Rapport d’activité

Mission 1 - FaceCast

Développement d’une application web basé sur un serveur Node.js, le framework Express.js ainsi qu’une base de données MongoDB via l’approche par Model de Mongoose .

07/11/2017

Crété Jonathan – Adelaïde Donovan - Lycée Léonard de Vinci

https://github.com/JonathanCrt/FaceCast

PLAN :

I. Cahier des charges
   a. Expression des besoins
   b. Analyse préalable

II. Mise en œuvre de la mission
   a. Structure de la mission
   b. API REST
   c. Test(s) Unitaire

III. Conclusion

https://github.com/JonathanCrt/FaceCast

I. Cahier des charges

a. Expression des besoins

Besoins :

L'agence cliente FaceCast, demandeuse de l'application, a besoin d'une application web afin de traiter au mieux ses candidatures. Le projet se découpe en 2 applications (web & Android) , l'application android permettra au figurant de postuler via l'application dont les données seront envoyées à l'application web nous intervenons alors dans la conception de l'application destinée au « back Office » à savoir l'application web.

Cette application étant destinée au gestionnaire de l'agence, celui-ci doit pouvoir ajouter une nouvelle offre ainsi que gérer en temps réel le traitement des candidatures (différents états).

Contraintes :

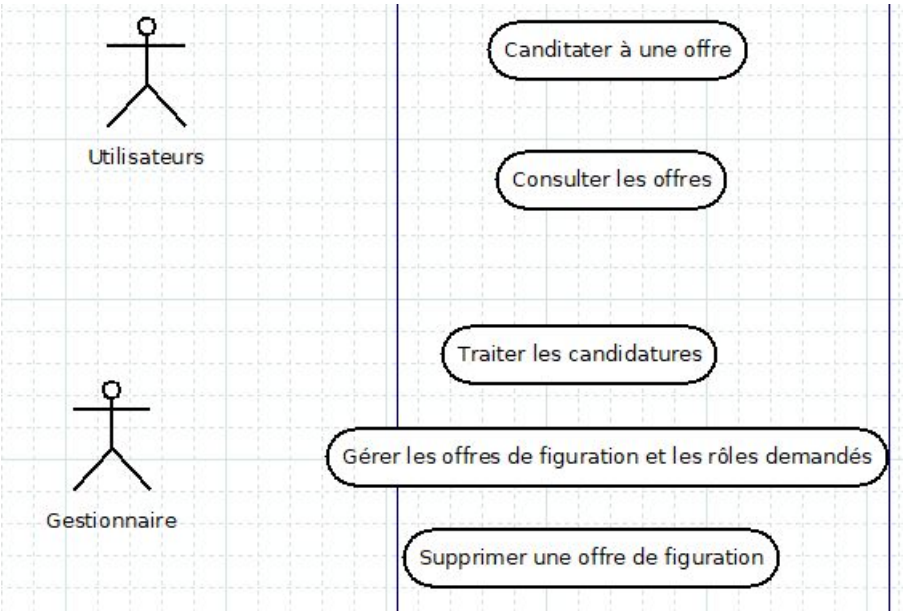
L'application Web du projet s'appuie sur un serveur Node.js, le framework Express.js et une base de données MongoDB via l'approche par Model de Mongoose.

L'application mobile dialogue avec l'application via le protocole HTTP et des messages au format JSON. De fait, nous avons développés une API basée sur REST pour les échanges entre l'application Android (développée par une autre entreprise) et l'application Node JS. Afin de tester le bon comportement de notre application web vis à vis de l'application mobile, nous avons programmés des tests fonctionnels avec java et Unirest . Cette solution nous as permis de programmer des requêtes JSON (envoi) et analyser les réponses JSON (réception) de notre API.

