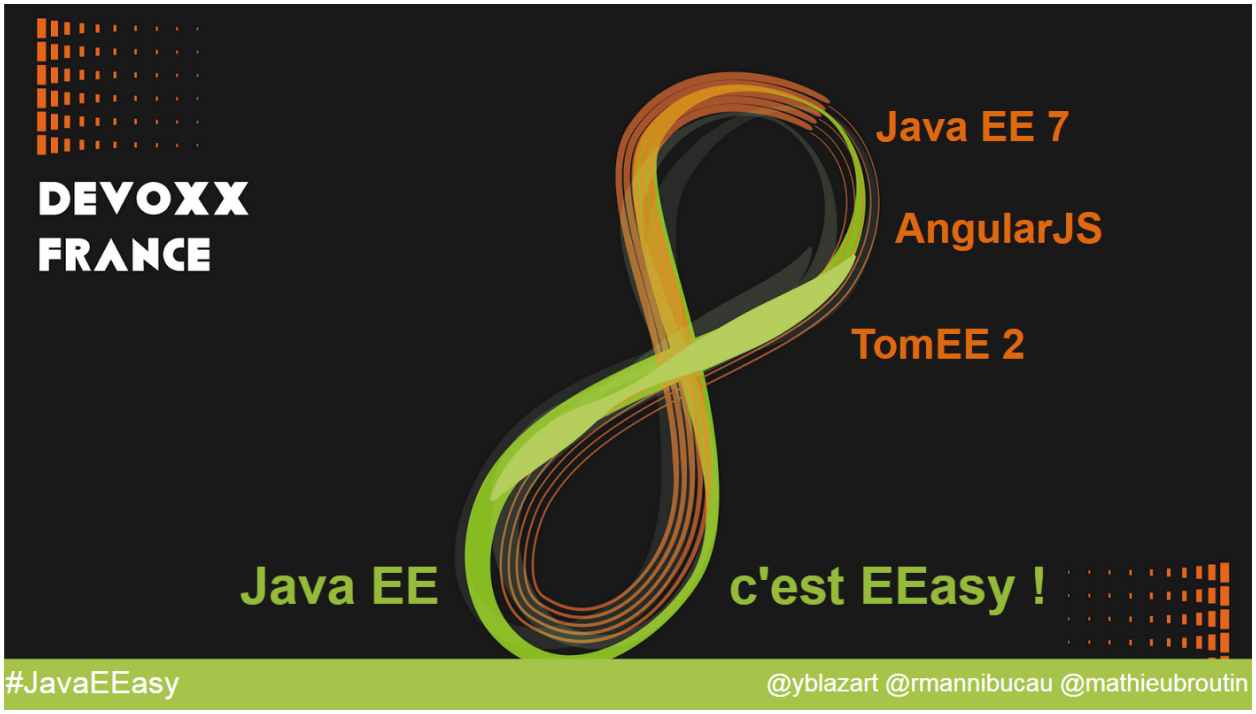


Introduction

Bonjour et bienvenu à ce Hand's on Lab sur Java EE 7, AngularJS et TomEE 2 !
Nous allons vous faire découvrir quelques points de ces technologies et comment elle s'articulent ensemble, c'est fun ; c'est EEasy !

Pré-requis

Votre poste doit avoir :

- un JDK8
- Maven 3.2.5+
- un IDE (eclipse, netbeans, intellij,etc…)
- Et GIT !

Qu'allons nous faire ?

Nous allons réaliser une application qui permet de récupérer l'agenda complet du devoxx en utilisant l'api cfp de devoxx http://cfp.devoxx.fr/api et nous l'afficherons sur le navigateur en mode responsive. Une autre fonctionnalité nous permettra de nous inscrire/retirer des conférences, en ayant un compteur qui se mettra à jour sur tous les clients connectés.

00 - Mise en place

Premiers pas

Une fois le projet récupéré, placez vous sur la branche initiale ; depuis la racine du projet bien sûr.

```
git checkout 00_mise_en_place
```

A la racine, se trouve un fichier devoxx-settings.xml qui est utilisé par maven pour utiliser un nexus local afin que tout le monde évite de télécharger les dépendance sur le net.

- Soit vous paramétrez votre IDE pour utiliser ce fichier
- Soit vous l'utilisez sur la ligne de commande avec l'option -gs

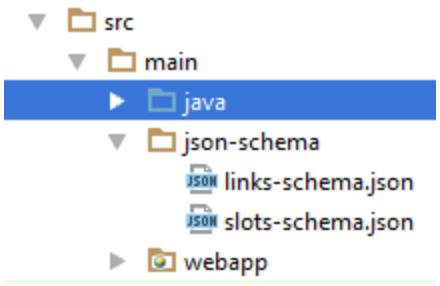
Lançons depuis la ligne de commande ou depuis l'IDE la première exécution maven :

```
mvn -gs devoxx-settings.xml clean compile
```

En lançant le build, nous avons déclenché un plugin maven qui génère un ensemble de classes Java qui servent de modèle pour les échanges avec l'api cfp de devoxx.

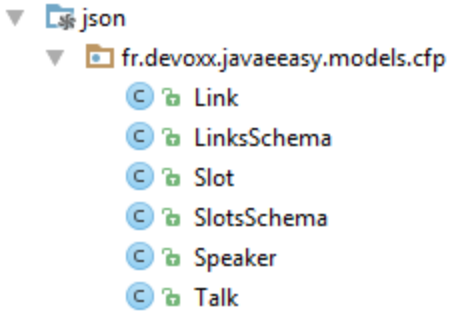
Ce plugin est capable de générer les classes Java depuis un schémas JSON.

