Pour aller plus loin avec Angular 1.4

- Accueil: Documentation support de formation
- Chapitre précédent: Démarrer avec Angular 1.4 (niveau débutant)

Pré-requis (niveau intermédiaire)

Cette formation est destinée à un public connaissant déjà les bases d'angular et l'ayant pratiqué sur un ou plusieurs projets.

Les pré-requis sont les sujets abordés dans le document "Démarrer avec Angular 1.4".

Sommaire

Cache par défaut (ng-cache)

Angular fourni un cache par défaut: ng-cache.

Création du cache:

```
var cache = $cacheFactory('users.resources');
```

Ajout d'un élément du cache:

```
cache.put('user.1', user1);
```

Lecture d'un élément du cache:

```
var user1 =  cache.get('user.1');
```

Limitations: * pas de méthode pour itérer sur tous les éléments du cache * pas de mécanisme d'expiration * cache mémoire, pas d'extension au localStorage ou sessionStorage

Resources:

- $cacheFactory

Cache alternatif: angular-cache

Cache avancé permettant de palier aux limitations de $cacheFactory: * itération sur tous les éléments du cache * mécanisme d'expiration configurable * cache mémoire, localStorage ou sessionStorage

Installation via bower:

```
bower install --save angular-cache
```

Ajout du module à l'application:

```
angular.module('myApp', ['angular-cache'])
```

Création et configuration du cache avec local storage:

```
var cache = CacheFactory('users.resources', {
    storageMode: 'localStorage'
});
```

Ajout d'un élément du cache:

```
cache.put('user.1', user1);
```

Lecture d'un élément du cache:

```
var user1 =  cache.get('user.1');
```

Lecture de tous les éléments du cache:

```
var users = cache.keys().reduce(function (values, key) {
    values.push(cache.get(key));
    return values;
}, []);
```

Resources:

- angular-cache

JS data: respectez vos données

- Sommaire: Pour aller plus loin avec Angular 1.4
- Page précédente: Les caches de données
- Page suivante: Sécurité

Introduction

Librairie de gestion des données, indépendante de tout framework, fonctionnant avec nodejs, angularjs, ou n'importe quelle application javascript.

Par défaut, toutes les données sont stockées dans un cache mémoire. L'auteur, Jason Dobry, est aussi le créateur de CacheFactory.

Installation via bower:

```
bower install --save js-data-angular js-data
```

Ajout du module à l'application:

```
angular.module('myApp', ['js-data'])
```

Configuration du module

```
    angular.module('tw.practice').config(configureModule);

    angular.module('tw.practice').run(runModule);

    /** @ngInject */
    function configureModule(DSProvider, DSHttpAdapterProvider) {

        DSProvider.defaults.basePath = '/api';
```

```
        DSProvider.defaults.idAttribute = '_id';
    }

    /** @ngInject */
    function runModule(DS) {

        var UserResource = DS.defineResource({
            name: 'users'
        });

    }
```

Contournement du cache ajax de IE10 via l'ajout d'un timestamp dans chaque requête:

```
    /** @ngInject */
    function configureModule(DSProvider, DSHttpAdapterProvider) {

        DSProvider.defaults.basePath = '/api';
        DSProvider.defaults.idAttribute = '_id';

     DSHttpAdapterProvider.defaults.queryTransform = function (resource, params) {
            // disable ie10 cache
            params.nocache = new Date().getTime();
            return params;
        };

    }
```

Resources:

- js-data
- DataStore configuration

Opérations de base

```
function findAll(params) {

    return DS.findAll('users', params);
}

function findOne(userId) {

    return DS.find('users', userId);
```

```
}

function createOne(user) {

    return DS.create('users', user);
}

function updateOne(user) {

    return DS.update('users', user);
}

function removeOne(userId) {

     return DS.destroy('users', userId);
}
```

Il est possible de bypasser le cache sur une requête particulière:

```
function findAll(params) {

    return DS.findAll('users', params, { bypassCache: true });
}

function findOne(userId) {

    return DS.find('users', userId, { bypassCache: true });
}
```

Pour vider manuellement le cache:

```
return DS.ejectAll('users');
```

Resources:

- js-data: Working with the data store
- js-data: Resources/Models

Adapters

Par défaut, JS-Data stocke les données sur serveur http (REST API) via l'adaptateur http.

