Plan de formation Web AngularJS

Jour 01 - Lundi - Introduction Web & AngularJS

Présentation de la formation + Tour de table

AngularJS: Kézako?!

Pourquoi utiliser AngularJS?

- Développé par une équipe d'ingénieur rattachée Google
- Framework client orienté testing
- Solution pour un développement frontend rapide
- Du HTML enrichi
- Conception simple

Les concepts d'AngularJS

Les modules

- Separation of concerns
- Paquet de fonctionnalités
- Indépendency injection

Les templates

- HTML + AngularJS Expressions
- Directive

Intéractions utilisateurs

- Fluidité de l'interface
- Click, mouse over, etc.
- Style & animations

Formulaires

- HTML5 Compliant
- Gestion avancée des validations / erreurs

Testing

- Testable à 100% (TU)
- Important : refactoring, securité, automatisation

Les données

- Dialogue server (XHR, JSONP)
- API : Custom, REST, JSON API, etc.

Installation de l'environnement de développement

Terminal

- Différents environnements
- Utilisation de commandes classiques

NodeJS

- Présentation NodeJS
- Utilisation scripting dans notre cas
- Utilisation de NVM (Node Version Manager)
- Installation Node v4 & v5

NPM

- Présentation: Gestionnaire de paquet NodeJS
- Installé avec NPM
- Paquet JavaScript (module, package.json)
- Système de versionning: semver.org
- Arbre de dépendance (npm v2.x) vs dépendance à plat (npm v3)

Bower

- Présentation: Gestion de paquet browser
- Différence avec NPM
- Système de versionning: semver.org
- Dépendance à plat!

Chrome Dev Tools

- Introduction d'articles Addy Osmani
- Brève présentation du Dév Tools (suite 2nd semaine)

Batarang

- Chrome extension
- Accès rapide au scope

$$watchers

- Chrome extensions
- Compteur de $watch présent sur la page

IDE

- Sublim Text (w/ ton of plugins)
- Webstorm
- Netbeans
- Atom
- Sublim Text
- Vi/Vim

Librairie JS

- Lodash
- jQuery
- angular-ui
- angular-translate

Jour 02 - Mardi - Concepts, Scopes, templates & filters

Structure et core concepts

Architecture MV*

- Explication **MVC**
- Explication **MVVM**/**MVW**
- Explication du **Whatever** par Igor Minar (Angular Core Dev) https://plus.google.com/+AngularJS/posts/aZNVhj355G2

Le scope

- Le scope représente la closure JavaScript qui hydrate la vue.
- C'est la sortie du contrôleur et l'entrée de la vue.

Contrôleurs

- Embarque la logique de la vue
- C'est une Fonction JavaScript
- Utilisé dans les routes, les directives/composants ou directement depuis une vue

Les vues

- Page HTML (suit les standards)
- Intégrer dans l'application via: directives/composants, ngInclude, routing
- Langage de templating (interpolation d'expression {{}})

Routing

- Pas de SPA robuste sans routing (Single Page Application)
- Router disponible:
    - ui-router (community driven)
    - angular-router
    - angular-route

Data binding

- AngularJS permet de faire du:
    - two-way data binding
    - one-way data binding
- Possibilité de faire des watchers d'expressions pour être au courant des changements

Evènements

- AngularJS met à disposition un système d'évènement simple
- Basé sur la hiérarchie des Scopes
- **$emit** (Haut) **$broadcast** (Bas) **$on** (écoute)

Directives

- Ce sont des autonomes et réutilisables
- Souvent un couple template/contrôleur
- Beaucoup de directive native dans AngularJS

Components

- Ce sont comme des directives mais très orientés AngularJS 2 & plus simple que les directives (moins de configuration possible)
- Arrivé dans AngularJS 1.5

Filters

- Pure fonction qui mute à la volée une donnée
- Des filtres natifs sont disponibles

Providers

- Factory, Service, Provider, Constant, Value
- Ce sont tous des singleton au sein de l'application AngularJS
- Embarque la logique métier de l'application
- Beaucoup de providers natif ($q, $http, etc.)

Dependency Injection (DI)

- Cela permet d'identifier l'instance d'un artefact AngularJS par un nom unique
- Gère les dépendances au sein de l'application

Les modules

- Permet de ranger le code dans un module
- Permet de cloisonner l'application
    - Gestion des dépendances
    - Tests simplifiés

