VITAM - Documentation de configuration des TNR

Version 0.20.0

VITAM

juil. 21, 2017

Table des matières

CHAPITRE 1

Introduction

1.1 Principes généraux

1.1.1 Tests de non régression

Les tests de non régression (TNR) ont pour objectif de tester la continuité des fonctionnalités de Vitam. L'ajout de nouvelles fonctionnalités pouvant entraîner des bugs ou indisponibilités (régressions) sur des fonctionnalités déjà existantes, l'outil de test de non regression va permettre de tester automatiquement le périmètre fonctionnel pré-existant afin de s'arrurer de son bon fonctionnement dans le temps.

L'aujout d'une nouvelle fonctionnalité doit donc s'accompagner d'un ou plusieurs TNR.

Idéalement, les développeurs doivent lancer les TNR avant d'effectuer une Merge Request visant à intégrer une nouvelle fonctionnalité.

1.1.2 Behavior-Driven Development (BDD)

Le BDD est une méthode de collaboration s'appuyant sur un langage de programation naturel permettant aux intervenants non techniques de décrire des scénarios de fonctionnement.

Les mots de ce langage permettent de mobiliser des actions techniques qui sont, elles, réalisées par les développeurs.

Le BDD est utilisé pour la réalisation des TNR, ce qui permet à tout intervenant du projet de pouvoir en réaliser.

L'outil utilisé dans le cadre de Vitam est Cucumber (https://cucumber.io/) qui utilise le langage Gherkin.

1.2 Pré-Requis

1.2.1 Dépot vitam-itest

L'ajout et la modification de TNR sont à éffectuer dans le dépôt vitam-itest.

Il est donc nécessaire de le cloner avant toute chose.

1.2.2 Git LFS

Afin de permettre la gestion de fichiers volumineux dans git, il est nécessaire d'installer l'extention Git-LFS (https://git-lfs.github.com/).

Une fois git lfs installé, il est néccessaire de l'activer pour le dépôt vitam-itest sur votre machine. Pour réaliser cette opération, se placer à la racine du dépôt et exéctuer la commande :

```
git lfs install
```

1.3 Méthodologie de test

1.3.1 Purge des bases

Les bases sont vidée avant le lancement de chaque campagne de test. Il est donc nécessaire que toutes les ressources mobilisées pour un test (SIP, référentiels...) soient chargées dans VITAM en amont de l'exécution de ce test.

CHAPITRE 2

Configuration

2.1 Structure des répertoires

Le répertoire du dépot vitam-test est strucutré de la façon suivante :

```
vitam-itests
                |------- data
```

Dossier vitam-itests : contient les fichiers de configurations des tests fonctionnels.

Dossier data : contient les éventuels jeux de données nécessaires à l'execution des tests.

2.2 Fichiers de Configuration

2.2.1 Nommage des fichiers

Un fichier regroupe tous les tests à effectuer sur une fonctionnalité. Il ne peut y avoir deux fonctionnalités dans un fichier de configuration.

On va par exemple réaliser :

- un fichier pour les tests sur l'Ingest
- un fichier pour les test sur l'accès aux unités archivistiques

Les noms des fichiers sont composés de la façon suivante :

```
EndPoint-Fonctionnalité.feature
```

Par exemple :

```
access-archive-unit.feature
admin-logbook-traceability.feature
```

2.2.2 Informations transverses

Les fichiers de configuration doivent contenir les informations suivantes qui s'appliqueront ensuite à l'ensemble des scénarios du fichier :

 # language : information obligatoire. Correspond à la langue utilisée pour les descriptions. Par exemple :

```
language : fr.
```

Annotation : information optionnelle. L'annotation permet par la suite de lancer uniquement un fichier de configuration en ligne de commande en utilisant son annotation en paramètre. Par exemple :

```
@AccessArchiveUnit
```

Fonctionnalité : information obligatoire. La fonctionnalité est une description qui permet d'identifier le périmètre testé. Il est notamment repris dans les rapports réalisés à la fin d'une campagne de test. Par exemple :

```
Fonctionnalité: Recherche une archive unit existante
```