Il est possible d'utiliser un autre adaptateur (localforage, firebase…) ou d'en écrire un sois-même.

LocalForage Installation via bower:

```
bower install --save localforage js-data js-data-localforage
```

Attention, vérifier que les dépendances sont déclarées dans le bon ordre dans bower.json:

```
"js-data": "~2.8.2",
"js-data-localforage": "~2.1.1",
"js-data-angular": "~3.1.0",
```

Remplacement de l'adapteur http par l'adapteur local forage:

```
angular.module('tw.practice.profile').run(runModule);

/** @ngInject */
function runModule(DS, DSLocalForageAdapter) {

    // make local forage the default adapter
    DS.adapters.localForage === DSLocalForageAdapter;
    DS.registerAdapter('localForage', DSLocalForageAdapter, {
        default: true
    });

}
```

Resources:

- js-data: Adapters
- dsLocalForageAdapter

Sécurité d'une application AngularJS

- Sommaire: Pour aller plus loin avec Angular 1.4
- Page précédente: JS Data
- Page suivante: Internationalisation et localisation

A propos de la sécurité côté client

La gestion de la sécurité côté client vise à :

- authentifier l'utilisateur auprès du serveur
- adapter l'interface en fonction des droits de l'utilisateurs

Elle a donc principalement un rôle de présentation, et en aucun cas elle ne se substitue à la sécurité côté serveur car elle est très facilement contournable.

Authentification par token

Principes L'authentification se fait auprès du serveur d'authentification via un mécanisme quelconque (par exemple login/password), et génère un token d'authentification unique au client.

Le client stocke ensuite le token (en mémoire, dans le local storage, dans un cookie…).

Ensuite, le client s'authentifie auprès des serveurs de données (API rest par exemple) en insérant le token en entête de toutes les requêtes. Les serveurs de données vérifient la validité du token auprès du serveur d'authentification.

L'exemple le plus répandu est OAuth 2 qui permet d'externaliser l'authentification d'une ou plusieurs applications.

Resources: * An Introduction to OAuth 2

Avantages Plus flexible, l'authentificaiton par token présente plusieurs avantages par rapport à l'authentification classique par cookie: * scalabilité: réplication aisée des serveurs (stateless) : mémoire: plus de sessions qui saturent la mémoire du serveur si beaucoup d'utilisateurs * API consommables depuis n'importe quel application (Web, Mobile, serveur…) car plus de problem CORS * authentification flexible car découplée des API Rest (e.g. Single Sign On) * plus de vulnérabilité cross-site request forgery (CSRF) car les URLs d'un site tiers n'auront pas le token dans le security header

Points importants: * les communications doivent se faire en https pour éviter le vol de token (man-in-the-middle) * les tokens doivent avoir une durée de vie limitée, et être renouvellés * les token étant stockés dans le navigateur, ils sont vulnérables aux attaques cross-site scripting (XSS) => injection de javascript depuis les commentaires, évitable en échapant le html

Resources: * Cookies vs Tokens. Getting auth right with Angular.JS * The ins and outs of token based authentication * Let's encrypt: des certificats SSL certifiés gratuits

JSON Web Tokens (JWT)

JWT vise à standardiser le mécanisme d'authentification par token, en y apportant au passage le cryptage. Il est compatible avec OAuth 2.

JWT défini le format du token, composé de 3 parties (encodées en base64) séparées par un point: * header: l'entête du message * payload: informations concernant le token * signature: la signature du token

Le header contient à minima le type de signature et l'algorythme de cryptage:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Le payload contient des informations complémentaires sur le token (sujet, origine, destinataire, date de création, date d'expiration, identifiant). Il peut être enrichi avec des meta-data personalisées, comme le profil de l'utilisateur dans le cas d'un JWT d'authentification.

