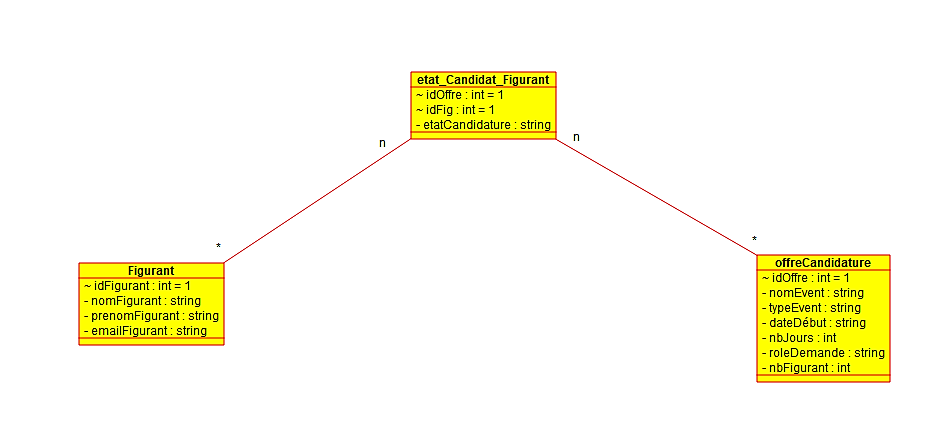
Mission n°1 : Le projet FaceCast

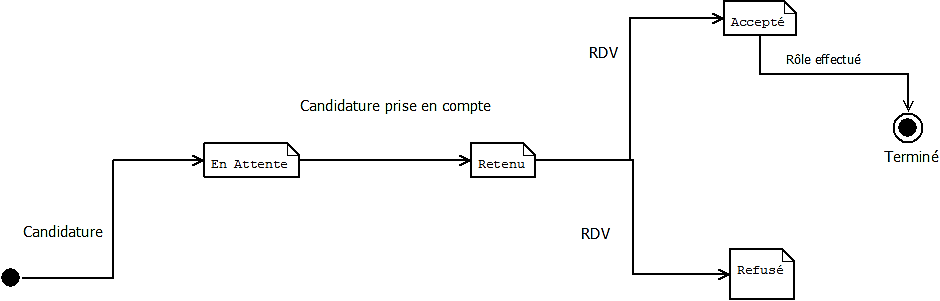
Dans le cadre des besoins de l’agence FaceCast, la mission consiste à offrir une application Web responsive en NodeJS sur lequel l’agence peut gérer ses offres de figuration ainsi que changer l’état des candidatures des figurants.

I. L’analyse du travail à effectuer

Avant de réaliser toute ébauche du site, un travail d’analyse a été élaboré de sorte à comprendre les enjeux du projet et dégager les différentes étapes à suivre pour la réalisation de notre site.

Ainsi, nous avons commencé par créer un diagramme UML représentant, d’après l’énoncé, les entités du modèle.

Comme nous pouvons le voir, nous avons constaté l’existence de trois entités distinctes : les offres, les figurants, et l’état de la candidature des figurants pour une offre. Un figurant peut donc candidater pour une ou plusieurs offres et une offre peut demander un à plusieurs figurants. De ce fait, on distingue l’état de la candidature du candidat en fonction de l’offre à laquelle il auditionne.

****Par la suite, nous avons constitué un diagramme UML E/T à partir du cycle de vie d’une candidature d’un figurant.

Nous pouvons alors observer, à travers ce diagramme, que dès qu’un figurant poste une candidature, celle-ci est de base en l’état « en attente », en ce sens qu’elle n’a pas encore été vue par l’offreur. Puis, une fois constatée, elle passe en l’état « retenu ». S’offrent alors deux possibilités : suite à un entretien, soit le figurant est retenu pour le rôle et son état passe en « accepté » ; soit l’état de sa candidature passe en « refusé » et la cela s’arrête là. Dans le cas où il sa candidature est accepté, le figurant effectue son rôle, puis la candidature passe à l’état final « terminé ».

A partir de ces deux analyses, des exemples de code JSON sont élaborés. Trois fichiers différents sont pour le moment décidés : offre.js, figurant.js, candidature.js.

