TD n°6



* parcours 1

- parcours 2 |

- Projet **boards**
- Application gestion de projets SCRUM

Objectifs

1. Factorisation du code
2. Réutilisation

Prise en main

Vous pouvez au choix :

- Partir de votre propre projet et y inclure les éléments du projet Github
- Partir du projet GitHub pour y intégrer vos propres fonctionnalités (recommandé)

Dans le cas 2 (récupération du projet Git) : Dézipper le projet, le renommer (en board-app), aller dans le dossier, puis exécuter :

```
npm install
npm install -g bower
bower install
```

Vérifier le nom de la base de données (propriété **namespace**) dans le fichier **app/adapters/application.js**

Démarrer les serveurs : mongodb, restheart, ember

Factorisation des vues

template partagé

Une route peut redéfinir la propriété **templateName** pour permettre l'utilisation d'un même template par

plusieurs routes :

C'est le cas des routes **projects/new** et **projects/update/:project_id** qui partagent le template **projects/frm.hbs** :

```
import...
...
export default Route.extend({
  templateName: 'projects/frm',
  ...
 }
```

Composants

Composant ui-form

Le composant **ui-form** permet de factoriser la création d'un formulaire de modification d'un model (il insère les parties **header** et **footer** du formulaire):

Exemple d'utilisation :

```
{{#ui-form
           newValue=model.newProject
           oldValue=model.oldProject
           validation="save" cancel="cancel"
           bigIcon='table' isNew=model.isNew
           header='Project'}}
  <div class="field">
    <label for="name">Name</label>
    {{input id="name" placeholder="Name" value=model.newProject.name}}
  </div>
  ...
{{/ui-form}}
```

Attribut	Rôle
oldValue	objet à modifier (model)
newValue	copie du model modifiée dans le formulaire
isNew	Détermine s'il s'agit d'une insertion ou d'une mise à jour
bigIcon	Icône affichée dans l'en-tête
header	Type de l'objet affiché dans l'en-tête
validation	Action à associer à la validation des données, l'action correspondante devra avoir pour prototype : (oldValue,newValue)
cancel	Action à associer à l'annulation des modifications

Composant ui-table

Le composant **ui-table** permet d'afficher une liste d'objets dans une table (semantic-ui) :

Exemple d'utilisation :

```
{{#ui-table
            elements=model.projects
            fields=model.fields
            operations=model.operations
            emptyMessage="No project to display"}}
{{/ui-table}}
```

Attribut	**Rôle**
elements	Tableau des objets (models) à afficher
fields	Tableau des champs à afficher : 2 formats acceptés pour chaque champ (chaîne ou objet du type **{name:'attribute',caption:'caption',component:'compo-name'}**)
operations	Tableau des opérations à afficher sous forme de bouton et associées à chaque objet, chaque élément permet de définir un lien : **{icon:'remove red',link:'projects.delete'}** ou une action **{icon:'edit red',action:'edit'}**
emptyMessage	Message à afficher si le tableau des objets est vide

L'utilisation de composants pour les champs affichés (**fields**) est disponible à partir du commit bf991db

Exemple d'utilisation : Affichage de l'identity du dev dans un label

Créer un composant pour le label :

```
ember g component lbl-value
```

Implémenter le composant (**value** représente la valeur du champ):

```
<span class="ui label"><i class="ui user icon"></i>{{value}}</span>
```

Mentionner le composant dans la déclaration fields de la route :

```
import Route from '@ember/routing/route';
import RSVP from 'rsvp';

export default Route.extend({
  model(){
    return RSVP.hash({
      developers:this.get('store').findAll('developer',{include:"projects"}),
      fields:[{name:'identity',component:'lbl-value',caption:'Labeled identity'}],
      operations:[{icon:'red
remove',link:'developers.delete'},{icon:'edit',link:'developers.update'}]
    });
  }
});
```

Factorisation des routes

Il est possible d'utiliser l'héritage pour factoriser les opérations effectuées par des routes ayant des rôles équivalents.