La sécurité est assurée par HTTPS, mais le payload peut également être encrypté via Json Web Encryption (JWE).

La signature permet au serveur de vérifier l'authenticité du token via la clé privée qu'il est le seul à connaitre.

Resources: * angular-jwt: intégrer JWT dans AngularJS * Introduction to JSON Web Tokens * The Anatomy of a JSON Web Token * Critical vulnerabilities in JSON Web Token libraries * 10 Things You Should Know about Tokens * Authenticate a Node.js API with JSON Web Tokens

JSON Web Token Tutorial: An Example in Laravel and AngularJS

Authentification JWT via AngularJS

Authentification Le mécanisme d'autentification dépend du serveur. Typiquement, il peut s'agir d'une authentification par login/password.

```
$http({
    method: 'POST',
    url: '/login',
    data: {
        login: login,
        password: password
    }
}).then(function successCallback(response) {
    // success
}, function errorCallback(response) {
    // authentication error
});
```

Si l'authentification réussi, le token JWT sera retourné au client:

```
{
"token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1NjM1ZjE4ODE1YmQ0MDI5MzgyN2MzMG
}
```

JSON Web Token Tutorial: An Example in Laravel and AngularJS

Manipulation du token Le token peut être manipulé via angular-jwt.

Décoder le token:

```
    var tokenClaims = jwtHelper.decodeToken(data.token);
```

Le token peut alors être décrypté afin d'extraire le payload et obtenir les informations concernant l'utilisateur:

```
var tokenClaims = jwtHelper.decodeToken(token);
// Token claims: Object { _id="5635f18815bd40293827c30c", role="admin", iat=1446387614, .
```

Sur cet exemple, le token claims contient l'identifiant de l'utilisateur ainsi que son rôle, que nous pouvons utiliser pour construire l'utilisateur connecté:

```
var user = {
    id: tokenClaims._id,
    role: tokenClaims.role
};
```

Stockage du token de sécurité En fonction du comportement souhaité, le token peut-être stocké: * dans le local storage: le token sera conservé dans le navigateur pour le domaine courant (pas de date d'expiration mais le token expirera côté serveur) => supporté par la plupart des navigateurs, limité à 5Mo par domaine * dans le session storage: le token sera conservé dans le navigateur pour l'onglet courant (permet d'avoir différents utilisateurs connectés dans différents onglets) => encore mal supporté par les navigateurs * dans un cookie: utile pour partager un token entre différents sous-domaines (mais créé un peu de traffic client/serveur et est limité à 4ko) * dans une variable, en mémoire: un simple rafraichissement de la page déconnectera alors l'utilisateur

Ce token sera ensuite accessible dans toute l'application.

Ajout automatique du token dans le security header Le token de sécurité peut alors être ajouté automatiquement à chaque requête via un http interceptor:

```
// register the previously created AuthInterceptor
app.config(function ($httpProvider) {
    $httpProvider.interceptors.push('BearerAuthInterceptor');
});

app.factory('BearerAuthInterceptor', function ($window, $q) {
```

```
    return {
        request: function(config) {
            config.headers = config.headers || {};

            // get token from local storage
            var token = $window.localStorage.getItem('token');

            if (token) {
              // may also use sessionStorage
                config.headers.Authorization = 'Bearer ' + token;
            }
            return config || $q.when(config);
        },
        response: function(response) {
          if (response.status === 401 || response.status === 403) {
                //  manage error (redirect, logout...)
            }
            return response || $q.when(response);
        }
    };
});
```

Au passage, les erreurs de sécurité http peuvent être interceptées et déclencher une redirection vers la page de login par exemple.

Renouvellement du token Si le serveur supporte le renouvellement du token, il est possible de tester la date de validité via angular-jwt et de renouveller le token avant de l'envoyer au serveur.