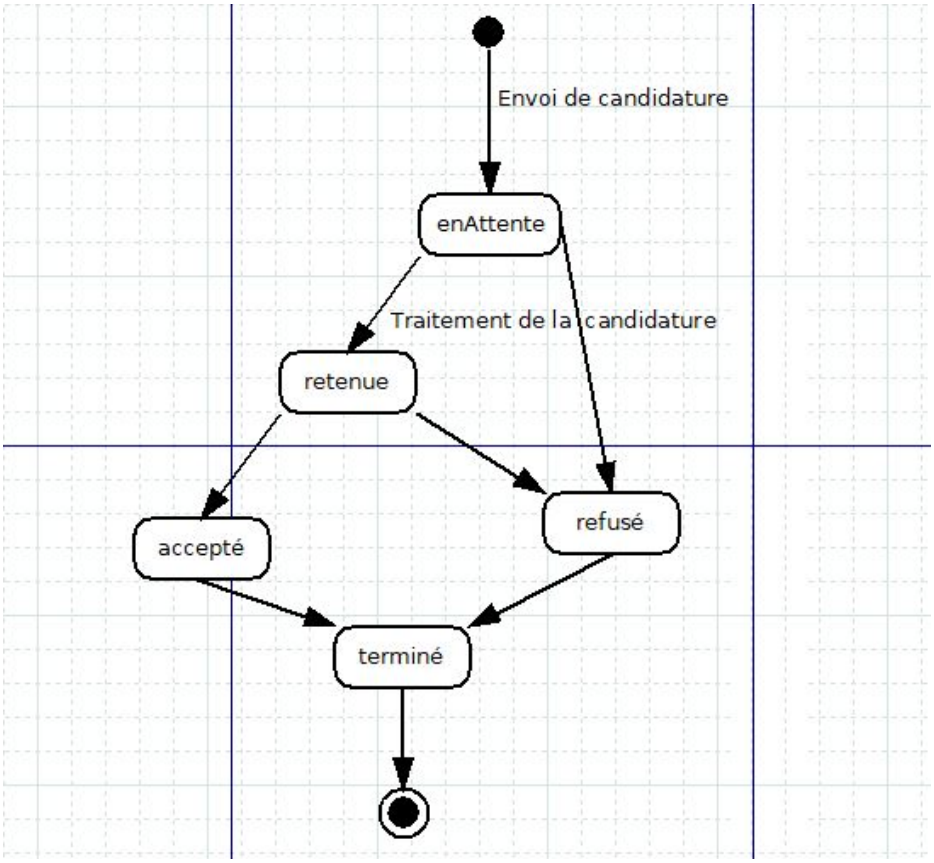
https://github.com/JonathanCrt/FaceCast

b. Analyse

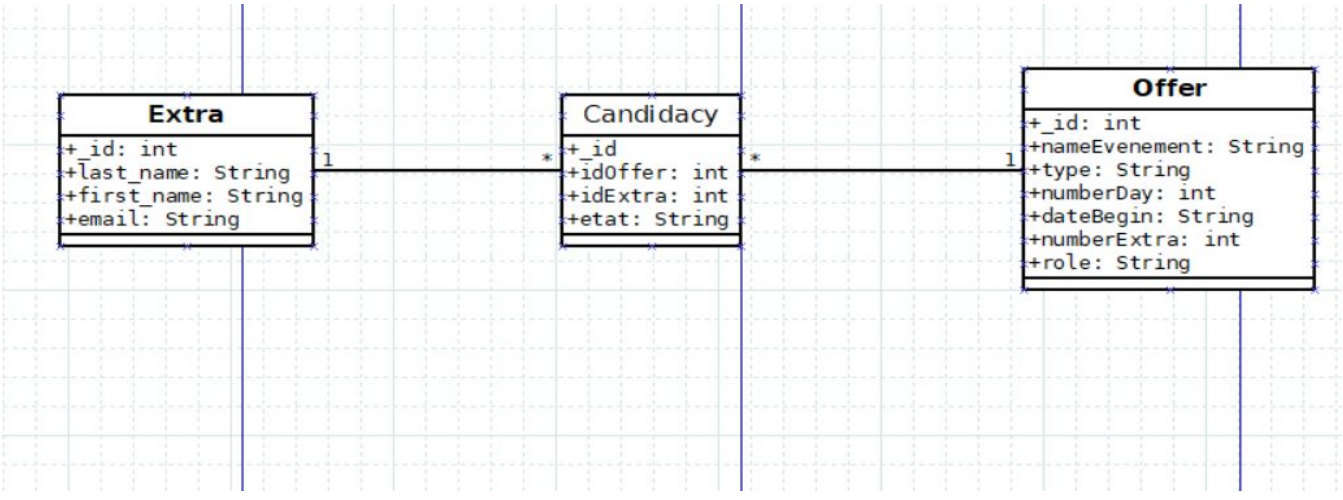
1- Diagramme des cas d'utilisation de FaceCast