Les schémas utilisé sont visibles dans les sources : src/main/json-schema



```xml
<plugin>
    <groupId>org.jsonschema2pojo</groupId>
    <artifactId>jsonschema2pojo-maven-plugin</artifactId>
    <version>0.4.8</version>
    <configuration>
        <sourceDirectory>${basedir}/src/main/json-schema/</sourceDirectory>
        <outputDirectory>${project.build.directory}/generated-sources/json</outputDirectory>
        <targetPackage>fr.devoxx.javaeeasy.models.cfp</targetPackage>
        <annotationStyle>none</annotationStyle>
        <useLongIntegers>true</useLongIntegers>
        <initializeCollections>true</initializeCollections>
        <includeConstructors>true</includeConstructors>
        <usePrimitives>true</usePrimitives>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>generate</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Les sources sont générées dans target/generated-sources :



Comment avons nous obtenu les schémas ?

Le service http://jsonschema.net/ permet de générer des schémas JSON depuis un document JSON valide, c'est une aide précieuse.

Voici les deux documents types qui nous ont servis à générer les schémas.

- Links

```
{
    "links":[{
```

```
      "href": "http://cfp.devoxx.be/api/conferences/DV14/schedules/monday/",
      "rel": "http://cfp.devoxx.be/api/profile/schedule",
      "title": "Schedule for Monday 10th November 2015"
    }]
}
```

- Slots (extrait)

```
{
   "slots":[{
     "fromTimeMillis":-1716607680,
     "toTimeMillis":-1705807680,
     "roomSetup":"theatre",
     "roomName":"Room 4",
     "roomId":"room4",
     "slotId":"uni_room4_monday_10_13h30_16h30",
     "notAllocated":false,
     "day":"monday",
     "fromTime":"13:30",
     "roomCapacity":364,
     "toTime":"16:30",
     "talk":{
       "id":"QKM-0129",
       "summary":"Ever wondered about all the new ..... ",
       "title":"Mastering xPaaS - get down and dirty in the OpenShift Cloud ",
       "talkType":"University",
       "track":"Cloud & BigData",
       "summaryAsHtml":"<p>Ever wondered about all t...",
       "speakers":[{
         "link":{
           "title":"Eric D. Schabell",
           "rel":"http://cfp.devoxx.be/api/profile/speaker",

"href":"http://cfp.devoxx.be/api/conferences/DevoxxFR2015/speakers/8c77c532774576a25bdc3fb6e51e067d5112d1c3"
         },
         "name":"Eric D. Schabell"
       }],
       "lang":"en"
     }
   }]
}
```

01 - Création du client REST de l'api CFP Devoxx et du premier service REST

En exécutant la commande

```
mvn -gs devoxx-settings.xml test
```

On s'aperçoit que le test échoue.

Au terme de cet exercice, nous allons réaliser un service REST qui récupérera les informations du devoxx via le service CFP (mis à disposition par les organisateurs du devoxx) et exposer cette information via notre propre service REST.

Création du service client de l'api cfp devoxx

Complétez la méthode "retreiveSlots" de fr.devoxx.javaeeasy.services.business.DevoxxClient
Cette méthode retourne une "List<Slot>". C'est l'ensemble des slots (conférences) du devoxx.
Cette récupération est faite en deux étapes :

Récupération des urls des services donnant les slots par jour

Cette liste est accessible en JSON à l'url suivante :

```
http://cfp.devoxx.fr/api/conferences/DevoxxFR2015/schedules
```

Ce qui donne le document suivant :


```
{"links":[{"href":"http://cfp.devoxx.fr/api/conferences/DevoxxFR2015/schedules/monday/","rel":"http://cfp.devoxx.fr/api/profile/schedule","title":"Schedule for Monday 10th November 2015"},{"href":"http://cfp.devoxx.fr/api/conferences/DevoxxFR2015/schedules/tuesday/","rel":"http://cfp.devoxx.fr/api/profile/schedule","title":"Schedule for Tuesday 11th November 2015"},{"href":"http://cfp.devoxx.fr/api/conferences/DevoxxFR2015/schedules/wednesday/","rel":"http://cfp.devoxx.fr/api/profile/schedule","title":"Schedule for Wednesday 12th November 2015"},{"href":"http://cfp.devoxx.fr/api/conferences/DevoxxFR2015/schedules/thursday/","rel":"http://cfp.devoxx.fr/api/profile/
```

schedule","title":"Schedule for Thursday 13th November 2015"},{"href":"http://cfp.devoxx.fr/api/conferences/DevoxxFR2015/schedules/friday/","rel":"http://cfp.devoxx.fr/api/profile/schedule","title":"Schedule for Friday 14th November 2015"}]}

Ce contenu sera mappé sur la Class LinksSchema générée précédemment (par le plugin maven).

Récupération des slots par jour

Les urls récupérée permettent ensuite de récupérer la liste des slots pour un jour donné.

Par exemple l'url :

http://cfp.devoxx.fr/api/conferences/DevoxxFR2015/schedules/friday/

Ce qui donne le document suivant (extrait) :

{"slots":[{"roomId":"a_hall","notAllocated":false,"fromTimeMillis":1428645600000,"break":{"id":"dej","nameEN":"Breakfast","nameFR":"Accueil et petit-déjeuner","room":{"id":"a_hall","name":"Exhibition floor","capacity":2300,"setup":"special","recorded":""}},"roomSetup":"special","talk":null,"fromTime":"08:00","toTimeMillis":1428649200000,"toTime":"09:00","roomCapacity":2300,"roomName":"Exhibition floor","slotId":"dej_friday_10_8h0_7h0","day":"friday"},{"roomId":"b_amphi","notAllocated":true,"fromTimeMillis":1428649200000,"break":null,"roomSetup":"theatre","talk":null,"fromTime":"09:00","toTimeMillis":1428650400000,"toTime":"09:20","roomCapacity":826,"roomName":"Amphi Bleu","slotId":"key_b_amphi_friday_10_9h0_9h20","day":"friday"},

Ce contenu sera mappé sur l'objet SlotSchema.

Le service DevoxxClient sera un service CDI, par défaut en java EE 7, tout bean est CDI dans une archive.

Quelques liens intéressants à voir :

http://docs.oracle.com/javaee/7/tutorial/cdi-basic008.htm#GJBBK (en)
http://antoniogoncalves.org/2012/12/19/test-your-jax-rs-2-0-web-service-uris-without-mocks/ (en)
http://docs.oracle.com/javaee/7/tutorial/cdi-basic014.htm#BABJFEAI (en)
http://rmannibucau.developpez.com/tutoriels/cdi/introduction-cdi/ (fr)

Consommer un service Rest

JAXRS 2.0 - API Rest client

Pour attaquer un service REST, utilisez l'api client JAXRS 2.0.

http://www.javabeat.net/creating-restful-webservice-client-api-jax-rs-2-0/
http://docs.oracle.com/javaee/7/api/javax/ws/rs/client/ClientBuilder.html
http://dreamand.me/java/jee7-websocket-decode-custom-message/

Pour décoder les messages JSON, le client jaxrs 2.0 doit connaître le provider qui transforme les documents JSON en objet (et vice versa).

Pour cela faisons un Class.forName("org.apache.johnzon.jaxrs.JohnzonProvider");

Ceci devra être passé au ClientBuilder :