Checker la date d'expiration (par exemple pour renouveller le token):

```
var tokenExpirationDate = jwtHelper.getTokenExpirationDate(data.token);

 var isTokenExpired = jwtHelper.isTokenExpired(data.token);
```

Filtrage du contenu par rôle

Création d'une directive pour filtrer le contenu par role.

“‘ js angular.module(‘tw.practice’).directive(‘twHasRole’, twHasRole); /** @ng-Inject */ function twHasRole(Auth) { return { scope: {}, link: function preLink($scope, element, attrs) {

```
        if (Auth.hasRole(attrs.twHasRole)) {
            // show
```

```
            element.removeClass('hidden');
        } else {
            // hide
            element.addClass('hidden');
        }

    }
};
```

} “‘
Utilisation:

```
handlebars <div tw-has-role="admin">    <a href="/administration">Page
admin</a> </div>
```

Sécurisation des URLs (ui-router)

Il est possible de rediriger automatiquement un utilisateur si il n'a pas le droit d'accéder à une page.

Association des roles aux urls via un enrichissement personnalisé de l'objet “state”:

```
js $stateProvider.state('edit-user', {   url: "/edit-user/:userId",
templateUrl: 'app/profile/profile.user-edit.html',     controller:
'TwProfileUserEditController',     controllerAs: 'vm',     roles:
['admin'] })
```

Puis interception des changements sur ui-router pour ajouter une vérification personalisée (et redirection si l'utilisateur n'a pas le rôle requis pour afficher la page): “‘ js function configureRoutesSecurity($log, $rootScope, $state, twSecurityService, twRouteSecurityService) {

```
// redirect if user does not has access to next route
var cb = $rootScope.$on('$stateChangeStart', function (event, nextState) {

    if (!twRouteSecurityService.hasAccess(nextState)) {
        // prevent current route change
        event.preventDefault();
        if (twSecurityService.isAuthenticated()) {
            // access denied
            $log.error('Access denied: redirect to home page.');
            $state.go('view-users');
        } else {
            // use not authenticated
        $log.error('User not authenticated: redirect to login page.');
            $state.go('login');
```

```
        }
    }

});
$rootScope.$on('$destroy', cb)

} “‘
```

Internationalisation et localisation

- Sommaire: Pour aller plus loin avec Angular 1.4
- Page précédente: Sécurité
- Page suivante: Interractions avec des bibliothèques non-angular - services

Localisation

Pour localiser les filtres currency et date, installer le module angular-i18n et inclure le fichier correspondant à la locale dans la page html:

```
<script src="i18n/angular-locale_fr-fr.js"></script>
```

Avec bower:

```
bower install --save angular-i18n
```

Puis surcharger la configuration par défaut dans le fichier bower.json du projet:

```
"overrides": {
    "angular-i18n": {
        "main": ["angular-locale_fr-fr.js"]
    }
}
```

C’est l’outil de build (gulp, grunt…) qui se chargera alors d’injecter ce fichier dans le fichier html.

Pour changer de locale dans l’application, il faut s’adresser à un module externe: angular-dynamic-locale.

Resources:

- i18n and l10n

angular-translate

Pour traduire son application, la solution la plus répandue est l'utilisation de la directive angular-translate.

Directive "translate": `handlebars <span translate="contacts.success">Your message has been successfully sent. You will receive an answer as soon as possible.</span>`

Resources:

- angular-translate

Angular 1.4 i18n

Pour un meilleur support i18n dans angular A partir de la version 1.4, Angular vise à fournir un meilleur support pour l'internationalisation.

