D

<?xml version="1.0"?><DocumentFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"> <GraphicCharterDefinitionId>0</GraphicCharterDefinitionId> <TemplateBaseTypeId>0</TemplateBaseTypeId> <CompanyId>4</CompanyId> <ConfidentialId>0</ConfidentialId> <ConfidentialDescription /> <CountryId>0</CountryId> <PageSizeId>1</PageSizeId> <PageOrientationId>1</PageOrientationId> <PrePrintedStationary>false</PrePrintedStationary> <Project>Outil de cartographie</Project> <Reference>20180601-150612-RR</Reference> <TemplateType>3</TemplateType> <CultureId>fr-FR</CultureId> <LanguageId>2</LanguageId> <Customer>Cellule architecture</Customer> <DocumentDate>2018-06-06T16:52:15.262368+02:00</DocumentDate> <Saved>false</Saved> <IsValid>true</IsValid> <FirstPageCover>false</FirstPageCover> <IsNew>false</IsNew> <CurrentVersion>1.00</CurrentVersion> <DocumentType>Dossier</DocumentType> <DocumentTypeId>1</DocumentTypeId> <Entity>Cellule architecture</Entity> <HasDistributionList>true</HasDistributionList> <HasForeword>false</HasForeword> <Title>Analyse de l'outil de cartographie</Title> <Status>1</Status> <StatusDescription>Travail</StatusDescription> <SetEdition>false</SetEdition> <SetVersion>false</SetVersion></DocumentFile>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  | |  |
|  |  |  |  | |  |
|  |  |  | | |  |
|  |  |  | | |  |
|  |  | Analyse de l'outil de cartographie | | |  |
|  |  | mercredi 6 juin 2018 | | |  |
|  |  | : | | |  |
|  |  |  | |  |  |

Destinataire(s)

|  |  |
| --- | --- |
|  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Origine de la mise à jour** | **Rédigée par** | **Validée par** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Sommaire

[1. Introduction 4](#_Toc516067265)

[2. Partie Back-end 5](#_Toc516067266)

[2.1. Création du modèle 5](#_Toc516067267)

[2.2. Alimentation initiale de la base 5](#_Toc516067268)

[2.2.1. Création des programmes en base 5](#_Toc516067269)

[2.2.2. Création des modules 4GL en base 6](#_Toc516067270)

[2.2.3. Création des fonctions techniques en base 6](#_Toc516067271)

[2.2.4. Association des modules 4GL et des programmes 6](#_Toc516067272)

[2.2.5. Association des fonctions et des programmes 6](#_Toc516067273)

[2.2.6. Extraction du dictionnaire des programmes 7](#_Toc516067274)

[2.2.7. Récupération des SMG 7](#_Toc516067275)

[2.2.8. Implémentation des connexions SVN 7](#_Toc516067276)

[2.3. Alimentation régulière 8](#_Toc516067277)

[3. Partie Front-end 9](#_Toc516067278)

[4. Chiffrage 10](#_Toc516067279)

# Introduction

Ce document présente l’analyse de l’outil de cartographie. Ici serons détaillées les étapes de développement à mettre en place pour réaliser l’outil du côté *back-end* avec les implémentations Java et Spring, et côté *front-end* avec les implémentations Angular.

Cette réalisation a pour but de proposer un outil de recensement, d’analyse et de visualisation de la cartographie du modèle de données Amplitude. Elle sera composée d’une partie *back-end* qui va comporter les fonctions d’analyse et de gestion des données d’Amplitude, et d’une partie *front-end* qui va gérer les affichages et les interfaces utilisateurs.

Parmi les étapes de développement pour l’outil, on retrouve la création du modèle en base de données, la gestion de l’alimentation initiale de la base, la mise en place des méthodes d’analyse d’Amplitude, la création des écrans, etc.

# Partie Back-end

La partie *back-end* de l’application regroupe les éléments de gestion des données présentes dans Amplitude. L’outil devra tenir compte des programmes d’Amplitude, des modules 4GL présents dans l’arborescence, des fonctions composant ces modules, et plus généralement des éléments présents dans le diagramme de classe. Le modèle de données pour l’outil sera réalisé avec *Spring Data*, qui permet un interfaçage automatique entre le code source en Java et le gestionnaire de base de données configuré (dans notre cas, Oracle). Une fois le modèle réalisé, nous distinguerons deux parties : l’alimentation initiale des tables de notre modèle et la mise à jour régulière des éléments de la base.

