Formations AngularJS

Société Générale

septembre 2017

onepoint.

01

Présentation

onepoint.

Objectif

Formation AngularJS avancée

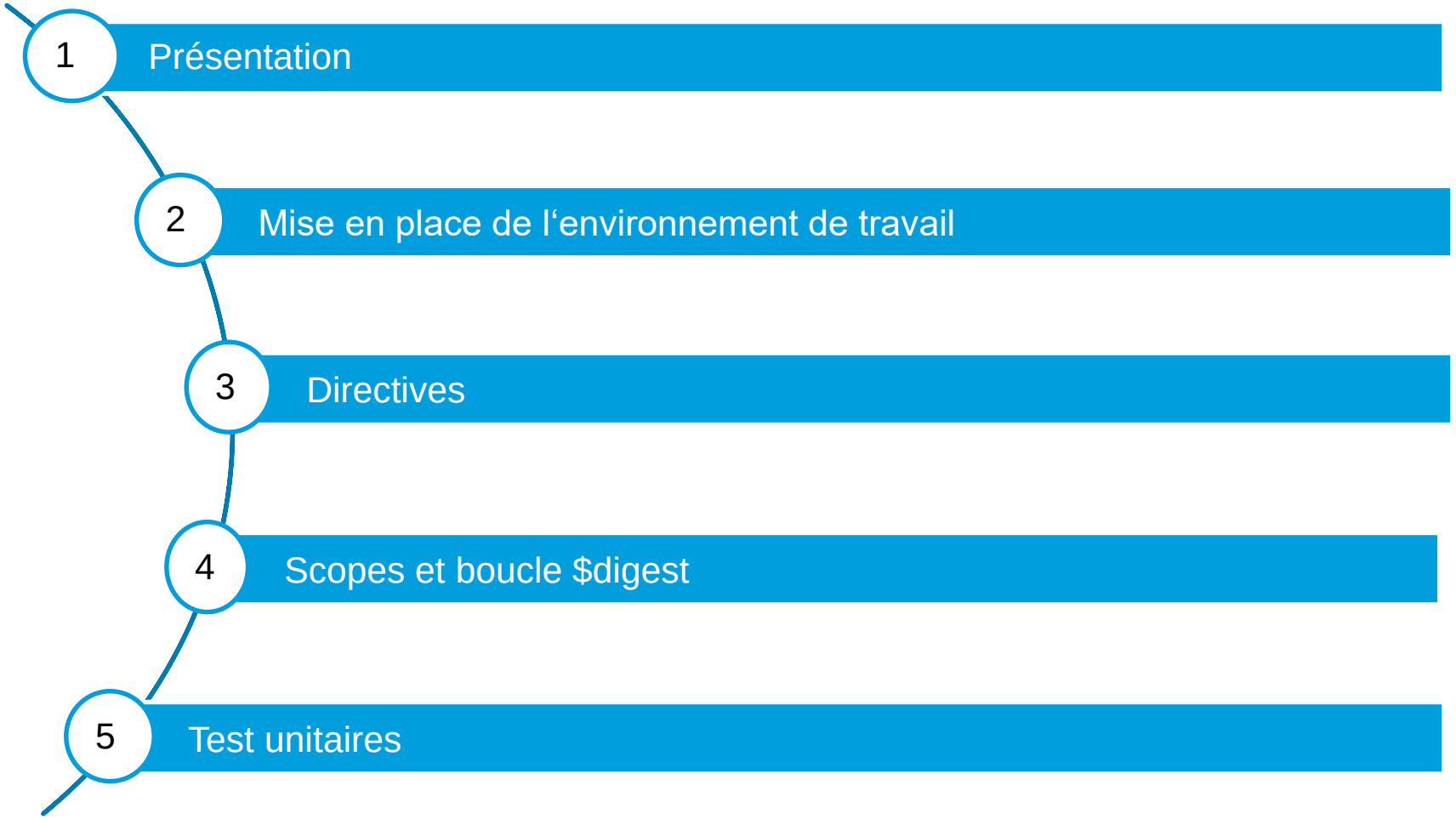
- Durée : 2 jours
- Prérequis: Connaitre les concepts de base du framework ainsi que de Node.js et des outils associés

  - Appréhender les concepts avancés du framework AngularJS
  - Mettre en place des tests unitaires automatisés
  - Mettre en place des tests automatisés de bout-en-bout
  - Optimiser une application AngularJS
  - Design et architecture d'une application AngularJS

onepoint.

Plan de la formation (jour 1)