Sur le document de travail, on peut trouver les reproches suivant concernant angular-translate:

- mauvais support de la pluralisation
- problèmes de sécurité si mal configuré(XSS / Cross-site scripting)
- problèmes de performances lors du chargement des traductions
- beaucoup de balises spécifiques
- système d'id pas pratique (pourrait être remplacé par le message lui-même)
- compliqué de traduire des fragments entiers de html

De plus, le projet vise à fournir des outils permettant de rendre plus simple l'industrialisation de l'i18n: * outils pour extraire les messages de l'application afin de les fournir aux services de traduction dans différents formats * outils pour transformer les traductions dans des formats exploitables par l'application angular * outils pour packager les traductions en plusieurs fragments pour chaque locale

Resources:

- Angular and Internationalization: The New World
- angular-ui security: escaping of variable content

MessageFormat extensions La première brique de cette refonte de l'internationalisation est MessageFormat, livrée avec Angular 1.4:

```
bower install --save angular-message-format
```

Chargement du module:

```
angular.module('app', ['ngMessageFormat']);
```

Exemple de pluralization: `handlebars  <div class="alert alert-info"> {{vm.users.length, plural, offset:0  =0  {There is no user}  =1  {There is only one user}  other {There are # users (including you {{ vm.currentUser.firstName }}).} }} </div>`

Exemple de gestion du genre:

`handlebars  <th>Gender</th> <td>  {{vm.user.gender, select, male {He is a man.}  female {She is a women.}  other {Unknown gender.}  }} </td>`

Resources:

- Angular I18N guide: MessageFormat Extensions
- better i18n for your angular apps
- What's new in AngularJS 1.4 ## Intégration de librairies non angular - services
- Sommaire: Pour aller plus loin avec Angular 1.4
- Page précédente: Internationalisation et localisation
- Page suivante: Interractions avec des bibliothèques non-angular - directives

Introduction

Afin d'intégrer dans une application angular des librairies javascript classiques, il faut les encapsuler dans des composants angular classiques (services, directives, filtres…).

Cela permet de les utiliser de façon classiques, et de limiter les conflits en n'ayant pas recours à des variables globales.

De plus, le niveau d'abstraction apporté par le service ou la directive angular permet de changer de librairie (ou de version) sans impacter le reste de l'application.

Il convient souvent de créer un module spécifique pour intégrer la librairie tierce.

Wrapper service

Pour intégrer un librairie non-graphique, il suffit de déclarer un service dit "wrapper", qui va exposer cette librairie au mécanisme d'injection de dépendances d'angular et la rendre ainsi accessible à l'ensemble de l'application.

Exemple avec moment.js:

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

    /** @ngInject */
    function twMoment($window, $log) {
        if (!$window.moment) {
           $log.error('Global moment variable is not available.');
            return null;
        }
        var service = $window.moment;

        return service;
    }

})();
```

Note: les variables globales sont accessibles via l'objet $window. Si la libraire n'est pas présente, il est judicieux de logger une erreur.

NoConflict

De nombreuses librairies proposent une solution visant à ne pas exposer de variable globale, afin d'éviter les conflits avec d'autres librairies.

Dans ce cas, il faut utiliser cette méthode dans le service wrapper.

Exemple avec leaflet:

```
    function twLeaflet($window, $log) {

        // ...

        var service = $window.L.noConflict();

        return service;
    }
```

Intégration de librairies non angular - directives

- Sommaire: Pour aller plus loin avec Angular 1.4
- Page précédente: Interractions avec des bibliothèques non-angular - services
- Page suivante: D3 - data-driven documents

Généralités

Si la librairie intégère des comportements graphiques, il convient de l'encapsuler dans une directive.

Afin d'accéder à la librairie, la directive utilisera un service wrapper.

Exemple:

```
angular.module('tw.practice.map').directive('twLeafletMap', twLeafletMap);

 function twLeafletMap() {
     return {
     templateUrl: 'app/components/map/leaflet/leaflet-map.directive.html',
         controllerAs: 'vm',
         scope: {
             // scope parameters here
         },
         bindToController: true,
         controller: TwLeafletMapController
     };
 }

 /** @ngInject */
function TwLeafletMapController($scope, $log, $timeout, twLeaflet) {

     var vm = this;

     // ...