```
restClient =
ClientBuilder.newBuilder().build().register(jsonProvider);
```

Une fois le service DevoxxClient correctement codé, le premier test devrait passer.

Création d'un service Rest pour que notre application puisse à son tour fournir les données

Complétez la méthode "Collection<Slot> getConferences() " de SlotRestService afin qu'elle fournisse la liste des Slot en format JSON depuis l'url /rest/conferences sur le verbe HTTP GET.

Il faut tout d'abord déclarer que notre application va proposer des services REST, et sur quel URL de base .

Pour ce faire explorez l'annotation @ApplicationPath

https://dwuysan.wordpress.com/2012/12/06/starting-with-jax-rs-configuring-javax-ws-rs-applicationpath/

Veuillez utiliser “/rest” comme basePath pour nos services.

Ensuite, veuillez rendre accessible le service accessible par l’url “/rest/conferences”

Explorez l’utilisation des annotations @Path et @GET. N’oubliez pas que les service REST sont accessible depuis “/rest” grace à la class annotée @ApplicationPath

http://www.mkyong.com/webservices/jax-rs/jax-rs-pathparam-example/

Ensuite assurez vous que la méthode renvoi bien un document JSON, explorez @Produces et MediaType.

http://docs.oracle.com/cd/E19798-01/821-1841/gipzh/index.html
https://docs.oracle.com/javaee/6/api/javax/ws/rs/core/MediaType.html

Le code de récupération des slots fonctionnant déjà dans DevoxxClient, utilisons CDI pour s’injecter le service, explorez l’annotation @Inject :

http://docs.oracle.com/javaee/6/api/javax/inject/Inject.html

Pour aller plus loin vous pouvez explorer les scopes des beans CDI : @ApplicationScoped, @RequestScoped etc…

http://www.byteslounge.com/tutorials/cdi-bean-scopes
http://www.byteslounge.com/tutorials/java-ee-7-cdi-bean-scopes

Une fois les tests opérationnels, vous pouvez lancer le TomEE embedded pour voir en vrai votre service REST !

```
mvn -gs devoxx-settings.xml tomee-embedded:run
```

```
http://localhost:8080/rest/conferences
```

Il y a deux utilisateurs configurés par défaut :

- devoxx
- devoxx2

Le mot de passe est identique au login.

La liste des utilisateur est configurer dans le pom.xml.

02 - CDI producers

Si vous n'avez pu aller au bout de l'exercice précèdent
vous pouvez récupérer le correctif :

```
git stash
git checkout 01_client_rest
```

Nous allons maintenant vous faire expérimenter ce qu'est un Producer CDI.

En gros il s'agit d'une factory.
Par défaut en Java EE 7, en CDI tous les beans d'une archive sont des beans CDI, donc lorsque CDI rencontre un @Inject, il construit simplement une instance du bean correspondant qu'il a pu trouver dans l'archive.

Mais parfois cela n'est pas suffisant. Le producer permet de construire spécifiquement une instance selon ses besoins, et selon le point d'injection.

Producer et point d'injection

Nous allons prendre le cas classique d'injection de Logger.

Afin de remplacer les sempiternel Logger.*getLogger*(DevoxxClient.class); que nous retrouvons fréquemment dans nos code, nous allons utiliser le code suivant :

```
@Inject
private Logger LOG;
```

Ajouter ce code par exemple dans la class DevoxxClient et utiliser le logger dans la méthode retreiveSlots() : "LOG.info("xxxx")".

Mais il faut maintenant produire une instance du logger qui correspond au besoins de la class DevoxxClient.

Explorez l'utilisation de @Produces et de l'InjectionPoint et créez une class LoggerProducer.

http://rmannibucau.developpez.com/tutoriels/cdi/introduction-cdi/
http://www.devsniper.com/injectable-logger-with-cdi/
https://docs.oracle.com/javaee/6/api/javax/enterprise/inject/spi/InjectionPoint.html

Vérifiez que votre producer de log est bien en ApplicationScope pour éviter la multi-instanciation de ce producer.

Attention aussi à vos noms de package pour les imports.

@Qualifier

Un qualifier permet de typer une injection lorsque l'on fait appel à un Producer.
Cela permet :

- D'injecter des types de base : String, Integer, Class
- De paramétrer l'injection

Nous allons utiliser ce qualifier et les producer attachés pour paramétrer la classe DevoxxClient.

http://piotrnowicki.com/2012/06/inject-java-properties-in-java-ee-using-cdi/

Commencez par créer un Qualifier @Configuration.
Ce qualifier devra avoir deux attribut de type String en @NonBinding, value() et otherwise().

value() servira à spécifier le nom de la propriété, otherwise() sa valeur par défaut.

Ce qualifier servira à annoter des @Inject sur DevoxxClient pour injecter les valeur de paramétrage.

Veuillez rendre au minimum paramétrable les propriétés suivantes sur la class DevoxxClient avec ce qualifier et des @Inject :

- url de base de l'api cfp du devoxx, avec pour clef : "client.url", de type String
- class du JSonProvider, avec pour clef : "client.provider", de type Class

N'oubliez pas de renseigner les valeur par défaut dans otherwise()

Il ne reste plus qu'a créer la class ConfigurationProducer qui servira à alimenter ces injections.

Cette classe doit aller chercher dans System.getProperties() la clef correspondant à l'attribut value() du qualifier. Si cette clef n'existe pas, elle doit alors utiliser la valeur de l'attribut otherwise().

Créez les méthode @Produces du producer correspondant à chaque besoin d'injection de votre class DevoxxClient.

Tips : Vérifiez vos @Inject

03 - Schedules et events

Si vous n'avez pu aller au bout de l'exercice précèdent
vous pouvez récupérer le correctif :