2- Diagramme UML E/T du cycle de vie de FaceCast



3- Diagramme UML des entités



https://github.com/JonathanCrt/FaceCast

4- Liste des modèles de collection nodejs

model offer:

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var offerSchema = Schema({
 nameEvenement: String,
 type : String,
 dateBegin: String,
 numberDay: Number,
 numberExtra: Number,
role: String,
},{ collection: 'offers' });
var Offer = mongoose.model('Offer', offerSchema,'offers');
// export le modèle
module.exports = Offer;
```

le model offer prend un nom d'événement , un type d'événement (spectacle ou film) ,la date de l'événement , le nombre de jour et le nombre de figurant .

le model offer correspond aux informations concernant l'offre.

model candidacy :

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var candidacySchema = Schema ({
   etat : {type : String , default: 'Waiting'},
   idExtra :{type:Schema.Types.ObjectId, ref : 'extras'},
   idOffer :{type:Schema.Types.ObjectId, ref : 'offers'}
},{ collection: 'candidacys' });

var Candidacy = mongoose.model('Candidacy', candidacySchema,'candidacys');
// export le modèle
module.exports = Candidacy;
```

le model candidacy prend un etat , l'idExtra (le figurant qui correspond a la candidature) ,l'idOffer (l'offre a laquelle correspond la candidature).

le model candidacy correspond à la candidature du figurant sur l'offre a laquelle il a postuler.

model extra :

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var extraSchema = Schema ({
   last_name: String,
   first_name: String,
   email: String,
```

https://github.com/JonathanCrt/FaceCast

```
},{ collection: 'extras' });

var Extra = mongoose.model('Extra', extraSchema,'extras');

// export le modèle

module.exports = Extra;
```

le model extra prend un nom, un prénom , un email .

le model extra correspond à l'identité du figurant.

5- Exemple Json

note : les données dans l'exemple des collections sont fictives

```
/* 1 */
Json de la collection Offers :
{
    "_id" : ObjectId("59fed81b069a78003b0bc631"),
    "nameEvenement" : "Gary Movie",
    "type" : "Spectacle",
    "dateBegin" : "2018-01-25",
    "numberDay" : 15,
    "numberExtra" : 5,
    "role" : "Plombier",
    "__v" : 0
}
////////////////////////////////////////////////
/* 1 */
Json de la collection extras :
{
    "_id" : ObjectId("59f1dbd830e2ea92834cc568"),
    "last_name" : "Wicket",
    "first_name" : "Harry",
    "email" : "Harry.wicket@usMail.com"
}

////////////////////////////////////////////////

/* 1 */
Json de la collection candidacy  :