     function renderMap() {

         // usage of twLeaflet wrapper service
        vm.map = twLeaflet.map('tw-map-id').setView(vm.config);
```

Intégration d'ue librairie - en pratique

Pour intégrer une librairie via une directive, il y a au moins 2 approches possibles:
* intégration progressive à partir d'un exemple (quick start ou exemple complexe)

* création “from scratch” à partir de la documentation

Une approche itérative en démarrant avec un exemple puis en consultant peu à peu la documentation permet de démarrer rapidement, tout en montant progressivement en compétence au fur et à mesure des besoins.

Intégration statique des fichiers Pour commencer, se référer à un exemple (simple de préférence) et intégrer le code brut CSS, JS et HTML dans les fichiers correspondants de l’application.

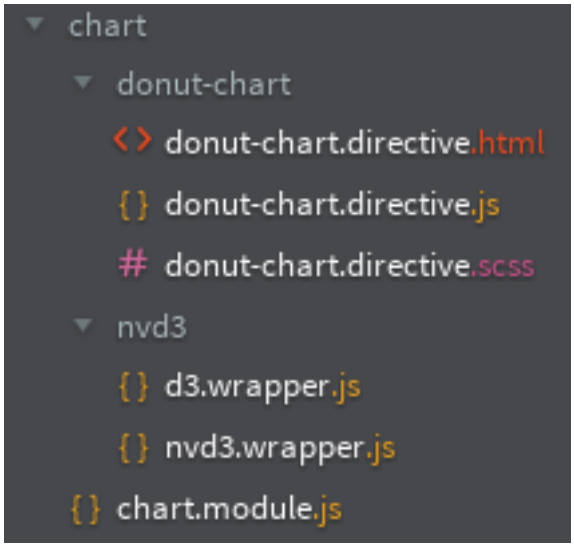

Figure 1: files tree

Vérifier que l’exemple se charge correctement dans la page:

Nettoyage de l’exemple Les différents étapes sont ensuite: * nettoyage du code, style, templates pour retirer les éléments inutiles (par exemple, ici, il y a 2 donuts, alors que notre exemple n’en requiert qu’un seul) * appel aux services wrappers au lieu des services natifs

Définir l’API

Avant de démarrer l’intégration d’une librairie, il faut commencer par concevoir l’API de la directive, exposée via les paramètres du scope.

Parmis les besoins fréquents: * de personaliser l’initialisation du composant * de synchroniser le modèle de données * de réagir aux évènements (ce qui peut être fait directement via le modèle de données)

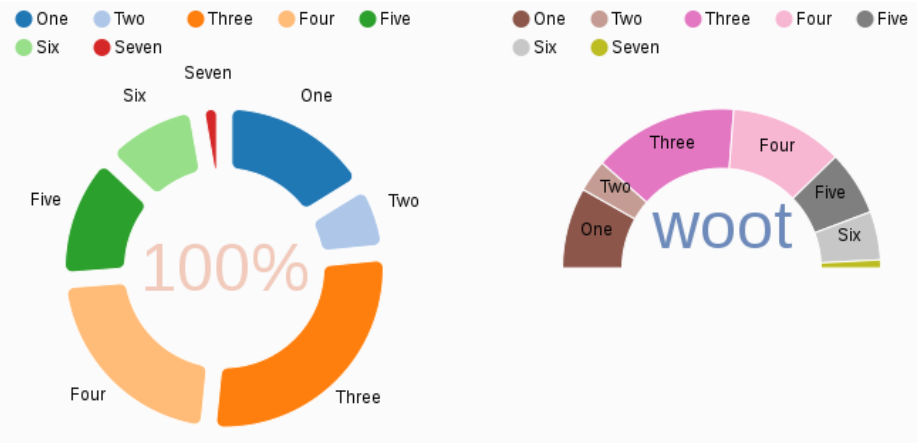

Figure 2: files tree

Pour un usage interne, il n'est pas nécessaire d'exposer à travers des paramètres toutes les fonctionnalités de la librairie tierce, mais seulement ce qui est utile à l'application.

Implémentation de l'API

- implementation des tests unitaires
- implémentation progressive de l'API

Synchronisation du modèle vers la librairie La librairie ne connaissant pas le contexte d'angular, il faut se mettre manuellement à l'écoute du modèle afin de mettre à jour la librairie en cas de changement:

Exemple avec D3:

