MOVING: Liens entre Tag et Scènes

*Spécifications*

Ce document synthétise les spécifications logicielles relatives au système à mettre en place pour la gestion des liens entre les tags (NFC/iBeacon/QRCode) ou les coordonnées GPS et les Scènes du BackOffice.

Thomas Badin

22/05/2014

MOVING: Liens entre Tag et Scènes

Spécifications

Table des matières

Introduction 2

Le serveur de lien 4

La base de données 4

Schéma 4

Explication 4

Détails des champs 5

Le cœur de l’application 5

L’interface d’Administration 6

Gestion des BOs 6

Administration du LinkServer 6

Les WebServices 7

WebServices Read 7

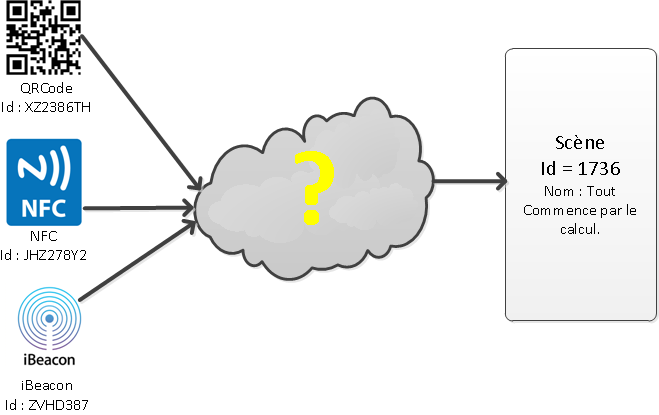
WebServices Post 10

L’API Java 11

# Introduction

L’application mobile a besoin d’interagir avec différents types de capteurs. Les différents type de tag (QRCode / NFC / iBeacon) contiendront des ids qui référenceront des Scènes du BO. Or les ids stockés dans les Tag ne seront pas nécessairement les même que les ids du BO.

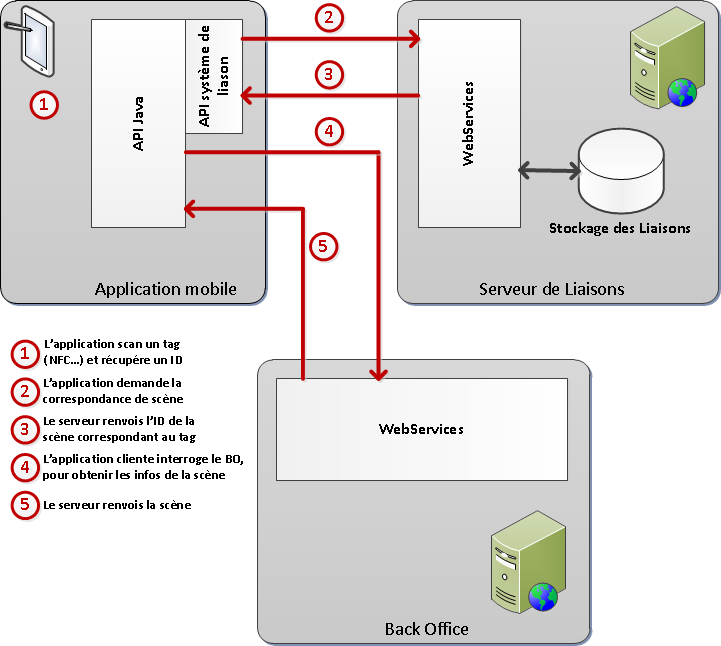
Nous avons donc besoin, et c’est ce que ce document va définir, du mécanisme suivant :



Un peu comme la manière du fonctionnement de DNS, le mécanisme qui fera le lien entre l’id d’un tag et l’id d’une scène dans le BO devra être indifférents du BO.

Pourquoi ce besoin d’indépendance ? Premièrement car l’administration doit être indépendante, ce ne seront pas les même personnes qui vont administrer le backoffice (créer les parcours, les scènes, etc.) et celle qui vont administrer les liens. De plus, a terme, le serveur de liens ne fera pas forcement la correspondance avec un serveur Moving-BO, mais pourra être utilisé pour autre chose. D’ou un serveur séparé, et une API Java dédiée.