Exemple avec la suppression d'une instance de Model :

Route de base permettant la supression d'une instance de model quelconque :

```
import Route from '@ember/routing/route';

export default Route.extend({
  getRedirectRoute(){},
  actions:{
    cancelDeletion(){
      this.transitionTo(this.getRedirectRoute());
    },
    delete(object){
      object.destroyRecord();
      this.transitionTo(this.getRedirectRoute());
    }
  }
});
```

Suppression d'un Project : **route projects/delete/:project_id** héritant de **delete-route** :

```
import DeleteRoute from '../delete-route';

export default DeleteRoute.extend({
  getRedirectRoute(){
    return "projects";
  }
});
```

Suppression d'un Developer : **route developers/delete/:developer_id** héritant de **delete-route** :

```
import DeleteRoute from '../delete-route';

export default DeleteRoute.extend({
  model(params){
    return
this.get('store').findRecord('developer',params.developer_id,{include:"projects"});
  },
  getRedirectRoute(){
    return "developers";
  }
});
```

Fonctionnalités à implémenter

CRUD

//TODO 1.1

Pour les models **tag**, **step**, **task**, ajouter les fonctionnalités de base CRUD :

- Listage des instances dans une table
- Suppression
- Modification
- Ajout

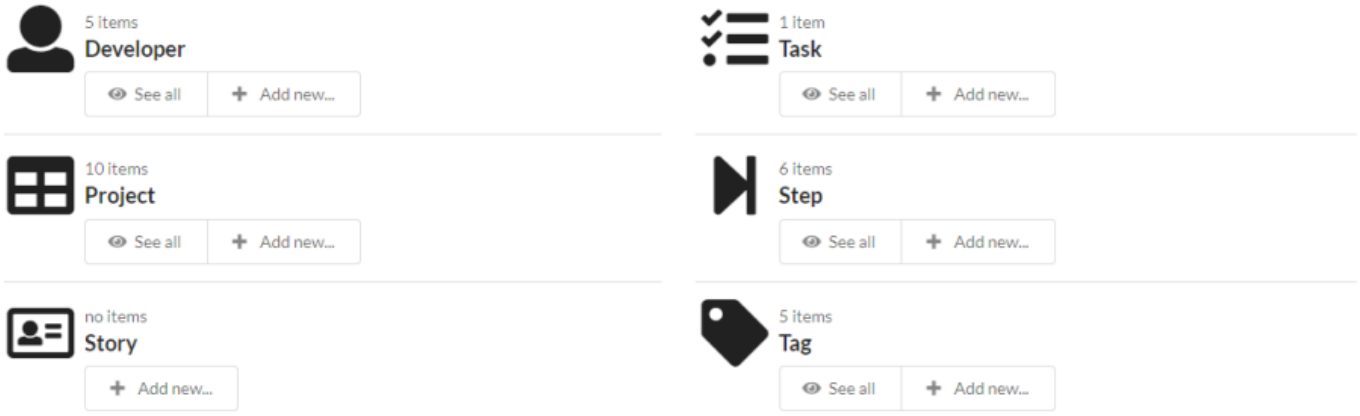
Consignes :

- Respecter la logique fonctionnelle et structurelle (implémentation) mise en place dans le projet
- Factoriser au mieux le code

Route index

//TODO 1.2

Modifier la route **index**, pour qu'elle affiche les éléments suivants, et qu'elle permette d'accéder à chacune des parties :



Il est possible d'obtenir la liste des models dans une route par :

```
const { Route, getOwner } = Ember;

export default Route.extend({
  model() {
    return getOwner(this).lookup('data-adapter:main').getModelTypes().map(type =>
type.name);
  }
});
```

Il ne reste plus qu'à charger chacun des models dans cette route, pour obtenir le nombre d'instance de chaque dans le template.

```
import Ember from 'ember'
import RSVP from 'rsvp';

const { Route, getOwner } = Ember;
```

```
export default Route.extend({
  model() {
    return RSVP.hash({
      models: getOwner(this).lookup('data-adapter:main').getModelTypes().map(type
=> {
                      return {name:
type.name,objects:this.get('store').findAll(type.name)};
                      }),
      icons: ['user','table','address card outline','tasks','step','tag']
    });
  }
});
```

Il suffit maintenant de parcourir **model.models** dans le template **index.hbs**, et d'afficher pour chaque classe **modelClass** sa propriété **name**: **modelClass.name**, ou le nombre de ses instances : **modelClass.objects.length**.

On pourra utiliser le composant semantic-ui Item, et le composant Grid pour le placement.

From:
http://slamwiki2.kobject.net/ - **SlamWiki 2.1**

Permanent link:
http://slamwiki2.kobject.net/richclient/emberjs/td6?rev=1521788987

Last update: **2019/08/31 14:38**