/* 1 */
{
    "_id" : ObjectId("59ff4e2f0b703667311e80c1"),
```

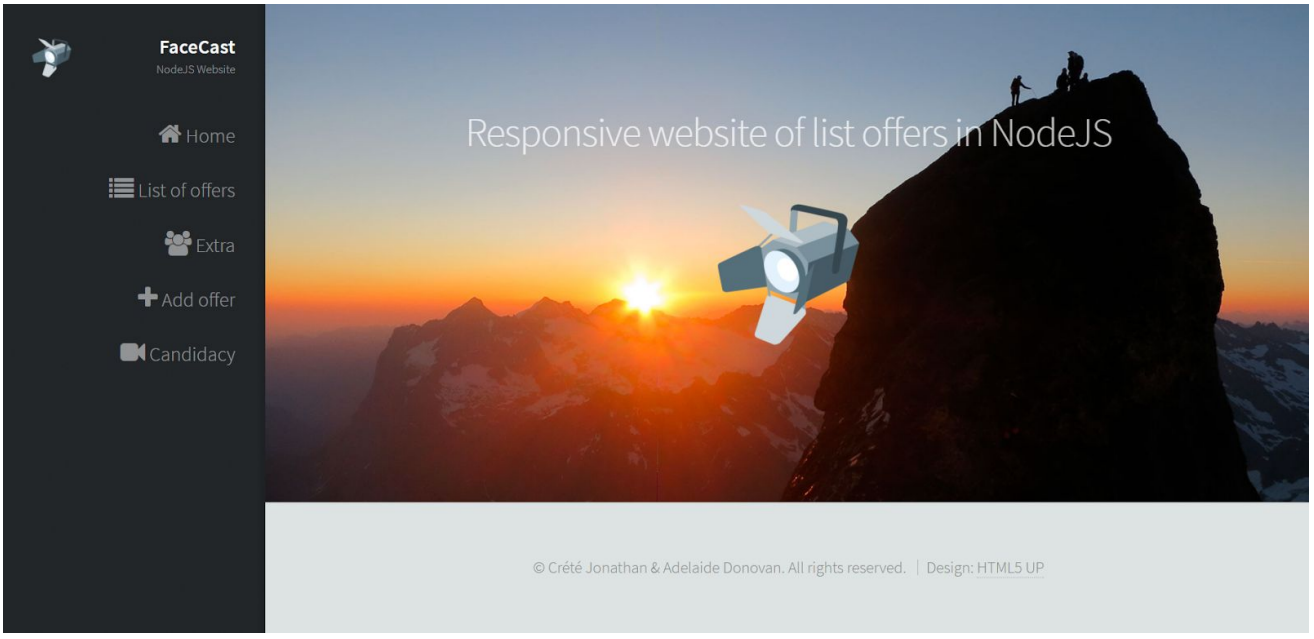
```
    "etat" : "Waiting",
    "idExtra" : ObjectId("59f1dbd830e2ea92834cc568"),
    "idOffer" : ObjectId("59fed81b069a78003b0bc631")
}
```

6- Tableau URI

Méthode HTTP	URI	Action résultante	Format échangé	Cas d'utilisation
Liste des Offres	FaceCast/Offers	GET	Json	Affichage des offre
Candidature a une Offre	FaceCast/Offers/Add	GET	Json	Candidature a une offre
Offre déjà postuler	FaceCast/Offers/id figurant, id offer	Post	Json	Affichage des offre déjà postuler par le figurant

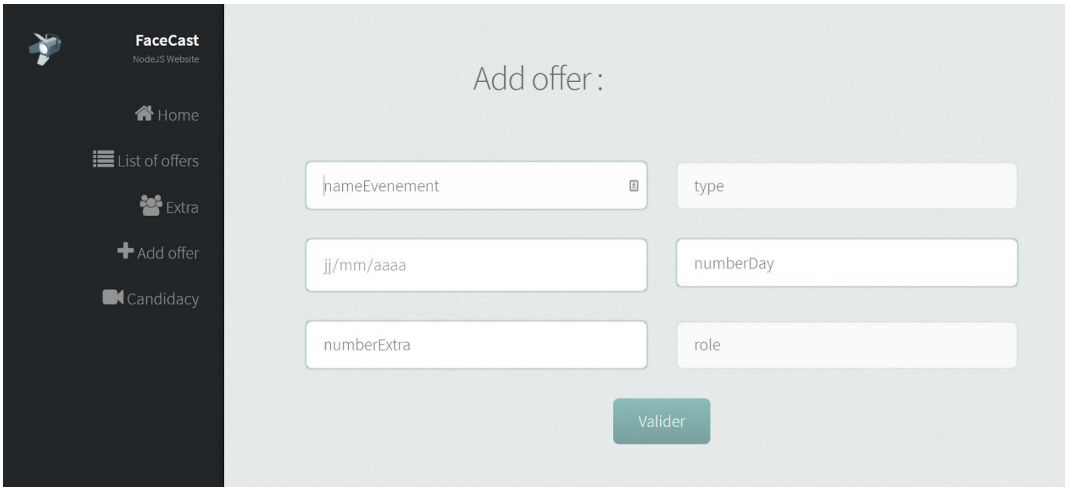
II. Mise en œuvre de la mission

a. Structure de la mission



https://github.com/JonathanCrt/FaceCast

FaceCast regroupe un ensemble de 5 pages (en comptant l'index). La page 'List of offers' permet d'afficher dans un tableau les différentes offres, que l'on peut bien entendu supprimé, mais aussi ajouter (en tant que gestionnaire fonction demandée). Ceci se fait via un formulaire de saisie de la page "Add offer" , ou on attend 5 saisies : le nom de l'événement, le type, la date, le nombre de jour, le nombre de figurants et le rôle(voir diagramme des entités)



De plus , nous avons d'ailleurs pris le temps d'ajouter une vérification de la cohérence des champs de saisie . Nous avons coder ceci en jQuery avec les expressions régulières (public/assets/fields.js)

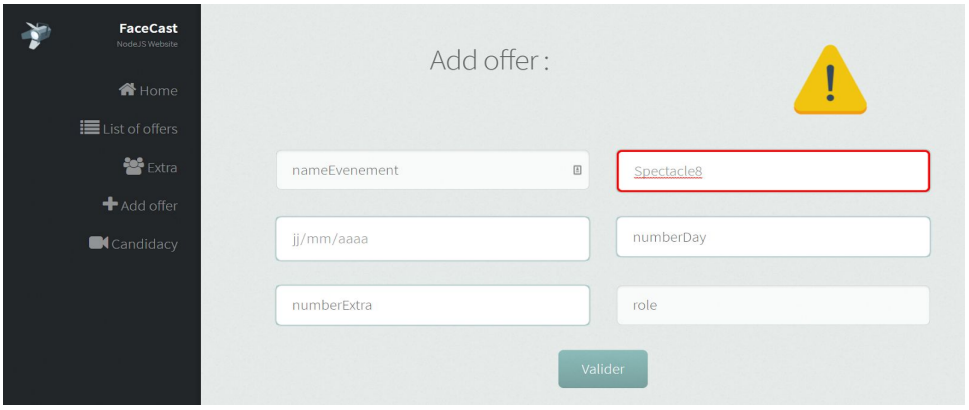
Exemple :

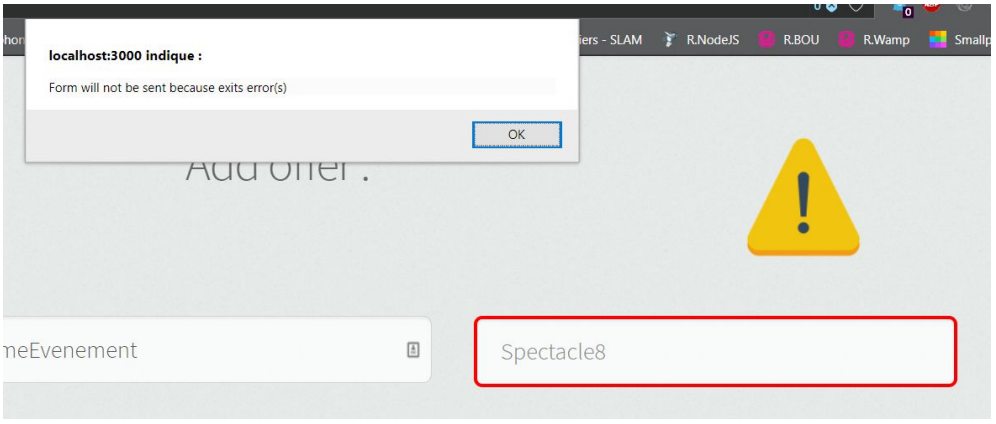
```
// JavaScript Document