Le fonctionnement devra donc être le suivant :



Le présent document détaillera donc la conception du serveur de liaison, de ses WebServices, ainsi que de l’API Java qui utilisera ces WebServices. Quand au BO, celui-ci ne connaitra pas l’existence du mécanisme de lien, il n’y aura donc aucune modification à effectuer.

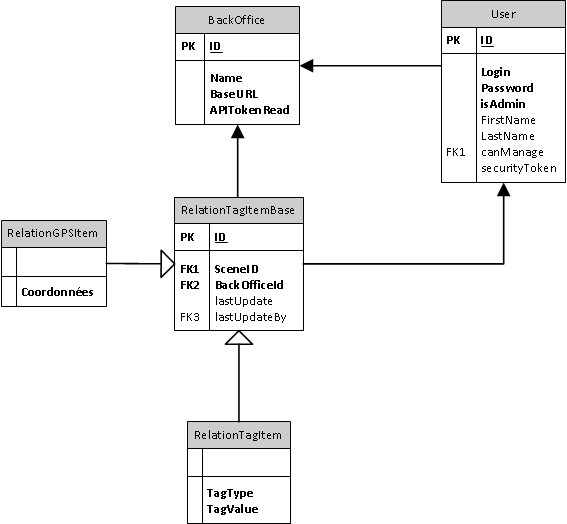
# Le serveur de lien

Nous avons choisis d’utiliser la technologie Java JEE afin de réaliser le serveur de lien. Nous avons choisie cette technologie car nous la maitrisons bien, qu’elle est open source, très répandue, et qu’il est facile de réaliser des WebServices. Afin de faciliter le développement et de rendre le code plus propre est standardisé (MVC), nous avons choisi d’utiliser le Framework Spring. De plus même s’il n’était pas indispensable d’utiliser un Framework aussi complet que Spring, cela nous permettra de nous former sur cette technologie.

## La base de données

### Schéma

La base de données sera assez simple. Elle contiendras deux parties ; la partie de lien proprement dite (qui stockera la relation entre les Tag / une zone GPS) et une partie administration.



### Explication

Le serveur de lien aura un fonctionnement multi-tenant, c’est à dire qu’il gérera les relations pour plusieurs BO. D’où la nécessité d’enregistrer les BO (entité BackOffice). Cependant dans la pratique, un utilisateur (en dehors des administrateurs du LinkServer) ne participe qu’à un seul BO (relation « canManage » du schéma). Dans le cas (rare) ou un utilisateur devra participer à 2 BO, il aura deux compte (ce qui n’est pas un problème, car c’est le même cas pour l’application Android, une application sera liée à 1 et 1 seul BO).

Un administrateur pourra bien sur gérer toutes relation de tous les BO.

### Détails des champs

|  |  |  |  |
| --- | --- | --- | --- |
| Entité | Champs | Valeur | Description |
| RelationTagItemBase | SceneID | Clef étrangère | Ce n’est pas une vrai FK, car l’entité référencée (la scène), ne fait pas partie de ce système. Il n’y aura pas donc d’intégrité référentiel |
| RelationTagItemBase | BackOfficeId | Clef étrangère | Indique à quel BO appartient le lien |
| RelationTagItemBase | lastUpdate | Date | Date de la dernière MAJ |
| RelationTagItemBase | lastUpdateBy | FK | Lien vers le user (admin ou non) qui est le dernier à avoir modifié cette ligne. |
| RelationTagItem | TagType | NFC/QRCode/iBeacon | Permet de stocker le type |
| RelationTagItem | TagValue | Identifiant | Identifiant du tag (chaine de caractère) |
| RelationGPSItem | Coordonées | V1 : Point (X,Y)  V2 : Polygon | Stock les coordonnées GPS de la scène. |
| BackOffice | Name | String (unique) |  |
| BackOffice | BaseURL | Url de la racine du BO (unique) |  |
| BackOffice | APITokenRead | Token (unique) | Token d’authentification pour l’accès aux WS en lecture |
| User | Password | String | Hash du password |
| User | isAdmin | Boolean | Indique si l’utilisateur est un admin |
| User | canManage | FK | Lien vers le BO que l’utilisateur peut gerer. (Si l’user est un admin canManage == null) |
| User | securityToken | Token (unique) | Token d’identification de l’utilisateur, utiliser pour l’authentification des WS Post. Note : si l’user est un admin le token est null. 🡺 Un admin ne peut pas gérer un BO par WS. |

La base de données utilisée sera une base de données MySQL. On utilisera l’extension GIS de mysql (<http://dev.mysql.com/doc/refman/5.0/fr/gis-introduction.html>) pour la partie géographique.

