ng-model** pour les variables des champs du formulaire
  4. Utiliser les « {{» et « }}» pour afficher le contenu des variables du **ng-model.**

Premier pas avec  (TP 2)

- Créer une page web simple (index.html) permettant d'effectuer la somme de deux entiers saisis dans un formulaire et afficher le résultat dans la même page :
  - Texte « Saisir la valeur de **A** » suivi d'une zone de saisi de texte « **valA** »
  - Texte « Saisir la valeur de **B** » suivi d'une zone de saisi de texte « **valB** »
  - La somme de **valA** et **valB** = **valA + valB**

Vues

- Utilisation des attributs **ng** (directives) sur les elements HTML
  - **ng-app**
    - Détermine quelle partie de la page va utiliser angularjs
    - Si il contient une valeur, il va charger le module d'application
  - **ng-controller**
    - Détermine quel contrôleur Javascript doit être utilisé pour la partie de la page
  - **ng-model**
    - Détermine quel modèle d'un champ de saisie sera lié à la valeur
    - Utilisé pour la liaison bidirectionnelle (Used for two-way binding)

Les modules

- La façon de construire l'application est spécifiée par des modules.
- Dans les modules, on spécifie ce qui est partagé dans l'application.
- Un module peut dépendre d'autres modules.
- L'application a un module principal.
- Dans le code HTML :

```
<!doctype html>
<html ng-app="myApp">
<!-- ... -->
</html>
```

- Côté JavaScript :

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

Les modules

- Un exemple avec la définition d'un nouveau contrôleur :

```
var myApp = angular.module('myApp', [ ]);
myApp.controller('MyCtrl', function MyCtrl($scope) {

    $scope.copy = function() {
        $scope.dst = $scope.src;
    }
});
```

- Utilisation du contrôleur côté HTML :

```
<!doctype html>
<html ng-app>
<!-- ... -->
 <body>
   <div ng-controller="MyCtrl">
        <input type="text" ng-model="src">
         <button ng-click="copy()">Copier</button>
         <h1>{{dst}}</h1>
     </div>
  </body>
</html>
```

Les modules

Trois sortes d'éléments peuvent être définis dans un module :

- Un service (un objet partageable);
- Une directive (nouvel élément côté HTML);
- Un filtre (voir le cours suivant).

- Il est recommandé de séparer les différents éléments :

```
var myAppService = angular.module('myApp.service', [ ]);
var myAppDirective = angular.module('myApp.directive', [ ]);
var myAppFilter = angular.module('myApp.filter', [ ]);
var myApp = angular.module('myApp', ['myApp.service', 'myApp.directive', 'myApp.filter']);
```

Les modules

Il est possible d'exécuter du code au lancement de l'application :

```
var myApp = angular.module('myApp', []);
myApp.run(function($rootScope) {
$rootScope.message = "Mon message";
});
```

```
<!doctype html>
<html ng-app="myApp">
<head><!-- ... --></head>
<body>
{{ message }} <!-- devient "Mon message" -->
</body>
</html>
```

Contrôleurs et Data Binding

Data Binding : les vues sont automatiquement modifiées lors d'un changement du modèle par un contrôleur.

- **Les contrôleurs sont attachés aux éléments du DOM et font évoluer** le modèle en fonction des actions de l'utilisateurs ou d'événements.

```
<!doctype html>
<html ng-app>
<!-- ... -->
 <body>
   <div ng-controller="MyCtrl">
         <input type="text" ng-model="src">
          <button ng-click="copy()">Copier</button>
          <h1>{{dst}}</h1>
     </div>
  </body>
</html>
```

Contrôleurs et Data Binding

Data Binding : les vues sont automatiquement modifiées lors d'un

changement du modèle par un contrôleur.

- **Les contrôleurs sont attachés aux éléments du DOM et font évoluer**

le modèle en fonction des actions de l'utilisateurs ou d'événements.

La fonction qui permet de construire une instance du contrôleur :

```
function MyCtrl($scope) {
    $scope.copy = function() {
        $scope.dst = $scope.src;
  }
}
```

La variable $scope référence un objet qui correspond à la partie du modèle attachée à l'élément du DOM contrôle.

