[[Home]] > [[Sommaire]]

[[Sommaire]]

[[1 BackOffice WebServices]]

- [[1.1 WS Auth]]
- [[1.2 WS Read]]
- [[1.3 WS Post]]

[[2 BackOffice Modifications]]

- [[2.1 I18n]]
- [[2.2 Historique Consultation]]
- [[2.3 Historique Contribution]]
- [[2.4 Recommandations]]
- [[2.5 IHM]]
- [[2.6 Versionning des éléments]]
- [[2.7 Datas Critiques]]
- [[2.8 Dédoublonnage]]
- [[2.9 Import/Export]]

[[3 Link Server]]

- [[3.1 Architecture]]
- [[3.2 IHM]]
- [[3.3 WS Read]]
- [[3.4 WS Post]]

[[4 API Java BO]]

- [[4.1 Général]]
- [[4.2 API Read]]
- [[4.3 API Post]]

[[5 API Java LS]]

- [[5.1 Général]]
- [[5.2 API Read]]
- [[5.3 API Post]]

[[6 Android]]

- [[6.1 Général]]
- [[6.2 Medias]]
- [[6.3 IHM]]
- [[6.4 Capteurs]]

[[7 Android Contribution]]

- [[7.1 Général]]
- [[7.2 Fonctions]]
- [[7.3 IHM]]

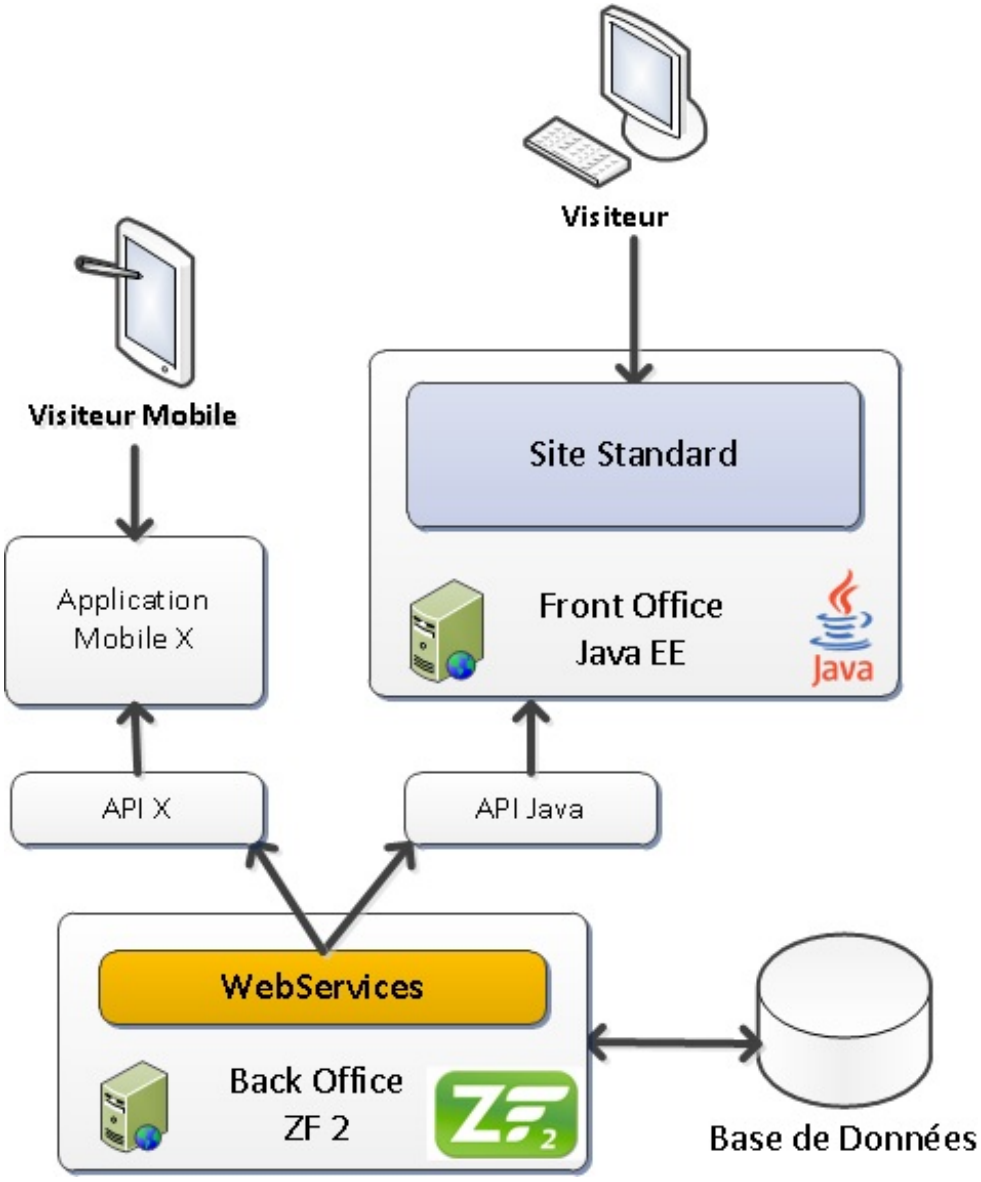
[[Home]] > [[Spécifications]] > [[1 BackOffice WebServices]]

[[Sommaire]]

[[1 BackOffice WebServices]]

Introduction

Les web services permettent d'exposer des objets métiers vers l'extérieur. Les principaux objets métiers sont les parcours et sous parcours, les scènes, et les éléments (médias ou artefact) liés aux scènes. Les WebServices seront utilisé à la fois par le Front Office Web et les applications mobiles :



Les WebServices seront en SOAP, protocole le plus répandu actuellement. Le Framework PHP utilisé pour le back office (Zend Framework 2), permet directement d'exposer des services (cf Zend/Soap/Server). C'est donc ce que nous utiliserons pour créer les WebServices. Dans l'ensemble de ce document, nous allons spécifier la réponse du WebService par un exemple. Ce sera moins formel qu'une définition des réponses au format XSD, mais plus simple à comprendre.

[[1.1 WS Auth]]

[[1.2 WS Read]]

[[1.3 WS Post]]

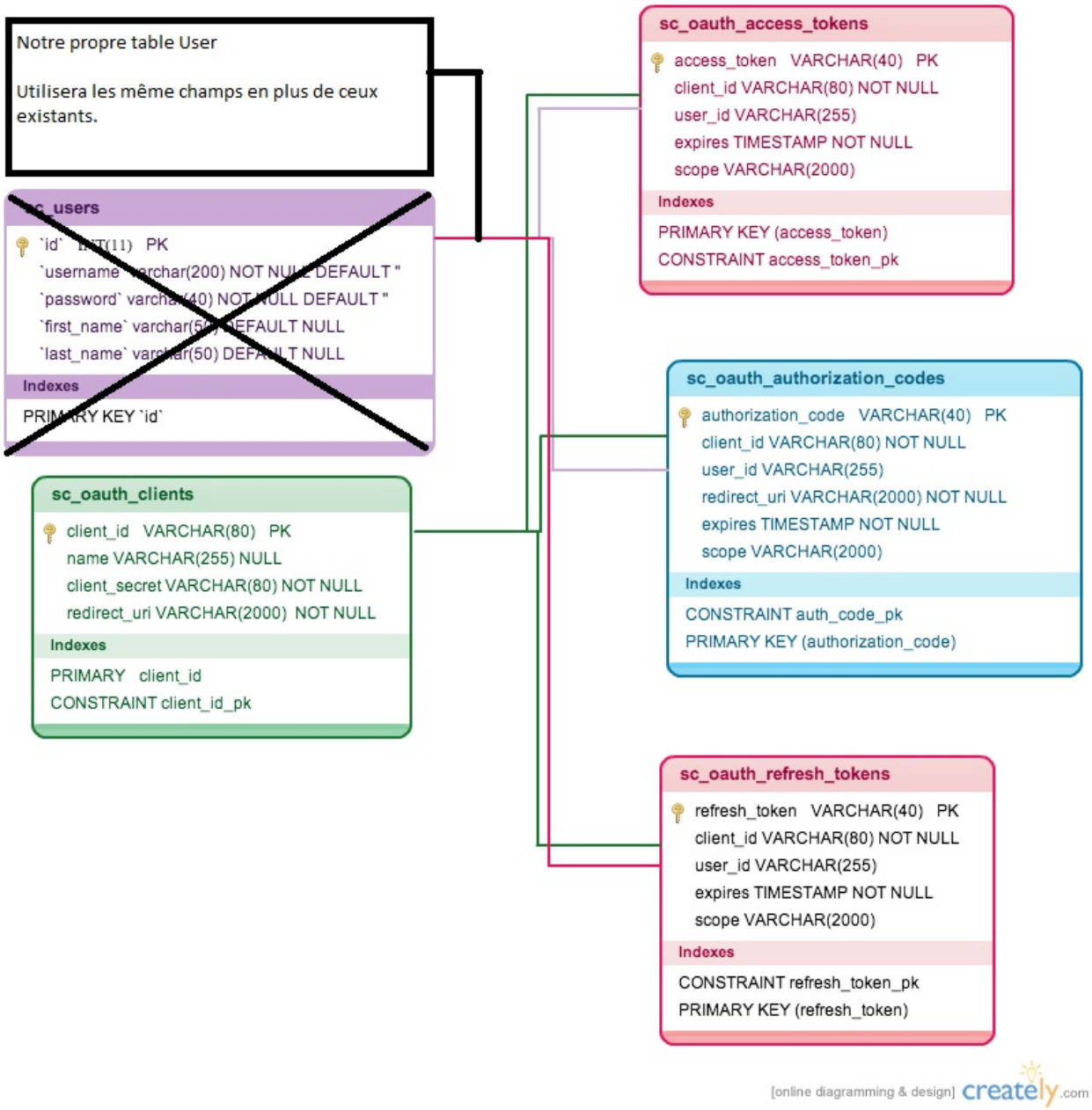
[[Home]] > [[Spécifications]] > [[1 BackOffice WebServices]] > [[1.1 WS Auth]]

[[Sommaire]]

[[1.1 WS Auth]]

1.1.1 Gestion des utilisateurs autorisés

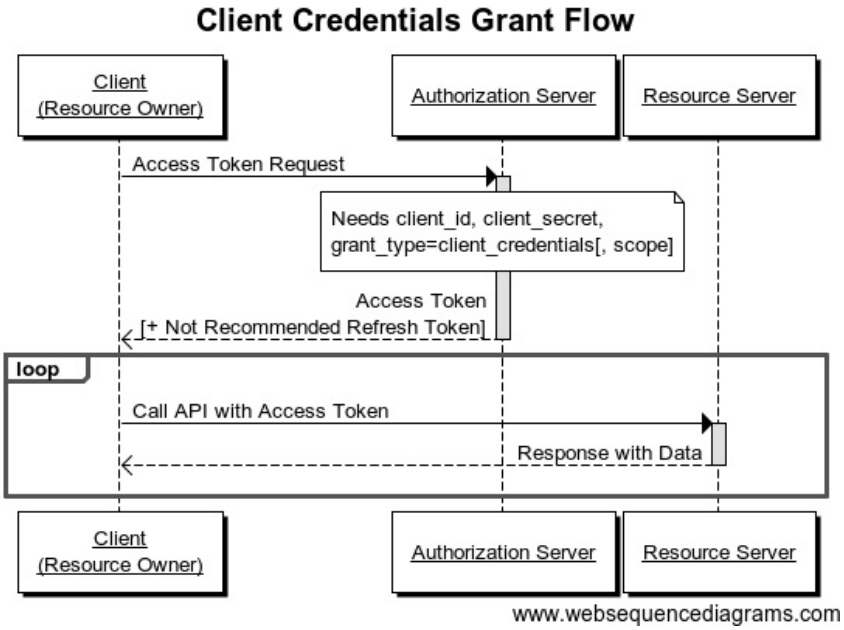
Le BackOffice doit être modifié afin de stocker des informations d'authentification pour les API. Nous utiliserons la base de données utilisateurs existante pour définir les utilisateurs autorisés à contribuer par le moyen des webservices POST. Aussi, nous stockerons les tokens d'accès et les tokens de refresh associés aux users. Le module d'authentification OAuth2 que nous implémenterons (voir WebServicesReadOnly.docx) met à disposition tout cela mais pose un problème concernant la table utilisateurs, le module est fait de façon à utiliser sa propre table d'users, nous allons donc le modifier pour utiliser notre propre table.



Un utilisateur pourra se voir attribué le rôle de Contributeur, un contributeur peut créer des éléments sur l'IHM du BackOffice ou en utilisant les webservices POST. Cependant chacune de ses contributions restera sous la forme d'un brouillon tant qu'un admin ne l'aura pas validé. De plus, les webservices de consultation ne permettrons pas d'exporter les éléments qui ne sont pas encore publiés, le contributeur ne pourra alors pas consulter un élément qu'il vient de créer (contribution) en utilisant les webservices GET.