La communication avec le serveur

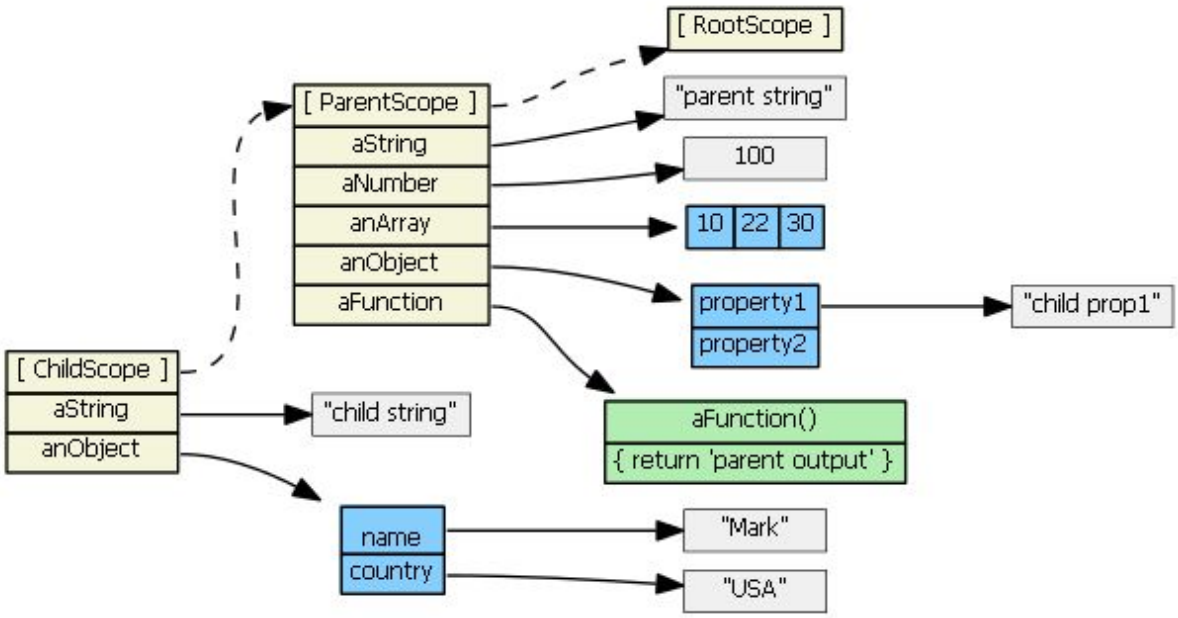
- Application côté client à besoin de données
- Communication via le protocole HTTP
- Communication avec des APIs (Json, xml, etc.)
- Utilisation de service AngularJS: $http, $resource

Le testing

- ngMock fournit beaucoup d'outils pour tester rapidement et facilement
- Deux types de testing:
    - UT (Unit Testing: Karma)
    - FT (Functional testing: Protractor)
- Démo de testing
    - Installation karma-cli
    - Installation des launcher karma, angular et jasmine
    - Initialiser Karma (karma init)
    - Ecrire des tests simples: Service, Filtre, Directives, Contrôleur
    - Lancer les tests

Le scope

L'héritage par prototypage dans JavaScript



Ex:

```
childScope.aString === 'parent string'
childScope.anArray[1] === 20
childScope.anObject.property1 === 'parent prop1'
childScope.aFunction() === 'parent output'
```

L'héritage par prototypage dans AngularJS

Un Scope peut être créé de plusieurs manière:

- Normal (héritage: ON)
- Isolé (aucun héritage)

Toujours un rootScope (accessible via le service $rootScope).

ControllerAs syntax

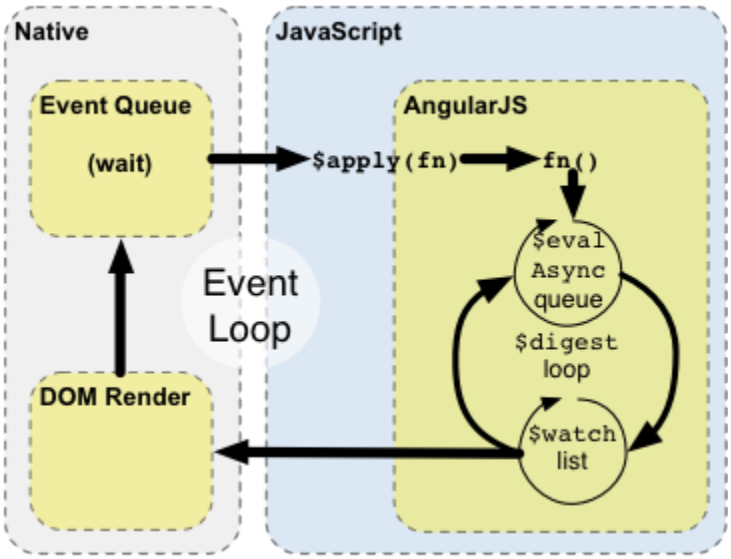
L'intéret de cette syntax est d'éviter l'héritage et de pouvoir accéder à tous les éléments.

Héritage des types primitifs

/!\ ng-include, ng-switch & ng-repeat et l'utilisation des cas primitifs (String, Number)

[DEMO/day_02/step_01]

Le cycle de vie



Dirty-checking => $digest
Update AngularJS Application => $apply

$apply et $digest

AngularJS permet le two-way data binding.
Cela veut dire que l'on peut écouter tout changement sur un élément du Scope avec la méthode $scope.$watch.

