Démarrer avec Angular 1.4

- Accueil: Documentation support de formation
- Chapitre suivant: Pour aller plus loin avec Angular 1.4 (niveau intermédiaire)

Pré-requis (niveau débutant)

Cette formation est destinée à un public débutant avec Angular, mais ayant déjà quelques bases de Web et de javascript.

Sommaire

Versions d'Angular

- Sommaire: Démarrer avec Angular 1.4
- Page suivante: Démarrage de l'application

Branche 1.x

La branche 1.4.x est à ce jour la version stable d'angular, sur laquelle s'appuie cette formation.

- **1.3.x** - 2014-10-13: fin de support ie8 (mais angular ie8-builds), bindToController, ngMessages…
- **1.4.x** - 2015-05-27: meilleures animations, i18n, cookies
- **1.5.0-beta.1** - 2015-09-29: migration, component, routeur

Ressources : * exploring angular 1.3 * angular 1.4 release * angular 1.5 milestone * angularjs ie8 builds for angular > 1.3

Branche 2.x

La branche 2.x est une ré-écriture d'angular basée sur TypeScript, ES6 et les Web components.

Elle comporte une nouvelle syntaxe pour les templates et est incompatible avec la version 1.x.

A noter aussi le support d'un rendering côté serveur, permettant de servir des pages statiques sans javascript.

- **2.0.0-alpha.46** - 2015-11-11: dernière version alpha (developer preview, instable)

Ressources : * blog officiel d'angularjs * angular.io documentation * angular github repository ## Démarrage de l'application angular

- Sommaire: Démarrer avec Angular 1.4
- Page précédente: Versions d'angular
- Page suivante: Modules

Chargement des scripts dans (index.html):

```
<!-- dépendance Angular -->
<script type="text/javascript" src="bower_components/angular/angular.js"> </script>

<!-- votre application -->
<script type="text/javascript" src="app.js"></script>
```

Déclaration de l'application dans le fichier principal (app.js):

```
angular.module('tw.practice', []);
```

Chargement de l'application (index.html):

```
<html ng-app="tw.practice">
    <!-- your application here -->
</html>
```

L'application peut-être chargée sur l'ensemble de la page, ou sur une balise quelconque. Il est donc possible (mais très peu commun) de charger plusieurs applications angular sur une même page.

Ressources : * bootstrap angular # Modules et organisation des fichiers

- Sommaire: Démarrer avec Angular 1.4
- Page précédente: Démarrage de l'application
- Page suivante: Templates

Composants

Découpage en fichiers: 1 fichier par composant: * contrôleur * template * service * directive * filter * feuille de style

Pour chaque fichier: * IFFE (Immediately Invoked Function Expression) * mode strict

```
(function () {
    'use strict';
    // ...
})();
```

Regroupage des composants lié à la même fonctionnalité par dossier:

Resources:

- johnpapa angular style guide: single responsibility

Modules

Composants regroupés en modules: * 1 fonctionnalité (ou ensemble) = 1 module * 1 module: scripts, templates, et styles * meilleure isolation des composants

Possibilité d'ajouter des fonctionnalités sans faire exploser la complexité de l'architecture

Déclaration d'un module

Déclaration du module dans le fichier principal du module (profile.js) :

```
angular.module('tw.practice.profile', []);
```

Injection du module dans l'application (app.js):

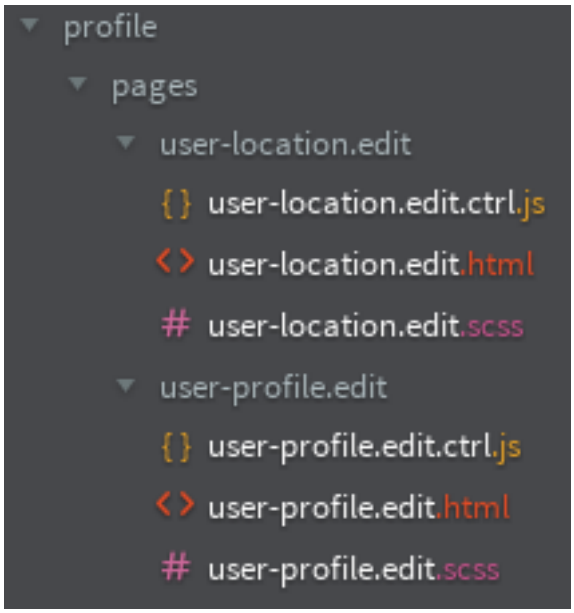

Figure 1: files tree

```
angular.module('tw.practice', ['tw.practice.profile']);
```