```
git stash
git checkout 02_producers
```

Nous allons maintenant vous faire expérimenter votre premier EJB (si si !) et les events CDI.

Partons du principe que l'agenda du devoxx peut changer à tout moment.

Mais que la quantité de donnée retournée représenterais une charge trop grande sur leurs serveurs si nous avions de nombreux client et si nous continuons à être simple passe plat.

Nous allons donc mettre en place un service en tâche de fond, un scheduler qui va récupérer les slots depuis l'api devoxx toutes les minutes et les mettre à disposition de nos autres services.

Le scheduler

Commencer par vous créer une classe RetreiveSlotScheduler.

Cette class devra toute les minutes utiliser le service DevoxxClient pour récuperer les slots.

N'oublier pas de faire une première récupération au démarrage de l'EJB ET au démarrage de l'application (@Startup).

```
http://www.adam-bien.com/roller/abien/entry/simplest_possible_ejb_3_16
```

http://www.mastertheboss.com/javaee/ejb-3/ejb-31-tutorial
http://docs.oracle.com/javaee/6/api/javax/ejb/Startup.html

Une fois ceci fait, en lançant le tomee embedded, vous devriez voir toutes les minutes le log d'appel de DevoxxClient.

```
mvn -gs devoxx-settings.xml tomee-embedded:run
```

Utilisons les events

Pour mettre à disposition des autre composants de l'application la liste des Slots, nous allons utiliser les events CDI.

http://docs.oracle.com/javaee/6/tutorial/doc/gkhic.html

Commençons par créer une class SlotUpdateEvent. Celle ci doit simplement contenir la collection de Slot.

Puis, faites en sorte que le schedule fire l'event SlotUpdateEvent toutes les minutes.

Ensuite, pour assurer un découplage maximal, créez une classe SlotHolder. Celle ci doit écouter l'évènement SlotUpdateEvent et récupérer les informations, pour les mettre à disposition des autres composant de l'application. Choisissez bien le scope de ce bean CDI.

Enfin, utilisez SlotHolder dans SlotRestService, en lieu et place de l'appel direct au service DevoxxClient.

En bonus, vous pouvez créer une classe supplémentaire SlotLogger, qui écoutera aussi l'evenement SlotUpdateEvent afin de logger le nombre de slots récuperés et pourra logger cette information.

Une fois ceci fait, en lançant le tomee embedded, vous devriez voir toutes les minutes le log d'appel de DevoxxClient, et eventuellement les traces du SlotLogger.

```
mvn -gs devoxx-settings.xml tomee-embedded:run
```

04 - Interceptors

Si vous n'avez pu aller au bout de l'exercice précèdent
vous pouvez récupérer le correctif :

```
git stash
git checkout 03_schedule_events
```

Les Interceptors (JSR-318) permettent d'ajouter du comportement à une méthode ou une classe.

Dans cet exercice, nous allons créer un Interceptor qui va afficher dans la log le temps passé pour exécuter une méthode.

Pour activer l'interceptor, commençons par créer l'annotation que sera posée sur le méthodes dont on veut tracer le temps d'exécution. Nommez cette annotation Benchmark, veuillez explorer l'annotation @InterceptorBinding :

http://docs.oracle.com/javaee/6/api/javax/interceptor/InterceptorBinding.html

Ensuite veuillez créer l'interceptor même : BenchmarkInterceptor.

http://www.mastertheboss.com/jboss-frameworks/cdi/interceptors-and-decorators-tutorial

N'oubliez pas de le déclarer dans le fichier beans.xml pour l'activer.

Vous pouvez maintenant utilisez cet interceptor sur vos services, sur DevoxxClient par exemple.

05 - AngularJS

Si vous n'avez pu aller au bout de l'exercice précèdent
vous pouvez récupérer le correctif :

```
git stash
git checkout 04_interceptors
```

Nous allons maintenant vous faire expérimenter AngularJS.

Actuellement, le script app.js est quasiment vide.

```
angular.module('DevoxxApp', []);
```

Ajoutons y la dépendance ui.bootstrap pour rendre compatible Angular avec ce framework css.