1.1.2 : Implémentation Oauth2

Nous utilisons le protocole d'Authentication et d'Authorization : OAuth2. L'utilisateur s'authentifie grâce à l'échange de 'tokens' entre le client mobile et le BackOffice. Il utilise ensuite ce token pour être autorisé à requêter les ressources du server (les webservices dans notre cas). De cette façon, aucun mot de passe n'est stocké sur le client mobile, seulement des tokens valides durant 3600 secondes. Bien qu'il soit possible de séparer les services d'authentication, d'authorization et de ressources sur différents serveurs, ici le BackOffice assumera tous ces rôles. Le protocole OAuth2 dispose de différents modes pour l'authentication et l'authorization, nous utilisons ici le 'credentials grant type' parfaitement adapté pour l'échange de données entre deux serveurs développés par la même entité, le client est également le propriétaire des données. Cas standard



On a alors une unique requête d'identification à effectuer, celle-ci renvoie un JSON avec l'« access token » nécessaire pour être autorisé à effectuer des requêtes de webservices. Exemple de requête d'identification (avec HTTPie) :

```
http --auth testclient:testpass -f POST http://http://10.67.64.196/cervin/moving-BO/public/oauth grant_type=client_credentials
```

Cette requête renvoie le JSON suivant :

```
{
        "access_token":"03807cb390319329bdf6c777d4dfae9c0d3b3c35",

        "expires_in":3600,

        "token_type":"bearer",

        "scope":null
}
```

Pour implémenter cette structure, nous utiliserons le module zf-oauth2 créé par zf-campus pour Zend Framework: https://github.com/zfcampus/zf-oauth2 En y apportant les modifications nécessaires pour utiliser notre base de données utilisateurs déjà existante dans le BackOffice au lieu de celle proposée. Ce module utilise la librairie oauth2-server-php créée par Brent Shaffer. https://github.com/bshaffer/oauth2-server-php

Cas Limites

L'utilisateur peut ne pas avoir les droits nécessaires pour accéder à une certaine ressource, son niveau d'accès est insuffisant. L' « access token » peut avoir expiré, répéter le processus d'authentification produira un nouveau token pour le client.

Cas d'erreurs

L'utilisateur n'est pas connu par le BackOffice et donc pas autorisé

[[1.2 WS Read]]

Introduction

Cette partie regroupe les fonctions d'accès aux différents éléments de Movin. Nous appliquons la convention suivante : quand nous requêtons un élément (fonction getXXXById), nous retournons le détail de l'élément, et la liste non détaillé (juste Id et nom affichable), des sous éléments. Nous rappelons que la hiérarchie des éléments est la suivante :

Parcours -> Sous-Parcours -> Scene -> Element(Media/Artefact)

De plus, chacune des fonctions suivantes prend en paramètre un 'access token' qui permet d'autoriser la requête si le token correspond à un client précédemment authentifié et autorisé et si le token est toujours valide.

1.2.1 Fonction getListAllParcours

Méthode permettant de retourner tous les parcours « Public » du BackOffice. La méthode ne prend pas de paramètre.

Cas standard

Il y a au moins un parcours « Public » dans le BO. Le XML retourné sera de la forme :

```
<parcoursListe>
    <parcours id="456" value="L'histoire de l'informatique à Grenoble" />
    <parcours id="486" value="Mon Parcours 2" />
</parcoursListe>
```

Note : La méthode ne prend pas d'argument, car un Back Office est associé à un seul Front. Dans le cas d'un fonctionnement multi-tenant (ce qui n'est pas prévus à l'heure actuelle), cette méthode devrait être supprimée, et remplacée par une méthode getListParcoursByClientId par exemple.

Cas Limites

Il n'y a aucun parcours en visibilité « Public » dans le BO. La fonction renvoie une liste vide.

Cas d'erreurs

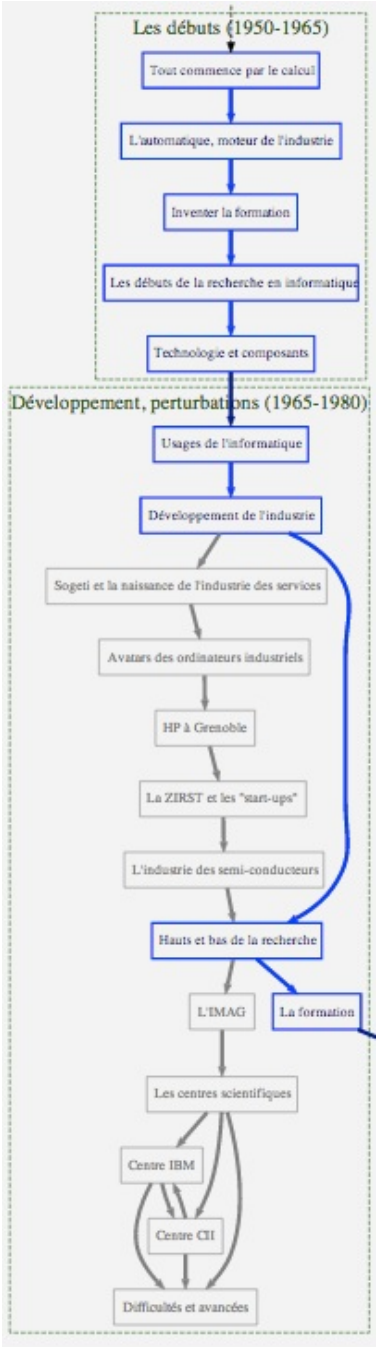
Il n'y a pas de cas d'erreur.

1.2.2 Fonction getParcoursArchitectureById

Méthode permettant de récupérer l'architecture d'un parcours. L'architecture d'un parcours correspond à sa décomposition en sous-parcours et scène, et les relations entre les éléments. La méthode prend en paramètre l'identifiant du parcours.

Cas Standard

Le parcours demandé existe dans la base de données, il est « Public » et possède au moins un sous parcours qui lui-



même contient au moins une scène. Prenons par exemple le parcours suivant :

Le XML reçu sera de la forme :

```
<!-- Exemple getParcoursArchitectureById -->
<parcours id="456" value="L'histoire de l'informatique à Grenoble"
firstSousParcoursId="12" >
    <sousParcours id="12" value="Les débuts (1950-1965)"
    firstSceneId="45" nextSousParcoursId="13">
        <scenes>
            <scene id="45" value="Tout commence par le calcul" recommandee="oui"/>
            <scene id="46" value="L'automatique, moteur de l'industrie" recommandee="oui"/>
            <scene id="47" value="Inventer la formation" recommandee="oui"/>
            <scene id="48" value="Les débuts de la recherche en informatique" recommandee="oui"/>
            <scene id="49" value="Technologie et composants" recommandee="oui"/>
        </scenes>
        <transitionsPrincipales>
            <transition id="236" from="45" to="46" semantiqueId="1" semantiqueName="Chronologique" />
            <transition id="237" from="46" to="47" semantiqueId="1" semantiqueName="Chronologique" />
            <transition id="238" from="47" to="48" semantiqueId="1" semantiqueName="Chronologique" />
            <transition id="239" from="49" to="50" semantiqueId="1" semantiqueName="Chronologique" />
        </transitionsPrincipales>
        <transitionsSecondaires />
    </sousParcours>
    <sousParcours id="13" value="Développement, perturbation (1965,1980)"
        firstSceneId="50"  nextSousParcoursId="" >
        <scenes>
            <scene id="50" value="Usages de l'informatique" recommandee="oui"/>
            <scene id="51" value="Développement de l'industrie" recommandee="oui" />
            <scene id="52" value="Sogeti et la naissance de l'insdutrie des services" recommandee="non"/>
            <scene id="53" value="Avatars des ordinateurs industriels" recommandee="non"/>
            <scene id="54" value="HP à Grenoble" recommandee="non"/>
            <scene id="55" value="La ZIRTS et les 'start-ups'" recommandee="non"/>
            <scene id="56" value="L'industrie des semi-conducteurs" recommandee="non"/>
            <scene id="57" value="Hauts et bas de la recherche" recommandee="oui"/>
            <scene id="58" value="L'IMAG" recommandee="non"/>
            <scene id="59" value="La formation" recommandee="oui"/>
            <scene id="60" value="Les centres scientifiques" recommandee="non"/>
            <scene id="61" value="Centre IBM" recommandee="non"/>
            <scene id="62" value="Centre CII" recommandee="non"/>
            <scene id="63" value="Difficultés et avancées" recommandee="non"/>
        </scenes>
        <transitionsPrincipales>
            <transition id="725" from="50" to="51" semantiqueId="1" semantiqueName="Chronologique" />
            <transition id="726" from="51" to="57" semantiqueId="1" semantiqueName="Chronologique" />
            <transition id="727" from="57" to="59" semantiqueId="1" semantiqueName="Chronologique" />
        </transitionsPrincipales>
        <transitionsSecondaires>
            <transition id="730" from="51" to="52" semantiqueId="1" semantiqueName="Chronologique" />
            <transition id="731" from="52" to="53" />
            <transition id="732" from="53" to="54" semantiqueId="1" semantiqueName="Chronologique" />
            <transition id="733" from="54" to="55" semantiqueId="1" semantiqueName="Chronologique" />
            <transition id="734" from="55" to="56" semantiqueId="1" semantiqueName="Chronologique" />
            <transition id="735" from="56" to="57" semantiqueId="1" semantiqueName="Chronologique" />
            <transition id="736" from="57" to="58" semantiqueId="1" semantiqueName="Chronologique" />
            ...
```

On ne renverra que les identifiants et noms, cette fonction permettra à afficher une vision globale du parcours, sans rentrer dans les détails.

Cas Limites

Le parcours ne contient aucun sous parcours ou un sous parcours ne contient aucune scène :

• Cela ne devrait pas arriver, un parcours ne contenant pas de sous parcours ou un sous parcours ne contenant pas de scène est probablement du au fait que le parcours est en construction, il n'aurait donc pas du être passé en « Public »

• Il convient donc de logger le problème et de renvoyer un message d'erreur à l'utilisateur. (Parcours incomplet).

Cas d'erreurs

L'id spécifié n'existe pas, ou n'est pas en visibilité public : retourner un message d'erreur à l'utilisateur (Not Found).

1.2.3 Fonction getParcoursById

Méthode permettant de récupérer les informations d'un parcours. On retournera toutes les informations du parcours, ainsi que la liste de ses sous-parcours. On ne détaillera pas les informations sur les sous-parcours, on se contentera de retourner l'id, le nom et les liens entre les sous-parcours.

Cas standard

Le parcours existe, il est en visibilité « Public » et contient au moins un sous parcours. Dans ce cas le XML retourné sera de la forme :

```
<!-- Exemple getParcoursById -->
<parcours id="456" value="L'histoire de l'informatique à Grenoble" firstSousParcoursId="12"
    description="Grenoble est l'un des principaux centres d'activité informatique..." >
    <sousParcours>
        <sousParcours id="12" value="Les débuts (1950-1965)" nextSousParcoursId="13"  />
        <sousParcours id="13" value="Développement, perturbation (1965,1980)"  nextSousParcoursId="14"   />
        <sousParcours id="13" value="Changement de visage (1980-1995)"  nextSousParcoursId=""   />
    </sousParcours>
</parcours>
```

Cas Limites

Le parcours ne contient aucun sous parcours : • Cela ne devrait pas arriver, un parcours ne contenant pas de sous parcours est probablement du au fait que le parcours est en construction, il n'aurait donc pas du être passé en « Public » • Il convient donc de logger le problème et de renvoyer un message d'erreur à l'utilisateur. (Parcours incomplet).

Cas d'erreurs

L'id spécifié n'existe pas, ou n'est pas en visibilité public : retourner un message d'erreur à l'utilisateur (Not Found).

1.2.4 Fonction getSousParcoursById

Méthode permettant de récupérer les informations d'un sous-parcours. On retournera toutes les informations du sous-parcours, ainsi que la liste de ses scènes. On ne détaillera pas les informations sur les scènes, on se contentera de retourner l'id, le nom et les liens entre les scènes.

Cas standard Le sous-parcours existe, le parcours auquel il appartient est en visibilité « Public » et contient au moins une scène. Dans ce cas le XML retourné sera de la forme :

```xml
<!-- Exemple getSousParcoursById -->
<sousParcours id="13" value="Développement, perturbation (1965,1980)"  parcoursId="456"
    firstSceneId="50"  nextSousParcoursId="" >
    <scenes>
        <scene id="50" value="Usages de l'informatique" recommandee="oui"/>
        <scene id="51" value="Développement de l'industrie" recommandee="oui" />
        <scene id="52" value="Sogeti et la naissance de l'insdutrie des services" recommandee="non"/>
        <scene id="53" value="Avatars des ordinateurs industriels" recommandee="non"/>
        <scene id="54" value="HP à Grenoble" recommandee="non"/>
        <scene id="55" value="La ZIRTS et les 'start-ups'" recommandee="non"/>
        <scene id="56" value="L'industrie des semi-conducteurs" recommandee="non"/>
        <scene id="57" value="Hauts et bas de la recherche" recommandee="oui"/>
        <scene id="58" value="L'IMAG" recommandee="non"/>
        <scene id="59" value="La formation" recommandee="oui"/>
        <scene id="60" value="Les centres scientifiques" recommandee="non"/>
        <scene id="61" value="Centre IBM" recommandee="non"/>
        <scene id="62" value="Centre CII" recommandee="non"/>
        <scene id="63" value="Difficultés et avancées" recommandee="non"/>
    </scenes>
    <transitionsPrincipales>
        <transition id="725" from="50" to="51" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="726" from="51" to="57" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="727" from="57" to="59" semantiqueId="1" semantiqueName="Chronologique" />
    </transitionsPrincipales>
    <transitionsSecondaires>
        <transition id="730" from="51" to="52" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="731" from="52" to="53" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="732" from="53" to="54" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="733" from="54" to="55" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="734" from="55" to="56" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="735" from="56" to="57" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="736" from="57" to="58" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="737" from="58" to="60" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="725" from="60" to="61" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="738" from="60" to="62" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="739" from="60" to="63" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="740" from="61" to="62" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="741" from="61" to="63" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="742" from="62" to="61" semantiqueId="1" semantiqueName="Chronologique" />
        <transition id="743" from="62" to="63" semantiqueId="1" semantiqueName="Chronologique" />
    </transitionsSecondaires>
</sousParcours>
```

Cas Limites

Le sous-parcours ne contient aucune scène: • Cela ne devrait pas arriver un sous-parcours ne contenant pas de scène est probablement dû au fait que le parcours est en construction, il n'aurait donc pas du être passé en « Public » • Il convient donc de logger le problème et de renvoyer un message d'erreur à l'utilisateur. (Parcours incomplet).