## Le cœur de l’application

Le cœur de l’application sera une application Spring MVC. Les modules Spring suivant seront utilisés :

* Spring MVC
* Spring data JPA pour la création et l’accès aux données
* Spring Security
* Spring WS pour les WebServices

## L’interface d’Administration

Un utilisateur devra être authentifié pour accéder à l’interface d’administration. (CF Spring Security).

Il y aura 2 interfaces d’administrations, une permettant simplement de gérer un BO (utilisateur standard), et une interface d’administration complète, pour l’administrateur du LinkServer.

### Gestion des BOs

Les fonctionnalités de l’interface seront les suivantes :

|  |  |  |
| --- | --- | --- |
| Catégorie | Nom | Remarques |
| Gestion des Admin | Créer un administrateur |  |
| Gestion des Admin | Supprimer un administrateur | (Il doit en rester 1 au minimum) |
| Gestion des Admin | Modifier son compte (password / nom / prénom) |  |
| Gestion des BO | Créer un BO |  |
| Gestion des BO | Modifier un BO (nom / URL) |  |
| Gestion des BO | Régénérer clef d’API (read /post) |  |
| Gestion des BO | Supprimer BO | Il ne doit pas être lié à une relation (pas de DROP CASCADE ! ) |
| Gestion des liens | Ajouter un lien Tag -> Item |  |
| Gestion des liens | Supprimer un lien (Tag / GPS ) -> item |  |
| Gestion des liens | Modifier un lien Tag -> Item |  |
| Gestion des liens | Ajouter un lien GPS -> Item | L’interface d’admin doit permettre à l’administrateur de selectionner des données sur une carte. (Regarder du coté d’Open Street Map, si jamais c’est trop compliqué utiliser Google Map) |
| Gestion des liens | Modifier un lien GPS -> Item |

### Administration du LinkServer

## Les WebServices

### WebServices Read

#### Fonction getSceneIdByTag

Cette fonction permet de faire la correspondance entre l’ID d’un Tag et l’Id de la scène correspondante dans le BO. Elle prend en paramètre :

* Un TokenAPIRead (correspondant au BO)
* Un Id de tag
* Un Type de tag (NFC / iBeacon / QRCode)

##### Cas standard

Le TokenAPI correspond à un BackOffice, le couple TagId / TagType existe.

Le XML retourné est très simple, il contient l’ID de la scène.

##### Cas d’erreurs

* Le Token n’est pas valide (il ne correspond pas à un APITokenRead du BO) : retourner une erreur (non autorisé)
* Le Token est valide mais le couple TagId / TagType n’existe pas : retourner une erreur (not found)