## Création du modèle

L’architecture du *back-end* sera divisée en deux parties principales : une partie cartographie fonctionnelle et une partie cartographie technique. La cartographie fonctionnelle correspond aux entités définies à partir du dictionnaire des données V3 dans un premier temps puis OCAPI à l’avenir. Pour le moment, l’outil ira chercher directement les informations dans la base de données de dico V3 et les exposera en REST à la partie *front-end*. La partie cartographie technique regroupera les analyses et les méthodes liées aux programmes et aux modules d’Amplitude.

## Alimentation initiale de la base

### Création des programmes en base

La première phase de l’alimentation initiale de la base est la création des programmes dans la base de données. La récupération des programmes se fait à partir du dossier contenant les *BaseProg*. Ceux-ci vont nous permettre de récupérer plusieurs données qui nous intéressent : le nom des programmes, leur type quand celui-ci est indiqué et certaines caractéristiques comme le client ou la zone géographique quand elles sont disponibles. Ces données peuvent être récupérées d’après les noms des *BaseProg* qui respectent la convention de nommage suivante :

*module.type.nom\_programme*

Dans certains cas, on retrouve le nommage suivant, ajoutant des caractéristiques:

*module.caractéristique.type.nom\_programme*

Ici, l’objectif sera de lister les fichiers de *BaseProg*, de parser chaque résultat en conservant les données que l’on souhaite, puis de créer les requêtes pour alimenter notre base.

### Création des modules 4GL en base

Une fois les programmes enregistrés, l’étape suivante est l’alimentation des modules 4GL en base. La récupération des modules 4GL se fait en parcourant l’arborescence des fichiers sources d’Amplitude. Pour chaque dossier de l’arborescence, on liste les modules présents et on conserve en base les noms des fichiers avec leur répertoire d’origine.

### Création des fonctions techniques en base

Plusieurs solutions sont possibles pour récupérer les fonctions techniques. Tout d’abord, on appelle fonctions techniques, les fonctions définies dans les modules 4GL. La méthode brute pour les récupérer serait donc de parser l’intégralité des fichiers sources pour y retrouver les fonctions définies. Cependant, il existe un outil en Python permettant de retrouver les fonctions appelées dans un programme : *check\_func\_xml.py*. L’idée est donc de modifier ce script afin de retrouver les fonctions définies par module 4GL. Les modifications à effectuer devront permettre de sortir un flux texte exploitable par la suite dans le programme Java avec les indications suivantes : nom de la fonction, module 4GL où elle est définie et la liste des fonctions appelées par cette fonction. Ces informations devront être ensuite conservées en base avec *Spring Data*.

### Association des modules 4GL et des programmes

À ce stade, les modules 4GL et les programmes ont été enregistrés en base. L’association des modules 4GL et des programmes se fera en utilisant le script Python *l4gl.py* qui affiche la liste des modules présents dans le *baseprog* d’un programme. À partir de cette liste, on associe le programme sélectionné aux modules affichés suite à l’exécution du script python. Parmi ces modules, on retrouve un module portant le même nom que le programme : le module principal du programme. Cette information sera à conserver en base.

### Association des fonctions et des programmes

L’association des modules et des programmes se fera en utilisant le même script que pour la création des programmes en base : *check\_func\_xml.py*. Pour rappel, ce programme permet d’afficher les fonctions appelées par un programme donnée. Dans notre cas, il suffit d’appeler le script pour chaque programme et d’associer les fonctions affichées avec le programme. Un modification du code Python sera nécessaire pour afficher une sortie qui correspond à notre besoin.

### Extraction du dictionnaire des programmes

L’utilisation des données du dictionnaire des programmes va nous permettre d’ajouter des informations supplémentaires aux programmes enregistrés dans notre base et aussi d’y ajouter les modules fonctionnels. Le dictionnaire des programmes se présente sous un fichier Excel avec deux feuilles de calcul : la première décrivant la liste des programmes avec leurs descriptions, leurs modules fonctionnels associés et d’autres informations, et la seconde lisant les modules fonctionnels ainsi que leur représentation sous forme de code à 3 caractères. L’objectif pour nous sera de parcourir la liste des modules et de les enregistrer en base dans un premier temps, puis, pour chaque programme du dictionnaire, on enregistre la description associée.