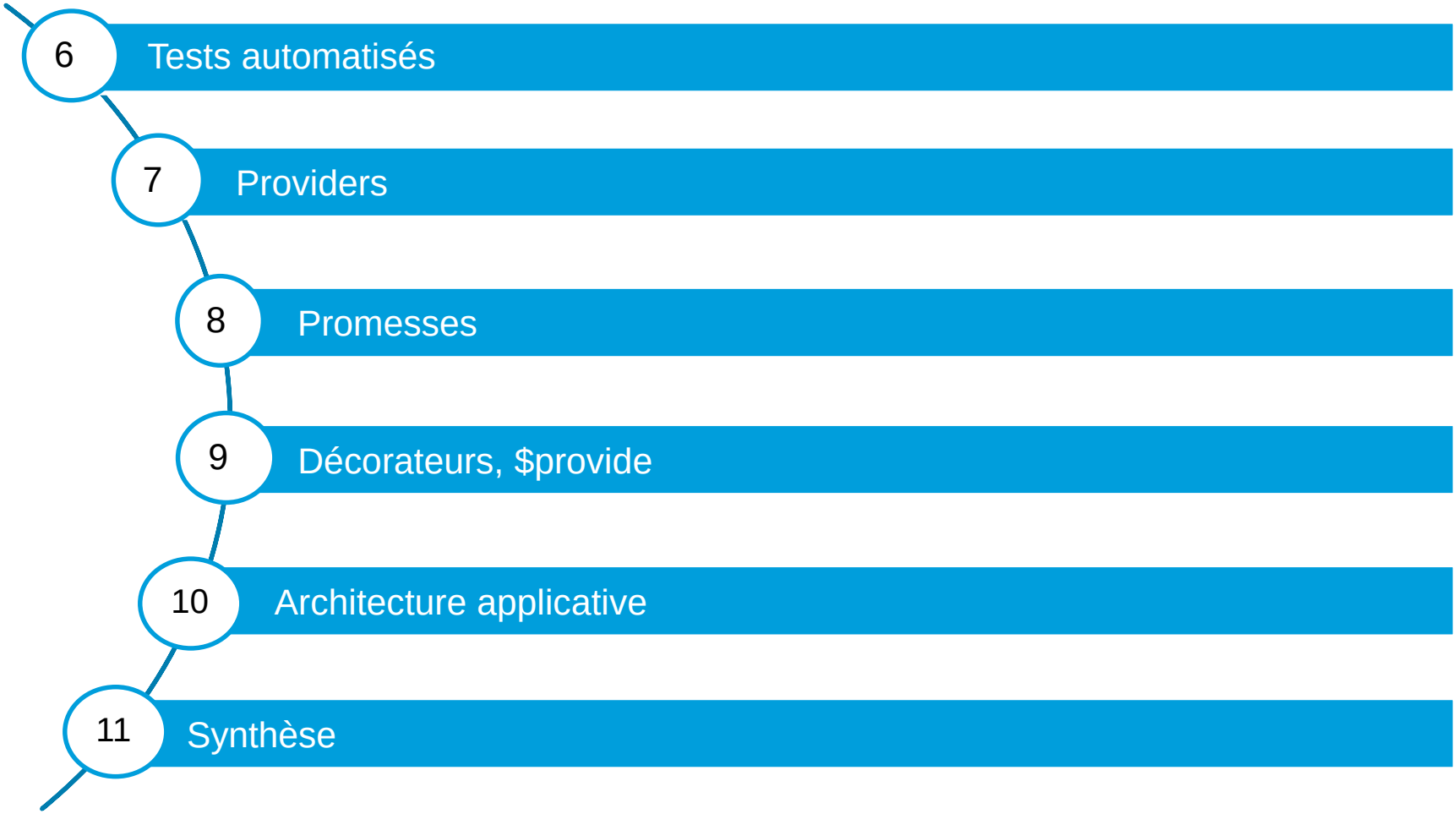
Formation AngularJS avancée



onepoint.

Plan de la formation (jour 2)

Formation AngularJS avancée



onepoint.

Vos objectifs

Formation AngularJS avancée

Vos objectifs et attentes ?

02

Environnement de travail

onepoint.

Mise en place de l‘environnement de travail

Formation AngularJS avancée

Installation des prérequis

- Node.js
  Dernière version stable (ou LTS) de Node.js: https://nodejs.org/en/
- Git
  Gestion de version Git: https://git-scm.com/
- Java JRE
  Pour l’exécution de Scelenium: https://www.java.com/fr/download/

Mise en place de l‘environnement de travail

Formation AngularJS avancée

Installation de l’application

- Dans un dossier décompresser l’archive fournie

- Installer les dépendances

Dans un terminal et à la racine du dossier de l’application :
npm install

Mise en place de l‘environnement de travail

Formation AngularJS avancée

Installation de l’application

- Démarrer l’application

Utiliser la commande suivante pour démarrer l’application:
npm start

L’application démarre sur http://localhost:8080
La page du serveur de développement: http://localhost:8080/webpack-dev-server/

Les autres scripts sont présents dans le package.json:

- npm run test
- Npm run e2e

Mise en place de l‘environnement de travail

Formation AngularJS avancée

Contenu de l’application

- WebPack
  - Construction en mémoire
  - Piloté par le code
  - Serveur de développement
  - Hot replacement
- ES6 (ES2015)
  - Organisation par module
  - Orienté objet
  - Meilleure lisibilité du code
- Express
  - API REST
  - http://localhost:8080/api

Ecrire une application en ES6 avec WebPack

Formation AngularJS avancée

L'application de la syntaxe ES6 et WebPack à une application AngularJS entraine plusieurs adaptations vis-à-vis de certaines recommandations:

https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md

- Favoriser l'usage des classes
- Disparition du « vm »
- Favoriser une syntaxe orientée composant pour les directives
- Factory vs Service
- Utiliser $onInit dans les contrôleurs
- Déclaration des composants dans le fichier « module »

03

Directives

onepoint.

Rappels sur les directives

Formation AngularJS avancée

- Marqueurs reconnus par le compilateur du framework
- Permet la manipulation du dom
- Composants réutilisables
- Principales propriétés:
  - restrict
  - template / templateUrl
  - controller
  - scope (true, false, {})
- Quand utiliser des directives ?

Directive vs Component

Formation AngularJS avancée

- Component à partir de angular 1.5
- Architecture plus proche de Angular 2+
- Basé sur le service $compile et donc les directives AngularJS
- Introduction d'évènements liés au cycle de vie du composant ($onInit, $onChange, $postLink …)
- Identique à une directive avec certains paramètres
- Component ou Directive ?

Processus de compilation

Formation AngularJS avancée

- Permet de cloner un template HTML et de le lier à un scope d'AngularJS, pour produire une vue avec data binding
- Découpé en 2 phases : « compile » et « link »
- Les fonctions compile, preLink et postLink permettent d'intervenir à des moments différents du processus

Processus de compilation – phase « compile »

Formation AngularJS avancée

- Le compilateur traverse le DOM et collecte toutes les directives
- Le retour de la fonction compile est soit une fonction postLink, soit deux fonctions, preLink et postLink
- compile() permet d'interagir avec le **template** de l'élément
- compile() permet de manipuler le template **avant** qu'il soit cloné

Processus de compilation – phase « link »

Formation AngularJS avancée

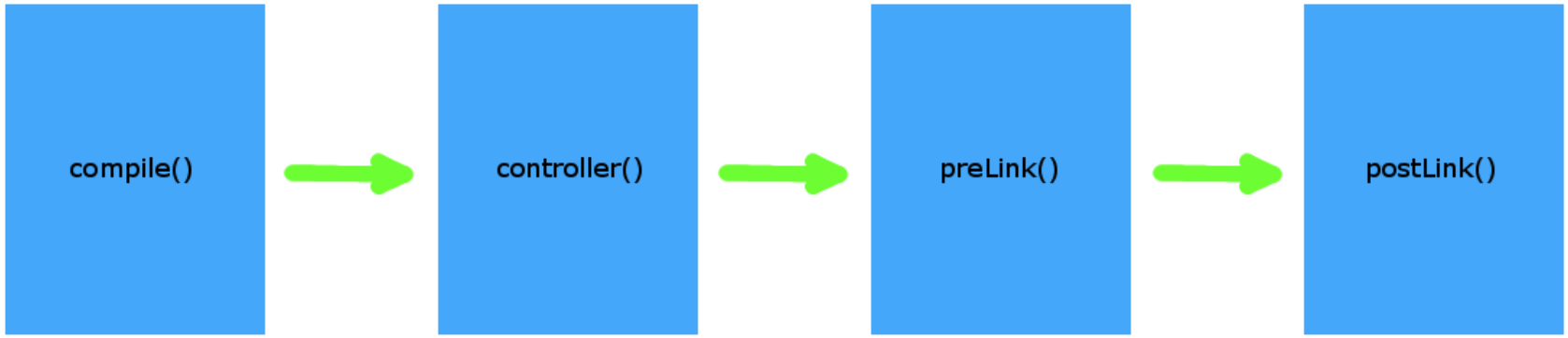
- Phase durant laquelle le template cloné et le scope de la directive sont liés, pour produire une vue avec data-binding
- 2 fonctions : **preLink** et **postLink,** exécutées à des moments différents
- preLink est exécuté avant postLink
- Manipuler la vue en ayant accès au scope

