ENSEIRB-MATMECA

Année 2016/2017

PFA - Réalisation d'un site web pour EirBot au-dessus d'OTF

Manuel de maintenance

Équipe :
Mathieu Arthaud
Maëva Grondin
Damien Haurat
Pascalie Henry
Laura Hing
Louise Mouret

Encadrant et client :
Laurent Simon

Vendredi 7 Avril 2017

Table des matières

1 Introduction

Ce manuel de maintenance a pour vocation de présenter les outils de développement sur lesquels repose l'application de gestion des composants d'Eirbot :

— Le framework OTF et son architecture générale
— La base de données
— Les liens entre les différentes parties du framework

Cette présentation vise à donner une vue globale du fonctionnement du framework dans le but de pouvoir rajouter à l'application des fonctionnalités par la suite, ou d'apporter des correctifs à d'éventuelles erreurs présentes dans l'application.

La maintenance nécessite d'avoir accès aux sources du projet et d'avoir deux terminaux de travail ouverts : l'un pour faire tourner l'application avec la commande **./start.sh** et y observer toutes les informations de debug, l'autre pour naviguer dans les sources et identifier les fichiers à modifier.

2 Présentation des outils de développement

2.1 Le Framework OTF²

L'architecture du framework OTF² est basée sur le **modèle MVC**.

2.1.1 Le contrôleur

Le contrôleur correspond aux **plans de vol**, chaque différent profil d'utilisateur (administrateur, utilisateur basique, etc) a ses plans de vol personnels. Ceux-ci sont dans le dossier *conf -> profiles*. Un plan de vol est composé de plusieurs sections, les dernières étant facultatives :

— Le **module** définit quel fichier appartenant au beans (Modèle) est utilisé.
— La **methode** définit quelle fonction dans le fichier module est utilisée.
— Le **screen** indique à quelle vue le plan de vol se réfère.
— Le **auth** correspond à un booléen indiquant si l'utilisateur doit être identifié ou non.
— Les **params_names** indiquent tous les champs qui doivent être envoyés au modèle afin de les ajouter à la base de données. [facultatif]
— Le **data_model** définit le document de la base de données que le plan de vol utilise. [facultatif]
— La **data_ref** recense les clés étrangères afin de faire des jointures de tables dans le modèle. [facultatif]
— Le **redirect** indique s'il y a une redirection ou non. [facultatif]
— Le **redirect_action** renseigne le plan de vol permettant de faire la redirection. [facultatif]
— Le **return_type** permet de définir des types de retours.

Deux types de plans de vol existent, les "GET" et les "POST". Les "GET" permettent d'obtenir une page et d'effectuer des traitements précédant son apparition tandis que les

"POST" servent à faire un traitement après l'action d'un utilisateur sur une page.

Un exemple de deux plans de vol est situé ci-dessous :

```
  "GET/rendre_composant": {
        "module": "finder",
        "methode": "populateInfinite",
        "screen": "formrendrecomposant",
        "auth": true,
        "params_names": ["_id"],
        "data_model": "Prets",
        "data_ref": [["id_composant"]]
    },
  "POST/addcom": {
        "module": "inserter",
        "methode": "newCommentaire",
        "screen": "forum_title",
        "auth": true,
        "params_names": ["id_conversation", "contenu_commentaire"],
        "redirect": true,
        "redirect_action" : "forum_conv"
     },
```

2.1.2 Le modèle

Le modèle est divisé en deux parties. Le première partie est définie dans les beans (dossier *beans* à la racine de l'application) comportant des fichiers Javascript rassemblés par fonctionnalités par exemple toutes les fonctions pour récupérer des données de la base de données sont dans le fichier finder.js, les fonctions permettant d'ajouter des données à la base sont rassemblées dans inserter.js etc. Dans chacun de ces fichiers se trouvent les fonctions permettant de réaliser la fonctionnalité mais de différentes manières. Par exemple dans le fichier finder.js, les fonctions permettent de récupérer des champs de la base de données selon si une seule table est utilisée, si plusieurs tables sont utilisées (réalisation de jonctions) etc. Ces dernières fonctions font appel à d'autres sous-fonctions qui font partie de la deuxième partie du modèle située dans le dossier *otf_ core -> lib -> otf_ mongooseGeneric.js*. Ces sous-fonctions sont un intermédiaire supplémentaire afin d'accéder à la base de données grâce à Mongoose.

2.1.3 Les vues

Les différentes vues sont rassemblées dans le dossier *views*, il s'agit de fichiers handelbars comprenant du langage HTML. Plusieurs vues sont isolées dans le sous-dossier *partials* car celles-ci se retrouvent sur plusieurs pages et sont systématiquement incluses dans les autres comme le *header*, le *footer* et les différents menus de navigation selon les types d'utilisateurs. Enfin, le dossier *layouts* contient le fichier main qui articule les différentes vues ensemble, celles communes du dossier *partials* et celle utilisée spécifiquement nommée **page**.

