Le Projet FaceCast

Contexte

L'agence FaceCast se spécialise dans la fourniture de figurants anonymes pour des événements comme des spectacles ou des films.

Le principe est simple : moyennant une rétribution à la journée, une personne, après acceptation de son CV, peut postuler à un spectacle ou à un film qui nécessite un grand nombre de figurants. Dans certains cas, il peut y avoir une tenue particulière (costume d'époque par exemple) mais il n'y a jamais de script à dire. La figuration par contre, peut-être statique ou dynamique en fonction du scénario.

Actuellement, le secrétariat gère le recrutement et l'organisation des contrats par l'intermédiaire de feuilles de calcul (tableur).

Le directeur de l'agence souhaite basculer une partie des traitements sur, d'une part une **application Web** et d'autre part, une **application mobile Android**. La première, de type Intranet est destinée au « back office » (non accessible aux utilisateurs finaux) et la deuxième au « front office » (pour les figurants).

*Vous êtes chargé, en tant que tout nouveau salarié, de participer à la création de l'**application web** de ce projet (mission1).*

Cahier des charges du projet

La gestion des offres de figuration (création, modification, navigation, suppression, états des candidatures) est sous la responsabilité des gestionnaires de l'agence. Les figurants potentiels pourront candidater à des offres exclusivement par l'intermédiaire de l'application mobile (hors du périmètre de cette mission). Une candidature peut-être, à un instant *t* stable, dans un des états suivants :

enAttente / accepté / refusé / retenu / terminé

L'état *retenu* est le signe d'un processus à 3 étapes : Dans ce cas, la première étape est l'état *retenu* qui déclenche une prise de rendez-vous pour un entretien (cas par exemple de nouveaux arrivants, ou de rôles très spécifiques). Suite à cet entretien, la demande passe à l'état *accepté* ou *refusé,* selon le cas. À la fin de la prestation, la demande passe à l'état *terminé*.

Pour candidater à une offre, l'utilisateur de l'application doit être connu de l'agence FaceCast. Cet aspect du projet est hors du périmètre de cette mission. On considérera donc, dans cette phase du projet, que toute personne ayant installé l'application mobile sur son appareil peut postuler à une offre de figuration.

L'application Web est destinée au gestionnaire. Elle doit lui permettre de traiter les candidatures (initiées par les figurants via l'application mobile). À noter que toute nouvelle candidature est initialement positionnée dans l'état en attente. L'application offrira des entrées par événement à l'origine des offres (spectacle ou film), offre de figuration, et par figurant. Le gestionnaire doit donc pouvoir gérer les offres de figuration et les rôles demandés. La suppression d'une offre de figuration entraîne la suppression des demandes de figurations associées.

Les propriétés minimales d'une **offre** de figuration seront : identifiant (automatique), nom de l'événement et son type (spectacle ou film), date de début, nombre de jours, rôle demandé, nombre de figurants et liste des figurants retenus. Exemple de rôles : "plombier des années 30", "passant (années 80)", "JF modèle d'origine indienne", "joueur de foot (actuel)", "senior avec barbe", "danseuse pour clip". Un figurant est identifié par un id (automatique), nom, prénom, email. **Dans cette release, on pourra ne pas considèrer un événement comme une entité à part (confondue dans l'offre)**

Notes :

- *on ne s'occupera pas de tout l'aspect financier (cachet du figurant, prix du contrat, etc.) qui est géré par le service comptable*

- *pour la bonne tenue des tests, les tables retenues devront être préalablement peuplées avec des données fictives*
- *dans la première release on ne s'occupera pas des modifications, juste de la lecture, de l'insertion et de la suppression des éléments.*

Contraintes :

L'application Web du projet s'appuiera 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 dialoguera avec l'application via le protocole HTTP et des messages au format JSON. Pour cela, vous développerez une API basée sur REST pour les échanges entre la future application Android (**développée par une autre entreprise, et que vous n'aurez donc pas à réaliser**) et l'application NodeJS.

Afin de tester le bon comportement de votre application web vis à vis de la future application mobile, vous programmerez des tests fonctionnels avec java et http://unirest.io/java.html. Cette solution vous permettra de programmer des requêtes JSON (envoi) et analyser des réponses JSON (réception) de votre API, voir l'exemple d'utilisation présenté en annexe.