Le contrôleur est déclaré dans le module (profile.user-edit.ctrl.js):

```
angular.module('tw.practice.profile')
    .controller('TwProfileUserEditController', Controller);
```

Tous les fichiers doivent être chargés dans le fichier html:

```
<link rel="stylesheet" href="profile/pages/user-profile.edit/user-profile.edit.css">
...
<script src="profile/profile.module.js"></script>
<script src="profile/pages/user-profile.edit/user-profile.edit.ctrl.js"></script>
```

Séquence de chargement de l'application: * les modules sont déclarés * les contrôleurs sont chargés dans les modules * les services ($scope, $log) sont injectés (inversion de contrôle) * les dépendances entre modules sont résolues * l'application peut démarrer

Templates et bi-directional data-binding

- Sommaire: Démarrer avec Angular 1.4

- Page précédente: Modules
- Page suivante: Contrôleurs

Angular fourni un mécanisme de synchronisation entre le modèle et la vue, appelé bi-directional data-binding.

Exemple (index.html):

```
<div>
    <label>Entrez votre prénom</label>
    <div>
        <input type="text" ng-model="firstName">
    </div>
</div>
<p ng-if="firstName">Bonjour {{ firstName }}!</p>

<ul>
    <li>ng-model="firstName"</li>
    <li>{{ firstName }}</li>
    <li>Bi-directional data-binding</li>
    <li>ng-if="firstName"</li>
</ul>
```

La valeur du champ firstName est synchronisée avec le modèle, et est donc automatiquement mise à jour dans template.

- Ressources :
- databinding guide

Contrôleurs

- Sommaire: Démarrer avec Angular 1.4
- Page précédente: Templates
- Page suivante: Filtres

Introduction

Déclaration du controller dans le module:

```
angular.module('tw.practice').controller('TwProfileUserEditController', Controller);

function Controller() {
    ...
}
```

Préférer une fonction nommée (plutôt qu'une fonction anonyme).

Injection des services via les paramètres du contrôleur:

```
/** @ngInject */
function Controller($scope, $logger, MyCustomService) {
     ...
}
```

L'annotation @ngInject permet à ngAnnotate de détecter ces fonctions comme des services à annoter avant la minification, pour ne pas casser l'injection.

La forme transformée ressemblera à la forme descriptive comportant le nom des services:

```
function Controller(['$scope', '$logger', 'MyCustomService',
    function($scope, $logger, MyCustomService) {
        // ...
    }
]);
```

Ceci peut aussi être généré par .

Resources:

- controller guide
- ng-annotate

Modèle bindé sur le scope (non-recommandé)

Bien que ce ne soit plus le pattern recommandé, il faut connaitre son existence pour comprendre les nombreux exemples qui l'utilisent sur le Web.

Affectation d'un attribut dans le scope:

```
function Controller($scope) {

    // public attributes
    $scope.total = 10;
    $scope.user = {
        firstName: 'John'
    };
}
```

Association d'une méthode au scope:

```
function Controller($scope) {

    // public methods
    $scope.openDetails = openDetails;
    $scope.getFullName = getFullName;

    function getFullName(){
        ...
    }

    function openDetails(){
        ...
    }

}
```

Utilisation du modèle dans le template:

```
<div class="body-area" ng-controller="TwProfileUserEditController">

    <div>
        <label>Entrez votre prénom</label>
        <div>
            <input type="text" ng-model="user.firstName">
        </div>
    </div>

    <p ng-if="user.firstName">Bonjour {{ user.firstName }}!</p>

</div>
```

Problèmes avec contrôleurs & $scope: * template fortement couplé au $scope * tests plus compliqués à réaliser * migration difficile vers un autre framework (ex: angular v2 n'a pas de $scope) * possible conflits entre les variables de contrôleurs imbriqués

Modèle bindé sur le controlleur: controller-as (recommandé)