2.2 La base de données

2.2.1 Une base de données NoSQL : MongoDB

Le framework OTF permet de se connecter à une base de données NoSQL à savoir ici MongoDB. Les bases de données NoSQL (Not Only SQL) se différencient des bases de données SQL en ce qu'elles permettent une plus grande liberté dans leur écriture : aucune vérification n'est faite sur les entrées d'une même table qui peuvent donc ne respecter aucun critère commun, aucun pattern obligatoire. De plus, les systèmes NoSQL ne gèrent pas le système de liens et clés-étrangères vers d'autres tables et permettent de stocker de grandes quantités d'informations à un coût moindre, et sont donc plus favorables à la redondance d'informations dans les différentes tables.

L'utilisation du NoSQL induit certaines contraintes. En effet, le caractère "libre" de MongoDB et du NoSQL en général peut aussi être source de nombreux dysfonctionnements d'une base de données : il est en effet nécessaire de s'assurer que chaque entrée de la base de données a bien la forme attendue (forme regroupant toutes les informations nécessaires au bon fonctionnement de l'application). De plus, le fait de ne pas avoir accès au système de liens et de clés étrangères dans MongoDB induit de regrouper plus d'informations dans des tables plus conséquentes en taille et moins nombreuses et donc de veiller à ce que les redondances n'induisent pas d'incohérence dans la base de données.

2.2.2 Les accès à la base de données via un module Node.JS : Mongoose

Le framework OTF^2 est implémenté en javascript, et plus précisément essentiellement en Node.JS qui est une extension du javascript "traditionnel" qui permet de faire du code javascript s'exécutant côté serveur.

Le framework OTF^2 permet d'effectuer toutes les interactions avec la base de données via l'utilisation de Mongoose, un module de Node.JS. Mongoose est en effet une sorte d'interface mettant à disposition des développeurs un ensemble de fonctions écrites en javascript et permettant des accès simples à une base de données MongoDB : recherche, insertion, modification ou suppression d'un ou de plusieurs éléments dans une table.

OTF^2, par dessus ce paquet, rend disponible des fonctions génériques prenant en paramètre un ou des objets json (permettant de signifier la donnée à rechercher, insérer, modifier ou supprimer) et effectuant l'appel à la fonction adéquate de Mongoose. Ainsi, le framework étant implémenté selon le pattern Modèle-Vue-Contrôleur, la partie Modèle du framework est cet ensemble de fonctions génériques faisant appel à Mongoose et se trouvant au chemin otf_core/lib/otf_mongooseGeneric.js.

Pour faire des appels à la base de données, il faut envoyer le bon objet javascript à la fonction effectuant l'action souhaitée. Une des fonctions proposées par Mongoose peut permettre de faire des jointures entre différentes tables : il s'agit ici de la fonction "populate". Cette fonction rend en effet possible l'accessibilité aux attributs d'une autre table si on possède leurs identifiants.

2.3 Les liens entre les différentes parties du code

En résumé, le framework OTF se divise en 5 parties principales :

- Le contrôleur :
  - Il existe un type de contrôleur par rôle (donc par droits d'utilisateur), celui de l'administrateur se trouve dans le fichier **conf/profiles/otf_admin.json**, celui d'un utilisateur basique dans le fichier **conf/profiles/otf_user.json**, etc...
  - Le contrôleur pour un rôle donné définit l'ensemble des pages accessibles pour ce rôle.
  - Pour une page donnée, un plan de vol récapitule comment la page est créée (notamment, l'attribut "module" indique le fichier du dossier **beans** dans lequel la fonction d'accès à la base de données que l'on va appeler se trouve ; l'attribut "methode" indique quelle fonction est appelée dans ce module ; l'attribut "screen" indique quelle vue est appelée).
- Le dossier **beans** contient toutes les fonctions d'accès à la base de données, fonctions qui elles-mêmes appellent des fonctions du fichier **otf_mongooseGeneric.js**. Ces fonctions permettent de traiter les données avant de les envoyer à la base de données via Mongoose. Les données sont contenues dans des fichiers *json*.
- Le fichier **otf_core/lib/otf_mongooseGeneric.js** contient toutes les fonctions de Mongoose.
- Le fichier **conf/directory_schema.json** contient tous les schémas d'accès à la base de données. Ils définissent quel pattern doit avoir respecté une entrée pour pouvoir être envoyé à la base de données.
- Les vues se trouvent dans le dossier **views** et accèdent au contenu à afficher par le biais de la variable **result** qui est calculée par la méthode appelée dans le module du contrôleur et envoyée à la vue. Cette variable contient généralement comme attributs ceux spécifiés dans le tableau **params_names** du contrôleur.