[DEMO/day_02/step_02]

Quand appeler $apply manuellement

Cette méthode est à appeler quand on veut utiliser du code Non AngularJS et impacter AngularJS.
Ex: tout code asynchrone setTimeout, request AJAX
[DEMO/day_02/step_03]

Performance

Les humains sont:

- Lent : <50ms est imperceptible et peut être considéeé comme "instantanné"
- Limité : Ne pas afficher plus de 2000 informations sur une même page (mauvaise UI)

Benchmarks (jsperf):
http://jsperf.com/angularjs-digest/6 => 10k watchers

L'avenir du dirty-checking

Object.Observe prévu pour ES7

Lien utile

https://www.youtube.com/watch?v=Mk2WwSxK218

Les templates

Les directives

ng-if, ng-show, ng-class, ng-src, ng-click, ng-copy, ng-paste, ng-href, ng-switch, etc...
[DEMO/day_02/step_04]

Les formulaires

HTML5 Validation API

HTML5 Attribute	ng Attribute	Registered Error
required="bool"	ng-required="..."	ngModel.$error.required
minlength="number"	ng-minlength="number"	ngModel.$error.minlength
maxlength="number"	ng-maxlength="number"	ngModel.$error.maxlength
min="number"	ng-min="number"	ngModel.$error.min
max="number"	ng-max="number"	ngModel.$error.max
pattern="patternValue"	ng-pattern="patternValue"	ngModel.$error.pattern

<input type="...">	Registered Error
type="email"	ngModel.$error.email
type="url"	ngModel.$error.url
type="number"	ngModel.$error.number
type="date"	ngModel.$error.date
type="time"	ngModel.$error.time
type="datetime-local"	ngModel.$error.datetimelocal
type="week"	ngModel.$error.week
type="month"	ngModel.$error.month

[DEMO/day_02/step_05]

Jour 03 - Mercredi - Les filtres, le routing, les directives/composant

Les filtres

Les filtres natifs AngularJS

Les filtres sont des “pure functions” qui permette de muter une variable passée en paramètre.
Beaucoup de filtre fournis avec AngularJS : currency, date, filter, json, limitTo, lowercase, number, orderBy, uppercase

[DEMO/day_02/step_06]

Création de filtre personnalisé

On crée un filter comme on crée un controlleur.
app.filter(‘NameFilter’, function() {});
Un filtre est une fonction, qui prend x paramètre en entré et retourne une valeur.
[DEMO/day_02/step_07]

TP Fil Rouge (Partie 1)

Prérequis

Création d’un projet depuis zéro

Création d’une application AngularJS de base

Gestion d’utilisateur

Le routing

Routing AngularJS

Documentation officiel : https://docs.angularjs.org/api/ngRoute
Manque de fonctionnalité.
Module externe à angular (**angular-route**).
Inclusion via une dépendance module [‘ngRoute’]
Expose des composants AngularJS:

- provider ($routeProvider)
- directive (ngView)
- services ($route, $routeParams)

Définition d’une route (link):

- path (pattern url)
- route config (controller, template, resolve, etc.)

[DEMO/day_03/step_03]

Routing UI-router

Projet Github : https://github.com/angular-ui/ui-router

Pourquoi utiliser ui-router ?

- Les vues imbriquées
- Les vues multiples nommées
- Liens générique vers un state (et non une url en dur)
- Decorator pour avoir des urls dynamique
- States (Au lieu des routes)

Documentation :

API Reference: http://angular-ui.github.io/ui-router/site/#/api
Guide: https://github.com/angular-ui/ui-router/wiki
FAQs: https://github.com/angular-ui/ui-router/wiki/Frequently-Asked-Questions
Sample Application: http://angular-ui.github.io/ui-router/sample/#/

Directives/componsants

Kezako

C'est la base du framework AngularJS.
Simplement : Un composant est une function JavaScript qui manipule et ajoute de nouveaux comportements au DOM HTML.
Elles peuvent être simple ou extrêmement compliqué.

Comment AngularJS détecte les directives

Le template HTML peut invoquer une directive AngularJS de quatre manières :

- En tant qu'attribut
  <span my-directive>
- En tant que class CSS

- En tant qu'element DOM
  <my-directive></my-directive>
- En tant que commentaire
  <!-- directive: my-directive expression →

Composant = Element only

La création

https://docs.angularjs.org/guide/directive
https://docs.angularjs.org/guide/component

Les options

restrict

Elle permet de définir à AngularJS le déclencheur dans le HTML.
'A' : Attribut

```
<span my-directive></span>
```

'E' : Element

```
<my-directive></my-directive>
```

'C' : Classe

```
<span class="my-directive"></span>
```

'M' : Commentaire

```
<!-- directive: my-directive -->
```

Template

Permet de défnir un template HTML directement dans la directive

```
template: '<div class=”myclass”></div>'
```

TemplateUrl

Permet de définir une url de template HTML à utiliser

```
templateUrl: 'templates/ng-sparkline-template.html'
```

Bindings

Le scope permet de définir l’héritage.
Toujours un scope isolé.
Pour mettre en place une discussion avec l’extérieur on a la possibilité de lui spécifier un comportement par valeur.

Scope (directive only)

Scope local
@ (or @attr)
Bi-directionel
= (or =attr)
Le context parent
& (or &attr)

Controller

Permet de définir un contrôleur

ControllerAs (optionnel depuis 1.5)

Permet de nommer le contrôleur sur le scope

Les étapes d’initialisation d’une directive

La compilation ($compile, compile)

Elle s’occupe de la manipulation du DOM avant d’effectuer le rendu.
Manipulation DOM = lent

Le contrôleur

S’occupe de la logique présentation.
Permet d’exposer une API pour cette directive et d’autoriser la communication avec des directives soeur et parente.