Cas d'erreurs

L'id spécifié n'existe pas, l'id du sous parcours ne correspond pas à un sous parcours dont le parent (le parcours) est public : retourner un message d'erreur à l'utilisateur (Not Found).

1.2.5 Fonction getSceneById

Méthode permettant de récupérer les informations d'une scène. On retournera toutes les informations de la scène, ainsi que la liste des transitions, des médias et des artefacts (en visibilité public) lié à cette scène. On ne détaillera pas les informations sur les médias et les artefacts, seulement les ids, les noms et les types (id et nom).

Cas standard

La scène existe et le parcours auquel elle appartient est en visibilité « Public ». Dans ce cas le XML retourné sera de la forme :

```
<!-- Exemple getSceneById -->
<scene id="50" titre="Tout commence par le calcul" recommandee="oui" sousParcoursId="13" >
    <narration>
        L'histoire de l'informatique à Grenoble commence...
    </narration>
    <artefacts>
        <artefact id="12" value="Jean Kutzmann" typeId="6" typeName="Personne" />
        <artefact id="13" value="Gamma 3" typeId="7" typeName="Matériel" />
    </artefacts>
    <medias>
        <media id="18" value="Photo calculateur" typeId="11" typeName="Image" />
        <media id="22" value="Video salle serveur" typeId="12" typeName="Video" />
    </medias>
    <transitionPrincipaleIn id="45" from="12" semantiqueId="1" semantiqueName="Chronologique"
        narration="Vers Tout commence par le calcul" />
    <transitionPrincipaleOut id="46" to="18" semantiqueId="1" semantiqueName="Chronologique"
        narration="Vers Développement de l'industrie" />
    <transitionSecondairesIn>
        <transition id="63" from="10" semantiqueId="1" semantiqueName="Chronologique"
            narration="Vers Tout commence par le calcul" />
        <transition id="67" from="8" semantiqueId="1" semantiqueName="Chronologique"
            narration="Vers Tout commence par le calcul" />
    </transitionSecondairesIn>
    <transitionSecondairesOut>
        <transition id="78" to="23" semantiqueId="1" semantiqueName="Chronologique"
            narration="Vers Difficultés et avancées" />
    </transitionSecondairesOut/>
</scene>
```

Cas d'erreurs

L'id spécifié n'existe pas, l'id de la scène ne correspond pas à une scène dont le parcours est public : retourner un message d'erreur à l'utilisateur (Not Found).

1.2.6 Fonction getTransitionById

Méthode permettant de récupérer les informations d'une transition. On retournera toutes les informations de la transition, ainsi que le lien vers la scène de départ et celle d'arrivée.

Cas standard La transition existe, et les 2 scènes qu'elle lie sont Public. Le XML retourné sera de la forme :

```
<!-- Exemple getTransitionById -->
<transition id="730" type="recommandee" estInterSousParcours="false"
    narration="Vers les usages de l'informatique" semenatiqueId="1" semantiqueNom="Chronologique" >
    <sceneDepart id="50" value="Usages de l'informatique" recommandee="oui"/>
    <sceneArrivee id="51" value="Développement de l'industrie" recommandee="oui" />
</transition>
```

Cas d'erreurs

L'id spécifié n'existe pas, une des scènes liée (ou les deux), ne sont pas public: retourner un message d'erreur à l'utilisateur (Not Found).

1.2.7 BackOffice WebServices \ WS Read \ Fonction getElementById/getMediaByID/getArtefactById

Méthode getMediaById

Méthode permettant de récupérer les informations d'un média. On retournera toutes les informations du média, ainsi que la liste de des médias et des artefacts (en visibilité public) auquel il est lié. On ne détaillera pas les informations sur les médias et les artefacts liés, seulement les ids, les noms et les types (id et nom).

Cas standard Le média existe, il est en visibilité « Public », et il contient au moins une donnée. Dans ce cas le XML retourné sera de la forme :

```
<!-- Exemple getMediaById -->
<media id="18" value="Photo calculateur" typeId="11" typeName="Image" >
    <description>Photo du premier calculateur prise par... </description>
    <datas>
        <data type="fichier" nom="Photo calculateur" URL="/media/images/12154545.jpeg"
            type="image/jpeg" nomType="Jpeg" longueur="300px" hauteur="200px" description="" />
    </datas>
    <linkedMedias>
        <media id="22" value="Video salle serveur" typeId="12" typeName="Video" />
    </linkedMedias>
    <linkedArtefats>
        <artefact id="12" value="Jean Kutzmann" typeId="6" typeName="Personne">
    </linkedArtefats>
</media>
```

Cas Limites

Le média ne contient aucune donnée :

• Cela ne devrait pas arriver, un media contient normalement 1 donnée (il peut en contenir plusieurs, mais ce ne sera pas courant). Le média est donc probablement est en construction, il n'aurait donc pas du être passé en « Public »

• Il convient donc de logger le problème et de renvoyer un message d'erreur à l'utilisateur. (Media incomplet).

Cas d'erreurs L'id spécifié n'existe pas, ou le média n'est pas public : retourner un message d'erreur à l'utilisateur (Not Found).

Méthode getArtefactById

Méthode permettant de récupérer les informations d'un artefact. On retournera toutes les informations de l'artefact, ainsi que la liste de des médias et des artefacts (en visibilité public) auquel il est lié. Le relation entre artefact ayant un sens on précisera si l'artefact en question est l'origine ou la destination, ainsi que sa sémantique (id et nom). On ne détaillera pas les informations sur les médias et les artefacts liés, seulement les ids, les noms et les types (id et nom).

Cas standard

L'artefact existe et il est en visibilité « Public ». Un artefact ne contient pas forcement de données, une description peut très bien suffire. Dans ce cas le XML retourné sera de la forme :

```
<!-- Exemple getArtefactById -->
<artefact id="12" value="Jean Kutzmann" typeId="6" typeName="Personne">
    <description> Jean Kutzmann fut un mathématicien... </description>
    <datas>
        <data type="date" nom="Date de décès" value="12/12/2012" format="dd/mm/yyyy"
            description="La date de décès de la personne" />
            <data type="date" nom="Date de naissance" value="12/06/2012" format="dd/mm/yyyy"
            description="La date de naissance de la personne" />
        <data type="geoposition" nom="Lieu de résidence" description="Lieux de résidence de la personne"
            latitude="12.5" longitude="15.3" adresse="3 rue de la Paix, 38000 Grenoble" />
    </datas>

    <!-- Cet artefact est l'artefact d'origine de la relation : -->
    <linkedOrigineArtefats>
        <link semantiqueId="12" semantiqueName="A pour chef">
            <!-- cela veut dire que Jean Kutzmann a Brad Pitt pour chef -->
            <aretact id="18" value="Brad Pit" typeId="6" typeName="Personne" />
        </link>
    </linkedOrigineArtefats>
    <linkedDestinationArtefats>
        <link semantiqueId="12" semantiqueName="A pour chef">
            <!-- cela veut dire que Ambroise croisat a Jean Kutzmann pour chef -->
            <aretact id="19" value="Ambroise Croizat" typeId="6" typeName="Personne" />
        </link>
    </linkedDestinationArtefats>
    <linkedMedias>
        <media id="18" value="Photo calculateur" typeId="11" typeName="Image" />
        <media id="22" value="Video salle serveur" typeId="12" typeName="Video" />
    </linkedMedias>
</artefact>
```

Cas d'erreurs

L'id spécifié n'existe pas, ou l'artefact n'est pas public : retourner un message d'erreur à l'utilisateur (Not Found).

1.2.8 Documentation WSDL

Une description des WebServices au format WSDL devra être générée automatiquement. Il servira de contrat sur l'utilisation des services entre le client (les APIs), et le serveur. Il servira aussi a générer plus rapidement les différentes APIs.

[[Home]] > [[Spécifications]] > [[1 BackOffice WebServices]] > [[1.3 WS Post]]

[[Sommaire]]

[[1.3 WS Post]]

1.3.1 Fonction Post Media (Lié à rien / à une scène / à un artefact)

Cette méthode permet d'envoyer un nouveau média au BackOffice. Elle prend en paramètre un objet média, l'access token (obligatoire) et deux paramètres optionnels : l'ID de la scène associée au média et l'ID de l'artefact associé au media. L'access token correspond à un user, si ce dernier y est autorisé alors la contribution (media) est stockée et associée avec l'ID du contributeur.

Cas Standard

L'utilisateur dispose des droits de contribution, l'access token est valide, les données du média sont complètes et l'ID de la scène (si fourni) correspond à une scène existante. Le retour est alors de ce type :

Cas Limites

Dans le cas d'une erreur connue, le message de la réponse de la méthode sera adapté. On aura par exemple un retour du type : Si le média est incomplet : > > > > >

Si l'access token est invalide : > > > > >

Si la scène parente précisée n'existe pas : > > > > >

Si l'user n'a pas les droits : > > > > >

Cas d'erreurs

Si un des paramètres obligatoires n'est pas fourni ou de format invalide.

1.3.2 Creation de Parcours

[[Sommaire]]

[[2 BackOffice Modifications]]

[[2.1 I18n]]

[[2.2 Historique Consultation]]

[[2.3 Historique Contribution]]

[[2.4 Recommandations]]

[[2.5 IHM]]

2.5.1 BackOffice Modifications \ IHM \ Afficher une miniature du parcours sur chaque page

[[2.6 : Versionning des éléments]]

2.6.1 : BackOffice Modifications \ Versioning des éléments \ Architecture BDD

2.6.2 : BackOffice Modifications \ Versioning des éléments \ Modification "Core"

2.6.3 : BackOffice Modifications \ Versioning des éléments \ Modification IHM BO

2.6.4 : BackOffice Modifications \ Versioning des éléments \ Modification WS

[[2.7 : Datas Critiques]]

2.7.1 : BackOffice Modifications \ Datas Critiques \ Modification BDD

2.7.2 : BackOffice Modifications \ Datas Critiques \ Modification "Core"

2.7.3 : BackOffice Modifications \ Datas Critiques \ Modification IHM Creation de type

2.7.4 : BackOffice Modifications \ Datas Critiques \ Modification IHM Creation de données (Rajout Warning si non remplie)

[[2.8 : Dédoublonnage]]

2.8.1 : BackOffice Modifications \ Dédoublonnage \ Modification BDD

2.8.2 : BackOffice Modifications \ Dédoublonnage \ Modification "Core" => fonction de déboublonnage

2.8.3 : BackOffice Modifications \ Dédoublonnage \ Modification IHM

[[2.9 : Import/Export]]

2.9.1 : BackOffice Modifications \ Import / Export \ WS Export

2.9.2 : BackOffice Modifications \ Import / Export \ WS Import (avec dédoublonage))

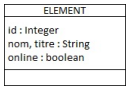
[[2.1 I18n]]

2.1.1 Architecture BDD

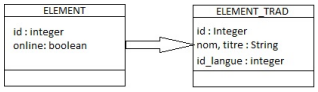
Pour pouvoir intégrer le changement de langue à l'application, il est nécessaire de modifier les classes pour qu'elles prennent en paramètre la langue sélectionnée.

La solution choisie est, pour chaque classe dont les attributs changent en fonction de la langue, nous créons un clone de cette classe contenant les attributs en question plus un attibut id_langue.

