Spécifications des ressources

**API REST**

# Introduction

Ce projet est une implémentation du célèbre site Web Stackoverflow (renommé Pokeflow dans notre cas).

Ce projet est divisé en deux parties : un service web implémenté avec Grails utilisant une architecture REST et l’interface utilisant les langages HTML/CSS/JavaScript qui communique avec le service Web.

# Pourquoi l'architecture REST ?

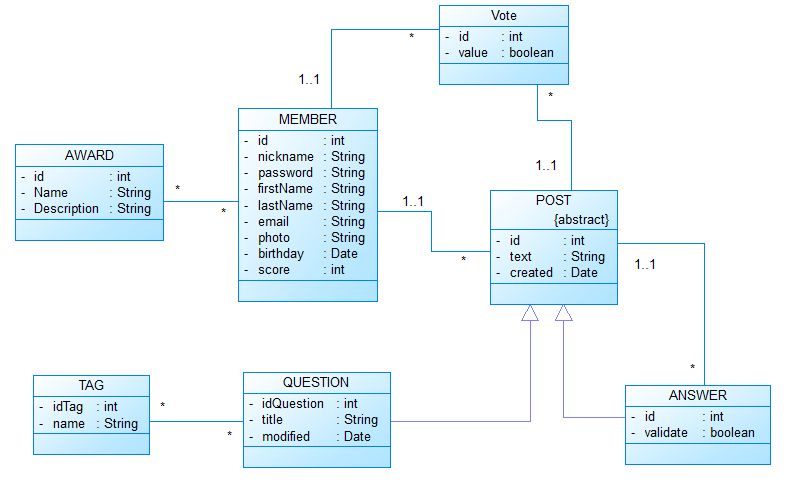
REST (**RE**presentational **S**tate **T**ransfer) est une architecture de services Web. REST part du principe selon lequel HTTP suffit à l'ensemble des besoins d'un service Web, pour peu qu'on utilise l'ensemble des méthodes de ce protocole : GET et POST notamment.

Le respect du style d’architecture REST nous permet d’avoir une interface simple et uniforme pour chacune des ressources.

Le principal avantage d'utiliser ce style d'architecture est que l'interopérabilité de notre application sera plus facile à mettre en œuvre.

# Diagramme UML

Voici ci dessous le diagramme UML qui a été utilisé pour l'élaboration de notre application :



**Détail des différentes classes implémentées :**

* **La classe "member" :** cette classe modélise un utilisateur, c'est-à-dire la personne qui va se connecter à l'application pour interagir avec celle-ci (poser des question, modifier son profil, etc). Elle possède de nombreux attributs décrivant l'utilisateur tels que son score, son mot de passe, ... En revanche, l'attribut "photo" ne sera pas utilisé dans notre application (elle peut par l'être ultérieurement) ;
* **La classe "post" :** représente ce qu'un utilisateur peut partager sur l'application. Cette classe est abstraite puisqu'un post peut être soit une question, soit une réponse. Elle possède un id, le contenu du post (le texte) ainsi que la date de création du post ;
* **La classe "question" :** modélise simplement la question posée par un utilisateur (hérite de la classe "post"). Elle possède un id, un titre (la question elle-même) ainsi que la date de modification du contenu de la question ;
* **La classe "answer" :** modélise la réponse à une question posée par un utilisateur (hérite elle aussi de la classe "post"). Elle possède un id et un booléen "validate" qui permet de valider une réponse (réponse validée par un utilisateur si sa question a été résolue par cette réponse) ;
* **La classe "tag" :** permet simplement d'ajouter des tags aux questions. Elle possède un id ainsi qu'une suite de tags représentés par une chaine de caractères ;
* **La classe "vote" :** modélise les votes des utilisateurs. Un post peut être votée. Un vote sert a expliciter la pertinence d'une réponse ou d'une question. Un vote possède un id ainsi qu'un booléen "value" qui permet de savoir si un utilisateur à déjà voté sur un même post ou non (en effet, un utilisateur ne peut voter deux fois un même post).
* **La classe "award" :** représente les récompenses qu'un utilisateur peut potentiellement obtenir. Plus un utilisateur aura un score élevé, plus il sera récompensé. Il obtient un meilleur score grâce aux points obtenus provenant des posts qu'il a partagé (une réponse efficace par exemple).