La syntaxe 'controller-as' est apparu avec angular 1.2, puis a été étendue ensuite: * le "viewModel" est lié au contrôleur, pas au scope * la vue associe le contrôleur à une variable

Affectation d'un attribut dans le scope:

```
function Controller() {
```

```
    // view model
    var vm = this;

    // public attributes
    vm.user;

    ...
    return vm;
}
```

Association d'une méthode au scope:

```
function Controller() {

    // view model
    var vm = this;

    // public methods
    vm.getFullName = getFullName;
    vm.reset = reset;

    function getFullName() {
        return vm.user.firstName + ' ' + vm.user.lastName;
    }

    function reset(){
        vm.user = {
            firstName: 'John',
            lastName: 'Smith'
        };
    }

    ...
    return vm;
}
```

Utilisation du modèle dans le template:

```
<div class="body-area" ng-controller="TwProfileUserEditController as vm">

    <div>
        <label>Entrez votre prénom</label>
        <div>
            <input type="text" ng-model="vm.user.firstName">
```

```
        </div>
    </div>

    <p ng-if="vm.getFullName()">Bonjour {{ vm.getFullName() }}!</p>

   <div><input type="button" value="Reset" ng-click="vm.reset()"></div>

</div>
```

Exemple avec des contrôleurs imbriqués:

```
<div ng-controller="MainCtrl as main">
    {{ main.title }}
    <div ng-controller="AnotherCtrl as another">
        Scope title: {{ another.title }}
        Parent title: {{ main.title }}
        <div ng-controller="YetAnotherCtrl as yet">
            Scope title: {{ yet.title }}
            Parent title: {{ another.title }}
            Parent parent title: {{ main.title }}
        </div>
    </div>
</div>
```

Le modèle n'est plus couplé au scope.

Si besoin spécifique, on peut quand même faire appel au scope:

```
$scope.$on('$destroy', function onDestroy() {
  // libération des resources, log, requête, message...
});
```

Resources:

- digging-into-angulars-controller-as-syntax
- controller guide
- scope guide

Contrôleur - bonnes pratiques

Quelques conseils à suivre:

- préférer les fonctions nommées

- déclarer les attributs et méthodes au début du controlleur
- préférer la syntaxe controller-as

Créez votre propre style-guide en vous inspirant de ceux-ci:

- johnpapa angular style guide
- mgechev angular style guide ## Les filtres
- Sommaire: Démarrer avec Angular 1.4
- Page précédente: Contrôleurs
- Page suivante: Services

Introduction

Les filtres permettent de transformer une variable afin de l'afficher. Par exemple, on peut formater une devise, filtrer une collection, compter le nombre d'éléments…

Formatage des devises:

```
{{ vm.user.salary | currency }}
```

Formatage des dates:

```
{{ vm.user.birthdate | date:'yyyy-MM-dd' }}
```

Resources:

- filter guide
- currency filter
- date filter

Tableaux: pagination et filtrage

Pagination avec limitTo

Le filtre limitTo:N:i permet d'extraire "N" éléments d'un liste, à partir de l'index "i".

On peut ensuite itérer sur la collection via ng-repeat.

Exemple:

```
<div ng-repeat="item in items | limitTo:10:30">{{ item }}</div>
```

Resources:

- limitTo filter
- ngRepeat directive

Filtrage du contenu

Le filtre filter:attributeName:value:strict permet de filtrer une collection en fonction de la valeur d'un attribut.

Si strict vaut true, la comparaison est exacte, sinon il s'agit d'un "contains".

Exemple:

```
<div ng-repeat="item in items | filter:{name:vm.name}">{{item}}</div>
```

Resources:

- 'filter' filter

Tri du contenu

Le filtre orderBy:attributeName:reverse permet de filtrer une collection en fonction de la valeur d'un attribut.