[DEMO/day_03/step_04]
[DEMO/day_03/step_05]

TP Fil Rouge (Partie 2)

Implementer un router

Ranger le code en composant

Aller plus loin: ui-router

Jour 04 - Jeudi - Les promises, les providers et le serveur

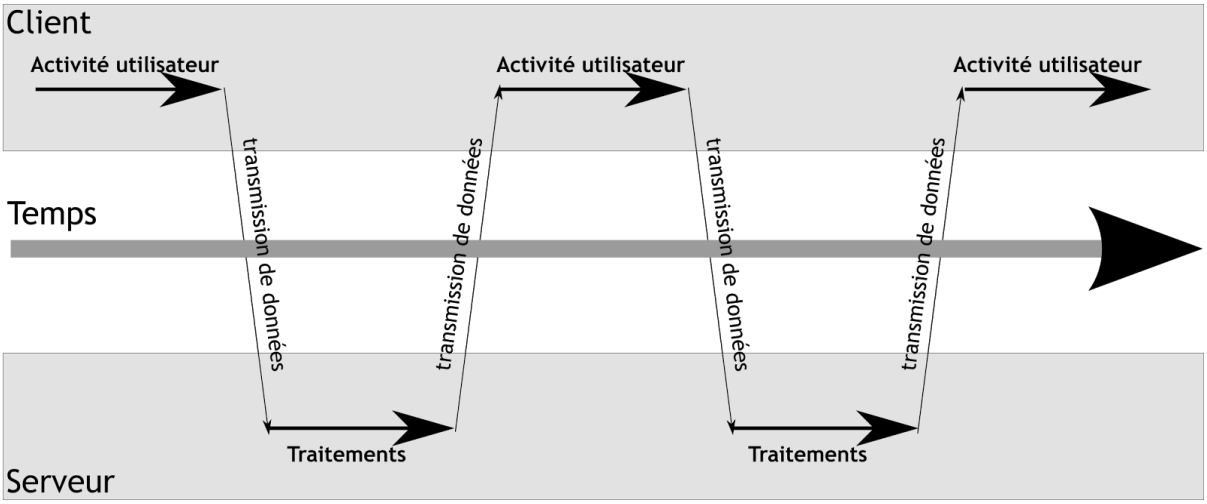
Les promesses

Kezako

Le JavaScript est un langage asynchrone, l'inverse du séquentielle.

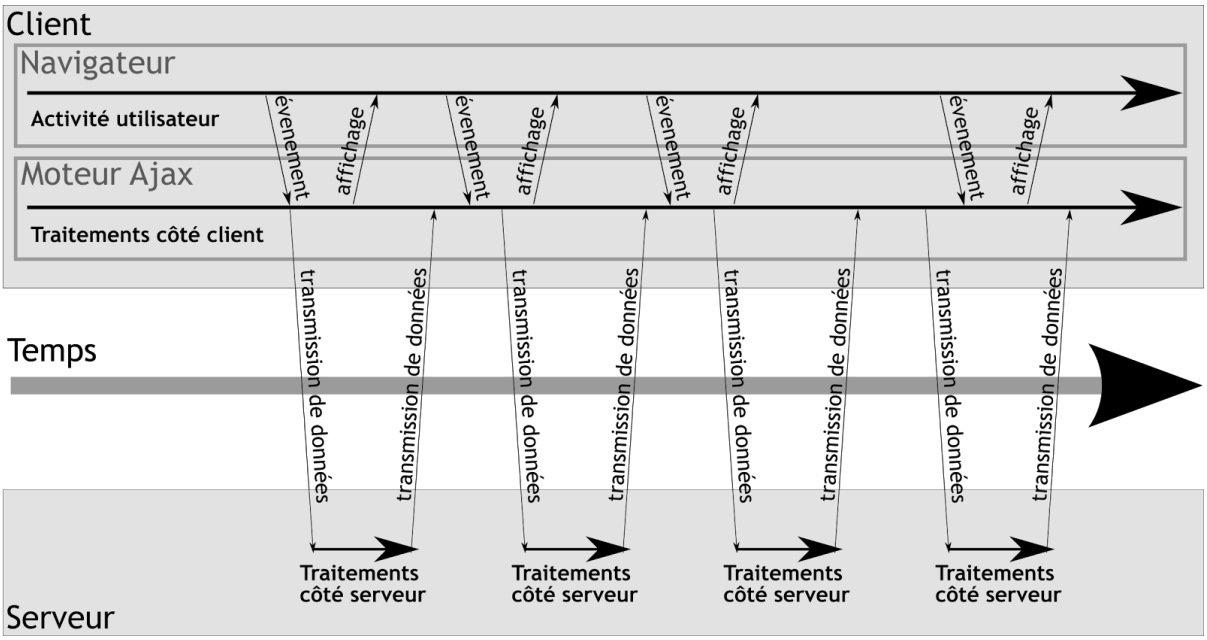
Non asynchrone

Ancienne application avec un appel serveur à chaque activité utilisateur



Asynchrone

Single Page Application avec une utilisation d'AJAX



Les promesses permettent de lancer l'exécution d'un code asynchrone et de traiter son retour au même endroit.