Exemple : La classe Element

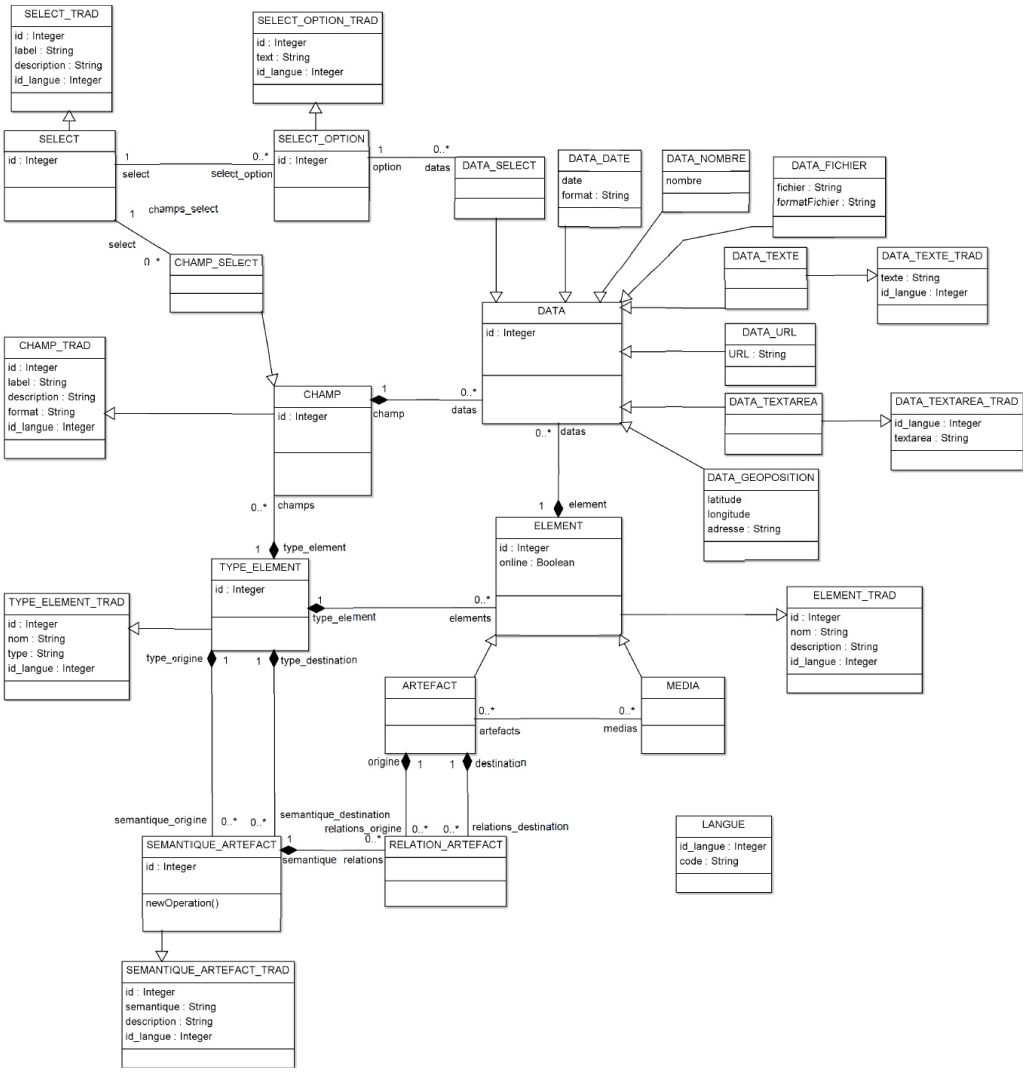


Devient :



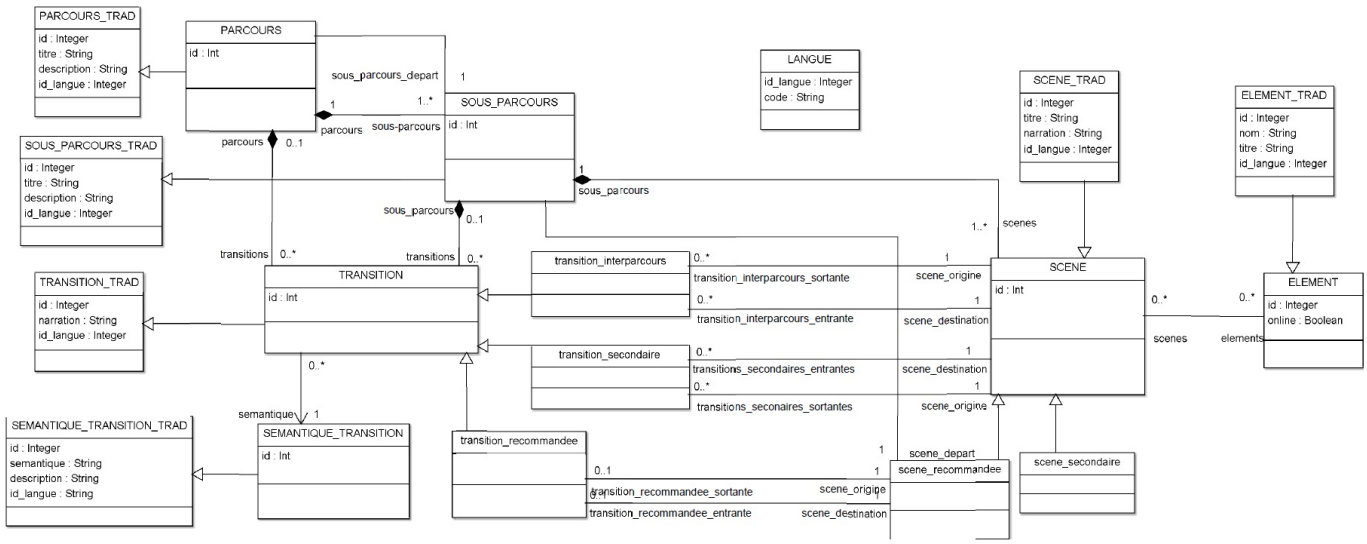
Cela permet de conserver un seul id par media, plutôt que d'avoir des id différents pour un même media existant en plusieurs langues.

Nouvelle modélisation de la collection :



La classe **LANGUE** est reliée aux classes **SELECT*TRAD*, *SELECTOPTION*TRAD, DATA*TEXTE*TRAD, DATA*TEXTEAREA*TRAD, ELEMENT*TRAD*, *SEMANTIQUE*ARTEFACT*TRAD*, *CHAMP*TRAD** et **TYPE*ELEMENT*TRAD**.

Nouvelle modélisation du parcours :



La classe **LANGUE** est reliée aux classes **PARCOURS*TRAD, SOUS*PARCOURS*TRAD, TRANSITION*T RAD, SEMANTIQUE*TRANSITION*T RAD, SCENE_T RAD** et **ELEMENT_T RAD**.

2.1.2 Modification "Core"

A l'inscription, l'utilisateur devra choisir sa langue, cela permettra d'afficher uniquement du contenu dans la langue correspondante. A tout moment, l'utilisateur doit être capable de changer sa langue.

Un administrateur, peut ajouter un artefact, un média, une sémantique transition ou un parcours. Dans les 4 cas, il devra sélectionner la langue dans laquelle il souhaite enregistrer ce contenu.

Un utilisateur lamba peut ajouter un média via un bouton contribution. La aussi, il devra sélectionner la langue d'enregistrement de son média.

2.1.3 Modification IHM BO

Il faut donc rajouter : * Un champ de sélection de la langue sur la page d'inscription. Langue choisie parmis la liste des langues disponibles. (Page Login, onglet Inscription : cervin/moving-BO/user/register) * Rajouter l'affichage de la langue et rendre possible la modification. (Page Mon compte : cervin/moving-BO/public/user) * Rajouter une liste de langue pour la création d'un artefact (Page Création d'un artefact : cervin/moving-BO/public/element/ajouter/artefact) * Rajouter une liste de langue pour la création d'un média (Page Création d'un média : cervin/moving-BO/public/element/ajouter/media) * Rajouter une liste de langue pour la création d'un parcours (Page Création d'un parcours : cervin/moving-BO/public/parcours/ajouter) * Rajouter une liste de langue pour la création d'une sémantique transition (Page Création d'un parcours : cervin/moving-BO/public/semantiquetransition/ajouter)

2.1.4 Modification WS

Les fonctions de webservices devront prendre en paramètre la langue de l'user et retourner la ressource dans la langue demandée.

Deux cas:

1: Aucun paramètre de langue n'est spécifié alors on récupère la langue du profil de l'user.

2: Le paramètre de langue est spécifié alors on renvoie la langue demandée.

[[Home]] > [[Spécifications]] > [[2 BackOffice Modifications]] > [[2.2 Historique Consultation]]

[[Sommaire]]

[[2.2 Historique Consultation]]

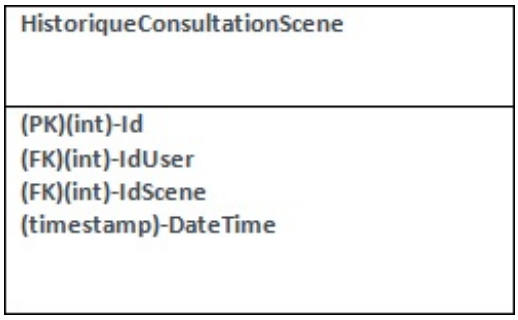
2.2.1 Architecture BDD

On veut pouvoir garder une trace chaque fois qu'une requête est effectuée contre les webservices. Pour les services GET, seules les scènes consultées nous intéressent pour pouvoir recommander des scènes en relation avec les scènes déjà visitée.

On stockera alors, pour chaque requête, dans une nouvelle table d'historique :

Id User, id Scene, timestamp

Schéma :



2.2.2 Modification WS

Les fonctions de webservices qui retournent une scène doivent enregistrer une trace de cette consultation dans la table d'historique.

2.2.3 Creation WS getHistoriqueByUserId (+modif WSDL)

Méthode getHistoriqueByUserId

Cette méthode prend en paramètre un ID User et retournera l'historique de consultation pour cet utilisateur. Cette fonction prend également un token d'accès, un admin peut récupérer l'historique de n'importe quel user, tandis d'un user basique peut retourner uniquement son historique.

Exemple de retour : > > > > >

[[Home]] > [[Spécifications]] > [[2 BackOffice Modifications]] > [[2.3 Historique Contribution]]

[[Sommaire]]

[[2.3 Historique Contribution]]

2.3.1 Modifications BDDMedia/Core: Validation de contribution

Pour les services POST, on veut connaître les médias créés par le contributeur. Le modèle actuel de la base de données nous permet de faire ceci sans ajouter de table, la table média possède un attribut IdUser c'est l'Id du créateur.

Nous pouvons donc utiliser ce modèle pour avoir l'historique des contributions. Il suffira de créer une fonction getMediaByUserId qui retournera les médias créés par un utilisateur en spécifié.

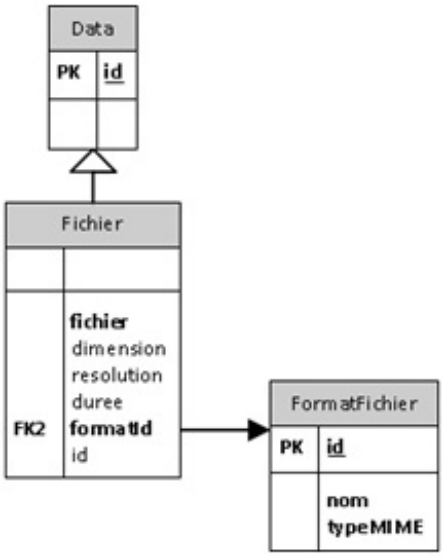
2.3.2 Correction BO Modification Media (utilisateur_id non renseigné)

La fonction de contribution PostMedia devra stocker l'ID du créateur associé au média. (Voir Webservices POST)

Pour le type de fichier, nous aurons besoin de stocker plus d'information, en fonction du type du fichier. Ces informations permettront d'afficher correctement le fichier au client. Nous rappelons que le front office devra permettre d'afficher les médias de type suivants : * Images (type a définir, mais au moins jpeg et png) * Vidéo (type à définir) * Audio (type à définir mais au moins mp3) * PDF

Nous pensons avoir besoin des éléments suivants pour l'affichage (liste non exhaustive) : * La durée (pour de l'audio, de la vidéo) * Les dimensions (pour de la vidéo, des images) * Une résolution (pour de la vidéo ? vraiment utile ?)

Pour le format de fichier, il sera préférable de créer une table séparée avec les différents types de fichier recensé. Au final le fichier devra être représenté dans la base de données de manière ci-contre. Au niveau des changements à réaliser dans l'interface d'administration, il faudra modifier la fonction d'upload afin de d'extraire les métadonnées (durée, dimensions, résolutions…) des fichiers. Actuellement, seule le type MIME est détecté en fonction de l'extension du fichier.



2.3.3 Creation WS getMediaByUserId (+modif WSDL)

Méthode getMediasByUserId Cette méthode permettra de récupérer tous les médias associés à un user. Elle prend en paramètre uniquement le token d'accès. Exemple de retour : > > > > >

[[2.4 Recommandations]]

2.4.1 Algorithme de recommandation (en fonction d'historique)

On veut pouvoir recommander des scènes à un user en fonction de son historique de consultation. La méthode (algo) reste à définir.

2.4.2 Creation WS getRecommandedSceneByUserId

Méthode getRecommandations Cette méthode prend en paramètre un access token qui permet d'identifier l'user et retourne la liste des recommandations de cet user.

Exemple de retour : > > > > >

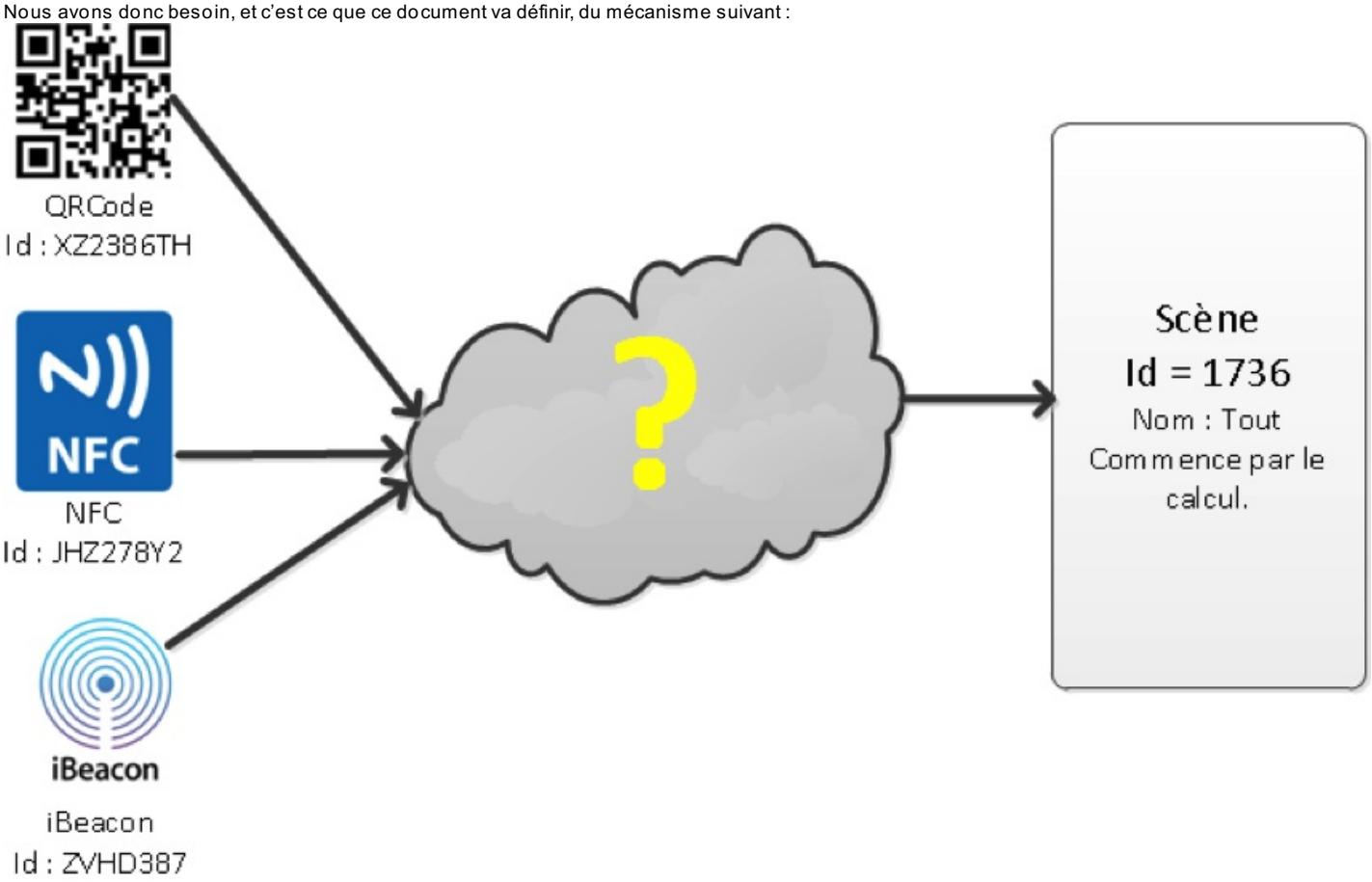
[[Home]] > [[Spécifications]] > [[3 Link Server]]

[[Sommaire]]

[[3 Link Server]]

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.