Utilisation du filtre "orderBy" pour trier le contenu:

```
<div ng-repeat="item in items | orderBy:date:true">{{item}}</div>
```

Resources:

- https://code.angularjs.org/1.4.7/docs/api/ng/filter/orderBy ## Les services
- Sommaire: Démarrer avec Angular 1.4
- Page précédente: Filtres
- Page suivante: Directives

Introduction

Les services sont des composants permettant d'isoler des fonctions spécifiques: * single responsability * injection de dépendances entres services, injection des services dans les contrôleurs

Il y a 4 façons de déclarer un service: * angular.module('tw.practice').**value()** pour déclarer une constante * angular.module('tw.practice').**service()** pour déclarer un service via une méthode * angular.module('tw.practice').**factory()** pour déclarer un service via une factory * angular.module('tw.practice').**provider()** pour déclarer un service via un provider (configurable)

Injection et utilisation dans un contrôleur:

```
function Controller(twSecurityService) {

    var user = twSecurityService.login('admin', 'pass');
    // ...
```

Resources:

- angular services guide
- les services angularjs
- Stack overflow: angular.service vs angular.factory
- Stack overflow: AngularJS : Service vs provider vs factory
- Service vs Factory - Once and for all

Service constante: .value()

La méthode .value() permet de déclarer une constante:

```
angular.module('tw.practice').value(twApiPrefix, '/api');
```

Service déclaré via une fonction constructeur: .service()

La méthode .service() permet de déclarer un constructeur pour notre service:

```
angular.module('tw.practice.security').service('twSecurityService', twSecurityService);

function twSecurityService($log, $http) {

    // private attributes
    var currentUser = null;
```

```
    // public methods
    this.login = login;

    function login(login, password) {
        // ...
    }
}
```

S'agissant d'une fonction constructeur et permet une migration simple vers une classe ES6.

Service déclaré via une fonction factory: .factory()

La méthode .factory() permet de déclarer un objet représentant notre service:

```
angular.module('tw.practice.security').factory('twSecurityService', twSecurityService);

function twSecurityService($log, $http) {

    var service = {};

    // private attributes
    var currentUser = null;

    // public methods
    service.login = login;
    service.logout = logout;
    service.getCurrentUser = getCurrentUser;

    function login(login, password) {
        // ...
        currentUser = buildSecurityUser(token)
        return currentUser;
    }

    // ...

    return service;
}
```

Ecriture très proche de service(). Adopter l'un ou l'autre dans l'application, mais pas les 2.

Service déclaré via un provider (configurable)

Exemple de configuration d'un service.

Déclaration du service provider:

```
var apiModule = angular.module("apiModule", []);

apiModule.provider("apiService", function() {
    var provider = {};
    var config   = { baseUrl : "/api" };

    provider.configureBaseUrl = function(baseUrl) {
        config.baseUrl = baseUrl;
    }

    provider.$get = function() {
        var service = {};

        service.getResources = function(name) {
            // TODO query the api server
        }

        return service;
    }

    return provider;
});
```

Configuration:

```
apiModule.config( function( apiServiceProvider ) {
    apiServiceProvider.configureBaseUrl("/restapi/v2");
});
```

Utilisation

```
function(apiService) {
    apiService.getResources('products');
});
```

Les méthodes value(), service() et factory() sont en fait des formes simpliées de provider().

Resources:

AngularJS Modularization & Dependency Injection: configuring a provider

Tests unitaires de services

Description d'un test Jasmine:

```
it('medium password (8 characters) should result to strongness 5', function () {
   var strongness = passwordService.checkStrongness('12345678');
    expect(strongness).toEqual(5);
});
```

Exécution automatique en mode watch (swiip):

```
gulp test:auto
...
PhantomJS 1.9.8 (Linux 0.0.0) form password service long password (> 12 characters) shou
    Expected undefined to equal 10.
        at /home/toub/Documents/dev/formation-angular/2015-09_rabat/angular-practice-1.4
PhantomJS 1.9.8 (Linux 0.0.0): Executed 3 of 3 (3 FAILED) ERROR (0.001 secs / 0.01 secs)
```

Resources:

- unit-testing guide
- Karma: moteur d'exécution de vos tests dans le browser
- Jasmine: descripteur de tests
- tâches gulp du générateur swiip ## Les directives
- Sommaire: Démarrer avec Angular 1.4
- Page précédente: Services
- Page suivante: Routeur

Introduction

"*At a high level, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler ($compile) to attach a specified behavior to that DOM element (e.g. via event listeners), or even to transform the DOM element and its children.*"

En résumé, des composants réutilisables permettant de modifier le contenu d'un élément et/ou son comportement.