```
$scope.$watch('vm.chartData', function (chartData) {
    if (chartData) {
          twD3.select('#' + vm.chartConfig.cssId)
            .datum(chartData)
              .transition().duration(1200)
            .call(chart1);
    }
});
```

Mise à jour du modèle depuis la librairie Pour mettre à jour le modèle depuis la librairie, il faut se mettre à l'écoute de ses listeners.

Exemple avec leaflet:

```
vm.map = twLeaflet.map('tw-map-id').setView(vm.config);

vm.map.on('click', function (e) {
    // propagate the event
    vm.mapConfig.events.click(e);

    // or update the model
    vm.item.location = e.latlng;
});
```

D3 - data-driven documents

- Sommaire: Pour aller plus loin avec Angular 1.4
- Page précédente: Interractions avec des bibliothèques non-angular - directives
- Page suivante: Web mapping avec leaflet

Présentation

D3.js permet d'afficher des données sous la forme de graphiques dynamiques.

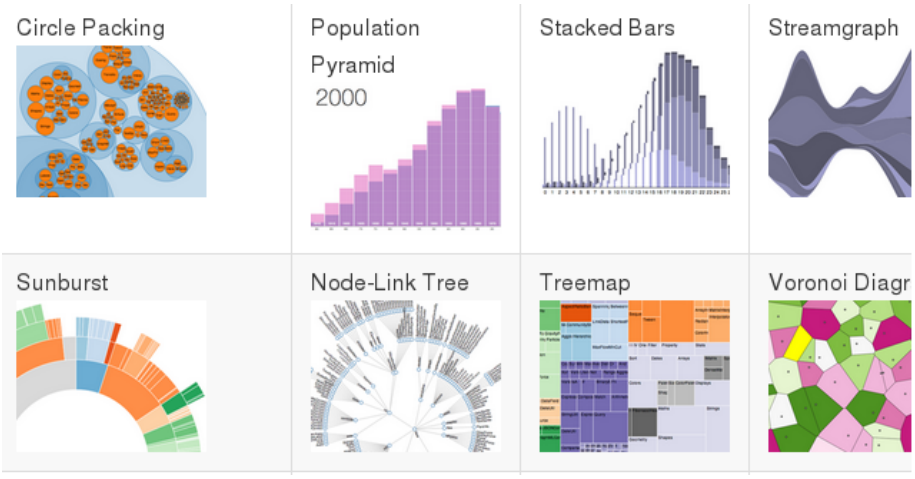

Figure 3: files tree

Resources:

- D3: data-driven documents
- D3 gallery: exemples de réalisation
- D3 and Angular: a list of D3 angular resources
- Vega and D3: a visualization grammar (pour aller plus loin)

NVD3 - composants D3 réutilisables

D3 étant une librairie puissante, mais bas niveau, NVD3 propose des composants réutilisables basés sur D3.

Figure 4: files tree

Resources: * NVD3: re-usable charts and chart components for d3.js

NVD3 - intégration à l'application

Installation de D3 et NVD3 via bower:

```
bower install --save nvd3
```

Création d'un module spécifique:

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

Création d'un wrapper pour chacune des librairies:

```
angular.module('tw.practice.chart').factory('twD3', twD3);