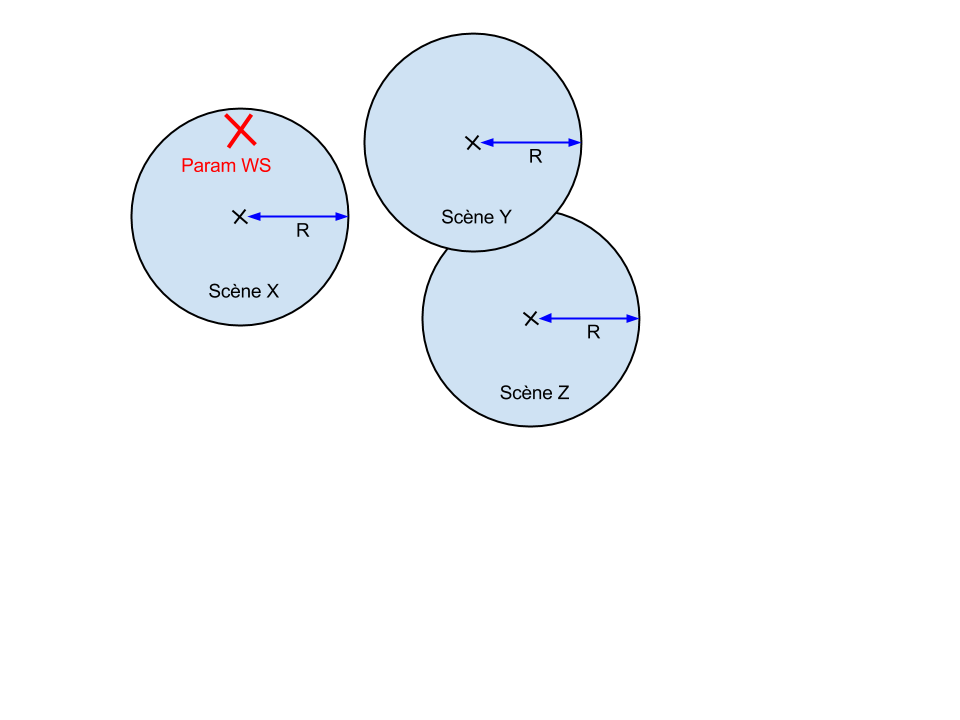
#### Fonction getSceneByGPSCoord V1

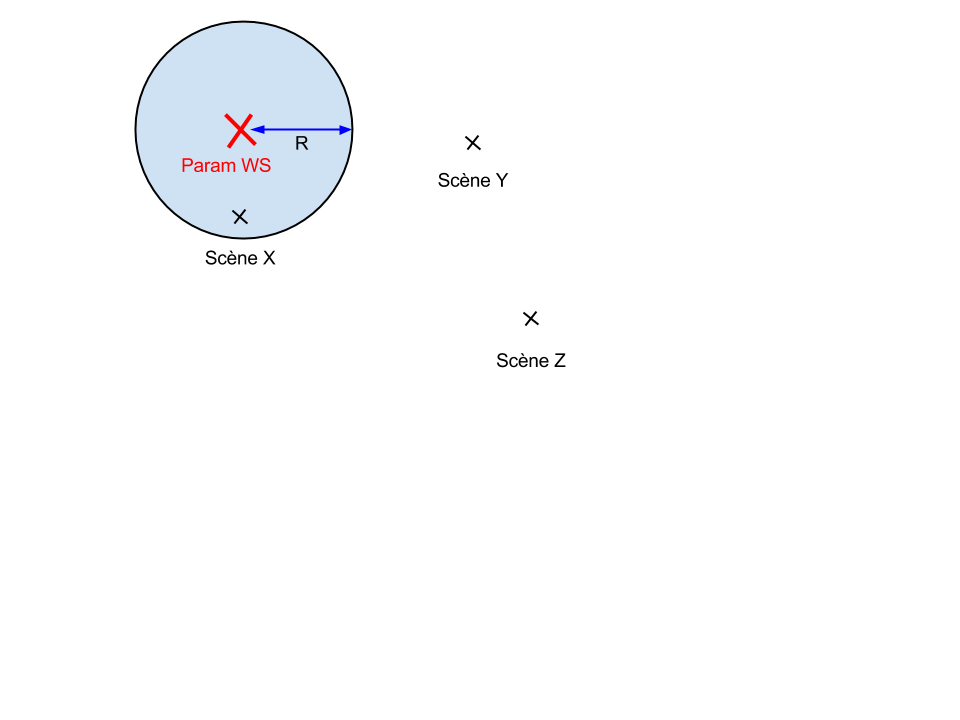
Fonction simple permettant de retourner la scène correspondant à une coordonnées géographique s’il y en a une. (Eventuellement les scènes, dans le cas ou 2 scènes se chevauchent géographiquement).

Cette fonction prend en paramètre :

* Un TokenAPIRead (correspondant au BO)
* Une coordonnée GPS (latitude, longitude)

Pour déterminer si une scène appartient ou non à une coordonnée GPS, on utilisera les coordonnées du point stocké dans la BDD et un rayon R qui sera une constante dans l’application (qui correspond à un rayon de tolérance). En résumé, le WebService retournera toutes les scènes dont le point passé en paramètre est à moins de R distance des centres.