Exemples: ng-model, ng-if, bs-datepicker…

Resources:

- angular directive guide

Directives standard

Quelques directives standard utilisables dans les templates:

- ng-if: création d'un élément dans le DOM
- ng-switch: ng-if version switch/case
- ng-show/ng-hide: affichage/masquage d'un élément du DOM
- ng-click: exécute une expression suite à un click
- ng-change: exécute une expression suite à un changement dans un champs de formulaire
- ng-model: association d'un attribut du modèle au champ input
- ng-class: assignation d'une ou plusieurs classes à un élément
- ng-repeat: parcours d'une collection

Ressources : * liste des directives standard

Directives: déclaration

Déclaration avec controllerAs:

```
angular.module('tw.practice.form').directive('someDirective', function() {
    return {
        templateUrl: 'app/some.directive.html',
        controllerAs: 'vm',
        scope: {
            param: '=',
            callback: '&amp;',
            value: '@'
        },
        bindToController: true,
        controller: SomeDirectiveController
    };
}
```

Utilisation:

```
<div some-directive param="vm.name" callbackmethod="vm.doIt" value="5"></div>
```

Resources:

- Directive avec controller as, bindToController et tests

Directives: transclusion

Permet d'insérer du code à l'intérieur du template de la directive.

Activation via l'attribut "transclude":

```
angular.module('tw.practice', []).directive('pane', pane);

function pane(){

    return {
            templateUrl: 'app/pane/pane.directive.html',
            controllerAs: 'vm',
            scope: {
                form: '=',
                attributeName: '@'
            },
            bindToController: true,
            controller: PaneController,
            transclude: true
        };
}
```

Puis dans le template de la directive (ici, "pane.directive.html"), à l'endroit souhaité:

```
<span ng-transclude></span>
```

Enfin, utilisation de la directive

```
<pane title="{{title}}">Le texte qui doit être inclu dans le "pane"</pane>
```

Resources:

- ngTransclude

Directives: tests

Se reporter à l'exemple tw-robustness-bar.directive.spec.js.

Chargement des modules à tester, ainsi que des templates compilés en JS (via ngHtml2JsPreprocessor):

```
// load module containing the directive
beforeEach(module('tw.practice.form'));

// load templates
beforeEach(module('ngHtml2JsPreprocessor'));
```

Création d'un nouveau scope avant chaque test:

```
beforeEach(inject(function (_$compile_, _$rootScope_, _$log_) {
    $compile = _$compile_;
    $rootScope = _$rootScope_;
    $scope = _$rootScope_.$new();
    $log = _$log_;
}));
```

Puis description du test:

```
it('tw-robustness-bar strong', function () {

    // compile the template
   var element = angular.element("<div tw-robustness-bar password=\"passModel\"></div>");
    var template = $compile(element)($scope);

    // update root scope with strong password
    $scope.passModel = 'Robu5tP@ssw0rd';
    // run a $digest cycle to update your template with new data
    $rootScope.$digest();
    // check the progressbar to contain progress-bar-success class
   expect(template.find('div').html().trim()).toContain('progress-bar-success');

});
```

Resources:

- unit-testing guide
- Directive avec controller as, bindToController et tests

Routeur

- Sommaire: Démarrer avec Angular 1.4
- Page précédente: Directives
- Page suivante: Formulaires

Routeurs disponibles

Plusieurs s'options existent:

- ngRoute: le routeur d'angular v1.x (< 1.5)
- ui-router: le routeur d'angular-ui
- angular router: futur routeur pour angular v1.5 et v2.x

Resources:

- difference between angular route and angular-ui router

ui-router - installation

Chargement du module:

```
angular.module('tw.practice', [..., 'ui.router']);
```

Chargement de la directive (index.html):

```
<div class="main-area" ui-view></div>
```

Configuration du routeur (app.js):

```
$stateProvider.state('edit-user', {
    url: "/profile/user/edit",
    templateUrl: 'profile/profile.user-edit.html',
    controller: 'TwProfileUserEditController'
});
$urlRouterProvider.otherwise('/profile/user/edit');
```

Resources:

- ui-router

ui-router - redirection

```
$state.go('edit-user', {
    userId: user._id
});
```

Resources:

$state.go

ui-router - paramètres URL

Exemple:

```
.state('edit-user', {
        url: "/edit-user/:userId",
        templateUrl: 'app/profile/profile.user-edit.html',
        controller: 'TwProfileUserEditController',
        controllerAs: 'vm'
    })
```