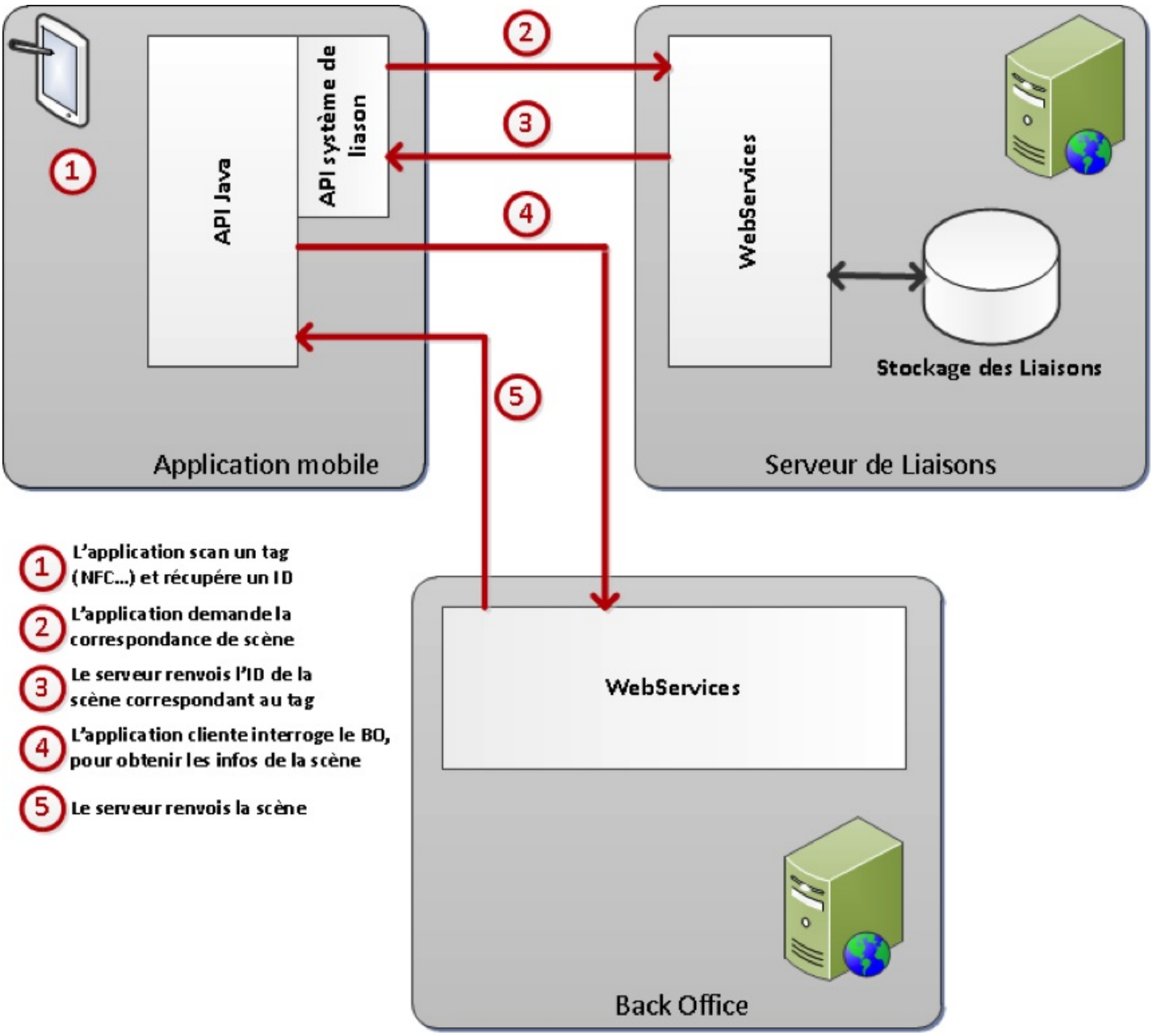
Nous avons donc besoin, et c'est ce que ce document va définir, du mécanisme suivant :



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 :



Cette partie 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.

[[3.1 Architecture]]

[[3.2 IHM]]

[[3.3 WS Read]]

[[3.4 WS Post]]

[[Home]] > [[Spécifications]] > [[3 Link Server]] > [[3.1 Architecture]]

[[Sommaire]]

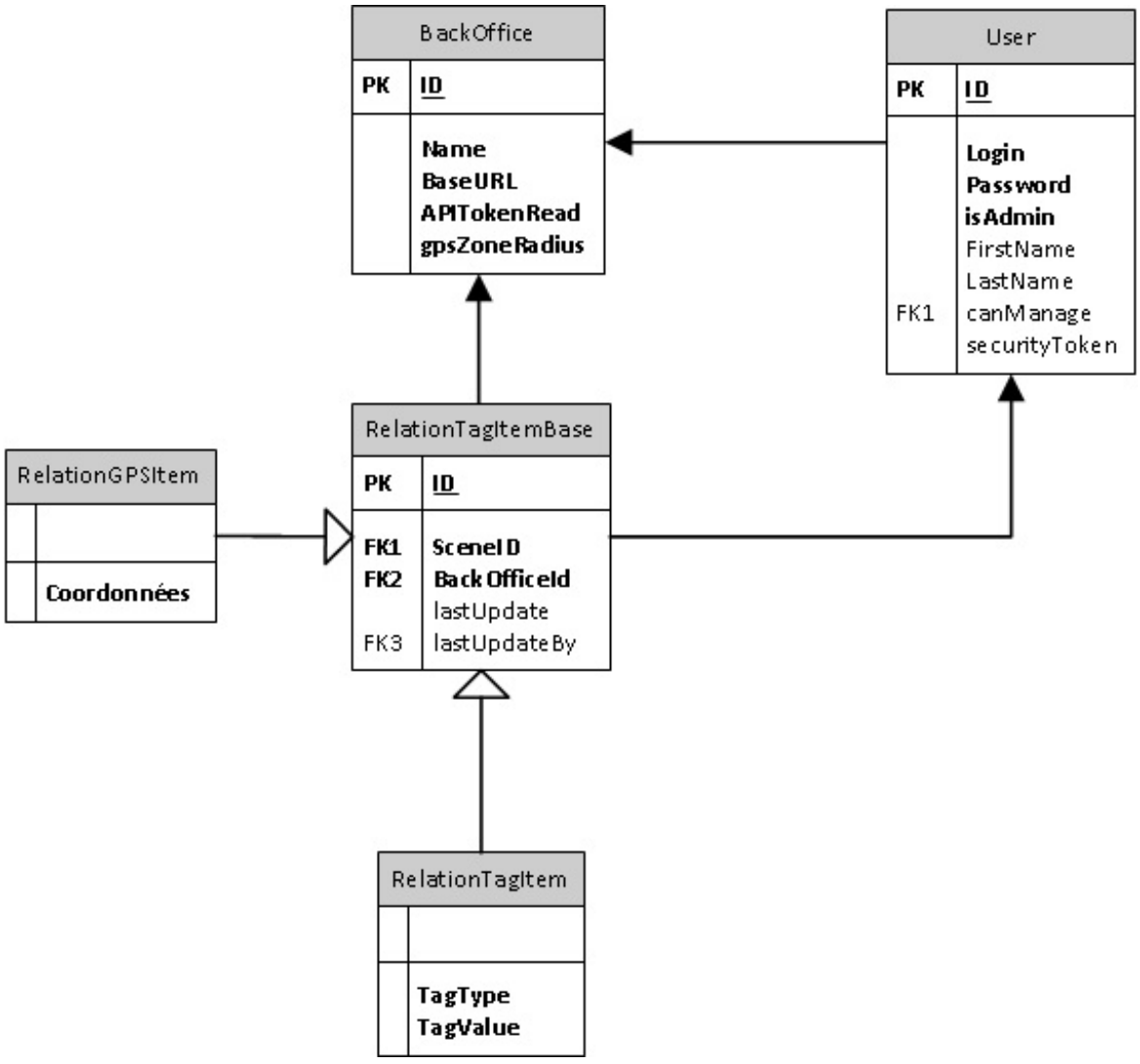
Nous avons choisis d'utiliser la technologie Java JEE afin de réaliser le serveur de lien. Nous avons choisie cette technologie car nous la maitrisons bien, qu'elle est open source, très répandue, et qu'il est facile de réaliser des WebServices. Afin de faciliter le développement et de rendre le code 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.

[[3.1 Architecture]]

3.1.1 Architecture

La base de données

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.



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.

Entité	Champs	Valeur	Description
			Ce n'est pas une vrai FK, car l'entité

RelationTagItemBase	SceneID	Clef étrangère	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)	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
BackOffice	gpsZoneRadius	int	Rayon (en metre) qui définit la zone autour d'un point correspondant à une scène
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 PostgreSQL. En effet, le choix initial s'était porté initialement vers une base MySQL, plus répandu (et également utilisé par le BackOffice). Cependant l'extension géographique de MySQL (MySQL GIS) est assez insuffisante et mal documentée. Contrairement à l'extension de PostgreSQL, PostGIS. Nous utiliserons les dernières versions : PostgreSQL 9.3 et PostGIS 2.1.

Le coeur de l'application

Le coeur 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

[[3.2 IHM]]

Nous avons séparer l'IHM par sa complexité. Dans la partie générale, nous avons regroupé toutes les interfaces classique (pouvant souvent être généré en fonction des données). Nous avons séparé la partie GPS, car c'est une partie complexe qui nécessitera l'affichage de maps (Surement OpenStreetMap), et la selection de coordonnées sur une carte.

3.2.1 Interface Web Générale

Le site n'aura pas d'interface publique. L'utilisateur devra toujours être authentifié.

Interface Standard

L'interface Standard regroupe toutes les fonctionnalités que pourra réaliser un utilisateur standard (partie GPS exclue).

L'utilisateur pour gérer les liens de son BackOffice. (Un utilisateur est lié à 1 et 1 seul BackOffice : cf. [[3.1 : Architecture]]).

Gestion des liens

Ajouter un lien (tag ==> scène)

Une interface permettant de saisir l'id de la scène, le type de tag (liste déroulante proposant NFC/iBeacon/QRCode), et l'id du tag.

Si le couple type tag / id tag existe déjà, une erreur sera affiché, lui demandant s'il souhaite afficher le lien.

Amélioration : A terme on pourra proposer une recherche de scène plutôt que de faire saisir l'id de la scène. Cependant cela serait compliqué à faire, notamment à cause du fonctionnement multi-tenant (cela voudrait dire que le LinkServer devrait posseder un compte sur chaque BackOffice, et qu'il se connecte au bon pour proposer une liste de scène).

Modifier un lien (tag ==> scène)

Après avoir sélectionné un lien, un utilisateur pourra le modifier. L'interface sera identique à celle de l'ajout.

Supprimer un lien (tous)

Un utilisateur pourra, après avoir sélectionné un lien, le supprimer.

Gestion de ses informations

Modifier ses informations

Une interface permettra à l'utilisateur de modifier son nom / prénom.

Regénérer son token

Un lien permettra de regénéré (aléatoirement) son token. Le token généré remplacera l'ancien. Cela veut dire que l'ensemble des applications de l'utilisateur ne pourra plus se connecter au LinkServer avant que celui-ci ne change leurs token.

Interface Administrateur

L'interface Administrateur ajoute des fonctionnalités à l'Interface Standard. L'administrateur pourra généré les lien de l'ensemble des BackOffice.