function isAlpha(s, mandatory) {
        var regEx;
        if (mandatory) {
                regEx = new RegExp(/^([a-zA-Z]+)$/);
        } else {
regEx=  new  RegExp(/^([a-zA-ZáàâäãåçéèêëíìîïñóòôöõúùûüýÿæœÁÀÂÄÃÅÇÉÈÊËÍÌÎÏÑÓÒÔÖÕÚÙÛÜÝŸÆŒ._
-]*)$/);
        }
        return (regEx.test(s));
}
```

Une fois appelée, cette fonction permet de vérifier (lors de la saisie du type par exemple) que celle-ci a bien les caractères demandés. (le gestionnaire doit donc bien saisir ses champs) Sinon le champs apparaîtra "rouge" et celui-ci ne pourra continuer (Bouton submit bloqué)

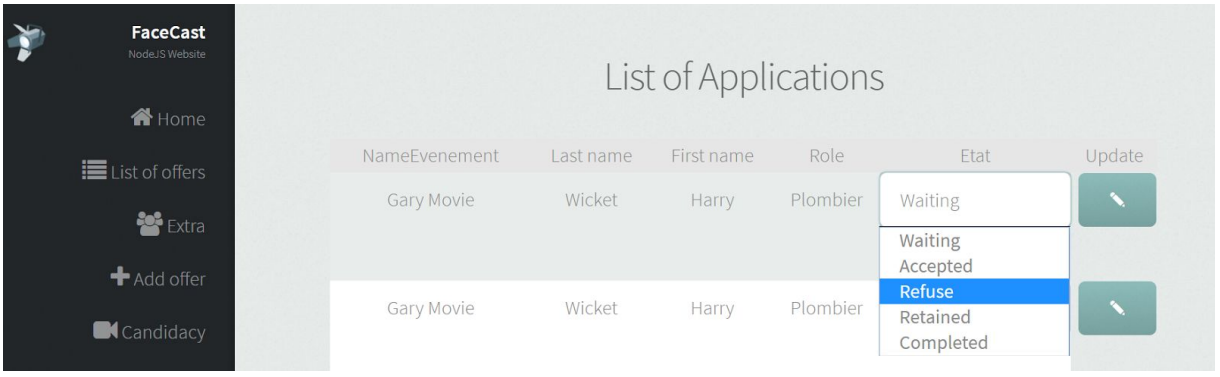
https://github.com/JonathanCrt/FaceCast





D'autre part l'application web comprend également une page "Extra" afin de lister les figurants (noms, email…) ainsi qu'une page "candidacy" regroupant les différentes candidatures (tableau avec le candidat et l'offre correspondante) . Le gestionnaire peut aussi changer l'état de la candidature (Waiting/Retained/Accepted/Refuse/Completed).

***La Mission a volontairement été faite en anglais ...**



https://github.com/JonathanCrt/FaceCast

b. API REST