3 Maintenance des fonctionnalités du framework

3.1 Identifier la section du contrôleur en cause d'une erreur

Si une page de l'application pose problème, procéder comme suit :
- Identifier l'URL de la page posant problème ainsi que le rôle (les droits d'accès) de la personne pour qui cette page pose problème.
- Ouvrir le contrôleur correspondant au rôle de la personne dans le dossier **conf/profiles**.
- Identifier dans ce fichier le plan de vol correspondant à l'URL relevée.

3.2 Identifier la vue et la section du modèle appelée

Une fois le bon plan de vol identifié :
- Identifier la vue appelée dans l'attribut "screen" du plan de vol et ouvrir le fichier views/nom_de_la_vue.hbs
- Identifier le module d'accès à la base de données et ouvrir le fichier beans/nom_du_module.js
- Une fois le module ouvert, identifier la fonction indiquée dans l'attribut "methode" du plan de vol.

— Une fois cette fonction trouvée, identifier la fonction de Mongoose appelée, celle-ci étant la première fonction appelée dans le bloc "try" de la méthode.

3.3 Apporter des correctifs

Une fois tous les fichiers concernés identifiés, seules des modifications dans ces fichiers pourront régler les problèmes aperçus. Cela nécessite ensuite d'étudier plus en détail la méthode javascript appelée, la vue appelée et la fonction mongoose appelée qui peuvent être modifiées ou remplacées par d'autres fonctions respectant la même architecture. De plus, dans otf_mongooseGeneric.js et dans le dossier beans, il y a possibilité d'appeler la fonction **logger.debug(attr)** avec attr étant une variable ou une chaîne de caractères. Ces informations seront ensuite affichées dans le terminal où *./start.sh* aura été lancé préalablement. Une observation attentive de ce terminal pourrait alors permettre, avec plusieurs affichages, de déterminer la source d'erreurs.

4 Maintenance de la base de données

4.1 Gérer "à la main" la base de données : liste de commandes MongoDB

Dans le but de corriger certaines erreurs, il est parfois nécessaire de regarder et/ou modifier directement les informations dans la base de données. Voici donc une liste des commandes de base de MongoDB permettant d'interagir avec les données :

— Lancer ./start.sh en tâche de fond
— Ouvrir MongoDB et se placer dans la base "otf_demo" : Ouvrir un nouveau terminal et entrer "mongo" suivi de "use_otf_demo"
— Voir les collections (tables) existantes : show collections
— Obtenir une aide sur les fonctionnalités accessibles : help
— Afficher les entrées de la collection "composants" : db.composants.find()
— Ajouter une entrée dans la table "composants" : db.composants.insert({nom_composant : vis, quantite_composant : 6, ...})
— Modifier une entrée identifiée par son nom dans la collection "composants" : db.composants.update({nom_composant : vis}, {$set : {quantite_composant : 4}})
— Vider la collection "composants" : db.composants.drop()
— Supprimer une entrée identifiée par son nom dans la collection "composants" : db.composants.remove({nom_composant : vis})
— Supprimer la base dans laquelle on se trouve : db.dropDatabase()
— Exporter la base otf_demo dans le dossier sauvegarde : se mettre dans un terminal normal et entrer "mongodump -d otf_demo -o sauvegarde"
— Importer la base se trouvant dans le dossier sauvegarde dans une base nommée new_base : mongorestore -d new_base sauvegarde/otf_demo

4.2 Ré-initialisation de l'administrateur en cas de suppression involontaire

En se basant sur les commandes citées précédemment, il est possible de recréer un administrateur dans le cas où le dernier compte administrateur du site aurait été supprimé. Il suffit alors de suivre les opérations suivantes :

- Lancer ./start.sh en tâche de fond dans un terminal
- Ouvrir MongoDB dans un autre terminal : mongo
- Ouvrir la base de données du site : use otf_demo
- Ajouter un administrateur :
  db.accounts.insert({ "_id" : ObjectId("543fbd936fae706dd241f2ac"), "login" : "admin", "password" :
  "$2a$10$zXV/hUZ3WGsmA14ZvkNHl.pbNR7.x5RGJAxZRTh0tNnVY/HID2w2q",
  "email" : "toto@truc.com", "role" : ObjectId("57601801cbb47caf29000001")})
- Une compte avec le login "admin" et le mot de passe "otf" sera alors de nouveau accessible. L'administrateur pourra ensuite réinitialiser son mot de passe et son login directement depuis l'application web.