### Récupération des SMG

Les SMG sont des programmes particuliers que l’on peut retrouver avec le type *webservice*. Les SMG ont néanmoins un autre nom qui désigne leur action (exemple d’un SMG: *createAccount*). La récupération de ces informations se fera à partir des fichiers *xcf* dans l’arborescence d’Amplitude. L’objectif sera de lister les fichiers *xcf*, de garder en mémoire les noms de fichier, et de parser chaque fichier *xcf* pour récupérer le nom du programme associé.

D’autres données concernant les SMG sont disponibles dans le dictionnaire des SMG. Il s’agit d’un fichier Excel similaire au dictionnaire des programmes qui va nous permettre d’ajouter des informations complémentaires, notamment la description du SMG.

### Implémentation des connexions SVN

L’outil devra interagir avec l’outil de versioning SVN pour pouvoir prendre en compte les modifications sur Amplitude. Ainsi, pour chaque version d’Amplitude que l’on souhaite étudier, il faudra effectuer un *checkout* d’Amplitude. C’est à partir des *working copy* récupérées que l’on va effectuer les analyses et les créations d’entités en base. Il sera nécessaire par la suite d’effectuer des *update* des versions lors des alimentations régulières pour observer les différences et mettre à jour les versions.

## Alimentation régulière

La mise à jour régulière des informations se fait à partir des versions d’Amplitude que l’on utilise sur nos *working copy*. Les modifications sur les versions seront visibles avec SVN en utilisant *svn status* ou *svn diff*.

Pour les fichiers ajoutés, le principe consistera à appeler le script *check\_func\_xml.py* pour voir la liste des fonctions du module 4GL qui a été modifié. On compare la liste obtenue en sortie du script et la liste présente en base et on ajoute ou modifie les fonctions qui diffèrent.

Pour les modules 4GL, on reprend les méthodes d’enregistrement des modules vu dans la partie [Création des modules 4GL en base](#_Création_des_modules) avec un appel du script python et l’ajout des fonctions pour le module sélectionné.

# Partie Front-end

La partie *front-end* de l’outil sera développée sous Angular et suivra une architecture *REST*. Les écrans devront être à même de récupérer les données exposées par la partie *back-end*. L’objectif des écrans est de donner un aperçu général de la cartographie. Un panneau de recherche des données devra permettre de retrouver les informations que l’on souhaite le plus simplement possible.

# Chiffrage

|  |  |
| --- | --- |
| Tâches | Nombre de jours |
| **Architecture du projet** |  |
| Partie Modèle | 1 |
| Partie View | 1 |
| **Back-end** |  |
| Création du modèle   * (tables + entities + dao + services) avec Spring Data | 10 |
| **Alimentation initiale** |  |
| Création des programmes en base :   * Récupération de la liste des programmes dans le BaseProg * Parsing des résultats (récupération du nom, du type, éventuellement des caracs pour certains programmes) * Alimentation de la table | 3 |
| Création des modules 4gl en base :   * Parcours de l'arborescence d'Amplitude * Alimentation de la table | 2 |
| Création des fonctions techniques en base :   * Modification du script check\_func\_xml.py | 3 |
| * Appel du script python et lecture du résultat | 2 |
| * Création des fonction techniques | 2 |
| * Liste des fonctions appelées par la fonction sélectionnée avec alimentation | 2 |
| Association des modules 4gl et des programmes |  |
| * Appel de l4g.py et lecture du résultat | 2 |
| * Alimentation | 2 |
| * Détermination du module 4gl principal | 1 |
| Association des fonctions et des programmes |  |
| * Modification du script check\_xml.py | 1 |
| * Appel du script précédent | 2 |
| * Alimentation de la base | 2 |
| Extraction du dico des programmes |  |
| * Lecture du fichier Excel | 3 |
| * Alimentation de la base de données | 2 |
| Récupération des SMG |  |
| * Lecture du XCF | 1 |
| * Parse du fichier XCF | 2 |
| * Alimentation | 1 |
| * Lecture du fichier Excel pour alimentation | 2 |
| Implémentation des connexions SVN | 2 |
| **Alimentation régulière** |  |
| Création du job | 1 |
| Liste des fichiers modifiés sur le repo avec svn status –q   * svn update * pour les fichiers modifiés * pour les fichiers ajoutés | 5 |
| **Front End** |  |
| Ecrans | 20 |