```
var express = require('express');
var router = express.Router();
var path = require('path');
var fs = require('fs');
var modelCandidacy = require('../models/candidacy');
var modelExtra = require('../models/extra');
var modelOffer = require('../models/offer');
var objectId = require('mongoose').Types.ObjectId;

router.get('/', function (req, res, next) {
    res.send('Hi ApiRest , Choose roads : /offer or  /extra or /candidacy');
});

// 1: Collection Offer
    router.get('/offer', function (req, res, next) {
        modelOffer.find({}, function (err, offers) {
            if (err) throw err;
            res.json(offers);
        });
    });
    router.get('/offer/:number', function (req, res, next) {

        modelOffer.find({}, function (err, offers) {
            if (err) throw err;
            if (req.params.number == 0) {
                res.json(offers[0]);
            }
            else {
                res.json(offers.splice(0, req.params.number));
            }
        });
    });

// 2 : Collection Extra
    router.get('/extra', function (req, res, next) {
        modelExtra.find({}, function (err, extras) {
            if (err) throw err;
            res.json(extras);
        })
    });

//1 + 2 :  Collection Candidacy
    router.get('/candidacy', function (req, res, next) {
        modelCandidacy.find({}, function (err, candidacys) {
            if (err) throw err;
            res.json(candidacys);
        });
    });
    router.post('/candidacy', function (req, res, next) {
        var candidacy = new modelCandidacy({
            etat: req.body.etat,
```

https://github.com/JonathanCrt/FaceCast

```
            idOffer: req.body.idOffer,
            idExtra: req.body.idExtra
        });
        candidacy.save(function (err) {
            if (err) throw err;
            res.json(candidacy);
        });
    });

// My applications
    router.get('/candidacy/:idExtra', function (req, res, next) {
        modelCandidacy.aggregate([
            {
                $match: { idExtra: objectId(req.params.idExtra) }
            },
            {
                $lookup: {
                    from: "offers",
                    localField: "idOffer",
                    foreignField: "_id",
                    as: "offer"
                }
            }]).exec(function (err, candidacy) {
                res.json(candidacy);
            });
    });

module.exports = router;
```

c. Test(s) Unitaire(s)

Un test unitaire est un programme qui vérifie le bon fonctionnement d'un module (unité fonctionnelle) au travers de situations déduites des spécifications du module testé ; à partir de données d'entrée prédéterminées (l'état du système en entrée est connu), le test sollicite le module et confronte les données réellement obtenues avec celles théoriquement attendues, puis en déduit un état de succès ou d'échec.Un test est souvent accompagné d'une procédure d'exécution. L'activité de test d'un logiciel est un des processus du développement de logiciels.

```
package faceCast;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class TestUnit {

	JsonNode body;
	HttpResponse<JsonNode> response;
	HttpResponse<JsonNode> responseExtra;

	@Before
	public void setUp() throws UnirestException {


response = Unirest.get("http://localhost:3000/rest/candidacy").asJson();
body = response.getBody();
responseExtra = Unirest.get("http://localhost:3000/rest/extra").asJson();

	}

	@Test
	public void testIdExtraAndIdOffer() {

		// Test IdExtra
		assertEquals("59f1dbd830e2ea92834cc568",
body.getArray().getJSONObject(0).getString("idExtra"));
		// Test IdOffer
		assertEquals("59fed81b069a78003b0bc631",
body.getArray().getJSONObject(0).getString("idOffer"));
```

Le test "testIdExtraAndIdOffer" nous permet de vérifier que l'id Offer et l'id Extra est bien récupérer au sein de la collection.