L'asynchrone pour une Web App

Les callbacks

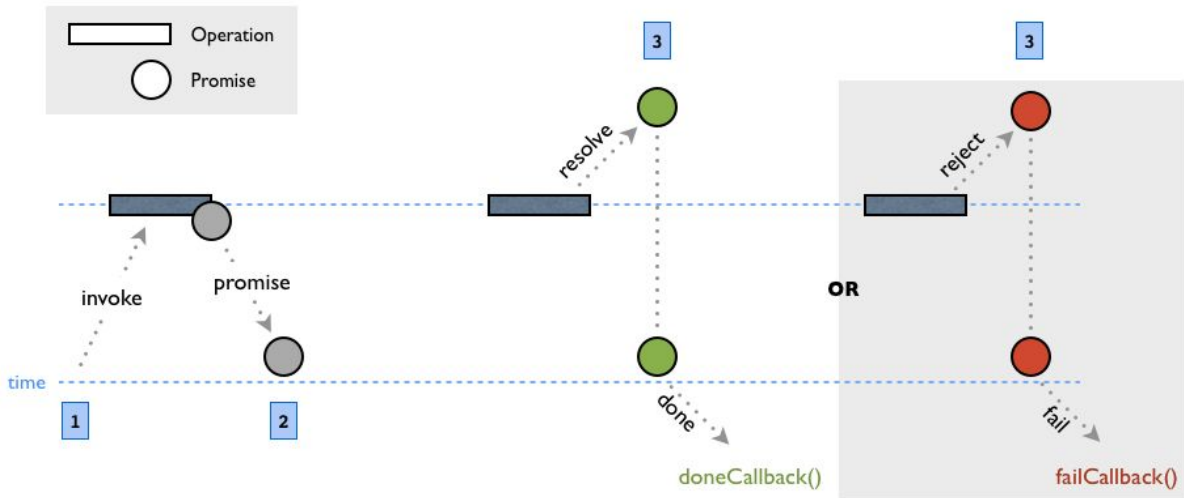
On peut gérer le code en asynchrone avec des callbacks.
Méthode utilisée jusqu'à récemment.
Peu lisible.

```
var myAsyncFunction = function (id, callback) {
  // Do some stuff
  setTimeout(function() {
    callback(); // Retour asynchrone
  }, 1000);
  return true; // Retour séquentielle
};
```

```
asncFunction1(function(err, result) {
    asncFunction2(function(err, result) {
        asncFunction3(function(err, result) {
            asncFunction4(function(err, result) {
                asncFunction5(function(err, result) {
                    // do something useful
                })
            })
        })
    })
})
```

Les promesses

Permets d'écrire en forme séquentielle comment traiter le résultat asynchrone.




```
function fetchData(id){
  return getDataFromServer(id)
          .then(transformData)
          .then(saveToIndexDB);
}
```

La gestion du résultat

Le code exécuté dans une promesses est protégé (try/catch).
On a 4 retours possible :

- resolve
- reject
- error
- notify

```
function fetchData(id){
  return getDataFromServer(id)
          .then(transformData, handleError)
          .then(saveToIndexDB);
}
```

Clôturer une promesse

Pour terminer une promesse et donner une réponse, on a deux possibilités: reject(msg) ou resolve(data).
reject(msg) refuse la promesse avec un message en string
resolve(data) valide la promesse avec la data (object, string, etc.)

La gestion des erreurs

Dans le second paramètre de la méthode then.

```
fetchData(1)
  .then(function(result){

  }, function(error){
    // exceptions dans transformData ou saveToIndexDB
  });
```

Les providers AngularJS

Kezako

Il existe 6 providers angularJS:

Provider	**Singleton**	**Instantiable**	**Configurable**
Constant	Yes	Yes	No
Value	Yes	No	No
Service	Yes	No	No
Factory	Yes	Yes	No
Decorator	Yes	No?	No
Provider	Yes	Yes	Yes

Constant

Une **constante** ne peut être modifié (même pas un décorateur).

```
angular.module('app', []);
app.constant('API_URL', 'http://api.twitter.com');
app.controller('MyController', function (API_URL) {
  console.log(API_URL === 'http://api.twitter.com');
});
```

Value

Une **value** est simplement une valeur injectable (number, function, string, etc.).

```
angular.module('app', []);
app.value('myApiUrl', 'http://api.twitter.com');
app.controller('MyController', function (myApiUrl) {
  console.log(myApiUrl === 'http://api.twitter.com';
});
```

Service

C'est un singleton (donc n'existe qu'une seule fois dans l'application).
Service applique un new de la function du service.

```
angular.module('app', []);
app.service('userService', function($http) {
  this.getUser = function () {
    return $http.get('url.com/users');
  };
});
app.controller('MyController', function (userService) {
  var self = this;
  userService.getUser().then(function(res) {
    self.users = res;
  });
});
```

Factory

Identique à service hormis le fait qu'AngularJS n'instancie pas la factory

```
angular.module('app', []);
app.factory('userService', function ($http) {
  var _privateUser = [];
  return {
    getUser: function () {
      return $http.get('url.com/users').then(function(res) {
        _privateUser = res.data;
        return _privateUser;
      });
    }
});
app.controller('MyController', function (userService) {
  var self = this;
  userService.getUser().then(function(res) {
    self.users = res;
  });
});
```