Conceptuellement, on pourra représenter les scènes de manières suivantes (cf ci contre). Dans cet exemple le WS retournera l’Id de la Scène X, car elle est contient le point passé en paramètre du WS. Conceptuellement, une scène est une zone mais d’un point de vue de l’algorithme il est plus simple de prendre le raisonnement inverse : une scène est un point, et on recherche tous les points à moins de R de distance.

Ce qui reviens donc simplement à chercher tous les points dans la zone de centre passé en paramètre et de rayon R.

##### Cas standard

Le TokenAPI correspond à un BackOffice

Le XML retourné est très simple, il contient une liste d’Id de scène. Cette liste sera :

* Soit vide, s’il n’y a aucune scène sur la zone
* Soit contenant s’il y a une scène sur la zone
* Soit plusieurs scènes dans le cas ou des scène se chevauchent

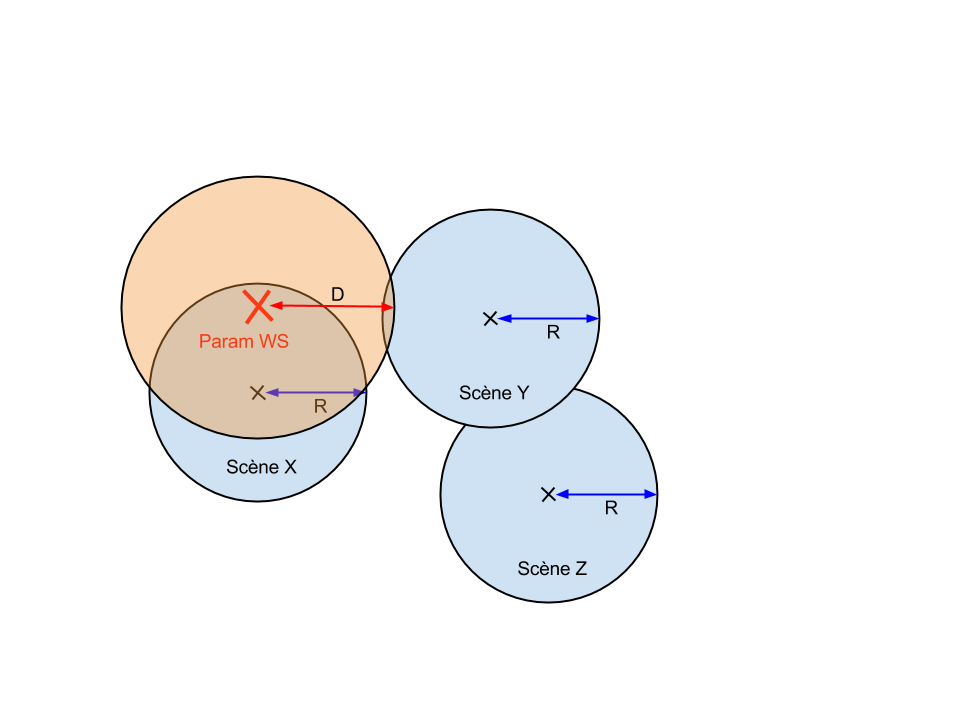
##### Cas d’erreurs

* Le Token n’est pas valide (il ne correspond pas à un APITokenRead du BO) : retourner une erreur (non autorisé)