Gestion des BackOffices

Ajouter un BackOffice

Modifier un BackOffice

Supprimer un BackOffice

Gestion des utilisateurs

Ajouter un utilisateur

Modifier un utilisateurs

Supprimer un utilisateurs

Regénérer le token d'un utilisateur

3.2.2 Interface Web GPS V1

3.2.3 Interface Web GPS V2

[[3.3 WS Read]]

Chaque WebService Read est identifié par un token. Il y a un token par BackOffice. C'est une méthode simple d'authentification (l'authentification sera transparente pour l'utilisateur), mais qui n'est pas fiable a 100% (le token stocké sur l'application mobile pourra être récupéré par un utilisateur mal intentionné pour l'utiliser sur une autre application). Mais cela nous semble suffisant, car récuperer ce token permet simplement l'accès en lecture des WebServices. (Et s'il l'on constate une utilisation malveillante d'un service, il sera possible de révoquer le token en question et de mettre a jour les applications).

3.3.1 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)

3.3.2 Fonction getSceneByGPSCoord / getCloseSceneByGPSCoord (V1)

Dans cette première version (V1), la zone géographique sera simplifiée; on définira une zone comme un point central et un rayon.

Fonction getSceneByGPSCoord V1

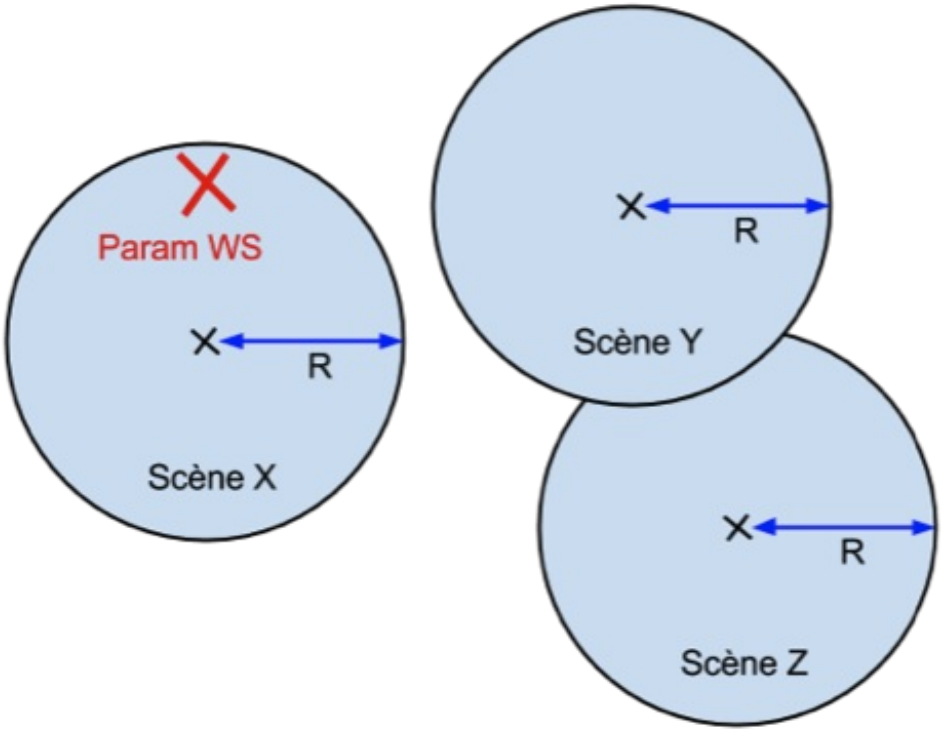
Fonction simple permettant de retourner la scène correspondant à une coordonnées géographique s'il y en a une. (Eventuellement les scènes, dans le cas ou 2 scènes se chevauchent géographiquement).

Cette fonction prend en paramètre : * Un TokenAPIRead (correspondant au BO) * Une coordonnée GPS (latitude, longitude)

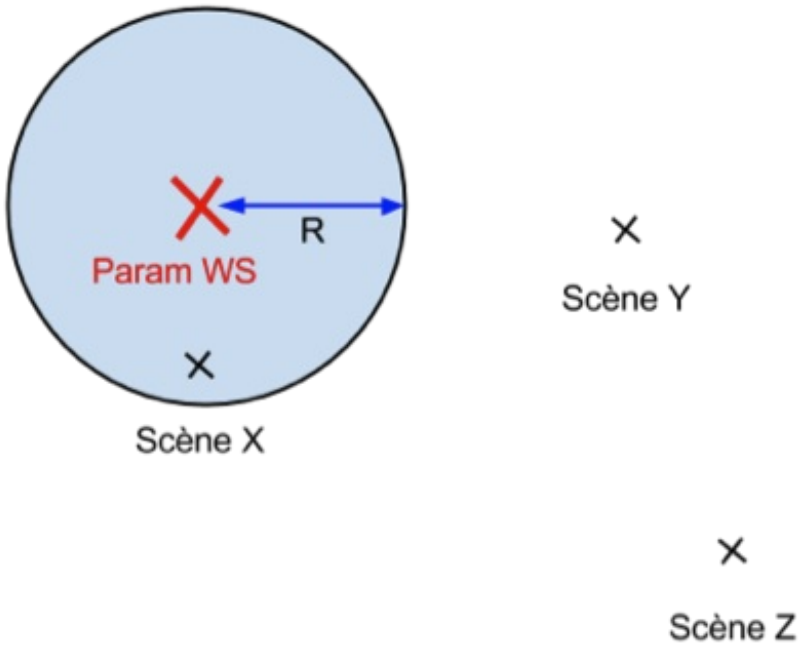
Pour déterminer si une scène appartient ou non à une coordonnée GPS, on utilisera les coordonnées du point stocké dans la BDD et un rayon R qui sera une constante liée au BackOffice (cf. gpsZoneRadius [[3.1 : Architecture]]) qui correspond à un rayon de tolérance. En résumé, le WebService retournera toutes les scènes dont le point passé en paramètre est à moins de R distance des centres.

Pour cela on utilisera la fonction PostGIS ST_Distance.

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)

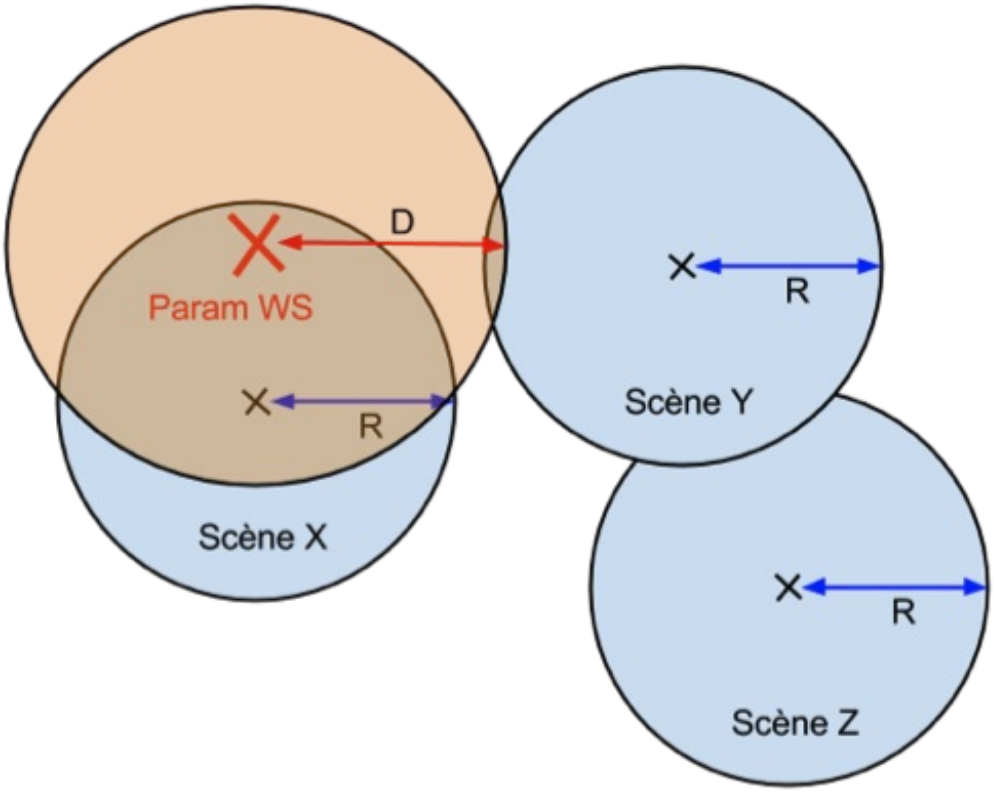
Fonction getCloseSceneByGPSCoord V1

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

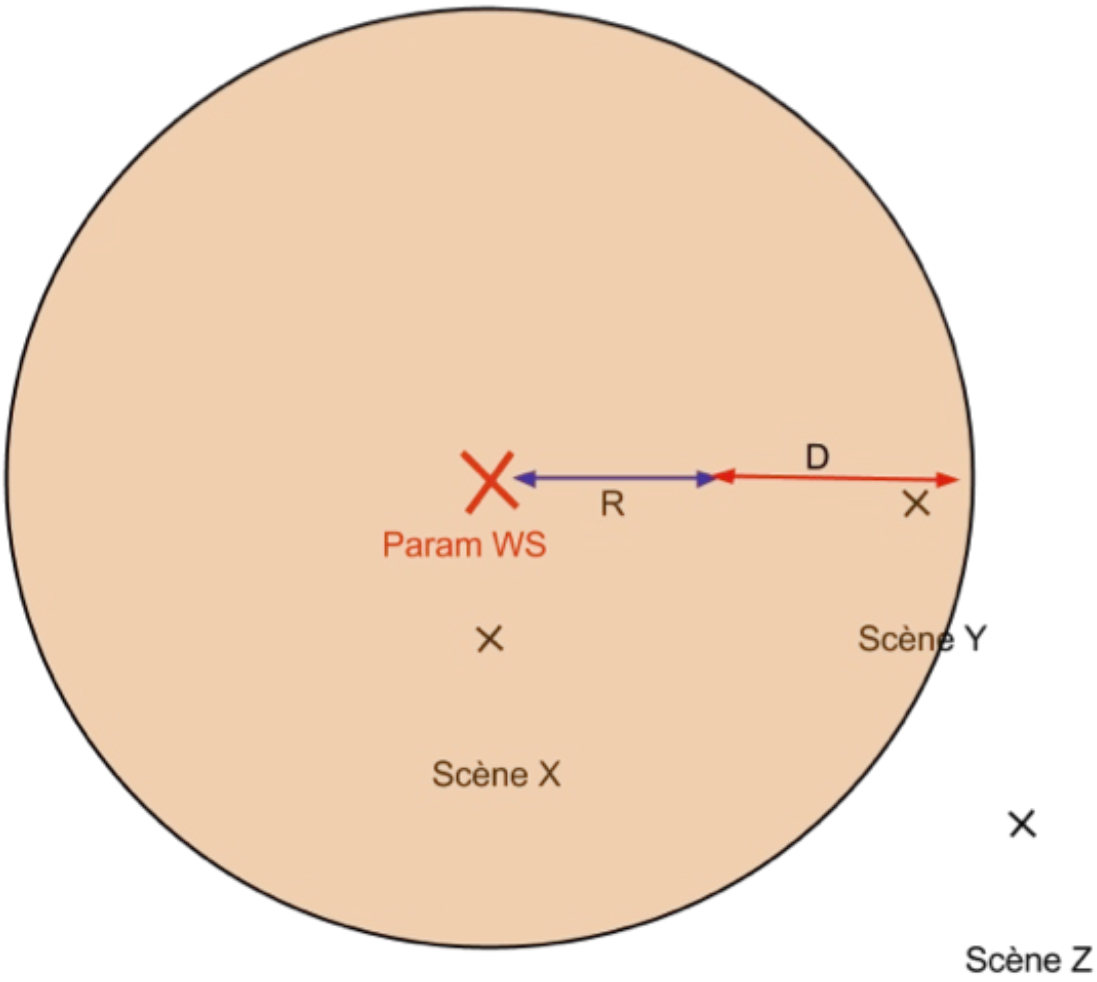
l'application liée au BackOffice (cf. gpsZoneRadius [[3.1 : Architecture]]) qui correspond à un rayon de tolérance.)

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.

Pour cela on utilisera aussi la fonction PostGIS ST_Distance. 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)

3.3.3 Fonction getSceneByGPSCoord / getCLoseSceneByGPSCoord (V2)

La version permet de définir une zone géographique grâce à un Polygon. Le polygon permet de définir précisément les contours des zones.

Fonction getSceneByGPSCoord V2

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)

Ce WebService très simple utilisera la fonction PostGIS ST_Contains.

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 : (ST_Contains retourne 0 pour tous les objets de la base) * Soit contenant s'il y a une scène sur la zone (ST_Contains retourne 1 pour un objets de la base) * Soit plusieurs scènes dans le cas ou des scène se chevauchent (ST_Contains retourne 1 pour plusieurs objets de la base)

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 V2

Fonction permettant de retourner les scènes correspondant proche d'une zone géographique. Cette fonction prend en paramètre : * Un TokenAPIRead (correspondant au BO) * Une coordonnée GPS (latitude, longitude) * Une distance D, correspondant au rayon de recherche.

La fonction sera simple, on retournera toutes les scènes dont la distance retournée par ST_Distance sera inférieur à D.

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)

[[Home]] > [[Spécifications]] > [[3 Link Server]] > [[3.4 WS Post]]

[[Sommaire]]

[[3.4 WS Post]]

Le token pour authentifier les WebServices en écriture est lié à un compte utilisateur. C'est a dire que c'est l'utilisateur qui est responsable de son token. Il ne doit pas le preter à un tiers (comme un mot de passe), et doit l'utiliser seulement sur son application mobile. C'est une méthode beaucoup plus fiable que l'authentification WS Read, mais qui demande à l'utilisateur de saisir son token dans l'application android.