```
	}
	@Test
	public void testEtat() {

		assertNotNull(response);
		assertEquals("Waiting",
body.getArray().getJSONObject(0).getString("etat"));
	}


}
```

https://github.com/JonathanCrt/FaceCast

Le test "testEtat" nous permet de vérifier que l'état d'une candidature n'est pas vide dans la collection et quelle est égal a la chaine de caractère "Waiting" (en attente).

Méthode de test mes candidatures (myApplications) :

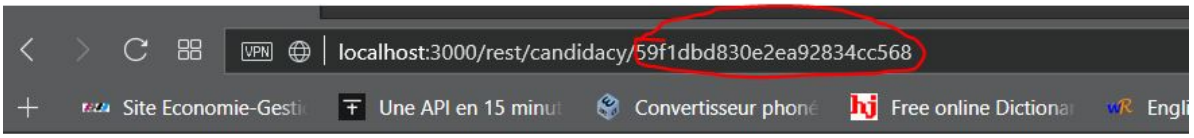
```
@Test
	public void myApplications() {
		try {
			HttpResponse<JsonNode> responseCandidacy = Unirest

.get("http://localhost:3000/rest/candidacy/59f1dbd830e2ea92834cc568").asJson();
			assertEquals(2,
responseCandidacy.getBody().getArray().length());
		} catch (UnirestException e) {
			// TODO Auto-generated catch block
			fail("Error : number of objects in the json");
			e.printStackTrace();
		}

	}
```

Le test "myApplications" nous permet de vérifier que la longueur du tableau correspond bien à celle attendue (soit 2 ici, correspondant aux 2 objets Json)

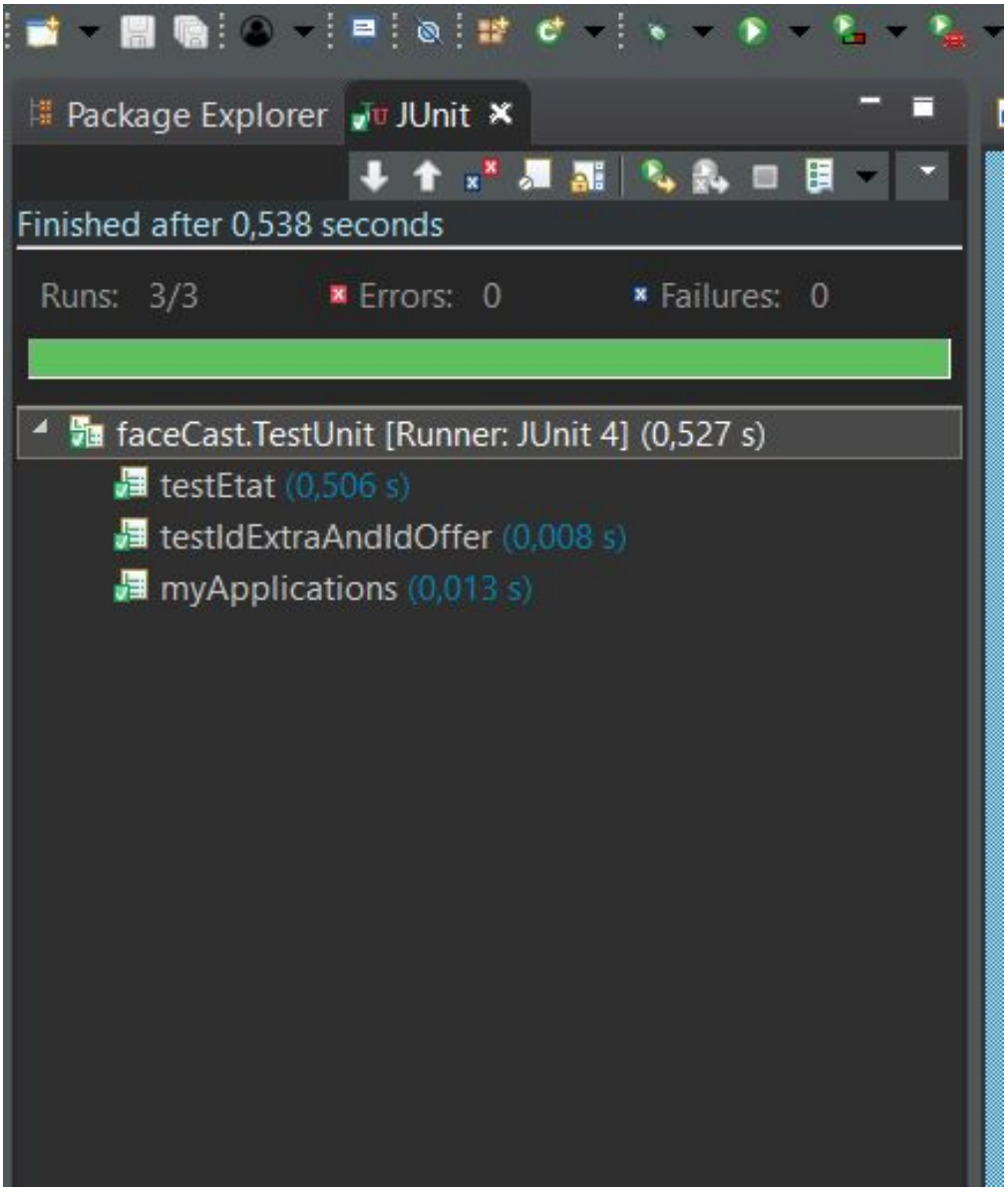
Le assertEquals nous permet de vérifier que la longueur du tableau responseCandidacy est bien de 2 (attendu).



Collapse all Expand all