Processus de compilation – ordre d'exécution

Formation AngularJS avancée

Exemple avec une directive :

```
<product data="prodCtl.product"></product>
```



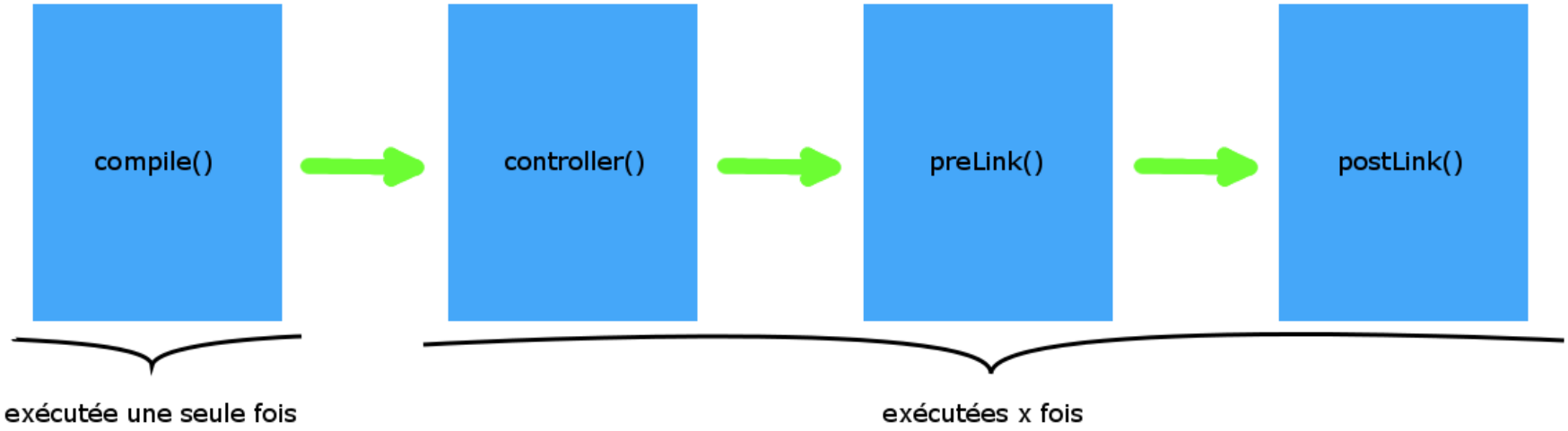
onepoint.

Processus de compilation – ordre d'exécution

Formation AngularJS avancée

Exemple avec une directive dans un ngRepeat :

```
<div class="col-sm-4 col-lg-4 col-md-4 product"
     data-ng-repeat="product in productListCtl.products">
    <product data="product"></product>
</div>
```



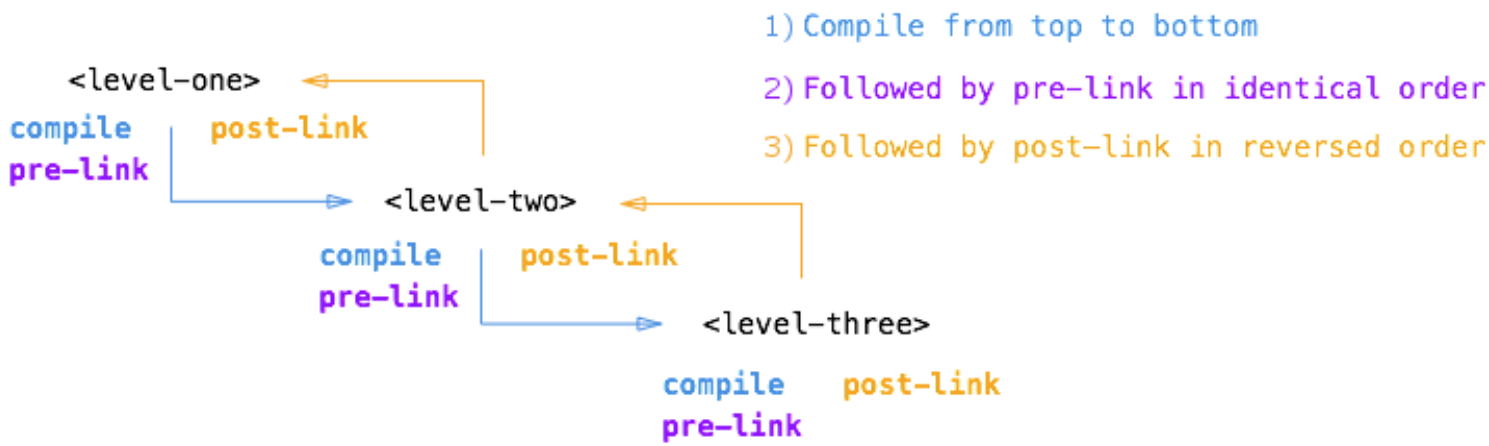
onepoint.

Processus de compilation – ordre d'exécution

Formation AngularJS avancée

Exemple avec plusieurs directives :

```
<level-one>
    <level-two>
        <level-three>
            Hello  {{name}}
        </level-three>
    </level-two>
</level-one>
```



onepoint.

Processus de compilation – TD

Formation AngularJS avancée

Instructions

- écrire la directive <product></product> qui sera utilisée dans la page « productList »
- Réutiliser le html présent dans la page de la liste des produits
- La directive doit permettre d'ajouter une class css à l'image du produit via un attribut

Aide

- s'appuyer sur header.directive.js pour la syntaxe
- écrire le code dans le dossier features/product
- Garder en tête l'optimisation ;)

onepoint.

