Gamification API

# But de l’API

Le but de l’API est de fournir un service de Gamification pour n’importe quelle application. Elle permet à des joueurs d’obtenir des points et des badges via un système de règles et d’évènements.

# Documentation de l’API

## Ressources

L’API reprend, en grande partie, la structure proposée par le cours.

Ainsi, on trouve les ressources suivantes.

### Applications

Affin de pouvoir utilisé le service pour plusieurs applications, l’API sépare les données par application.

La ressource « Application » permet ainsi d’enregistrer le nom de l’application et de définir une clé et un mot de passe permettant l’authentification.

### Players

La ressource « Players » permet de gérer les joueurs et leur nombre de points.

### Badges

Les badges peuvent être gagnés par des joueurs (voir events et rules). Chaque badge possède un nom, une description et un icône.

### Rules

La ressource « Rules » permet de définir les règles à appliquer lors d’un évènement particulier. Elles contiennent le nombre de points et les badges reçus par un joueur lors d’évènements d’un certains type.

### Events

Les évènements sont liés à un joueur et à un type. Lors de la création d’un évènement la liste des règles est parcourue et les points et les badges sont attribués au joueur.

### Leaderboard

## Apiary.io

La documentation complète de l’API a été réalisée avec l’application en ligne « apiary.io ». Elle peut être consultée à l’adresse suivante : « docs.aurelienthevoz.apiary.io » ou dans le fichier « apiary.io.txt ».

# Structure du code

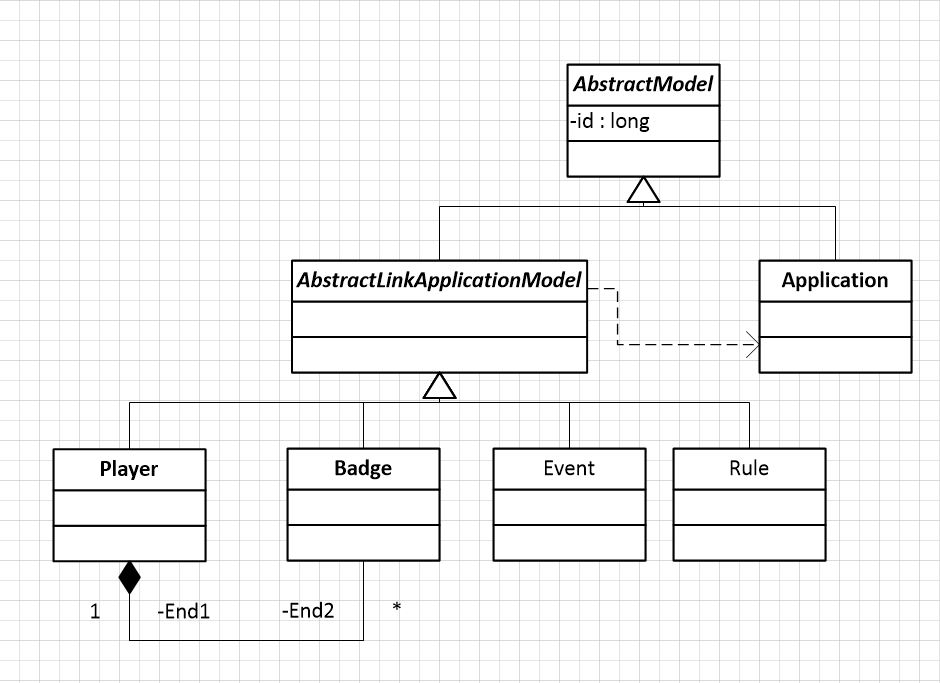
## Diagrammes de classes

### Modèle des données

Le modèle de données reprend les ressources vues précédemment (excepté la ressource leaderboard qui est simplement calculée.) Chaque classe du modèle hérite de la classe « AbstractModel » contenant des méthodes et attributs de bases (getId(), setId(), hash(), etc.)

A part la classe « Application », les autres classes héritent de « AbstractLinkApplicationModel » qui ajoute un lien vers la classe « Application ». Ainsi, chaque donnée est liée à une application.