* Le fichier offre.js

var mongoose = require(‘mongoose’) ;

var Schema = mongoose.Schema ;

var offre = Schema({

\_id : Schema.Types.ObjectId,

nomEvent : String,

typeEvent : String,

dateDebut : String,

nbJour : Number,

roleDemande : String,

nbFigurant : Number,

figurants : [{type : Schema.Types.ObjectId, ref:'Figurant'}]

});

var Offre = mongoose.model('Offre', offre);

module.exports = Offre;

* Le fichier figurant.js

var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var figurant = Schema ({

nomFigurant : String,

prenomFigurant : String,

emailFigurant : String,

candidature : [{type : Schema.Types.ObjectId, ref: 'Offre'}]

});

var figurant = mongoose.model('Figurant', figurant);

module.exports = Figurant;

* Le fichier candidature.js

var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var candidature = Schema ({

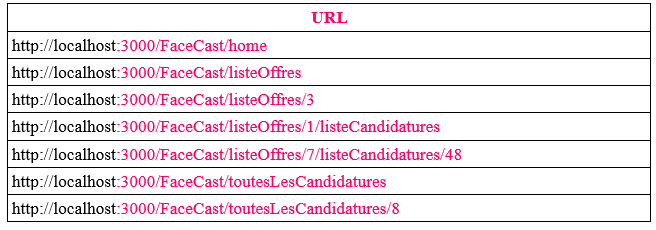
offre : [{type : Schema.Types.ObjectId, ref: 'Offre'}],

figurant : [{type : Schema.Types.ObjectId, ref:'Figurant'}],

etatCandidature : String

});

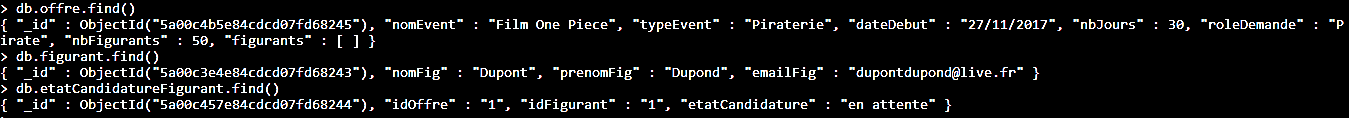
Une fois ces trois travaux d’analyse en main, la tâche à effectuer est la suivante : créer un tableau proposant des URL en vue du site en NodeJS à venir. Nous proposons le tableau suivant :



Dans ce tableau sont proposées différentes URL imaginées à partir de nos analyses et de l’énoncé proposé. Le projet s’appellerai donc FaceCast, lequel possèderait une page home, une page proposant la liste des offres et la liste des candidatures, et qui offrirait la possibilité de visualiser les candidatures en fonction de l’offre. Nous n’excluons pas le fait de proposer au gestionnaire la faculté à ajouter, supprimer voire modifier une offre.

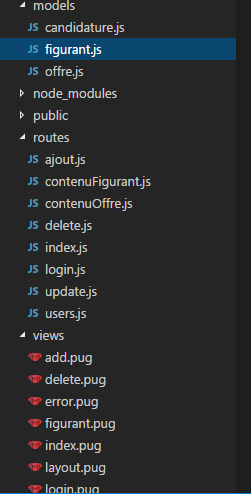
II. La réalisation de l’application Web

Suite à la création du fichier facecast, qui contiendra tout le code propre au site, à la racine du dossier nodejs lui-même créé suite aux TP orientés NodeJS, nous avons, dans la base de données propre à NodeJS, MongoDB, réalisé une base de donnée du nom de facecast, avec 3 tables : offre, figurant et etatCandidatureFigurant. Nous avons ensuite entré des données fictives pour constater le bon fonctionnement du site, qui effectue un lien entre NodeJS et MongoDB grâce à mongoose.