Mise à jour du dépôt

Formation AngularJS avancée

Récupérez la version du code comprenant la correction

Dans un terminal de commande à la racine du projet:

git reset chapter-directive-transclude --hard

Transclusion

Formation AngularJS avancée

- Permet de passer un fragment html en paramètre d'une directive ou d'un composant qui sera compilé à l'intérieur
- ngTransclude ou $transclude
- Transclusion multi-slot

```
export default () => ({
  restrict: 'A',
  transclude: true,
```

```
  transclude: {
    'title': '?paneTitle',
    'body': 'paneBody',
    'footer': '?paneFooter'
  },
```

```
    this.$transclude((clone) => {
      this.$element.find('div').append(clone);
```

```
<pane>
  <pane-title><a ng-href="{{link}}">{{title}}</a></pane-title>
  <pane-body><p>{{text}}</p></pane-body>
</pane>
```


onepoint.

Transclusion – TD

Formation AngularJS avancée

Instructions

- écrire une directive « user-is-logged » permettant de simplifier le fait d'afficher / masquer des éléments
- Reprendre fonctionnellement la mécanique en place dans les directive header et footer
- Écrire la directive inverse « user-is-not-logged »
- Utiliser ces directives dans le header pour le simplifier

Aide

- s'appuyer sur formField.directive.js pour l'exemple de transclusion
- écrire le code dans le dossier features/security
- Il n'est pas nécessaire d'écrire deux composants …

onepoint.

Mise à jour du dépôt

Formation AngularJS avancée

Récupérez la version du code comprenant la correction

Dans un terminal de commande à la racine du projet:

git reset chapter-directive-interpolate --hard

Services $parse, $interpolate et $compile

Formation AngularJS avancée

- $parse, permet de convertir une expression Angular en fonction. La fonction est exécutable dans un contexte (scope)
  - Très utile lors de la conception de composant pour passer des paramètres complexes
  - Utilisé en interne lors de binding '&'
  - Equivalent à $scope.$eval()
- $interpolate, permet la compilation de chaine pouvant contenir des éléments de markup (ex: {{user.name}} => « John Doe »)
  - Peut-être utilisé lors du binding de textes non initialisés ou changeants
  - Utilisé en interne par le service compile pour le binding
- $compile, permet de compiler un fragment html en template Angular pouvant ensuite être lié à un scope
  - Rarement utilisé directement peut s'avérer utile si l'on souhaite manuellement compiler du code html en dehors des mécaniques d'inclusion Angular (routing, transclusion, include)

$interpolate – TD

Formation AngularJS avancée

Instructions

- **Modifier la directive « formField.directive.js » pour lui permettre de configurer le label à l'aide de chaîne de caractère pouvant contenir du markup.**
- **ex: {{label field}}**
- **La modification ne doit pas modifier le comportement d'origine du composant**

Aide

- $sce permet de désactiver l'échappement de contenus html

- ng-bind-html permet d'afficher du html

onepoint.

Mise à jour du dépôt

Formation AngularJS avancée

Récupérez la version du code comprenant la correction

Dans un terminal de commande à la racine du projet:

git reset chapter-directive-require --hard

Require

Formation AngularJS avancée

- Permet de définir les dépendances d'une directive (?, ^, ?^, ^^, ?^^)
- Permet d'injecter les contrôleurs des dépendances
- Contrôleurs des dépendances directement liées au contrôleur si « bindToController: true »
- Cas d'usage les plus fréquents
  - Validateur de champs de formulaire
  - Transformation de données sur un champ de formulaire
  - Composant complexe nécessitant plusieurs éléments

require & $parse – TD

Formation AngularJS avancée

Instructions

- **Mettre en place une directive implémentant un validateur pour les comparaisons de mot de passe**
- **Créer la directive dans le dossier /common/directives sur le même modèle que la directive « passwordComplexity.directive »**
- **La validation devrait se faire également lors de modification dans le champ source**

Aide

- Require permet de travailler sur le controller du model
- $parse permet de récupérer la valeur du champ source

onepoint.

Mise à jour du dépôt

Formation AngularJS avancée

Récupérez la version du code correspondant à ce chapitre

Dans un terminal de commande à la racine du projet:

git reset chapter-scope --hard

04

Scopes et boucle $digest

onepoint.

Rappel sur les scopes

Formation AngularJS avancée

- Objets servant de modèles applicatifs
- Héritage prototypal
- Contexte d'exécution
- Un seul $rootScope, plusieurs scopes enfants
- Egalement une API :
  - $watch() : écouter les changements
  - $broadcast : émettre un évènement vers les nœuds enfants
  - $emit : émettre un évènement vers les nœuds parents
  - $on : recevoir des évènements
  - $watchCollection() : écouter les changements sur une collection