/** @ngInject */
function twD3($window, $log) {
   // ...
}

angular.module('tw.practice.chart').factory('twNvd3', twNvd3);
```

Se reporter au chapitre sur les wrappers.

Resources: * Intégration de librairies non angular

NVD3 - création d'une directive

La documentation étant limitée, la meilleure approche est l'intégration d'un exemple.

Resources: * Intégration de librairies non angular - directives * Exemple donutChart ## Web mapping avec Leaflet

- Sommaire: Pour aller plus loin avec Angular 1.4
- Page précédente: D3 - data-driven documents
- Page suivante: JSON Patch

Introduction

Librairie permettant de manipuler des cartes, d'interragir avec elles et d'y afficher des données (Web mapping).

Les fonds de cartes (tiles) peuvent provenir de Google Map, Bing, Open Street Map, ou encore de fond personnalisés générés à partir de données mixtes (publiques, internes).

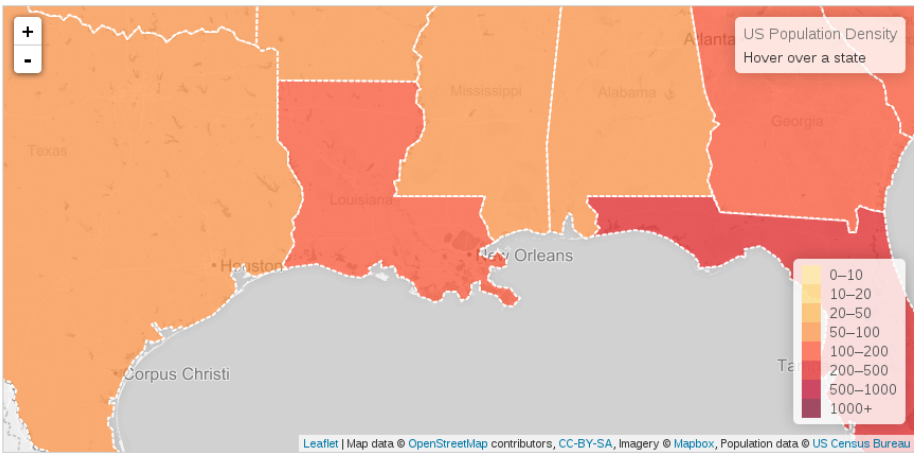

Figure 5: files tree

Resources:

- leaflet
- tilemill: excellent outil de création de tiles
- kosmtik: nouveau projet proche de tilemill
- Mapbox Studio Classic: équivalent de tilemill intégré à Mapbox

Leaflet - utilisation d'une directive tierce

Une directive permet d'utiliser simplement leaflet dans une application angular. Le projet vient d'être intégré à angular-ui.

Elle est pour le moment sous-documentée, mais, en fonction du besoin, il peut-être pertinent de s'appuyer dessus quand elle le sera.

Resources:

- ui-leaflet: directive leaflet

Leaflet - intégration à l'application

Installation via bower de la version en cours de développement:

```
bower install --save leaflet#1.0.0-beta.2
```

Création d'un module spécifique:

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

Création d'un wrapper (en mode noConflict):

```
angular.module('tw.practice.map').factory('twLeaflet', twLeaflet);

/** @ngInject */
function twLeaflet($window, $log) {
    if (!$window.L) {
        $log.error('Global L variable is not available.');
        return null;
    }

    var service = $window.L.noConflict();

    return service;
}
```

Se reporter au chapitre sur les wrappers.

Resources: * Leaflet * Intégration de librairies non angular

Leaflet - création d'une directive

La documentation et les exemples étant bien fournis, il est possible d'encapsuler leaflet via un exemple ou from scratch.

Resources: * Intégration de librairies non angular - directives * Leaflet - documentation * Leaflet - exemples

JSON Patch

- Sommaire: Pour aller plus loin avec Angular 1.4
- Page précédente: Web mapping avec leaflet

Structure

Structure JSON permetant de définir le format d'un patch.

Exemple:

```
{
    "op": "replace",
    "path": "/user/firstname",
    "value": "Paul"
}
```

Structure en 3 parties:

- **op**: opération à effectuer: "add", "remove", "replace", "move", "copy", or "test"
- **path**: chemin de la propriété à modifié dans l'objet spécifié
- **value**: valeur (pour "opérations add, replace et test)
- **from**: chemin d'origine (pour move et copy)

Resources:

- RFC6902: JavaScript Object Notation (JSON) Patch