**Détail des associations :**

Un utilisateur peut poster une infinité de questions/réponses et un post ne peut appartenir qu'a un seul utilisateur.

Chaque membre peut voter autant de fois qu'il y a de post (le vote d'un utilisateur particulier n'appartient qu'à un seul post). Un post peut avoir bien entendu autant de votes qu'il y a d'utilisateurs.

Un post peut avoir une infinité de réponse et une réponse n'appartient qu'a un seul post (la question posée).

Un membre peut posséder autant de récompenses qu'il en existe et une récompense peut appartenir à autant d'utilisateurs existants.

Une question possède une infinité de tags et un tag peut être possédé par une infinité de questions.

Remarque sur le cas des associations de type n-n : Groovy ne permet pas de gérer plusieurs associations n-n sur une même classe. C'est pourquoi, une classe supplémentaire est créée pour gérer ce cas.

# Détails des différentes méthodes implémentées

## /rest/auth/session

**Fonction :** sert à un utilisateur de se connecter (pour qu'il puisse ainsi poser des questions).

**Détail sur le fonctionnement :** une nouvelle session est créée pour l’utilisateur. Le type d’authentification est HTTP BASIC.

**Méthode utilisée :** POST.

**Données envoyées :**

* "username": "fred" ;
* "password": "freds\_password".

**Réponses possibles :**

* 200 : Successful (aucune erreur), les valeurs récupérables en JSON sont les suivantes :
  + "id" : "1" ;
  + "nickname" : "fred" ;
  + "password" : "iOZKe21=" ;
  + "firstName" : "Frédéric" ;
  + "lastName" : "Farouzza" ;
  + "email" : "farouzza@gmail.com";
  + "birthday" : "date" ;
  + "score" : "0".
* 401 : Invalid credentials ;
* 400 : Bad Request (la syntaxe de la requête est erronée).

## /rest/api/search

**Fonction :** sert à effectuer une recherche grâce à la barre de recherche.

**Détail sur le fonctionnement :** envoie une requête de recherche au serveur. Le champ "request" présente le motif à chercher partout (titre ou contenu de la question). Envoie de toutes les questions si le motif est nul.

**Méthode utilisée :** GET.

**Données envoyées :**

* "request" : "motif" ;
* "startAt":"0" ;
* "maxResults":"100".

**Réponses possibles :**

* 200 : Successful (aucune erreur), les valeurs récupérables en JSON sont les suivantes :
  + "id" : 2 ;
  + "author" : "Elytio" ;
  + "created" : "08/02/2014 15:00:53" ;
  + "modified" : "08/02/2014 15:00:53" ;
  + "title" : "How can i get a Pikachu level 100 ??" ;
  + "text" : "I have a yellow version of Pokemon and i want to know if it is possible to push the starter Pikachu to the level 100 ? If yes, how can i do it ??" ;
  + "votes" : 0 ;
  + "tags" : "tag1 tag2 tag3" ;
  + "note" : 4 ;
  + "answers" : 1.
* 400 : Bad Request (la syntaxe de la requête est erronée).

## /rest/api/question

**Fonction :** permet de créer une nouvelle question.

**Détail sur le fonctionnement :** c’est le serveur qui se charge de remplir les autres champs à savoir les dates, l’auteur grâce à l’identification contenue dans l’en-tête et l’id qu’il attribue.

**Méthode utilisée :** POST.

**Données envoyées :**

* "title" : "Question test" ;
* "text" : "Ceci est un test !" ;
* "tags" : "tag1 tag2".

**Réponses possibles :**

* 200 : Successful (aucune erreur), la valeur récupérable en JSON est la suivante :
  + "id" : 3.
* 401 : Invalid credentials ;

## /rest/api/question/{id} (GET)

**Fonction :** récupération des informations de la question répondant à l’id demandé.

**Méthode utilisée :** GET.

**Réponses possibles :**

* 200 : Successful (aucune erreur), les valeurs récupérables en JSON sont les suivantes :
  + "id" : 2 ;
  + "author" : "Elytio" ;
  + "created" : "08/02/2014 15:00:53" ;
  + "modified" : "08/02/2014 15:00:53" ;
  + "title" : "How can i get a Pikachu level 100 ??" ;
  + "text" : "I have a yellow version of Pokemon and i want to know if it is possible to push the starter Pikachu to the level 100 ? If yes, how can i do it ??" ;
  + "votes" : 0 ;
  + "tags" : "tag1 tag2 tag3" ;
  + "note" : 4 ;
  + "answers" : 1.