Decorator

Les décorateurs permettent de modifier ou encapsuler d'autres providers (sauf les constants).

```
angular.module('app', []);
app.value('apiUrl', "api.com");
app.config(function ($provide) {
  $provide.decorator('apiUrl', function ($delegate) {
    return $delegate + '/v2';
  });
});
app.controller('MyController', function (apiUrl) {
  console.log(apiUrl, 'api.com/v2');
});
```

Provider

C'est la plus compliqué des providers.
Permet d'avoir une phase de configuration (un **factory** configurable).

```
angular.module('app', []);
app.provider('apiUrl', function () {
  var version;
  return {
    setVersion: function (version) {
      version = version;
    },
    $get: function () {
      return url = "api.com/" + version;
    }
});
app.config(function (apiUrlProvider) {
  apiUrlProvider.setVersion('v2');
});
app.controller('MyController', function (apiUrl) {
  console.log(apiUrl, 'api.com/v2');
});
```

La communication avec le serveur

Kezako

AJAX est le coeur des Single Page Applications.
Une application client à besoin d'aller chercher des informations sur un serveur.

Communiquer avec $http

Ce service AngularJS peut être injecter en dépendance dans un contrôleur.
Il facilite la communication avec le serveur en HTTP (XMLHttpRequest ou JSONP).
Il est compatible avec tout les verbe HTTP : GET, POST, HEAD, DELETE, PUT, JSONP
Ce service utilise les Promises pour faciliter l'asynchrone.

```
$http.get('url/someUrl')
  .success(function(data, status, headers, config) {  })
  .error(function(data, status, headers, config) {  })

$http.post('/someUrl', {msg: 'Message sent to the server!'})
```

```
  .success(function(data, status, headers, config) {  })
  .error(function(data, status, headers, config) {  })

// Disponible : $http.get, $http.head, $http.post, $http.put, $http.delete,
$http.jsonp, $http.patch
```

Aller plus loin

https://docs.angularjs.org/api/ng/service/$http

Communiquer avec une API de type REST avec $resource

REST: http://goo.gl/FvfGMV

Pour résumer, REST permet de suive une convention de nommage des routes pour accéder aux ressources. Ce qui nous permet de faire des méthodes génériques de dialogue avec le serveur.

On utilise les méthodes suivantes : Get, Save, Query, Remove et Delete.

Le service $resource permets de facilement les mettre en place.

```
angular.factory("userService", function($resource) {
  return $resource('/users/:userId', {userId: '@id'});
});
```

1. Service **/users** est appelé
2. **:userId** sera remplacé par la valeur passé lors de l'appel de la méthode Get
3. @id permet de savoir quel champs mettra à jour le userId

Utilisation

Récupere une liste (query)

Méthode statique, un tableau d'objet :

```
$scope.users = [
  {userObject1},
  {userObject2},
  {userObject3}
];
```

Méthode dynamique avec la factory userService :

```
$scope.users = userService.query();
```

Récupérer une entrée (get)

```
$scope.user = userService.get({userId: 5});
```

Créer une entrée (save)

```
var user = new userService();
user.name = 'my name';
user.$save();
```

Modifier une entrée

Si une API RESTful, il faut définir une méthode *update* qui utilise la méthode PUT.

```
angular.factory("userService", function($resource) {
  return $resource('/users/:userId', {userId: '@id'}, {'update', {method: 'PUT'}
});
});
```

```
//
$scope.user = userService.get({userId: 5});
$scope.user.name = 'new name';
$scope.user.$udapte();
```

TP Fil Rouge (Partie 3)

Ecrire un service User

Mettre à jour l'application

Utiliser les données d'un serveur

Architecture composant

Jour 05 - vendredi - Pratiques, Questionnaire et Mini-TPs

TP Fil Rouge

Ordonner le code

Allez plus loin

Questionnaire

Questions fermées

Questions ouvertes

Explications et rappels sur les réponses

Mini-TPs

Formulaire d'achat

Login

Menu app

Timer app

Jour 06 - lundi - DévTool & Pratique

DévTools

Introduction DévTools

Interface globale

Panel Element

Panel Console

Panel Source

Panel Network

Panel Timeline, Profil & Audits (introduction rapide)

Panel Resources

Panel AngularJS (Batarang & $$Watchers)

Débug pas-à-pas!

TP Fil Rouge

Jour 07 - mardi - Firebase

Firebase + AngularJS

- Solution SaaS (https://www.firebase.com/pricing.html)
- Présentation globale
- DEMO (jsfiddle + gh-pages)