Ci-dessous le diagramme de classe du modèle :

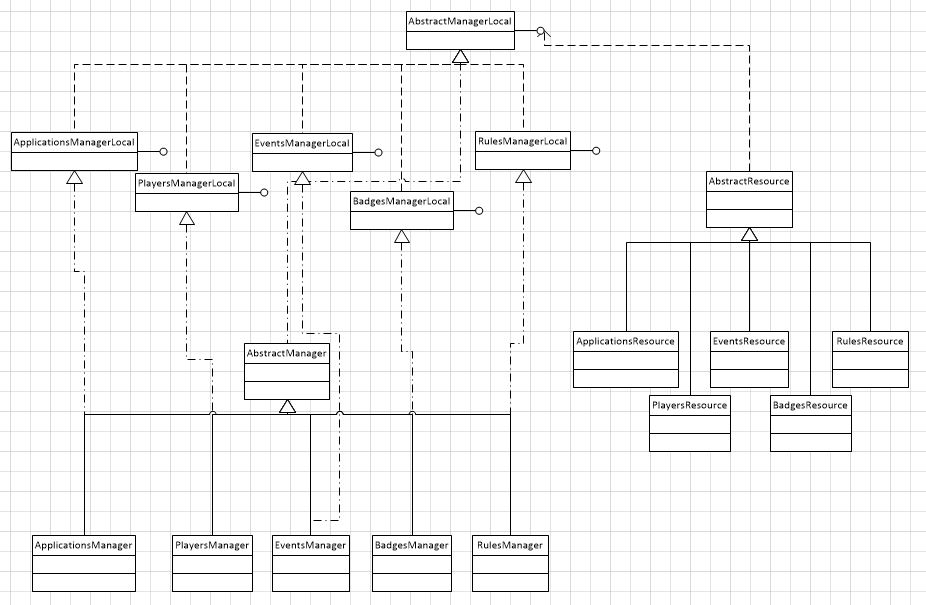


### Managers et ressources

Les classes « Managers » exposent les méthodes permettant les interactions avec la base de données. Chaque manager à la responsabilité de gérer une classe du modèle. Les classes « manager » héritent de la classe « AbstractManager », proposant une implémentation de base pour les quatre opérations (create, read, update et delete.)

Les classes de ressources permettent de définir les méthodes et les paramètres de l’API. Elles utilisent des liens vers les classes « Manager » pour récupérer et modifier les données. Elles héritent de la classe « AbstractResource » définissant les attributs utilisés pour l’authentification (voir le chapitre authentification pour plus de détails.).

Ci-dessous, le diagramme de classes des managers et des ressources :

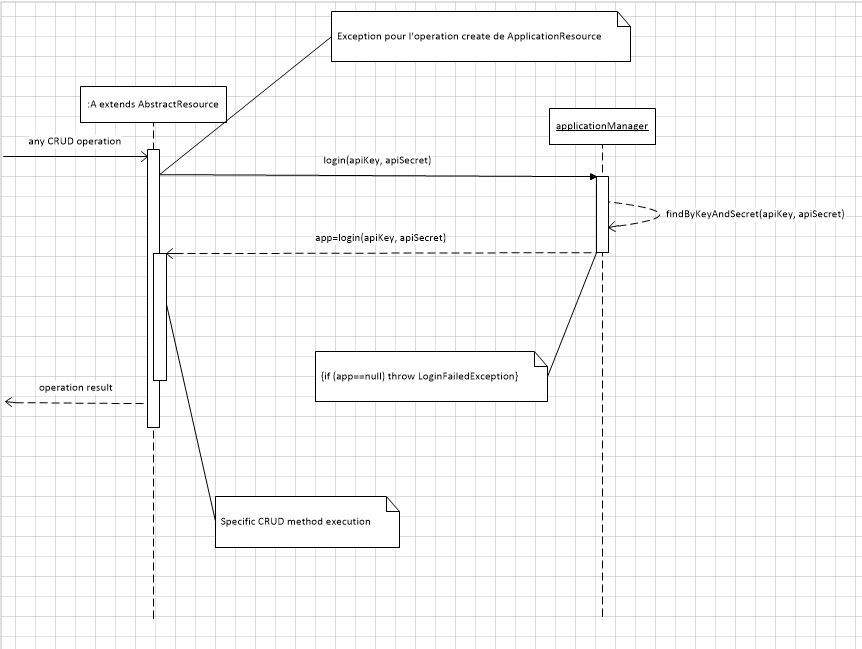


## Diagrammes de séquences

### Authentification

Lors de la création d’une ressource « Application » l’application cliente définit une clé et un mot de passe (apiKey et apiSecret.) Ceux-ci, sont ensuite utilisés pour l’authentifier. Lors de chaque appel à l’API, le client doit transmettre dans l’entête http les paramètres « apiKey » et « apiSecret ». Ceux-ci sont vérifiés par le manager et retourne un status « unauthorized » en cas d’erreur.

Ci-dessous, le diagramme de séquence de l’opération :



### Gestion des évènements

Lors de la création d’un évenement, l’ « EventsManager » notifie le « RulesManager ». Ce dernier récupère le « player » en utilisant le « PlayersManager », parcoure les règles du type de l’évènement et met à jour les points et les badges du Player.

Ci-dessous, le diagramme de séquence de la gestion des évenements :