Note: le token sera le token d'un utilisateur standard. Un compte administrateur ne pourra pas son token pour s'authentifier (dans le cas normal l'administrateur n'aura même pas de token). La raison est qu'un administrateur n'est pas lié à un BackOffice, mais peut gérer tout les backoffice (et si on ne sais pas à quel BackOffice on fait référence, il ne sert à rien de rajouter un lien vers une scène d'un BackOffice inconnu).

3.4.1 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 limite

- Le couple Id tag / Id Scene existe déjà. Dans ce cas on fera la mise a jour l'Id de la scène, et on reverra un message indiquant qu'iil s'agit d'une modification (en retournant l'Id de scène remplacé).

Cas d'erreurs

- Le Token n'est pas valide : renvoyer une erreur (non autorisé)

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.

3.4.2 Fonction postLinkGPSCoordScene V1

Fonction permettant d'ajouter une liaison entre une zone géographique et une scène. Dans cette première version (V1), la zone géographique correspond à un point central et un rayon. Le rayon est un paramètre du lié à un BackOffice (cf gpsZoneRadius [[3.1 : Architecture]]).

Elle prend en paramètre : * Un Token (un TokenPost) * Un point GPS (latitude / longitude), correspondant au point central de la zone * Un Id de Scène

Cas standard

Le token est valide. 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)

Améliorations

Comme pour la liaison tag / scène, 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.

Une autre amélioration serait la detection des positions similaires. Par exemple si l'utilisateur posterait un point central à

moins de X metre d'un autre point central, il devrait pouvoir soit créer un nouveau point (cas standard) soit mettre à jour le point à proximité.

3.4.3 Fonction postLinkGPSCoordScene V2

Fonction permettant d'ajouter une liaison entre une zone géographique et une scène. Dans cette seconde version (V2), la zone géographique est définie par un Polygon.

Elle prend en paramètre : * Un Token (un TokenPost) * Un Polygon (liste de points GPS (latitude;longitude)) * Un Id de Scène

Cas standard

Le token est valide. 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)

Améliorations

Comme pour la liaison tag / scène, 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.

Une autre amélioration serait la detection des similarité entre polygon. Par exemple si le polygon passé en paramètre couvre une zone commune avec un autre polygon de la base associé au back office, l'utilisateur devrait pouvoir soit créer un nouveau polygon (cas standard) soit mettre à jour le polygon similaire.

[[4 API Java BO]]

[[4.1 Généralité]]

[[4.2 API Read]]

[[4.3 API Post]]

[[4.1 Général]]

4.1.1 Architecture de l'API Java

--> A faire

4.1.2 Authentification

L'authentification consiste à utiliser OAuth 2 du serveur de WebServices. (cf. [[1.1 WS Auth]]).

Fonctionnement : avant chaque requête aux WebServices, si l'application n'a pas de token, ou que celui-ci a expiré (ou est sur le point d'expirer), l'API fera appel aux services d'authentification.

[[4.2 API Read]]

Ici, l'API Java consiste à faciliter l'appel des webservices du BO, les spécifications sont donc simplement correspondantes aux specs des webservices. La seule différence se situe au niveaux du type de données; les WebServices travaillent avec du XML, et dans l'API nous utiliserons des objets métiers (qui correspondent au XML).

4.2.1 Fonction getListAllParcours

cf. [[1.2 WS Read]]

4.2.2 Fonction getParcoursArchitectureById

cf. [[1.2 WS Read]]

4.2.3 Fonction getParcoursById

cf. [[1.2 WS Read]]

4.2.4 Fonction getSousParcoursById

cf. [[1.2 WS Read]]

4.2.5 Fonction getSceneById

cf. [[1.2 WS Read]]

4.2.6 Fonction getTransitionById

cf. [[1.2 WS Read]]

4.2.7 Fonction getElementById

cf. [[1.2 WS Read]]

4.2.8 Fonction getHistoriqueByUserId

cf. [[1.2 WS Read]]

4.2.9 Fonction getRecommandedSceneByUserId

cf. [[1.2 WS Read]]

4.2.10 Fonction getMediaByUserId

cf. [[1.2 WS Read]]

[[Sommaire]]

[[4.3 API Post]]

Ici, l'API Java consiste à faciliter l'appel des webservices du BO, les spécifications sont donc simplement correspondantes aux specs des webservices. La seule différence se situe au niveaux du type de données; les WebServices travaillent avec du XML, et dans l'API nous utiliserons des objets métiers (qui correspondent au XML).

4.3.1 Fonction Post Media (Lié à rien / à une scène / à une artefact)

cf. [[1.3 WS Post]]

4.3.2 Creation de Parcours

cf. [[1.3 WS Post]]

[[5 API Java LS]]

5.1 API Java LS \ API Java LS

5.1.1 API Java LS \ API Java LS \ Architecture de l'API Java

5.2 API Java LS \ API Java LS : Read

5.2.1 API Java LS \ API Java LS : Read \ Fonction getSceneIdByTag

5.2.2 API Java LS \ API Java LS : Read \ Fonction getCloseSceneByGPSCoord

5.3 API Java LS \ API Java LS : Post

5.3.1 API Java LS \ API Java LS : Post \ Fonction postLinkTagScene

5.3.2 API Java LS \ API Java LS : Post \ Fonction postLinkGPSCoordScene (V1 : juste pts central + rayon)

5.3.3 API Java LS \ API Java LS : Post \ Fonction postLinkGPSCoordScene (V2 : polygon)

[[5.1 Général]]

5.1.1 Architecture de l'API Java

[[5.2 API Read]]

5.2.1 Fonction getSceneIdByTag

5.2.2 Fonction getCloseSceneByGPSCoord

[[5.3 API Post]]

5.3.1 Fonction postLinkTagScene

5.3.2 Fonction postLinkGPSCoordScene (V1 : juste pts central + rayon)

5.3.3 Fonction postLinkGPSCoordScene (V2 : polygon)

[[6 Android]]

[[6.1 Général]]

[[6.2 Medias]]

[[6.3 IHM]]

[[6.4 Capteurs]]

[[6.1 Général]]

6.4.1 Architecture

6.4.2 Intégration API

6.4.3 Connexion/Deconnexion via API

[[6.2 Medias]]

6.2.1 Afficher une vidéo YouTube

Les médias d'une scène peuvent être une vidéo youtube.

La solution retenue est de lire ces vidéos dans l'application plutôt que de passer par l'application youtube.

Pour cela nous utiliserons l'API youtube Android.

https://developers.google.com/youtube/android/player/ http://fr.openclassrooms.com/informatique/cours/creez-des-applications-pour-android/le-multimedia-3

6.2.2 Afficher un PDF

Android ne gère pas les pdf en natif, pour afficher un fichier pdf il faudra donc que l'utilisateur possède une application de lecture de pdf que nous lancerons depuis notre application.

6.2.3 Jouer un MP3 (streaming)

Le MP3 est un format que connait android de base.

http://fr.openclassrooms.com/informatique/cours/creez-des-applications-pour-android/le-multimedia-3

6.2.4 Jouer une vidéo (streaming)

http://fr.openclassrooms.com/informatique/cours/creez-des-applications-pour-android/le-multimedia-3

6.2.5 Afficher une vidéo Dailymotion

6.2.6 Afficher une page Web

Solution : Réaliser un mini navigateur WEB

http://a-renouard.developpez.com/tutoriels/android/realiser-navigateur-web/

http://blogpeda.ac-poitiers.fr/lp2i-si/2013/02/03/ouvrir-une-page-html-dans-une-application-android-avec-app-inventor/
http://flo.dauran.com/169-android-webview/

http://charlesen.fr/mon-blog/35-tutoriel/95-construire-des-applications-android-a-laide-de-webview.html

6.2.7 Afficher un média du BO

[[6.3 IHM]]

Maquette Ninja Mock

6.3.1 Afficher une scène/artefact du BO

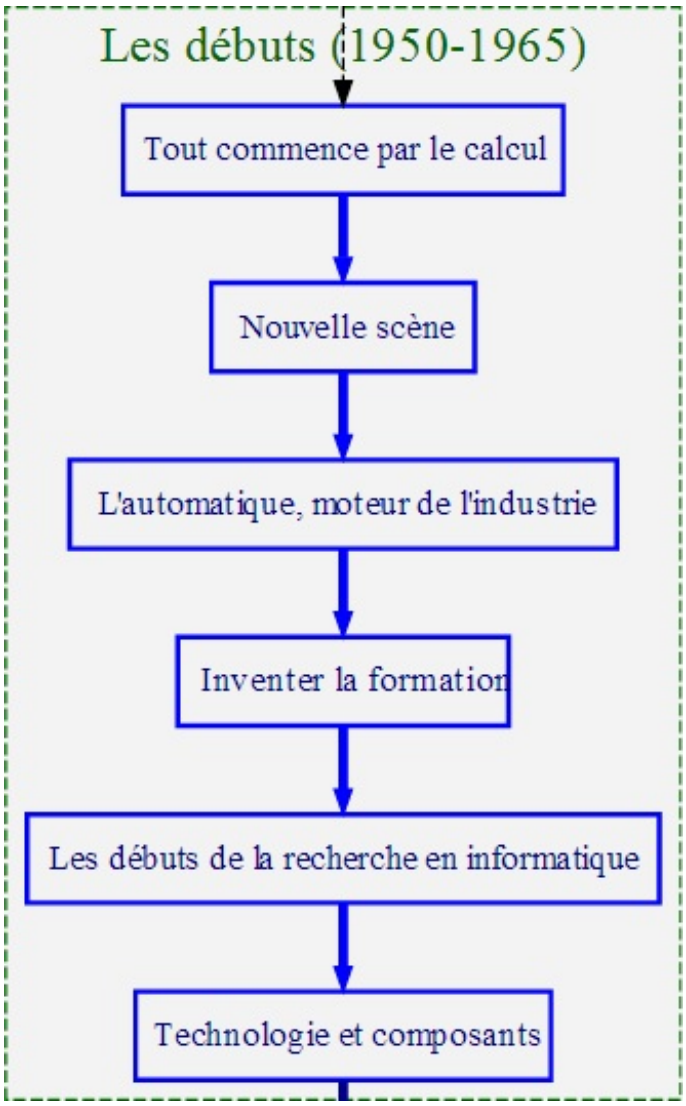
Affichage d'une scène :

Affichage d'un artefact :

6.3.2 Afficher un parcours du BO

Le Parcours s'affiche en glissant l'écran de droite à gauche depuis une scène. Apparaît alors un le parcours interactif sur lequel se trouve l'utilisateur.

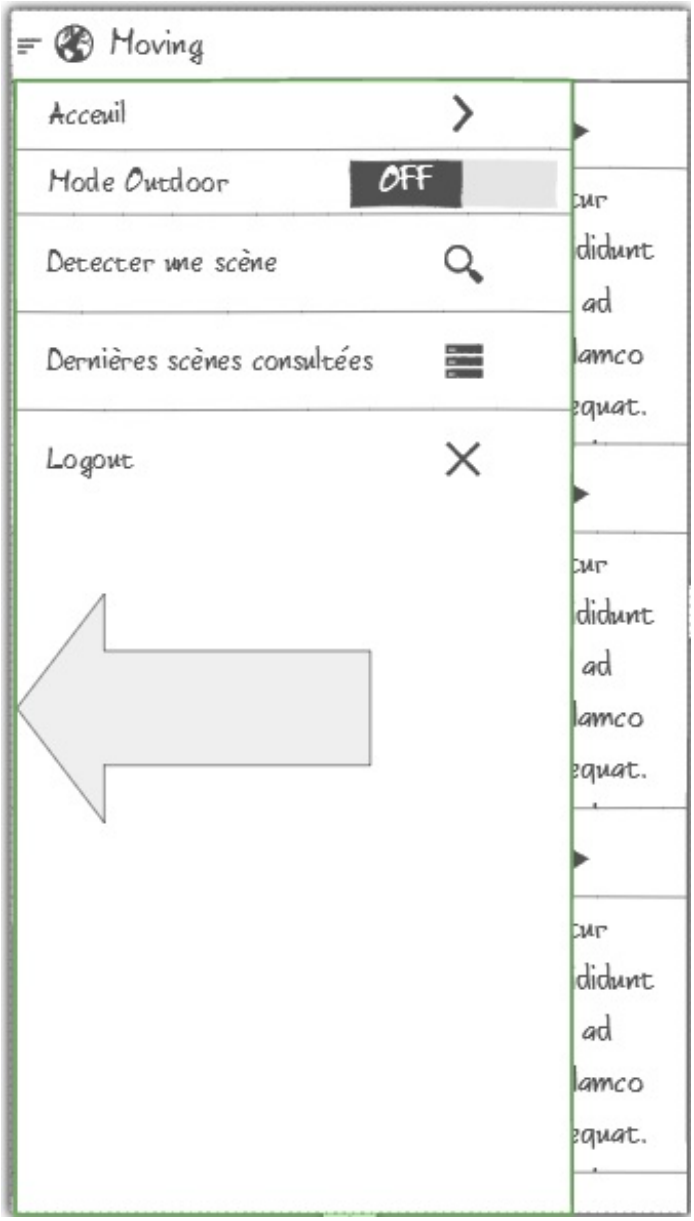
Affichage du parcours :



6.3.3 Navigation

La navigation s'effectue de plusieurs façons : * Le panel de gauche comporte un lien qui permet de revenir à l'accueil. * Les scènes comportent des liens vers les éléments qui la composent (artefacts ou médias), vers les scènes voisines (scène recommandée ou scène(s) secondaires) et des liens de retours (scène précédente, dernière scène recommandée visitée et retour au début du parcours). cf 6.3.1 * Les artefacts comportent des liens vers les médias liés à eux, vers les artefacts liés à eux et un lien de retour à la scène parente. cf 6.3.1