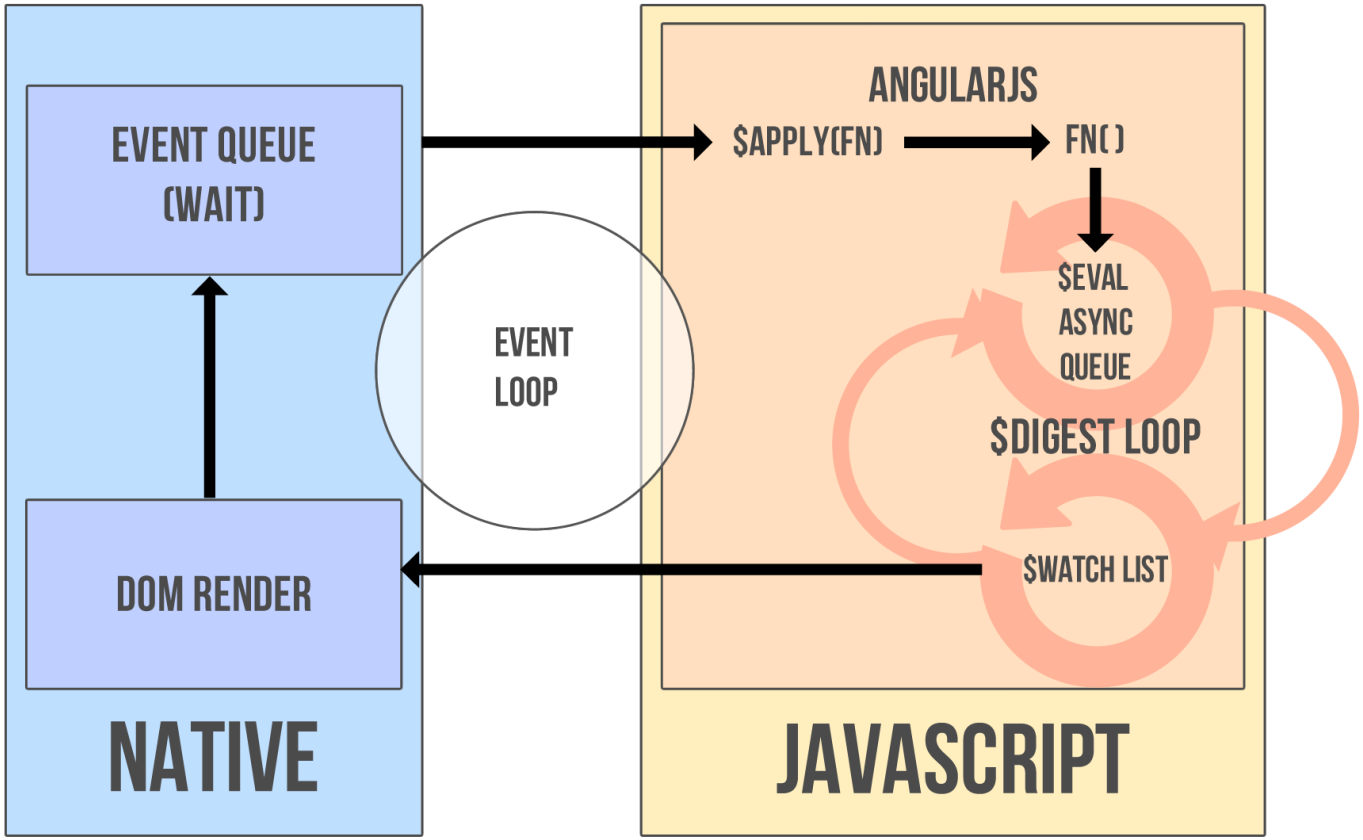
Boucle de digestion

Formation AngularJS avancée

- Permet d'actualiser l'affichage en fonction du model
- Exécutée depuis le $rootScope
- Exécutée au maximum 10 fois
- Dirty checking
- Déclenchée automatiquement par les composant Angular
- Peut-être déclenchée manuellement pas $scope.$eval ou $scope.$evalAsync

Boucle de digestion

Formation AngularJS avancée



onepoint.

Performance

Formation AngularJS avancée

La boucle de digest présente une grande force mais également la plus grande faiblesse d'AngularJS. La plupart des critiques au sujet des performances du framework sont liées à cette mécanique :

- Listeners d'un scope sont exécutés au moins 2 fois en cas de modification
- Coût significatif d'une boucle sur les pages ayant beaucoup de watchers
- Deep watch dangereux
- Favoriser l'usage du one time binding
- Pattern observable et programmation réactive ?

Boucle de digestion – TD

Formation AngularJS avancée

Instructions

- Ajouter un évènement sur la directive **« formField.directive.js »** permettant d' d'effacer les message d'erreur lors d'un clic dans le champ.

Aide

- $setUntouched() permet d'indiquer au champ qu'il n'a pas été modifié par l'utilisateur

- Devrait être ajouté lors de l'init du champ dans la directive

onepoint.

Mise à jour du dépôt

Formation AngularJS avancée

Récupérez la version du code correspondant à ce chapitre

Dans un terminal de commande à la racine du projet:

git reset chapter-test --hard

05

Test unitaires

onepoint.

Outils de test

Formation AngularJS avancée

Outils utilisé pour les tests unitaires

- Karma, exécute les tests
- Jasmine, BDD framework pour l'écriture des tests
- angular-mock, mocks du framework AngularJS
- PhantomJS, Navigateur « headless »
- Les tests unitaires peuvent être lancés depuis la console en lançant la commande:
  npm run test

Configuration de karma avec WebPack

Formation AngularJS avancée

- Karma configuré pour WebPack et support d'ES6

Le fichier main.test.js sert de point d'entré principal pour WebPack

```
files: [
  'app/main.test.js'
],
// Define entry points for tests
preprocessors: {
  'app/main.test.js': ['webpack']
},
// karma-webpack plugin config
webpack: {
  devtool: webpackConfig.devtool,
  module: {
    loaders: webpackConfig.module.loaders
  }
},
```

```
import 'angular';
import 'angular-mocks';

const testsContext = require.context('.', true, /.spec.js$/);
testsContext.keys().forEach(testsContext);
```

Le chargement de l'ensemble des tests ainsi que des mocks est réalisé ensuite dans le point d'entré

Tester une directive

Formation AngularJS avancée

```
describe('shopping.directives.form: shopPasswordComplexityDirective', () => {

  let $compile;
  let $rootScope;
  let scope;
  beforeEach(angular.mock.module('shopping.directives.form'));

  beforeEach(angular.mock.inject(function(_$compile_, _$rootScope_) {
    $compile = _$compile_;
    $rootScope = _$rootScope_;
    scope = $rootScope.$new();
  }));

  it('should throw exception on bad param', testExceptionOnBadParam);
  it('should override default params', testOverrideDefaultParam);

  function testExceptionOnBadParam() {
    // Given
    let template = angular.element('<input type="test" ng-model="test" shop-password-complexity="param">');
    scope.param = 'badparamtype';

    // When
    let compileExecution = () => $compile(template)(scope);

    // Then
    expect(compileExecution).toThrow();
  }
```

Tester une directive
Formation AngularJS avancée

Tester une directive c'est :

- S'assurer que le fragment html compilé est conforme
  - Possible en utilisant le template « angular.element » lors de l'initialisation du test
- S'assurer du comportement des méthodes du controller de la directive
  - Possible en récupérant le contrôleur associé à la directive

```
// When
let compiledElement = $compile(template)(scope);
$rootScope.$digest();

// Then
expect(template.controller('shopPasswordComplexity').complexityParams).toEqual(scope.params);
```

Tester un contrôleur

Formation AngularJS avancée

Dans l'exemple ci-dessous tester l'init du contrôleur nous oblige à mettre en place un mock du service productService :

```
// Standard writing syntax
  describe('shopping.feature.product: ProductListController', () => {

  let $controller;
  let $rootScope;
  let scope;
  let productService;
  let $q;

  beforeEach(angular.mock.module('shopping.feature.product'));
  beforeEach(angular.mock.module(function($provide) {
    $provide.service('productService', () => {
      let productList = null;
      return {
        getList: () => ($q((resolve) => resolve(productList))),
        setList: (list) => productList = list
      }
    });
  }));
```


onepoint.