Scopes et RootScope

La variable $RootScope contient l'intégralité du modèle alors que la variable $scope contient uniquement la partie du modèle attachée à l'élément du DOM :

-
```
<body>
    <div ng-controller="MyCtrl">
        <input type="text" ng-model="src">
        <button ng-click="copy()">Copier</button>
    </div>
    <h1>{{dst}}</h1>
</body>
```

-
```
function MyCtrl($scope, $rootScope) {
$scope.copy = function() { $rootScope.dst = $scope.src; }
}
```

Scopes et RootScope

Portée des scopes :

```
<body>
<div ng-controller="MyCtrl">
{{ dst }}
</div>
{{ dst }}
</body>
```

```
function MyCtrl($scope, $rootScope) {
$rootScope.dst = "rootScope";
$scope.dst = "scope";
}
```

Chaque contrôleur peut donc avoir une partie du modèle comme contexte.

Template – ng-repeat

```
<body ng-controller="ListCtrl">
<input ng-model="item">
<button ng-click="add()">add</button>
<ul><li ng-repeat="e in items">
{{ e }}
</li></ul>
</body>
```

```
function ListCtrl($scope) {
$scope.items = [];
$scope.add = function() {
$scope.items.push($scope.item);
}
}
```

Template – visibilité

Il est possible de masquer ou d'afficher certains éléments :

```
<body>
    <input type="checkbox" ng-model="checked">
    <span ng-show="checked">coché</span>
    <span ng-hide="checked">non coché</span>
    <input type="text" ng-model="data">
    <ul ng-switch on="data">
        <li ng-switch-when="toto">data==toto</li>
        <li ng-switch-when="truc">data==truc</li>
        <li ng-switch-default>data!=toto and data!=truc</li>
    </ul>
</body>
```

Template – style

Il est possible de modifier la classe et le style des éléments :

```
<body ng-init="style={color:'green'}">
<button ng-click="style={color:'red'}">Rouge</button>
<button ng-click="style={color:'black'}">Noir</button>
<span ng-style="style">zadza</span>
<pre>style={{style}}</pre>
</body>
```

```
<body ng-init="cls='green'">
<button ng-click="cls='red'">Rouge</button>
<button ng-click="cls='black'">Noir</button>
<span ng-class="cls">zadza</span>
</body>
```

Template – formulaire

```
<form name="myForm">
    <input name="input" type="text" name="input"
    ng-model="text" ng-pattern="/^[a-z]{3,6}$/" required>
    <button ng-disabled="!myForm.$valid" type="submit">
        Envoyer
    </button>
    <span class="error" ng-show="myForm.input.$error.required">
        Nécessaire
    </span>
    <span class="error"  ng-show="myForm.input.$error.pattern">
        3 à 6 lettres miniscules
    </span>
</form>
```

Template – événements

```
<body ng-controller="MyCtrl">
    <div ng-click="onEvent('click')">click</div>
    <div ng-mousedown="onEvent('down')">down</div>
    <div ng-mouseup="onEvent('up')">up</div>
    <div ng-mouseenter="onEvent('enter')">enter</div>
    <div ng-mouseleave="onEvent('leave')">leave</div>
    <div ng-mouseover="onEvent('over')">over</div>
    <input ng-model="data" ng-change="onEvent('change')">
</body>
```

```
function MyCtrl($scope) {
$scope.onEvent = function(s) { console.log(s); }
}
```

Template – les liens et les images

```
<body ng-controller="MyCtrl">
    <div ng-repeat="item in items">
        <a ng-href="{{item.img}}">
            <img ng-src="{{item.rimg}}">
        </a>
    </div>
</body>
```

```
function MyCtrl($scope) {
$scope.items = [
    {img : "a.jpg", rimg : "ra.jpg" },
    {img : "b.jpg", rimg : "rb.jpg" }
];
}
```

Template – utilisation des filtres

```
<body ng-controller="ListCtrl">
    <input ng-model="item">
    <button ng-click="add()">add</button>
    <input ng-model="query">
    <ul><li ng-repeat="e in items | filter:query">
    {{ e }}
    </li></ul>
</body>
```

```
function ListCtrl($scope) {
    $scope.items = [];
    $scope.add = function() {
    $scope.items.push($scope.item);
}
}
```

TP 3

- Créer un fichier indexTP3.html et un ficheir app.js pour tester exemples offerts dans les slides précédents pour valider les fonctionnalités offertes par AngularJs :
  - Séparer les démos par des titres expliquant la fonctionnalité testée (utiliser la balise « H1 »).
  - Le fichier app.js doit contenir les contrôleurs et les traitements des différentes démonstrations.
  - N'oublier pas d'importer les fichiers JS d'AngularJS ainsi le app.js dans la page indexTP3.hml