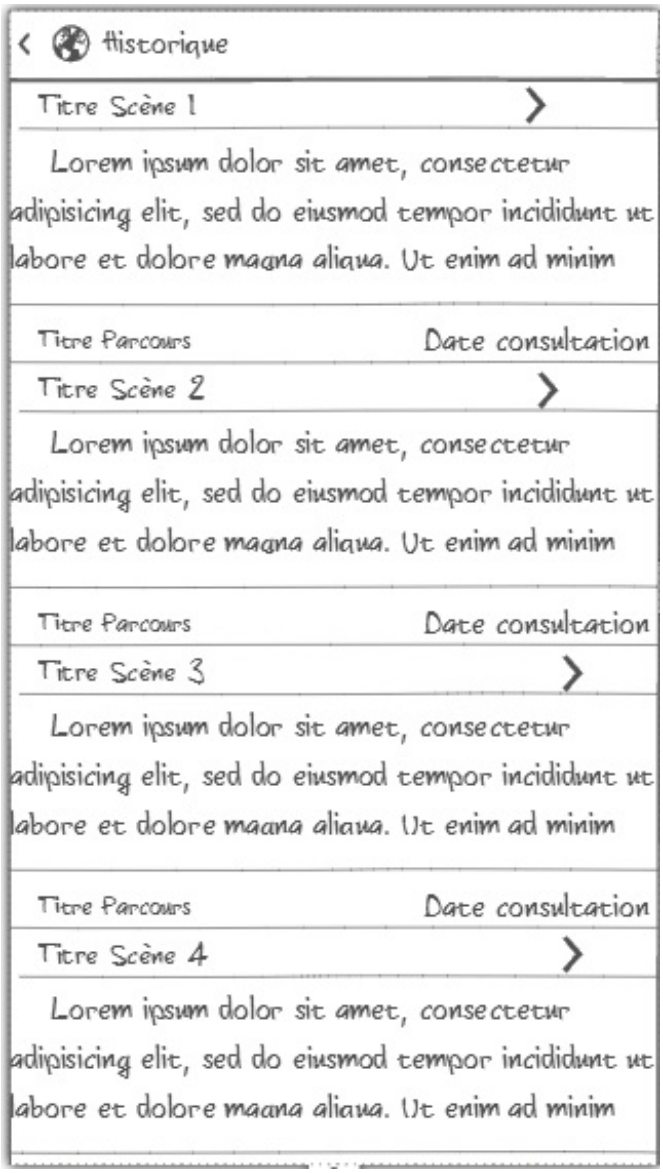
Affichage du panel gauche :



6.3.4 Consultation Historique

L'historique s'affiche depuis le panel gauche. Le bouton de la bare d'action en haut permet de revenir à la scène courante.

Affichage historique :



6.3.5 Consulation Recommandations

[[Home]] > [[Spécifications]] > [[6 Android]] > [[6.4 Capteurs]]

[[Sommaire]]

[[6.4 Capteurs]]

L'application doit permettre la lecture de différents capteurs, dans tous les cas l'user doit "déclencher" l'écoute des capteurs. Aucun des capteurs ne doit être actif en permanence, pour économiser de la batterie.

6.4.1 Lire un QRCode

Deux possibilités : * Utiliser une appli du téléphone pour lire le QR code. * Intégrer un lecteur de QR code dans l'appli.

6.4.2 Lire un NFC

Les puces NFC seront rattachées à des scènes indoor. Le lecture d'une puce NFC devra récupérer comme information l'id de la scène pour afficher la scène correspondante dans l'application.

http://sberfini.developpez.com/tutoriaux/android/nfc/

6.4.3 Lire les coordonnées GPS

Les coordonnées GPS auront plusieurs fonctions dans l'application: * Délimiter la zone géographique d'une scène outdoor. * Récupérer la position de l'utilisateur et afficher la ou les scènes correspondantes. * Diriger l'utilisateur vers la scène outdoor choisie où la scène la plus proche (par défaut).

http://fr.openclassrooms.com/informatique/cours/creez-des-applications-pour-android/la-localisation-et-les-cartes

6.4.4 Gestion de modes

La gestion de modes est un paramètre modifiable par l'utilisateur pour choisir si il est en extérieur ou en intérieur.

6.4.5 Mode intérieur

Le mode intérieur déclenchera le NFC et le bluetooth.

6.4.6 Mode extérieur

Le mode extérieur déclenchera le GPS.

6.4.7 Lire un Ibeacon

iBeacon est un système de positionnement en intérieur à basse consommation énergétique. iBeacon est une technologie qui permet à un périphérique d'envoyer une push notification à un périphérique à proximité.

http://beekn.net/developing-ibeacon-app/

[[7 Android Contribution]]

[[7.1 Général]]

[[7.2 Fonctions]]

[[7.3 IHM]]

[[7.1 Général]]

7.1.1 Architecture

7.1.2 Intégration API Post

7.1.3 Interface parametrage

[[Sommaire]]

[[7.2 Fonctions]]

7.2.1 Ajout de photo Serveur de fichier

Envoi de fichier par webservices, le serveur de fichier c'est le BO. Création de la fonction PostFile() dans le BO et l'API Java.

PostFile:

-Entrées: TypeFile(string 'PDF'/'Video'/'MP3'/'Photo'/'Autre'), FileName(string 'Nom.mp4'), Data(le fichier). -Sorties: Success?, FilePath(string 'path/du/fichier/sur/le/server/BO/file.mp4'

Une fois le fichier envoyé on peut utiliser la fonction de webservice PostMedia() avec en paramètre l'URI fichier reçue en retour de PostFile()

7.2.2 Ajout de vidéo Youtube

API Youtube: Upload video Une fois la video uploadée, on utilise la fonction de webservice PostMedia() en passant en paramètre l'URL de la video sur youtube.

7.2.3 Ajout de vidéo Dailymotion

Pas d'API Java. On peut utiliser javascript? Une fois la video uploadée, on utilise la fonction de webservice PostMedia() en passant en paramètre l'URL de la video sur dailymotion.

7.2.4 Ajout de vidéo Serveur de fichier

Même méthode que l'ajout de photos(7.2.1): fonction de webservice PostFile() puis PostMedia().

7.2.5 Ajout de MP3 Serveur de fichier

Même méthode que l'ajout de photos(7.2.1): fonction de webservice PostFile() puis PostMedia().

7.2.6 Ajout autre fichier Serveur de fichier

Même méthode que l'ajout de photos(7.2.1): fonction de webservice PostFile() puis PostMedia().

[[Home]] > [[Spécifications]] > [[7 Android Contribution]] > [[7.3 IHM]]

[[Sommaire]]

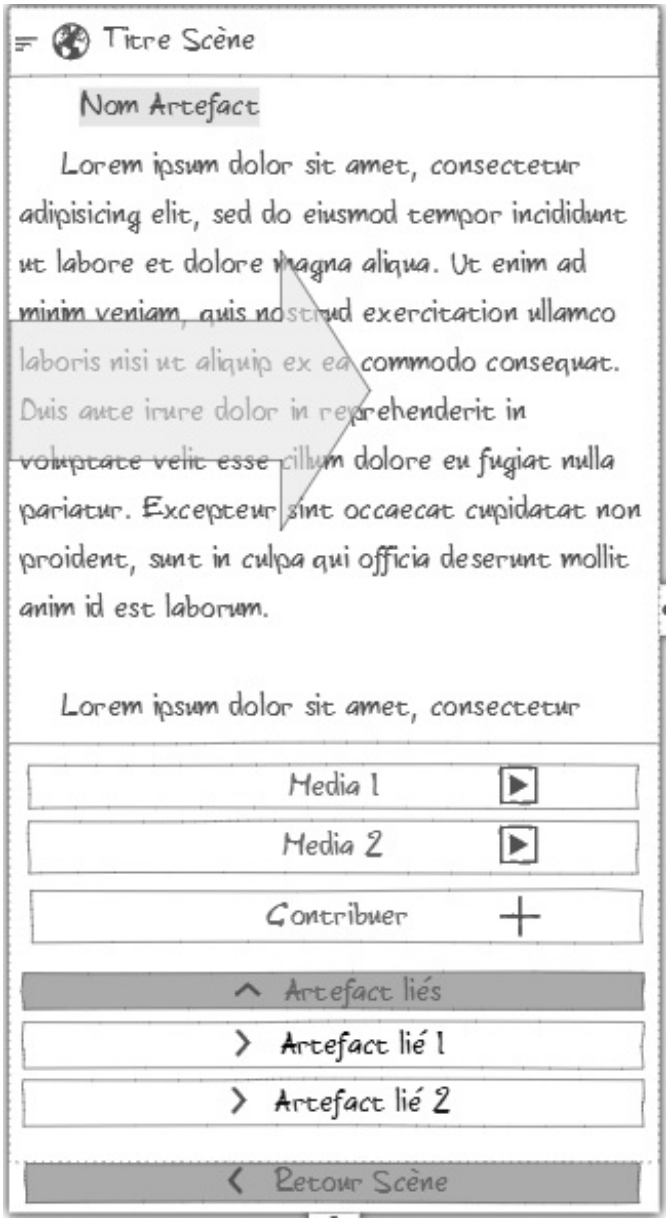
[[7.3 IHM]]

7.3.1 Poster un media

La contribution est possible depuis l'interface d'une scène où d'un artefact. Le média sera automatiquement lié à la scène ou l'artefact depuis lequel il est posté.

Affichage Scène contribution :

Affichage Artefact contribution :

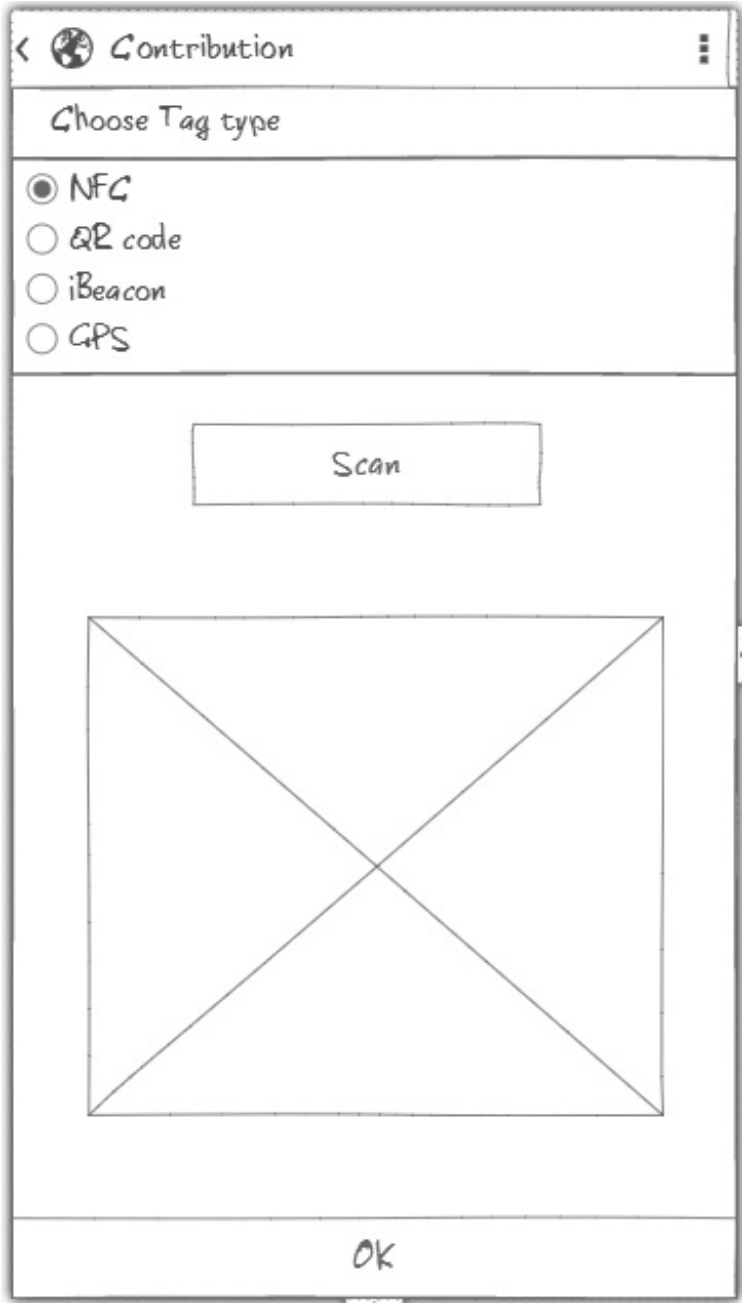


Affichage interface sélection média :

7.3.2 Lier un Tag à une scène

Pour lier un Tag à une scène, il faut passer par le bouton "lier un tag" depuis l'interface de la scène (cf 7.3.1) qui envoi sur une interface de scan où l'on peut choisir le type de capteur et le lier à la scène courante.

Affichage interface scan :



7.3.3 Lier un Point GPS à une scène (V1 : pts central + rayon)

L'interface de scan (cf 7.3.2) permettra de lier la scène en cours avec la position actuelle de l'utilisateur.

7.3.4 Lier un Point GPS à une scène (V2 : polygon)

7.3.5 Consulter historique Contribution

7.3.6 Interface parametrage

L'interface paramètre est accessible depuis le panel gauche. Elle permet de modifier les différents paramètres de l'application.

Affichage interface paramètre :


Paramètres
URL Back-office
Adresse link-Serveur
API Key
Infos Compte Youtube