```
angular.module('DevoxxApp', ['ui.bootstrap']);
```

Ensuite, il faut indiquer dans le fichier index.html la section qui correspond à l'application angular :

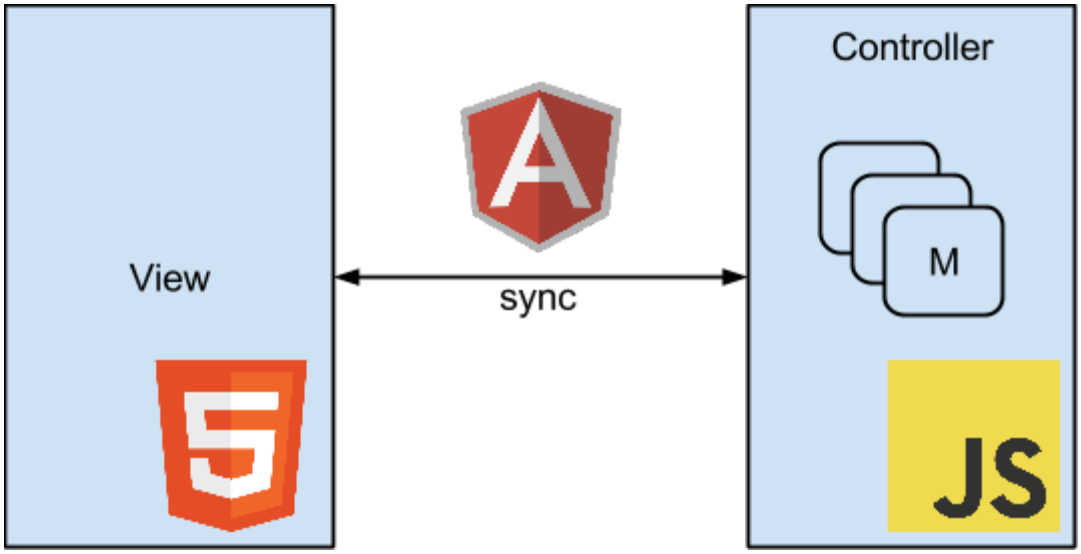
```
<html ng-app="DevoxxApp">
```

Nous pouvons voir que dans le fichier index.html, il y a une inclusion du fichier views/conferences.html.

```
<div ng-include="'views/conferences.html'"></div>
```

Ce fichier est vide actuellement.

AngularJS implémente le pattern MVC :



La **vue** est contenue dans le fichier HTML, pour une portion du fichier HTML, on peut lui lier un **contrôleur**.
Dans ce contrôleur on peut exposer des données à la vue, en placant les données à exposer dans le **$scope** du contrôleur. AngularJS s'occupe de synchroniser automatiquement la vue avec les données exposées du contrôleur.

Commençons par placer dans la vue une liste de choix pour sélectionner le jour et un champs de saisie pour filtrer

```
<div class="toolbar row">
  <div class="col-md-6">
    <select ng-model="day" class="form-control">
      <option value="wednesday">Mercredi</option>
      <option value="thursday">Jeudi</option>
      <option value="friday">Vendredi</option>
    </select>
  </div>
  <div class="col-md-6">
    <input type="text" ng-model="selected.talk.title" class="form-control" placeholder="Rechercher une conference"
/>
  </div>
</div>
```

L'attribut ng-model indique que la valeur sélectionnée sera placée dans la variable *day* du $scope du controller associé à cette vue.

De même que la saisie dans le champs input sera placé dans une structure JSON (selected.talk.title) dans le $scope du controller.

Il nous fait maintenant spécifier le controller de cette vue. Pour ce faire ajouter le dans le fichier index.html :

```
<div ng-include="'views/conferences.html'" ng-controller="ConferenceCtrl"></div>
```

Ensuite nous devons ajouter ce controller au niveau du module angular (app.js).

```
angular.module('DevoxxApp', ['ui.bootstrap'])
.controller('ConferenceCtrl', function ($scope) {
});
```

Bien ! Mais ce n'est pas suffisant, il nous faut afficher les conférences !

Commençons par récupérer l'ensemble des données des conférences. Dans le fichier app.js modifiions le controller :

```
angular.module('DevoxxApp', ['ui.bootstrap'])
.controller('ConferenceCtrl', function ($scope, $http) {
    $http.get('rest/conferences')
     .success(function (result) {
        $scope.agenda = result.filter(function (key) {
            return !!key.talk;
        });
     });
});
```

Nous avons ici ajouter l'injection du service **$http**. Ce qui nous permet de déclencher un appel REST vers /rest/conferences et de stocker le resultat dans le controller ($scope) sous le nom *agenda*.
Le résultat de l'appel REST est filtré afin d'éliminer les slots ne concernant pas de conférences (les breaks par exemple).

Une fois les données acquises, il faut les afficher. Nous allons vous montrer comment réaliser une directive simple.

Une directive est une notion AngularJS, qui permet de créer ses propres composants graphique. Une directive peut contenir sont propre contrôleur et son propre scope. Cet indépendance de scope permet de faire de cet directive un composant réutilisable.

Ajoutons la définitions de la directive dans le fichier apps.js

```
angular.module('DevoxxApp', ['ui.bootstrap'])
.controller('ConferenceCtrl', function ($scope, $http) {
   $http.get('rest/conferences')
    .success(function (result) {
       $scope.agenda = result.filter(function (key) {
          return !!key.talk;
       });
    });
})
.directive('slot', function () {
  return {
     scope: {
        event: '='
     },
     templateUrl: 'views/slot.html'
  };
});
```

Ceci indique que lorsque angular rencontre un tag <slot />, il le remplace par le contenu du fichier template views/slot.html. On voit aussi que le tag slot définira une variable de context par tag slot, 'event' qui sera transmise tel quel ('=').

Il ne nous reste plus qu'à écrire le template de la directive slot !

Dans le fichier slot.html, ajoutez le code suivant :

```
<div class="content" title="{{event.slotId}}">
  <div class="time">{{event.fromTime}} - {{event.toTime}} </div>
  <div class="title">{{event.talk.title}}</div>
  <div class="details">{{event.roomName}}:
    <span ng-repeat="speaker in event.talk.speakers">
    {{speaker.name}} {{$last ? '' : ', '}}
    </span>
  </div>
</div>
```

Un event sera un contenu de slot. On affiche donc le slotId et différentes informations. L'attribut ng-repeat est ici l'équivalent d'un for-each et nous voyons ici l'utilisation d'un variable spéciale $last.

Remarquez la notation **{{..}}**. Celle-ci permet d'afficher la valeur d'une variable contenue dans le scope du contrôleur (ou plus généralement évaluer une expression).

Il ne reste plus qu'a itéré sur les données récupérées pour générer tous les "slots".

Dans le fichier conferences.html, ajoutez l'utilisation de la directive slot :