Resources:

- ui-router URL-Routing & $stateParams
- how to pass parameters using ui-sref in ui-router to controller

ui-router - mode html5

Définir la balise “base” dans index.html:

```
<base href="/">
```

Activation du mode html5 (app.js):

```
$locationProvider.html5Mode(true);
```

Configuration du serveur pour rediriger les urls de fallback:

Pour apache, nginx, azure IIS, express, asp.net: ui-router: how to configure your server to work with html5Mode

Pour gulp connect:

```
gulp.task('connect', function() {
  connect.server({
    port: 3000,
    root: 'app',
    livereload: true,
      fallback: './app/index.html'
  });
});
```

Pour browser sync:

```
browserSync.use(browserSyncSpa({
    selector: '[ng-app]' // Only needed for angular apps
}));
```

Les formulaires

- Sommaire: Démarrer avec Angular 1.4
- Page précédente: Routeur
- Page suivante: API REST

Principes de base

On associe au formulaire un objet spécial du modèle, puis chaque champ à un autre attribut du modèle:

```
<form name="userForm" novalidate>
    <div class="form-group required">
        <label class="control-label">First name</label>
        <input name="firstName" class="form-control" type="text"
            ng-model="vm.user.firstName">
    </div>
</form>
```

A noter:

- form name=: l'objet contenant le contrôleur du formulaire
- attribut novalidate pour désactiver la validation html5
- ng-model permet de binder le champ à l'attribut correspondant du modèle

Il est possible d'ajouter automatiquement une étoile rouge sur les champs requis, en CSS:

```
.form-group.required .control-label:after {
    content: "*";
    color: red;
    margin-left: 3px;
}
```

Se reporter au TP pour les exemples complets.

Validation

Validation - ng-messages

Directive ng-messages pour afficher les messages d'erreur (ajoutée en angular 1.3, améliorée ensuite):

```
<div ng-messages="userForm[firstName].$error" >
   <div class="error" ng-message="required">Field is required.</div>
    <div class="error" ng-message="minlength">Too short.</div>
</div>
```

Resources: * directive ngMessages * exploring angular 1.3 ngMessage

Validation - required L'attribut required active la contrainte de champ obligatoires:

```
<input name="firstName" ng-model="vm.user.firstName" required>
```

Validation - minlength/maxlength La présence de minlength/maxlength active les contraintes de taille min/max:

```
<input name="firstName" ng-model="vm.user.firstName" ng-minlength="2" ng-maxlength="10">
```

Validation - email Le type "email" active la validation du format de l'email:

```
<input name="email" type="email" ng-model="vm.user.email">
```

Validation - number Le type "number" active la validation du format nombre:

```
<input name="salary" ng-model="vm.user.salary" type="number">
```

Validation - password match La validation de 2 champs identiques peut se faire via la directive validation-match: `handlebars <input name="password" type="password" ng-model="vm.user.password"> <input name="passwordConfirmation" type="password" ng-model="vm.user.passwordConfirmation match="vm.user.password">`
Resources:

- angular-validation-match

Gestion fine de l'état du formulaire

Il est possible de contrôler plus finement quand le message est affiché via les propriétés $touched et *submitted* : “‘*handlebarsng* − *if* = *userForm.firstName*.touched || userForm.*submitted*“‘*Etdemodifierl'tatduformulairequandoncliquesur*
“‘*handlebars* < *ang* − *click* = *userForm*.setSubmitted()“>Save Reset `On peut tester dans le code si le formulaire est valide:` js
if (form.$valid) … “‘
Resources:

- form.FormController
- angular difference between pristine/dirty and touched/untouched

Date picker

Champ date avec directive bs-datepicker de angular-strap: `handlebars <label class="control-label">Birthdate</label> <input name="birthdate" class="form-control" type="text" ng-model="vm.user.birthdate" bs-datepicker data-date-format="yyyy-MM-dd" data-max-date="today" data-autoclose="1" data-date-type="iso">`
Alternatives:

- input[type=“date”] natif pour les navigateurs récents
- angular-boostrap

Resources:

- http://mgcrea.github.io/angular-strap/#/datepickers ## Rest API
- Sommaire: Démarrer avec Angular 1.4
- Page précédente: Formulaires
- Page suivante: Service $http

Introduction

REpresentational State Transfer (REST) propose de considérer que toutes les données sont des ressources auxquelles on accède via des URL uniques.