Tester un contrôleur

Formation AngularJS avancée

L'écriture est verbeuse et assez contraignante

```
beforeEach(angular.mock.inject(function(_$controller_, _$rootScope_, _$q_, _productService_) {
  $controller = _$controller_;
  $rootScope = _$rootScope_;
  $q = _$q_;
  productService = _productService_;
  scope = $rootScope.$new();
}));

it('should init controller with products', testInitWithProducts);

function testInitWithProducts() {
  // Given
  productService.setList([{id: 1}, {id: 2}, {id: 3}]);

  // When
  let ctl = $controller('ProductListController', {$scope: scope});
  ctl.$onInit();
  $rootScope.$apply();

  // Then
  expect(ctl.products.length).toBe(3);
}
```

ngDescribe

Formation AngularJS avancée

ngDescribe est un bibliothèque fournissant des aides construits sur la base de angular-mock:

```
ngDescribe({
  modules: 'shopping.feature.product',
  mocks: {'shopping.feature.product': {productService:
mockProductService()}},
  inject: ['$controller', '$rootScope', '$q', 'productService'],
  tests: function(deps) {
    it('should init controller with products', testInitWithProducts);

    function testInitWithProducts() {
      // Given
      deps.productService.setList([{id: 1}, {id: 2}, {id: 3}]);

      // When
      let ctl = deps.$controller('ProductListController', {$scope:
deps.$rootScope.$new()});
      ctl.$onInit();
      deps.step();

      // Then
      expect(ctl.products.length).toBe(3);
    }
  }
});

function mockProductService() {
  let productList = null;
  return {
    getList: ($q) => ($q((resolve) => resolve(productList))),
    setList: (list) => productList = list
  }
}
```

Tester les filtres et service

Formation AngularJS avancée

Les filtres sont parmi les éléments les plus simples à tester avec les services. Dans le cas des filtres il faut simplement passer par le service $filter pour injecter le filtre à tester:

```
ngDescribe({
  modules: 'shop.filters.multiply',
  inject: ['$filter'],
  tests: function(deps) {
    it('should multiply by given factor', testMultiplyByFactor);

    function testMultiplyByFactor() {
      // Given
      let multiplyFilter = deps.$filter('multiply');

      // When
      let result = multiplyFilter(2, 3);

      // Then
      expect(result).toEqual(6);
    }
  }
});
```

Tests – TD

Formation AngularJS avancée

Instructions

- Ecrire 3 tests pour notre directive de comparaison de mot de passe permettant de vérifier son fonctionnement:
- Doit ajouter une erreur si différent
- Ne doit pas ajouter d'erreur si égaux
- Doit ajouter une erreur quand le mot de passe d'origine change

Aide

- Ce qui doit être testé est l'ajout d'erreurs dans le contrôleur du ngModel
  - Vous pouvez vous inspirer des autres tests (ex : passwordComplexity.direcive.spec.js header.directive.spec.js)

onepoint.

Mise à jour du dépôt

Formation AngularJS avancée

Récupérez la version du code correspondant à ce chapitre

Dans un terminal de commande à la racine du projet:

git reset chapter-e2e --hard

06

Tests automatisés

onepoint.

Outils

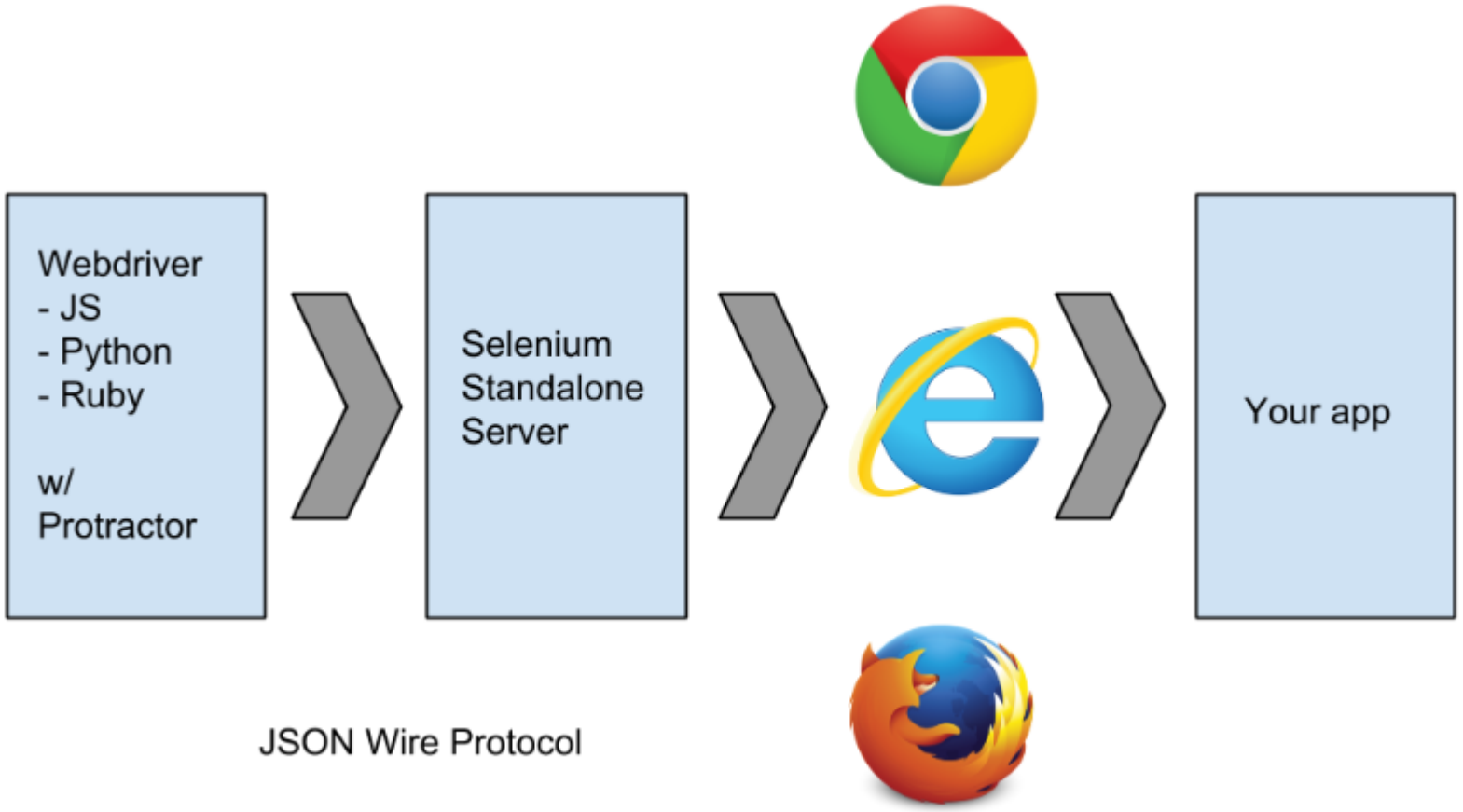
Formation AngularJS avancée

Outils utilisés pour les tests e2e

- Protractor, framework e2e pour Angular basé sur WebDriver
- Selenium WebDriver
- Jasmine, BDD framework pour l'écriture des tests
- Chrome
- Les tests e2e peuvent être lancés depuis la console en lançant la commande:
  npm run e2e
  (Le server de développement doit être lancé au préalable via la commande npm start)

Principe

Formation AngularJS avancée



onepoint.