Un travail préalable de recherche d'informations effectué par un stagiaire en BTS SIO a permis de trouver ceci :

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

Travail à faire en préalable des missions :

1/ Analyse : à rendre et à **faire valider** avant de passer au code !

Documents attendus :
(les items marqués d'une étoile ne font pas partie de la première version de votre analyse – à réaliser après le cours sur ces concepts)

- ***** *Diagramme des cas d'utilisation de l'application*
- Diagramme UML des entités du modèle (diagramme de classes métier)
    - à l'image de l'exemple : http://mongoosejs.com/docs/populate.html

  Diagramme UML E/T du cycle de vie d'une candidature (voir documentation associée à la mission)
- Exemple JSON illustrant votre implémentation de votre modèle : schéma de données MongoDB avec mongoose sur la base de données de test.
  voir : http://mongoosejs.com/docs/populate.html
- Un tableau proposant les différentes URI supportés par votre API. Il comprendra la méthode HTTP utilisé, l'URI s'y rapportant, l'action qui en résulte, le format échangé et le cas d'utilisation concerné.

Date butoir : ***mardi 10 octobre, 12h***

Mission 1 : *back-office*

Réalisation d'une première version orientée **back-office** (à destination des personnels de l'agence FastCast)

Technologies utilisées : serveur **nodejs**, **mongodb**, **MVC** avec express, **tests unitaires** avec unirest.

Cas d'utilisation attendus :

- Gestion des offres
- Gestion de l'état des candidatures (selon le cycle de vie d'une candidature représenté par un diagramme d'état-transition de votre analyse)
- Tests unitaires de l'API, développés en java, à destination de la futur application android (voir annexe A)

Date butoir : *retour des vacances de Toussaint*

L'évaluation portera sur l'analyse et la maîtrise de la réalisation (implémentation et tests)

Mission 2 : *front-office*

Réalisation d'une première version orientée **front-office** (à destination des figurants potentiels)

Technologies utilisées : **android natif**, interaction avec des **services distants** (online) de type REST avec JSON comme format d'échange.

Cas d'utilisation attendus :

- Gestion d'une API key (la société FasCast souhaite contrôler l'accès à ses services API). L'application mobile devra pouvoir :
  - Demander et sauvegarder l'APIKEY sur le mobile
  - Tester la validité de l'APIKEY
  - Faire une demande de renouvellement de l'APIKEY, si périmée

Aide : voir https://www.tutorialspoint.com/android/android_shared_preferences.htm

- Gestion des candidatures (acteur : le figurant)
  - Voir mes candidatures en cours
  - Voir mes candidatures passées

- Consultation des offres (acteur : le figurant)
  - Voir les offres en cours
  - Voir les offres passées
  - Voir les offres passées auxquelles j'ai postulé, avec leur état

Date butoir : *retour de stage*

L'évaluation portera la maîtrise de la réalisation (implémentation et tests) et sa cohérence avec les uses cases attendus.

Note : *la mise en place du projet sur BitBucket est attendue. Un « leader » se charge de sa création et de la constitution de l'équipe (obligatoire même si vous êtes seul).*

ANNEXE A : Exemple d'un test automatisé d'une API avec UniREST

```
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 TestRestAPI {
  /** (api.json à placer sur un serveur pour le bon déroulement du test)
   *
   * <pre>
  {
  "lotto":{
   "lottoId":5,
   "winning-numbers":[2,45,34,23,7,5,3],
   "winners":[{
      "winnerId":23,
      "numbers":[2,45,34,23,5,3]
      },{
      "winnerId":54,
      "numbers":[2,45,23,7,5,3]
      }]
   }
  }
   * </pre>
   */

  @Test
  public void test() {
    try {
      // http://unirest.io/java.html
      HttpResponse<JsonNode> response =
          Unirest.post("http://localhost/~kpu/api.json")
           .header("accept", "application/json")
           .queryString("apiKey", "123")
           .field("parameter", "value")
           .field("foo", "bar")
           .asJson();

      int code = response.getStatus();
      JsonNode body = response.getBody();

      assertNotNull(response);
      assertEquals(200, code);
      assertEquals(5,
          body.getObject().getJSONObject("lotto").getLong("lottoId"));

    } catch (UnirestException e) {
      e.printStackTrace();
      fail(e.getMessage());
    }
  }

}
```