```
(2) [
- {
    _id: "59ff4e2f0b703667311e80c1",
    etat: "Waiting",
    idExtra: "59f1dbd830e2ea92834cc568",
    idOffer: "59fed81b069a78003b0bc631",
  - offer: (1) [
    - {
        _id: "59fed81b069a78003b0bc631",
        nameEvenement: "Gary Movie",
        type: "Spectacle",
        dateBegin: "2018-01-25",
        numberDay: 15,
        numberExtra: 5,
        role: "Plombier",
        __v: 0
      }
    ]
  },
- {
    _id: "5a00876819d6d9006eb00fb0",
    idOffer: "59fed81b069a78003b0bc631",
    idExtra: "59f1dbd830e2ea92834cc568",
    etat: "Waiting",
    __v: 0,
  - offer: (1) [
    - {
        _id: "59fed81b069a78003b0bc631",
        nameEvenement: "Gary Movie",
        type: "Spectacle",
        dateBegin: "2018-01-25",
        numberDay: 15,
        numberExtra: 5,
        role: "Plombier",
        __v: 0
      }
    ]
  }
]
```


https://github.com/JonathanCrt/FaceCast



Les trois méthodes de test avec JUnit test sont valides ainsi on obtient une couverture de tests de niveau 0 minimum (Test Unitaire).

https://github.com/JonathanCrt/FaceCast

III. Conclusion

Pour conclure ce rapport, l'application web semble bien correspondre aux besoins de l'agence cliente FaceCast . Celle-ci peut bien traiter ses candidatures et administrer les différentes offres tout en ayant une liste des figurants. A cet effet, elle peut ajouter de nouvelles offres avec un simple formulaire de saisie(avec une vérification de la cohérence des champs) et les supprimer si elle le souhaite dans le listing.

D'autre part, les tests unitaires permettent de déceler une partie des bugs actuels sur l'application Web

Ce rapport fait figure de présentation de l'application web FaceCast.

SUPPORT PRODUITS

Outils : Ordinateur, Internet,Espace de travail

Logiciels: Visual Studio Code , Eclipse , PsPad, Robo3T, Serveur Linux (Bash on ubuntu) , Git

Sources: Stack Overflow , w3School, GitHub, CodeSchool

https://www.frugalprototype.com/api-mongodb-mongoose-node-js/

https://blog.nicolashachet.com/niveaux/confirme/larchitecture-rest-expliquee-en-5-regles/

https://zestedesavoir.com/tutoriels/312/debuter-avec-mongodb-pour-node-js/

https://codequs.com/p/B108Pqbt/build-a-restful-api-with-node-js-and-mongodb/

http://unirest.io/java.html

https://github.com/JonathanCrt/FaceCast