```
<div class="toolbar row">
  <div class="col-md-6">
    <select ng-model="day" class="form-control">
      <option value="wednesday">Mercredi</option>
      <option value="thursday">Jeudi</option>
      <option value="friday">Vendredi</option>
    </select>
  </div>
  <div class="col-md-6">
    <input type="text" ng-model="selected.talk.title" class="form-control" placeholder="Rechercher une conference"
/>
  </div>
</div>
<slot
  ng-repeat="evt in agenda | filter:{day : day} | filter:selected | orderBy:'fromTimeMillis'"
  event="evt">
</slot>
```

Ici nous pouvons constater que :

- la directive slot est démultiplié grace à l'utilisation d'un ng-repeat sur les données comprises dans l'agenda, récupéré lui même grace à l'appel REST initiale.
- Le contenu de ce resultat est filtré :
    - en correspondance avec le jour selectionné (day)
    - en correspondance avec l'objet JSON contenu dans la variable selected (talk.title)
- Le tout ordonné par la date de départ.
- Chaque élement de l'agenda est ensuite transmis au context de la directive sous le nom 'event'.

06 - JPA

Préparations

Si vous n'avez pu aller au bout de l'exercice précèdent
vous pouvez récupérer le correctif :

```
git stash
git checkout 05_angular
```

Introduction

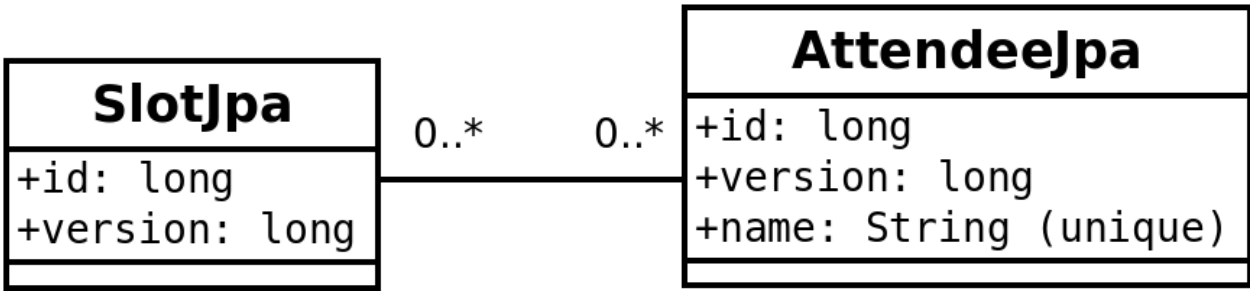
Nous allons maintenant vous faire expérimenter l'utilisation de JPA. La première étape sera de modéliser notre stockage en base de donnée.

Pour définittion :

- un Slot est une conférence/HOL/BOF/Quickies
- Un Attendee est un participant, qui s'incrirera et se retirera de un ou plusieurs Slots.

Modèle JPA

Pour cette exercice on restera très simple en utilisant le model suivant :



Ce modèle nous servira dans l'exercice à associer aux slots leurs participants.

Commencez par créer ces entités dans le package **fr.devoxx.javaeeasy.services.jpa**.

Si vous n'êtes pas familier avec JPA regardez les documentations des classes suivantes :

```
http://docs.oracle.com/javaee/7/api/javax/persistence/Entity.html
http://docs.oracle.com/javaee/7/api/javax/persistence/Id.html
```

http://docs.oracle.com/javaee/7/api/javax/persistence/ManyToMany.html
http://docs.oracle.com/javaee/7/api/javax/persistence/Version.html

Une fois votre modèle crée, il faut créer le descripteur de fichier JPA:

- persistence.xml.

Pour un war celui-là se met généralement dans le dossier WEB-INF de notre application.

il y a un exemple de ce fichier sur cette page :

https://docs.jboss.org/hibernate/entitymanager/3.5/reference/en/html/configuration.html

- Utilisez “devoxx” comme nom de persistence unit.
- Pour la datasource JTA, utilisez le nom : “devoxx”
- Pour la datasource non JTA, utilisez le nom : “devoxxNonJTA”
- Listez vos deux classes modèle.
- Enfin, comme avec TomEE 2.0 nous utilison OpenJPA comme implémentation de l’API JPA 2.0, ajoutez la propriété suivante pour générer le schéma de la base au démarrage de la datasource :

  <property name="openjpa.jdbc.SynchronizeMappings"
  value="buildSchema(ForeignKeys=true)"/>

Une fois votre modèle et notre descripteur prêts et pour s’assurer qu’il fonctionne sans surprises il faut instrumenter les classes (c’est ce qui permet au provider JPA de suivre ce que vous faites pour appliquer les modifications au moment du commits).

Comme nous utilisons OpenJPA nous utilisons le plugin maven le openjpa-maven-plugin (http://openjpa.apache.org/enhancement-with-maven.html).

Dé-commentez ce plugin dans le fichier pom.xml afin de le déclencher sur la phase process-classes du build.

Création des services

Ouf ! Maintenant tout est en place pour développer nos services.
Ajoutez et complétez la classe suivante :

```
package fr.devoxx.javaeeasy.services.business;
import fr.devoxx.javaeeasy.services.jpa.SlotJpa;
import javax.enterprise.context.ApplicationScoped;
import javax.transaction.Transactional;
import java.util.Collection;
import java.util.Map;
@ApplicationScoped
public class SlotService {
  /*
   * récupère un SlotJpa par son ID
   */
  @Transactional
  public SlotJpa findById(final String id) {
  }
  /*
   * Retourne le nombre d'attendee pour le slot passé en paramètre
   */
  public int count(final SlotJpa slot) {
  }
  /*
   * Retourne la liste des slotId auquel un attendee est inscrit
   */
  @Transactional
  public Collection<String> getAllocatedSlotIdsForAttendee(String name) {
  }
  /*
   * Retourne une map dont les clefs sont les slotId et les valeur
   * le nombre d'attendee par Slot
   */
  public Map<String, Integer> getAttendeesCountGroupedBySlotId() {
  }
}
```

Puis celle ci :