La forge

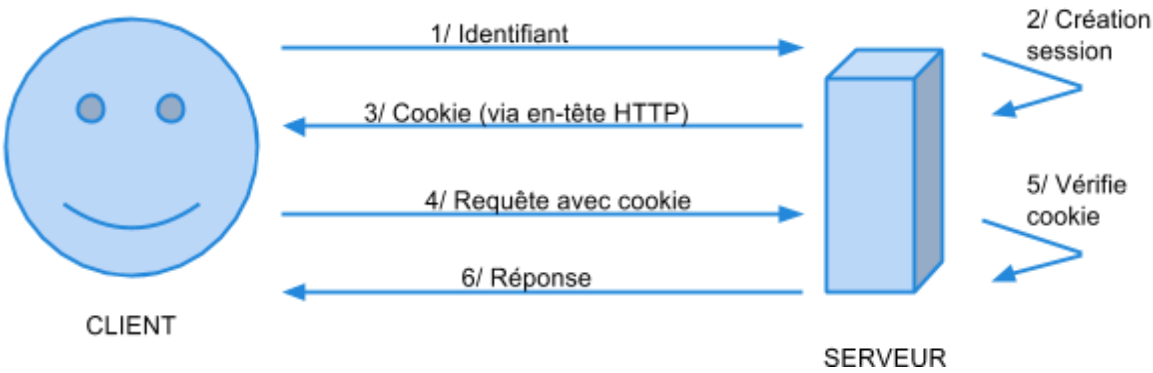
- IHM user-friendly
- Importer/Exporter les données
- Métriques d'utilisation
- Sécurité

Les données

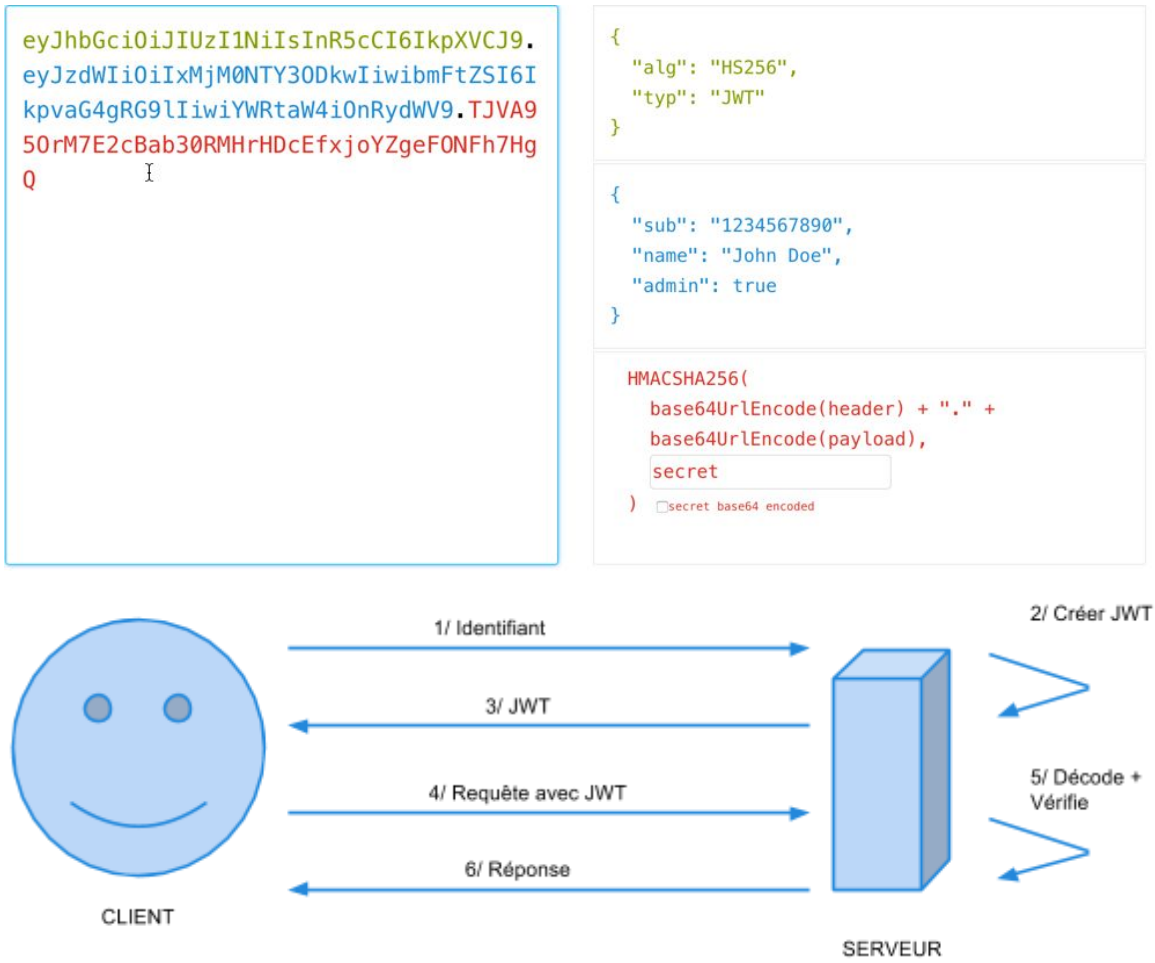
- Représenter sous forme hierarchique
- Le chemin de la donnée est disponible dans l'url

L'authentification

- Mise en place assez complexe
  - Nécessite ses propres serveurs
  - Suivre l'actualité pour les failles de sécurité
  - Synchronisation hors-ligne lourde à mettre en place
  - Intégration des enregistrement via réseau sociaux
- L'authentification via cookie



- L'authentification via JWT
  JWT : JSON Web-Token (http://jwt.io/)