## /rest/api/question/{id} (PUT)

**Fonction :** mise à jour de la question spécifiée.

**Méthode utilisée :** PUT.

**Données envoyées :**

* "text" : "Y a des carottes magiques !".

**Réponses possibles :**

* 200 : Successful (aucune erreur) ;
* 401 : Invalid credentials.

## /rest/api/post/{id} (POST)

**Fonction :** création d'une nouvelle réponse au post spécifié.

**Méthode utilisée :** POST.

**Données envoyées :**

* "text" : "Ceci est une réponse au post d'id id".

**Réponses possibles :**

* 200 : Successful (aucune erreur) ;
* 401 : Invalid credentials.

## /rest/api/post/{id} (DELETE)

**Fonction :** suppression du post spécifié (question ou réponse).

**Méthode utilisée :** DELETE.

**Réponses possibles :**

* 200 : Successful (aucune erreur) ;
* 401 : Invalid credentials.

## /rest/api/post/{id}/answers

**Fonction :** récupération des informations de la question répondant à l’id demandé.

**Méthode utilisée :** GET.

**Réponses possibles :**

* 200 : Successful (aucune erreur), les valeurs récupérables en JSON sont les suivantes :
  + "id" : 5 ;
  + "author" : "georges" ;
  + "created" : "28/01/2014" ;
  + "modified" : "29/01/2014" ;
  + "text" : "Texte de la question" ;
  + "votes" : 4 ;
  + "answers" : 2.

## /rest/api/post/{id}/vote

**Fonction :** envoi d'un vote appliqué au post spécifié.

**Méthode utilisée :** POST.

**Données envoyées :**

* "vote" : "+1".

**Réponses possibles :**

* 200 : Successful (aucune erreur).

## /rest/api/user

**Fonction :** création d'un nouvel utilisateur.

**Méthode utilisée :** POST.

**Données envoyées :**

* "nickname" : "woute" ;
* "password" : "e2uZm2==" ;
* "firstName" : "Jerome" ;
* "lastName" : "delajungle" ;
* "email" : "vroum@coucou.net" ;
* "birthday" : "03/11/1923".

**Réponses possible :**

* 200 : Successful (aucune erreur), les valeurs récupérables en JSON sont les suivantes :
  + "id" : "8" ;
  + "nickname" : "woute" ;
  + "password" : "e2uZm2==" ;
  + "firstName" : "Jerome" ;
  + "lastName" : "delajungle" ;
  + "email" : "vroum@coucou.net" ;
  + "birthday" : "03/11/1923" ;
  + "score" : "0".

## /rest/api/user/{id}

**Fonction :** mise à jour des informations concernant un utilisateur.

**Méthode utilisée :** PUT.

**Données envoyées :**

* "password" : "e2uZm2==" ;
* "firstName" : "Jeveuxplusjerome" ;
* "lastName" : "delajungle" ;
* "email" : "vroum@coucou.net" ;
* "birthday" : "03/11/1923".

**Réponses possibles :**

* 200 : Successful (aucune erreur), les valeurs récupérables en JSON sont les suivantes :
  + "id" : "8" ;
  + "nickname" : "woute" ;
  + "password" : "e2uZm2==" ;
  + "firstName" : " Jeveuxplusjerome" ;
  + "lastName" : "delajungle" ;
  + "email" : "vroum@coucou.net" ;
  + "birthday" : "03/11/1923" ;
  + "score" : "0".
* 401 : Invalid credentials.

# Note de conclusion

Le projet est malheureusement incomplet et même si le squelette de l'application est implémenté, certaines fonctionnalités ne sont pas terminées. De plus, les test unitaires ont dû être négligés car la plus grosse partie du projet a été réalisée en HTML/JS.