```
package fr.devoxx.javaeeasy.services.business;
import fr.devoxx.javaeeasy.models.event.SlotUpdateEvent;
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.event.Observes;
import javax.transaction.Transactional;
@ApplicationScoped
public class SlotBackgroundService {
  /*
   * Sauve les Slot non déjà présent en base sur réception de l'event
   */
  @Transactional
  public void saveConferences(@Observes final SlotUpdateEvent conferences) {
  }
}
```

Enfin, vous pouvez brancher les services REST écrit lors du précèdent exercice sur Angular :

- Faites en sorte que SlotRestService.getAttendeesBySlotId appel SlotService.getAttendeesCountGroupedBySlotId.
- Faite en sorte que AttendeeRestService.getSlotIds appel SlotService.getAllocatedSlotIdForAttendee

Vous aurez peut être besoin de lire ce document :

http://docs.oracle.com/javaee/6/tutorial/doc/gmgli.html

07 - WebSocket

Si vous n'avez pu aller au bout de l'exercice précèdent
vous pouvez récupérer le correctif :

```
git stash
git checkout 06_JPA
```

Afin de permettre à nos utilisateur de voir en direct-live les choix des participants (et mettre un peu de stress pour leur propre choix) on va afficher le nombre de participant par conférence. Lors de l'affichage du détail de la conférence on pourra alors s'enregistrer pour une conférence ou se retirer si on y participait.

Pour cela deux parties sont nécessaires:

1. deux endpoints REST pour dire que "je participe" ou "je ne participe plus". Cela ajoutera/enlévera le participant de la relation avec la conférence.
2. une websocket qui gérera les utilisateurs connecté et nous autorisera à broadcaster la mise à jour lors de l'appelle à aux endpoints précédents. On broadcastera du json de la forme {"slotId":"xxx","attendees":nombre}

Je participe à une conf ! Ou non, je n'y vais plus ….

Coté Java

Commencez par ajouter une méthode REST dans SlotRestService pour que le front Angular puisse signifier que l'on souhaite participer à une conf. Faite en sorte que cette méthode réponde au verbe HEAD et sur l'url REST /rest/conferences/attendees/increment/{id}

Cette méthode doit ensuite appeler le service SlotService pour attacher l'attendee au slot. N'oubliez pas de créer l'attendee si celui ci n'existe pas encore.

Prévoyez aussi que SlotRestService devra broadcaster l'information d'inscription. Prévoyez de broadcaster l'objet SlotJpa une fois l'inscription effectué.

Faites les même opération pour se desincrire de la conf. Faite en sorte que cette méthode réponde au verbe HEAD et sur l'url REST /rest/conferences/attendees/decrement/{id}

ATTENTION aux transaction ! ;)

Ensuite, veuillez ajouter un nouveau service REST qui devra répondre au verbe HTTP GET en JSON sur l'ulr /rest/attendee/slots. Ce service renverra une Collection<String> pour donner tous les slotId auquel l'utilisateur connecté est inscrit. Pour ce faire votre service utilisera la méthode getAllocatedSlotIdsForAttendee de SlotService.

Coté Angular

Commençons par faire surgir une popup détaillant le talk, et nous permettant de nous inscrire/désinscrire.

Ajouter les différent modules angular nécessaires pour la suite :

```
angular.module('DevoxxApp', ['ui.bootstrap', 'ngSanitize', 'ngAnimate'])
```

Dans le fichier conferences.html, dans la directive slot, nous pouvons ajouter un attribut ng-click qui appel un select(evt). Ceci appelera donc la méthode javascript select du controller ConferenceCtrl.

Il faut en premier lieu s'injecter l'objet $modal, dans le script app.js :

```
.controller('ConferenceCtrl', function ($scope, $http, $modal) {
```

Ensuite, ajoutez le code suivant dans le controller :

```
$scope.select = function (conference) {
  $modal.open({
    templateUrl: 'views/conference-dialog.html',
    controller: 'ConferenceDialogCtrl',
    resolve: {
      conference: function () {
        return conference;
      }
    }
  }).result.then(function (result) {
      var action = result.attending ? 'increment' : 'decrement';
      $http.head('rest/conferences/attendees/' + action + '/' + result.conference.slotId)
    });
};
```

Nous voyons ici que :

- L'appel de cette méthode va ouvrir une fenêtre modale.
- Son template est dans views.
- Cette fenêtre modal utilisera un controller nommé ConferenceDialogCtrl

- Et dans le $scope de ce controller on insère une variable nommée “conference” qui prend en valeur l’argument passé à la fonction select, en l'occurrence “evt” qui vient lui même du ng-repeat présent dans la directive slot.
- Le “result.then” permet de réagir soit quand l’internaute aura “validé” le formulaire, ou cliqué en dehors.

Il nous faut donc maintenant ajouter ConferenceDialogCtrl. Ajoutez le dans le module DevoxxApp.

```
.controller('ConferenceDialogCtrl', function ($scope, $modalInstance, conference, $http) {
  $scope.conference = conference;
  $http.get('rest/attendee/slots').then(function (result) {
    $scope.isAttending = result.data.indexOf(conference.slotId) >= 0;
  });
  $scope.submit = function () {
    $modalInstance.close({attending: !$scope.isAttending, conference: conference});
  };
  $scope.cancel = function () {
    $modalInstance.dismiss();
  };
})
```

Ici nous pouvons constater que :

- Le controller reçoit la valeur de conference transmise par l’attribut “resolve” du $modal.open.
- Immédiatement un premier appel REST est fait avec $http.get pour vérifier que l’utilisateur est enregistré à la conf ou non. Le résultat est mis dans $scope.isAttending.
- la fonction $scope.submit permet de fermer la fenêtre modal en envoyant un objet JSON qui sera reçu dans la variable result de la fonction result.then de la méthode select précédemment vue.
- le $scope.cancel est appelé par Angular quand l’internaute clique ailleurs que dans le fenêtre modale.

Il nous reste maintenant à écrire le code html de cette fenêtre de dialogue modale. Remplissez le fichier conference-dialogue.html :