Configuration de protractor

Formation AngularJS avancée

La configuration de base de protractor est très simple.

Il est possible également de lancer les tests sur plusieurs navigateurs, de spécifier l'adresse du serveur Selenium … .

```
require('babel-register');

exports.config = {
  specs: ['./e2e/**/*.spec.js'],
  localSeleniumStandaloneOpts: {
    loopback: true
  },
  capabilities: {
    browserName: 'chrome'
  },
  baseUrl: 'http://localhost:8080'
};
```

```
"babel": {
  "presets": [
    "env"
  ]
},
```

L'écriture du code des tests e2e est possible en ES6 grâce à l'utilisation de babel

Page Object

Formation AngularJS avancée

Le « Page Object » est un principe de design des tests e2e qui visent à encapsuler les éléments d'une page dans un objet la représentant.

Ceci permet la réutilisation d'éléments entre les tests mais aussi de fournir une abstraction suffisante permettant aux testeurs d'une équipe de participer à l'écriture de ceux-ci.

```
import BasePage from '../base.page';

export default class extends BasePage {
  constructor(url) {
    super(url);
  }

  getProducts() {
    return element.all(by.css('.main-container product'))
  }
}
```

Exemple de test

Formation AngularJS avancée

```
import ProductListPage from './productList.page.js';

describe('Home Page:', () => {

  let page;
  beforeEach(() => page = new ProductListPage('/#/'));

  it('should display products', testDisplayProduct);

  function testDisplayProduct() {

    page.getProducts().count()
      .then((productCount) => {
        expect(productCount > 0).toBe(true);
      });
  }

});
```


onepoint.

Test e2e – TD

Formation AngularJS avancée

Instructions

- Ecrire le test e2e de connexion d'un utilisateur. Avec
- Clic sur le lien pour remplir le formulaire
- Clic sur le bouton de soumission du formulaire
- Vérification de la redirection

Aide

- Créer un « page object » pour la page de login mais aussi pour la page cible (account)
  - Inspirez-vous du test productList présent dans le dossier e2e/product

onepoint.

Mise à jour du dépôt

Formation AngularJS avancée

Récupérez la version du code correspondant à ce chapitre

Dans un terminal de commande à la racine du projet:

git reset chapter-provider --hard

07

Providers

onepoint.

Rappel sur les « recettes » angular

Formation AngularJS avancée

Angular s’appui sur un certain nombre de recettes pour créer les services

- Value, constant, service, factory, provider
- Toutes des sucres syntaxiques de provider hormis constant
- Seul provider et constant sont accessibles durant la phase de « config »
- Les providers ne sont plus accessibles une fois l’application démarrée
- Les constantes sont toujours accessibles

Des providers pourquoi ?

Formation AngularJS avancée

Les providers permettent de modifier un service avant qu'il soit injecté

- Permet de configurer les services avant le démarrage de l'application
- Permet d'effectuer des tâches avant la phase de « run »
- Utile pour les modules réutilisables
- Les routeurs sont de bons exemples de cas d'usage

Exemple de provider

Formation AngularJS avancée

L'exemple ci-dessous présente un service de gestion d'actions asynchrones au démarrage de l'application

```
angular.module('shopping.services.bootstrap', [])
  .provider('shopBootstrapService', BootstrapProvider);
```

```
export default class BootstrapProvider {
  constructor() {
    this.bootstrapQueue = [];
  }

  register(task) {
    this.bootstrapQueue.push(task);
  }

  $get($rootScope, $q, $state, $injector) {
    return new BootstrapService($rootScope, $q, $state, $injector, this.bootstrapQueue);
  }
}
```

Tester un provider

Formation AngularJS avancée

Tester notre provider s'avère facile ici puisque nous l'avons déclaré dans une classe que nous pouvons ensuite importer.

```
import angular from 'angular';
import BootstrapProvider from './bootstrap.provider';
import BootstrapService from './bootstrap.service';

ngDescribe({
  name: 'shopping.services.bootstrap: BootstrapProvider',
  module: 'ui.router',
  inject: ['$rootScope', '$q', '$state', '$injector'],
  tests: function(deps) {
    it('should register task', testConfigRegistration);
    it('should return a bootstrap service instance', testInstantiateBootstrapService);
```

Provider – TD

Formation AngularJS avancée

Instructions

- Ecrire un provider permettant d'enregistrer des configurations lors de la phase de configuration de l'application
- Le service renvoyé par le provider permet d'accéder aux paramètres de configuration ajoutés avant le démarrage de l'application
- Ecrire au moins 1 test pour le provider et le service

Aide

- Vous pouvez vous inspirer du provider bootstrap présent dans le dossier /common/services/bootstrap

onepoint.

Mise à jour du dépôt

Formation AngularJS avancée

Récupérez la version du code correspondant à ce chapitre

Dans un terminal de commande à la racine du projet:

git reset chapter-decorator --hard

08

Promesses

onepoint.

Rappel sur les promesses

Formation AngularJS avancée

- Introduite par Kris Kowal avec bibliothèque « q »
- API de base, then, catch, finally (done), all
- Permettent de gérer plus facilement l'exécution des tâches asynchrones
- Peuvent être chainées
- Promise en ES6
- $q dans Angular, doit être utilisé afin de déclencher les cycles de digest de l'application
- $http renvoi une promesse, $resource expose une promesse au travers de la propriété $promise

Les promesses dans Angular

Formation AngularJS avancée

Peuvent être écrites de 2 façons :

```
return $q(function(resolve, reject) {
  $http.get()
    .then(function(response) {
      resolve(response.data);
    })
    .catch(function(error) {
      reject(error.status);
    });
});
```

```
var deferred = $q.defer();
$http.get()
  .then(function(response) {
    deferred.resolve(response.data);
  })
  .catch(function(error) {
    deferred.reject(error.status);
  });
return deferred.promise;
```

Chainer et gérer les erreur

Formation AngularJS avancée

- Exécutées dans l'ordre
- Chaque promesse prend en paramètre la résolution de la précédente
- Les rejets ou exceptions interrompent l'exécution de la chaine
- Un rejet ne lance pas d'exception
- La méthode finally est toujours appelée

```
service1
  .then(service1)
  .then(service2)
  .then(service3)
  .then(service4)
  .catch(function(error) {
    // error depend on which service has reject or throw
  })
  .finally(finalTask);
```


onepoint.