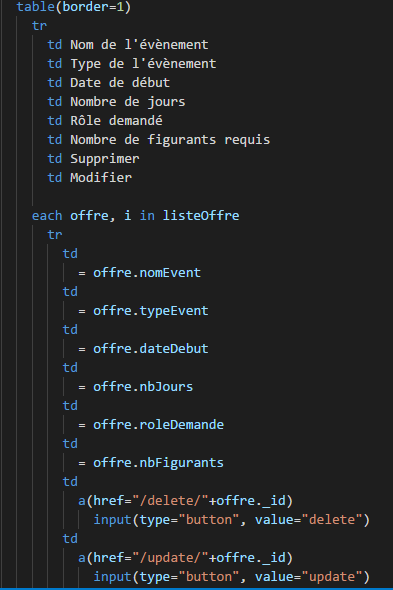


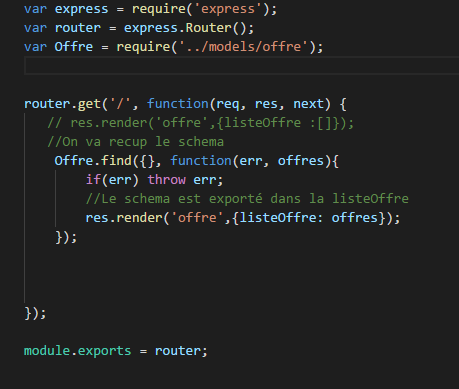
Ceci est un exemple de données contenues dans la base.

Les fichiers offre.js, figurant.js et candidatures.js ont été ajoutés sous le dossier « models ». Les fichiers js assurant la manipulation des données situées dans MongoDB par mongoose se trouveront sous le dossier « routes », et les fichiers permettant d’afficher les données seront des fichier pug placés sous le dossier « views ».



Dans un objectif de manipulation des données, nous allons faire appel aux opérations CRUD, autrement dit Read Create Update Delete. Nous avons donc, dans un premier temps, cherché à lire les données de la table offre.

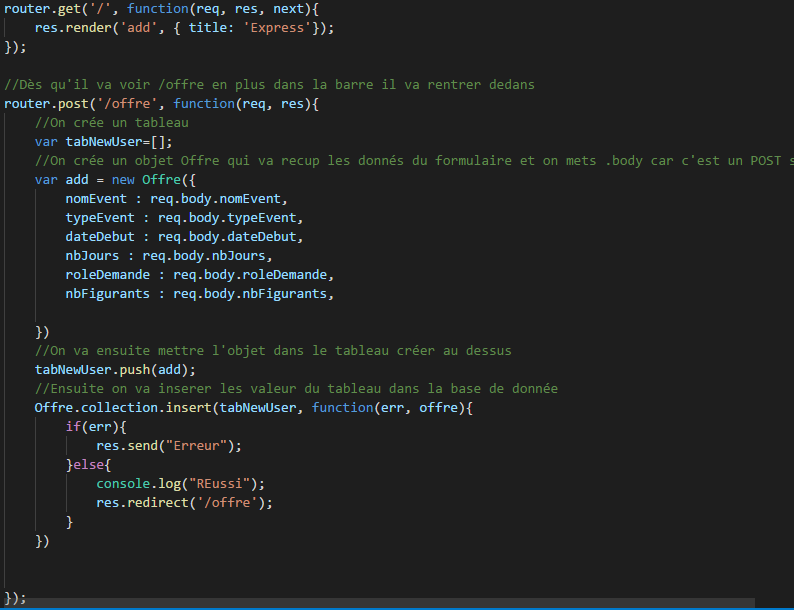




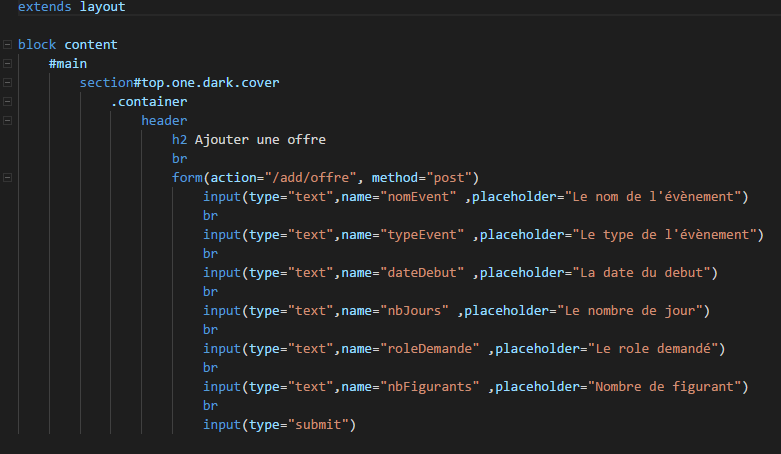
Comme il est montré ci-dessus (image 1), nous avons stocké dans une variable nommé Offre le chemin menant au modèle présent dans offre.js. Par la suite, nous avons utilisé la méthode Offre.find() qui va nous permettre de lire les données de la table offre. Afin d’exporter ces données récupérées, nous allons faire appel à une liste qui sera récupéré dans le fichier offre.pug, la vue permettant l’affiche des données (image 2). Ainsi, pour chaque offre contenue dans la liste listeOffre, les données de la table seront affichées. On obtient la page suivante :