#### Fonction getCloseSceneByGPSCoord V1

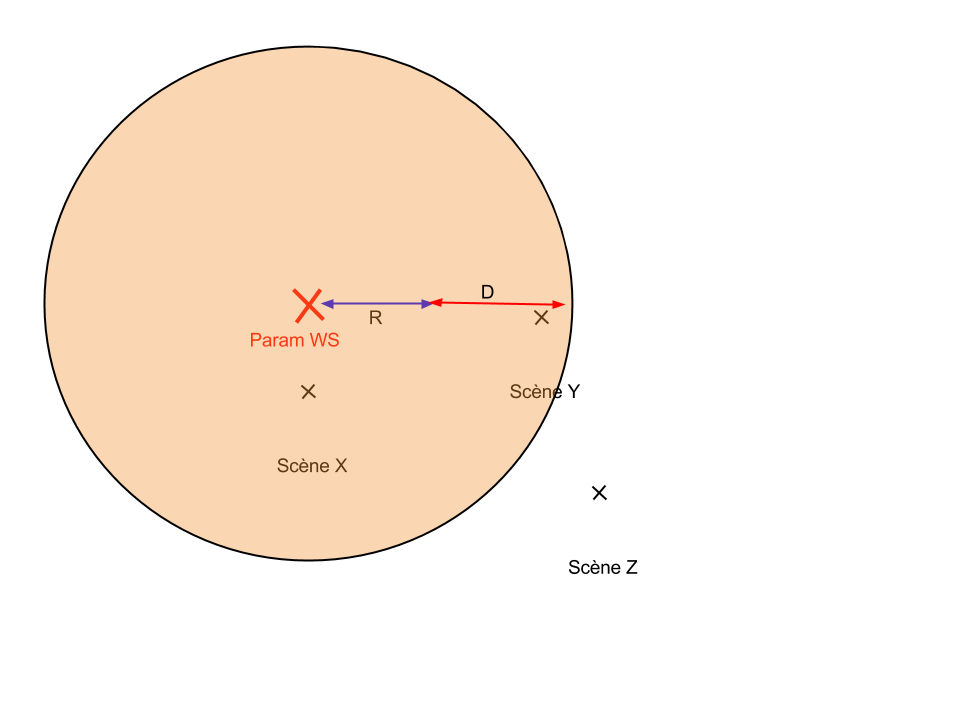
Fonction simple permettant de retourner les scènes correspondant proche d’une coordonnée géographique.

Cette fonction prend en paramètre :

* Un TokenAPIRead (correspondant au BO)
* Une coordonnée GPS (latitude, longitude)
* Une distance D, correspondant au rayon de recherche.

Conceptuellement, on peut représenter la recherche de manière ci contre. Il y a deux intersection entre le rayon de recherche (D) et les Scènes (toujours stockée de la manières suivante un centre dans la BDD, et un rayon R, constante de l’application).

Dans cet exemple, le WebService retournera les ids des scènes X et Y.

L’algorithme de recherche sera très simple car cette recherche est identique à la recherche de points qui ont une distance inférieure à R+D du point passé en paramètre. Cela correspond au schéma ci contre.

##### Cas standard

Le TokenAPI correspond à un BackOffice

Le XML retourné est très simple, il contient une liste d’Id de scène. Cette liste sera :

* Soit vide, s’il n’y a aucune à moins de D distance
* Soit la liste des ids de scène dans la zone

##### Cas d’erreur

* Le Token n’est pas valide (il ne correspond pas à un APITokenRead du BO) : retourner une erreur (non autorisé)

#### Fonction getSceneByGPSCoord V2

#### Fonction getCloseSceneByGPSCoord V2

### WebServices Post

#### Fonction postLinkTagScene

Fonction permettant d’ajouter une liaison entre un tag et une scène. Elle prend en paramètre :

* Un Token (un TokenPost)
* Un type de tag (NFC / QRCode / iBeacon)
* Un Id de tag
* Un Id de Scène

##### Cas standard

Le token est valide et le couple TagType / TagScene n’existe pas encore.

La fonction insert les valeurs dans la base de données (avec le BO qui correspond au token) et renvois un message indiquant que tout c’est bien déroulé.

##### Cas d’erreurs

* Le Token n’est pas valide : renvoyer une erreur (non autorisé)
* Le couple Id tag / Id Scene

##### Améliorations

Une amélioration intéressante serait de vérifier que l’id du media existe bien sur le BO concerné. Cela est possible a réaliser, le LinkServer connaît de chaque BO, il peut très bien faire appel au WS getSceneById sur le bon BO pour vérifier que l’Id de la scène est bon. Mais cela reste compliqué à gérer, il faudrait que le LinkServer ait un compte sur chaque BO et qu’il utilise OAuth2 pour authentifier sa requête. Donc dans un premier temps, il n’y aura pas de vérification d’Id Scène.

#### Fonction postLinkGPSCoordScene V1

#### Fonction postLinkGPSCoordScene V2

# L’API Java