09

Décorateurs, $provide

onepoint.

Décorateurs, $provide

Formation AngularJS avancée

Le service $provide est responsable de l'enregistrement des différents composants de l'application. La plupart de ses méthodes sont exposées au travers de angular.module (provider, service, controller …)

Ce service permet également de déclarer des décorateurs pour les services enregistrés permettant ainsi de modifier ou remplacer des méthodes spécifiques de services

- Doit être déclaré durant la phase de configuration du module (.config)
- Meilleurs granularité dans les surcharges
- Marche pour l'ensemble des services enregistrés
- Ne s'applique pas aux providers
- Peux également servir lors des tests

Exemple de décorateur

Formation AngularJS avancée

L'enregistrement d'un décorateur prend en argument une fonction « decorator » qui sera appelée avec le service décoré en paramètre ($delegate)

```
angular.module('myModule')
  .config(($provide) => {
    $provide.decorator('$state', ($delegate) => {

      const originalFunction = $delegate.go;

      $delegate.go = (to, params, options = {}) => {
        if (angular.isUndefined(options.reload)) {
          options.reload = true;
        }
        return originalFunction.go.apply(null, [to, params, options]);
      };

      return $delegate;
    })
  });
```


onepoint.

Tester provider et décorateur

Formation AngularJS avancée

Tester un décorateur et provider peut s'avérer plus complexe. La déclaration peut éventuellement les rendre non testable.

Par exemple service $log est déjà décoré par angular-mocks.

Une astuce reviens à différentier la déclaration du décorateur de celui-ci le rendant ainsi injectable durant les tests :

```
.config([ '$provide', function($provide) {

  $provide.decorator('$log', [
    '$delegate',
    function $logDecorator($delegate) {

      var originalWarn = $delegate.warn;
      $delegate.warn = function decoratedWarn(msg) {
        msg = 'Decorated Warn: ' + msg;
        originalWarn.apply($delegate, arguments);
      };

      return $delegate;
    }
  ]);
```

```
export function configureRouterDecorator($provide) {
  $provide.decorator('$state', ($delegate) => new RouterDecorator($delegate).decoratedService);
}
```

Tester provider et décorateur

Formation AngularJS avancée

Tester la class résultante deviens ensuite beaucoup plus facile

```
import {configureRouterDecorator, RouterDecorator} from './router.decorator';

ngDescribe({
  name: 'shopping.services.router: RouterDecorator',
  tests: function(deps) {
    it('should add reload option by default', testAddTimestampToDebug);
    function testAddTimestampToDebug() {
      // Having
      let originalService = {go: (name, params, options) => [name, params, options]};
      let decorator = new RouterDecorator(originalService);

      // When
      let [name, params, options] = decorator.decoratedService.go('fakestate', null);

      // Then
      expect(options.reload).toBe(true);
    }
  }
});
```

Tests – TD

Formation AngularJS avancée

Instructions

- Ecrire un décorateur pour le service $log qui permettra de préfixer tous les appels à la méthode debug par un timestamp
- Ecrire le test de ce décorateur
- Debug peut prendre plusieurs paramètres

Aide

- Ajouter le décorateur dans la partie common/services de l'application
- Vous pouvez vous inspirer du décorateur du service $state dans common/services

onepoint.

Mise à jour du dépôt

Formation AngularJS avancée

Récupérez la version du code correspondant à ce chapitre

Dans un terminal de commande à la racine du projet:

git reset final --hard

10

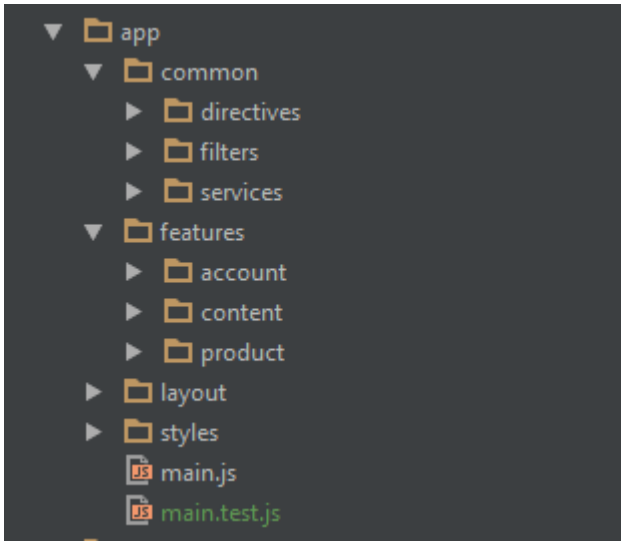
Architecture applicative

onepoint.

Architecture applicative

Formation AngularJS avancée

Présentation détaillée du squelette de notre l'application et organisation des modules :




```
angular
    .module('shopping', [
        'shopping.layout',
        'shopping.feature.product',
        'shopping.feature.account'
    ]);
```


onepoint.

Organiser le code
Formation AngularJS avancée

Points clés lors de la mise en place du socle applicatif

- Fixer les conventions en début de projet
- Regrouper par modules fonctionnels
- Eviter le couplage entre les modules
- Les tests permettent de s'assurer de la bonne gestion des dépendances
- Ne pas hésiter à refactoriser
- Attention aux « bonnes pratiques »

Configuration

Formation AngularJS avancée

L'application est dépendante de son environnement d'exécution. En fonction du contexte du projet plusieurs options s'offrent

- Construire l'application avec sa configuration
- Configuration fournie par le serveur d'application
- Ou un peu des deux …
- Problématique du chargement de la configuration au démarrage

Livrable final

Formation AngularJS avancée

Tout comme pour la configuration ces questions doivent intervenir au plus tôt lors de la mise en place du socle

- Livré ou non à part du serveur d'application ?
- Page de démarrage de l'application
- Sécurité
- Versioning de l'application mais aussi de l'API
- Tests d'intégration

11

Conclusion

onepoint.

Conclusion

Formation AngularJS avancée

- Rappel des points abordés
- Liens pour aller plus loin et références
  - http://www.ecma-international.org/ecma-262/6.0
  - https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md
  - http://pascalprecht.github.io/slides/e2e-testing-with-protractor
  - https://www.html5rocks.com/fr/tutorials/es7/observe/
  - http://home.heeere.com/tech-intro-programmation-reactive.html
  - https://medium.com/google-developer-experts/angular-introduction-to-reactive-extensions-rxjs-a86a7430a61f#.4de1smf9n
- Foire aux questions