REST est composé de plusieurs éléments, décomposés en niveaux par Léonard Richardson: * niveau 1: accés aux données par ressources /users, /products… * niveau 2: utilisation des verbes HTTP (GET, POST, PATCH…) et codes de retour HTTP (200, 201, 403…) * niveau 3: enrichi les resources avec les liens associés aux ressources (contrôles hypermédia)

Exemple classique d'une API REST niveau 2 pour la resource “user”:

- **GET /users?max=100**: retourne tous des utilisateurs (maximum 100 résultats)
- **GET /users?group=admin**: retourne tous des utilisateurs appartenent au groupe “admin”
- **GET /users/:id**: retourne un utilisateur à partir de son id
- **POST /users**: créé un nouvel utilisateur
- **DELETE /entities/:id**: supprime un utilisateur à partir de son id
- **PUT /users/:id {body}**: mise à jour complète d’un utilisateur à partir de son id
- **PATCH /users/:id {body}**: mise à jour partielle d’un utilisateur à partir de son id

La documentation de l’API doit être précise, afin d’indiquer quels attributs possède la ressource, quelle est la réponse du serveur, et quelles sont les paramètres obligatoires et facultatifs pour chaque requête.

Resources: * REpresentational State Transfer sur Wikipedia * REST : Richardson Maturity Model par Xebia

Standardisation, documentation, outils

Différents projets proposent de décrire de façon standardisée une API afin de permettre de gérérer une documentation compréhensible par les humains comme par les machines.

Ces projets proposent, selon les cas: * une spécification permettant de représenter l’API dans un format standard * des outils pour éditer ou générer ce format à partir de code ou de modèle * des outils pour générer le code client ou serveur de l’API * des outils pour tester la documentation * des outils pour monitorer l’utilisation de l’API

Testez vos API avec postman.

Les principaux projets sont:: * Swagger * API Blueprint * RAML

Resources: * The data, the hypermedia and the documentation * Swagger for Node.js HTTP API Design * Strongloop Swagger compatible LoopBack framework * Writing tests with postman

Service $http

- Sommaire: Démarrer avec Angular 1.4
- Page précédente: API REST
- Page suivante: Service ng-resource

> The $http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP.

Ce service permet d'effectuer des requêtes HTTP (ajax) vers un serveur. Il ne prend pas en compte les spécificités d'une API REST.

Exemples

Récupération de la liste des utilisateurs:

```
js $http.get('/users?max=100').then(function successCallback(response) { // success var users = response.data; }, function errorCallback(response) { // error });
```

Création d'un utilisateur:

```
js $http.post('/users', {name: 'John', age: 30}).then(function successCallback(response) { // success var user = response.data; }, function errorCallback(response) { // error });
```

Resources:: * $http documentation

Service $resource

- Sommaire: Démarrer avec Angular 1.4
- Page précédente: Service $http

> A factory which creates a resource object that lets you interact with RESTful server-side data sources.

Service dédié à la communication avec une API REST (basé sur $http).

Exemples

Configuration de la resource:

```
js var User = $resource('/users/:userId', {userId:'@id'});
```

On indique ici que le paramètre "userId" dans l'URL correspond à l'attribut "id" de la ressource.

Récupération de la liste des utilisateurs:

```
js  var user = User.query({group='admin'}, function(user) {     //
success }, function(err) {     // error }); Récupération et modification
```
d'un utilisateur:

```
js  var user = User.get({userId:15}, function(user) {     // success
user.name = 'Peter';       user.$save(function(user){            //
success       }, function(err) {            // error       }); },
function(err) {     // error });
```

Création d'un utilisateur:

```
js  var newUser = new User({name: 'John', age:'10'});    newUser.$save(function(user){
// success     }, function(err) {         // error });
```

Resources:

- https://code.angularjs.org/1.4.7/docs/api/ngResource
- https://code.angularjs.org/1.4.7/docs/api/ngResource/service/$resource
- Consuming restful apis
- http://stackoverflow.com/questions/20584367/how-to-handle-resource-service-errors-in-angularjs