- Authentification via OAuth

Email / Password

```
var ref = new Firebase("https://myfirebase.url");
ref.authWithPassword({
    email: "john@doo.com",
    password: "monpassword"
}, function (error, authData) {
    // Utilisateur authentifié
});
```

Email / Password

```
var ref = new Firebase("https://myfirebase.url");
ref.onAuth(function (authData) {
    if (authData) {
        // Utilisateur authentifié
    } else {
        // Utilisateur non connecté
    }
});
```

Email / Password

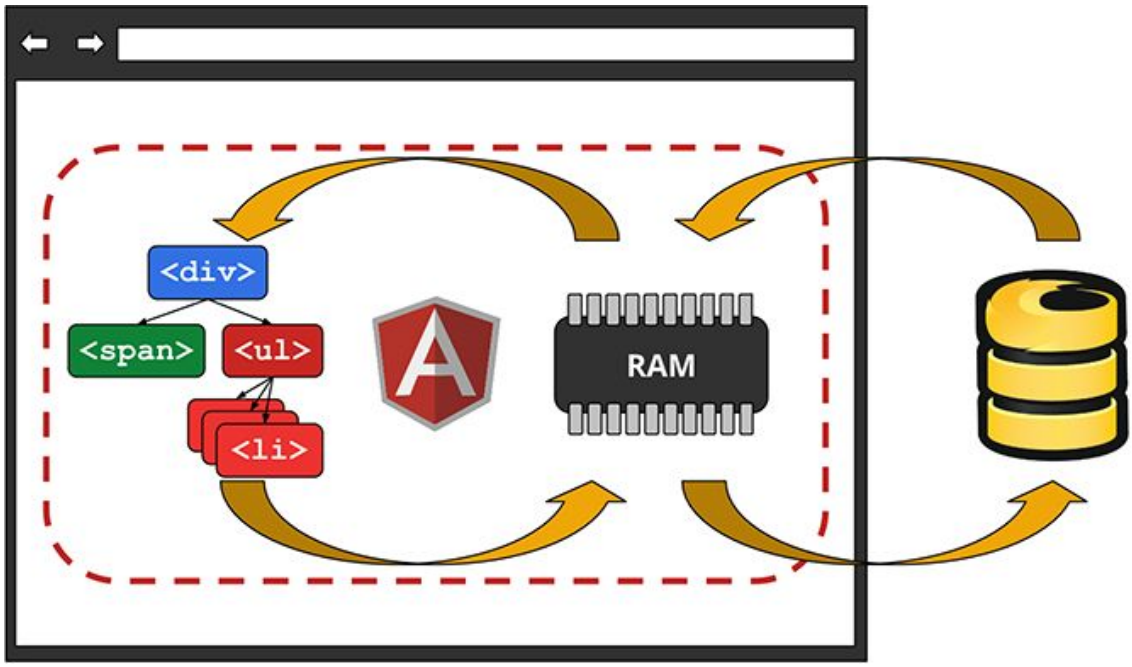
```
var ref = new Firebase("https://myfirebase.url");
ref.createUser({
    email: "john@doo.com",
    password: "monpassword"
}, function (error, authData) {
    // Utilisateur créé
})
```

Mode Hors-ligne

- Semi-automatique
- https://www.firebase.com/docs/web/guide/offline-capabilities.html

AngularFire

- Firebase + AngularJS = angularfire
  Homepage : https://goo.gl/JFiYp6
  Doc : https://goo.gl/z3J6fN
    - Data binding entre la vue et la base de données
    - Authentification



- $firebaseArray, $firebaseObject & $firebaseAuth

TP Fil Rouge

Mise en pratique de firebase sur le TP Fil Rouge

Jour 08 - mercredi - Les animations, Git, Grunt & TPs

Les animations dans AngularJS

- Beaucoup de MAJ dans AngularJS 1.4
- CSS & JavaScript

Les directives natives compatibles

Les animations CSS

- AngularJS synchro avec les animations CSS
- Séquencer des animations
- Classes css

Les animations en JS

- Artifact AngularJS: angular.animation
- Synchronisation avec le CSS

Les services natifs

- $animate
- $animateCss

L'anchoring

- ng-animate-ref
- Les routes

TP Fil Rouge (Partie 5)

Git

Installation

Création d'un nouveau repository

Cloner un repository

Le processus de Git

- Trois arbres de modifications
    - Working Directory
    - Index
    - HEAD

Ajouter/Commiter des fichiers

Pousser des changements sur le serveur

Les branches

Mettre à jours son repository

Merger une branche

Les tags

Les logs

Remplacer des changements locaux

Astuces

Liens

GUI

Guides

SaaS

Grunt

Introduction

Développement d'un environnement de dév + build

- copy
- jslint
- csslint
- concat
- uglify
- ngTemplates
- clean
- watch

Jour 09 - jeudi - Formulaire & directive/composant avancé

Formulaires

Création d'un formulaire

Les classes CSS

Le FormController

Le ngModelController

Directive/Composant avancé

L'option require

$formatters

$parsers

$validators

$asyncValidators

Utiliser ngModelController

Utiliser ngModelOptions

Utiliser ngMessages

Pratique

Jour 10 - vendredi - Liens, Astuces & Questions ouvertes