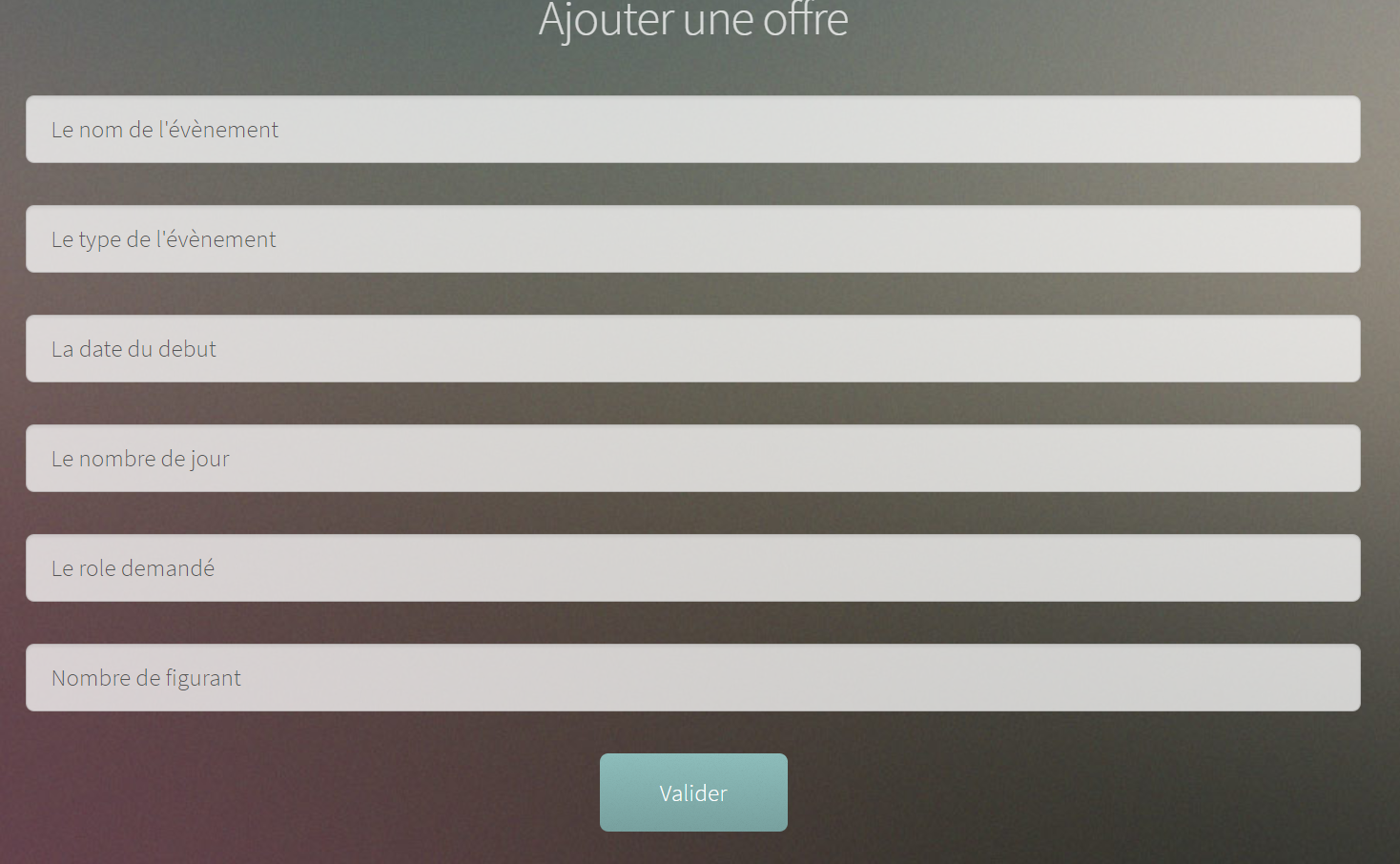


Vient l’opération Create, qui nous amènera donc à ajouter une offre à la liste.