```
<div class="modal-header">
  <span class="pull-right close" ng-click="cancel()">&times;</span>
  <h3 class="modal-title">{{conference.talk.title}}</h3>
</div>
<div class="modal-body row">
  <div class="col-md-8">
    <div ng-bind-html="conference.talk.summaryAsHtml"></div>
  </div>
  <div class="col-md-4">
    <div>
```

```
        {{conference.fromTime}} - {{conference.toTime}}
      </div>
      <div>
        Where : {{conference.roomName}}
      </div>
    </div>
  </div>
  <div class="modal-footer">
    <button class="btn btn-warning" ng-click="submit('unregister')" ng-show="isAttending">Se desabonner</button>
    <button class="btn btn-primary" ng-click="submit('register')" ng-show="!isAttending">S'enregistrer</button>
  </div>
```

Quelques éléments intéressants à noter ici :

- La directive ng-bind-html
- La directive ng-show

Diffusons les informations.

Coté Java

Nous voulons afficher pour chaque slot le nombre de participants.

Pour ceci commencez par ajouter une methode sur SlotRestService, qui répondra sur le verbe HTTP GET, sur le chemin /rest/conferences/attendees en JSON, et qui renverra une Map<String,Integer>. Cette méthode devra utiliser le service getAttendeesCountGroupedBySlotId du service SlotService.

Puis explorez comment utliser les websocket en Java EE 7.

https://netbeans.org/kb/docs/javaee/maven-websocketapi.html
https://javaee-spec.java.net/nonav/javadocs/javax/websocket/server/ServerEndpoint.html
http://docs.oracle.com/javaee/7/api/javax/websocket/OnOpen.html
http://docs.oracle.com/javaee/7/api/javax/websocket/OnClose.html

Créez un endpoint websocket sur l'url "/ws/planning"

Créez une classe SlotAttendees avec deux champs slotId et attendees (int). attendees qui contiendra le nombre de participants au slot.

Puis spécifiez un JSONEncoder à votre endpoint websocket pour traduire la classe SlotAttendees en JSON.

```
public class JsonEncoder implements Encoder.TextStream<SlotAttendees> {
  private JsonGeneratorFactory factory;

  @Override
  public void init(final EndpointConfig endpointConfig) {
    factory = Json.createGeneratorFactory(Collections.<String, Object>emptyMap());
  }

  @Override
  public void encode(final SlotAttendees object, final Writer writer) throws EncodeException, IOException {
    factory.createGenerator(writer)
            .writeStartObject()
              .write("slotId", object.getSlotId())
              .write("attendees", object.getAttendees())
            .writeEnd()
          .close();
  }

  @Override
  public void destroy() {
    // no-op
  }
}
```

Prévoyez une méthode pour l'envoi de SlotAttendees à tous les peers connectés.
Puis branchez votre SlotRestService à votre système de broadcat.

Coté Angular

au niveau du module DevoxxApp, ajoutez l'écoute de la websocket :

```
.run(function ($rootScope, $location) {
  var websocket_url = "ws://" + $location.host() + ":" + $location.port() + "/ws/planning";
  var ws = new WebSocket(websocket_url);
  ws.onmessage = function (message) {
    $rootScope.$apply(function () {
      $rootScope.$broadcast('registred', angular.fromJson(message.data));
    });
  };
})
```

Ce code écoute les messages du websocket puis les broadcast dans angular.

Il faut ensuite écouter ces messages, ajoutez le code suivant dans le controller ConferenceCtrl :

```
$scope.$on('registred', function (event, slot) {
  if ($scope.attendees) { // in case this message arrives before attendees is initialized
     $scope.attendees[slot.slotId] = slot.attendees;
  }
});
```

Ce bout de code insère le message dans la variables $scope.attendees. Cette variable est l'image correspond à la map du nombre d'attendees par slotId.

Il faut bien sur que cette map soit initialisé au départ. Pour ce faire ajoutez ce bout de code dans le controller ConferenceCtrl :

```
$http.get('rest/conferences/attendees')
  .success(function (result) {
     $scope.attendees = result;
  });
```

Ceci utilise la méthode getAttendeesBySlotId de la classe SlotRestService.

Ensuite nous devons afficher ces informations.

Dans le fichier conferences.html dans la directive slot ajoutez la récuperation du nombre d'attendees pour le slot :

```
<slot
    ng-repeat="evt in agenda | filter:{day : day} | filter:selected | orderBy:'fromTimeMillis'"
    event="evt"
    attendee="attendees[evt.slotId]"
    ng-click="select(evt)">
</slot>
```

Dans le fichier slot.html, ajoutez l'affichage du nombre d'attendee.

```
<div class="content" title="{{event.slotId}}">
 <div class="pull-right">
   <span class="label" ng-class="{'label-success': attendee > 0, 'label-default': !attendee}">
     {{attendee || '0'}}
     <i class="fa fa-user"></i>
   </span>
 </div>
 <div class="time">{{event.fromTime}} - {{event.toTime}} </div>
 <div class="title">{{event.talk.title}}</div>
 <div class="details">{{event.roomName}}:
   <span ng-repeat="speaker in event.talk.speakers">
```

```
    {{speaker.name}} {{$last ? '' : ', '}}
    </span>
  </div>
</div>
```

Et enfin dans app.js, ajoutez la transmission de la variable attendee dans la directive slot :

```
.directive('slot', function () {
  return {
    scope: {
      event: '=',
      attendee: '='
    },
    templateUrl: 'views/slot.html'
  };
```

Voilà qui clôture l'ensemble des exercices.

Note: pour les plus rapides utilisez un DTO pour le mesasge broadcasté et implémenté un javax.websocket.Encoder utilisant JSON-P (javax.json.stream.JsonGeneratorFactory) pour généré le JSON.

08 In fine

Si vous n'avez pu aller au bout de l'exercice précèdent
vous pouvez récupérer le correctif :

```
git stash
git checkout 07_websocket
```