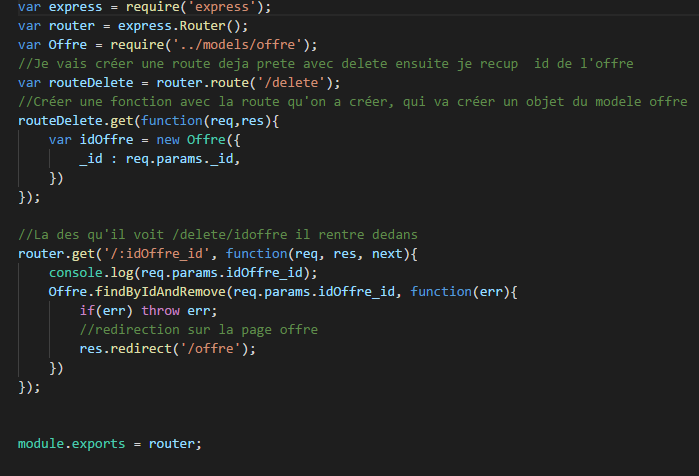


Après avoir fait appel au modèle Offre dans une variable du même nom, nous avons créé un tableau dans lequel sera contenu les nouvelles données, lesquelles seront-elles-mêmes créées à l’aide de la création d’un nouvel objet new Offre. Les données créées, elles sont insérées par la méthode db.collection.insert() dans la table offre et ajouter à la liste des offres. Les données créées et récupérées dans tabNewUser sont élaborés à partir de la page ajout.pug.



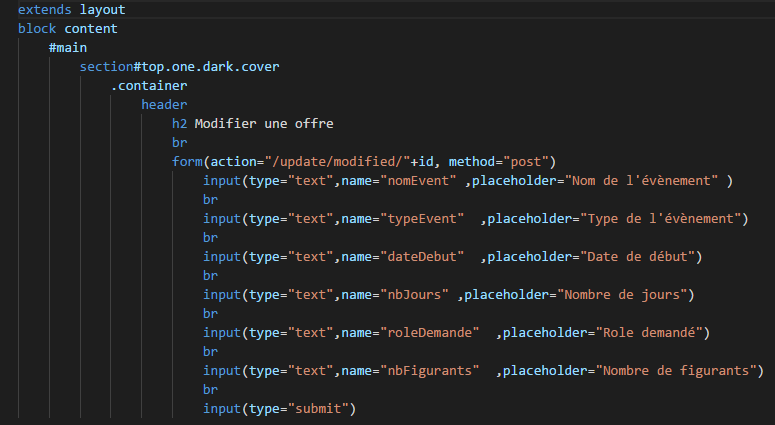


Une fois les étapes Read et Create réalisées survient le Delete.



Après avoir récupéré le modèle Offre et créé un nouvel objet permettant la récupération de l’id de l’Offre à supprimer, on utilise une méthode Offre.findByIdAndRemove() qui, en ayant en paramètre l’id de l’offre, va supprimer celle-ci. C’est une méthode plus rapide et moins lourde que si on était passé par un Offre.findById() puis par un Offre.Remove(). Une fois supprimée, la méthode redirige vers la page affichant la liste des offres.

La dernière étape des opérations CRUD est l’opération Update. Nous allons récupérer une offre grâce à son id, puis, une fois l’id trouvé, la méthode Offre.update() sera utilisé pour modifier chaque critère trouvé d’une offre en fonction de cet id.

La manipulation du Update est permise par la view update.pug : 

Nous obtenons alors la page suivante :



L’affichage des figurants et de leurs candidatures, la modification de leur état, l’utilisation de l’API REST et les Tests Unitaires n’ont pu être réalisés par manque de temps, les opérations CRUD ayant été laborieuses à réaliser.

L’agence FaceCast dispose néanmoins d’un site responsive sur lequel il lui est possible de mettre en évidence les offres qu’elle propose comme il avait été demandé au départ.