Contexte : information optionnelle. Les informations contenues dans contexte sont des actions qui vont s'exécuter pour chacun des scénarios. A ce titre, elles s'écrivent comme les actions d'un scénario. Le constexte doit être indenté de 1 par rapport aux autres éléments. Par exemple :

```
Contexte:
Etant donné les tests effectués sur le tenant 0
```

2.3 Configuration d'un scénario

2.3.1 Structure d'un scénario

Un scénario correspond à un test. Son nom doit être défini de la façon suivante :

```
Scénario: Description du scénario
```

Il doit être sur la même indentation que le contexte, soit 1 par rapport à la fonctionnalité, annotation et language.

Un scénario est constitué d'une succession d'actions, chacune décrite sur une ligne.

Les actions sont composées des trois informations suivantes :

- Contexte
- Fonction
- Paramètre (pas toujours obligatoire)

Contexte : permet d'introduire l'action, de l'insérer par rapport à l'action précédente. La liste des contextes disponibles se trouve en annexe.

Fonction : mobilise, via un langague naturel, une fonction de Vitam. La liste des fonctions disponibles se trouve en annexe.

Paramètre : certaines fonctions ont besoin d'être suivies d'un paramètre. Ils sont listés dans le tableau des fonctionnalités disponibles en annexe.

Les actions doivent être indentées de 1 par rapport aux scénarios.

Exemple d'un scénario constitué de trois actions :

```
Scénario: SIP au mauvais format
  Etant donné un fichier SIP nommé data/SIP_KO/ZIP/KO_SIP_Mauvais_Format.pdf
  Quand je télécharge le SIP
  Alors le statut final du journal des opérations est KO
```

2.3.2 Insérer une requète DSL

Certaines fonctions nécessitent l'entrée de requêtes DSL en paramètre. Celles-ci doivent etre insérées entre guillemets (" "), après un retour à la ligne à la suite de la fonction.

Voici un exemple d'une action suivie d'une requète DSL :

```
Et j'utilise la requête suivante
        """
        { "$roots": [],
                "$query": [
                { "$and": [
                        { "$gte": {
                        "StartDate": "1914-01-01T23:00:00.000Z"
                                } }, {"$lte": {
                                "EndDate": "1918-12-31T22:59:59.000Z"
                                } } ],
                        "$depth": 20}],
                        "$filter": {"$orderby": { "TransactedDate": 1 }
                        }, "$projection": {
                        "$fields": {"TransactedDate": 1, "#id": 1, "Title": 1, "
↪#object": 1, "DescriptionLevel": 1, "EndDate": 1, "StartDate": 1}}}
        """
```

2.3.3 Insérer un tableau

Certaines fonctions attendent un tableau en paramètre. Les lignes des tableaux doivent simplement être séparées par des "pipes" (|).

Voici un exemple de fonction prenant un tableau en paramètre.

```
  Alors les metadonnées sont
| Title            | Liste des armements |
| DescriptionLevel | Item                |
| StartDate        | 1917-01-01          |
| EndDate          | 1918-01-01          |
```

2.4 Lancer les tests

2.4.1 IHM

Des écran dédiés aux tests fonctionnels sont disponibles dans l'IHM de recette. Leurs fonctionnements sont détaillés dans le manuel utilisateur.

2.4.2 Ligne de commande

Comming soon

2.5 Annexes

2.5.1 Liste des contextes disponibles

Action
Etant donné
Et
Quand
Mais
Alors

2.5.2 Liste des fonctions disponibles

Fonctionnalité	Doit être suivi par
les metadonnées sont	un tableau
le nombre de résultat est	un nombre
j'utilise la requête suivante	une requête
je recherche les unités archivistiques	une autre action
un fichier SIP nommé (.*)	un fichier
je télécharge le SIP	une autre action
le statut final du journal des opérations est (.*)	un statut
le[s] ? statut[s] ? (? :de l'événement|des événements) (.*) (? :est|sont) (.*)	un ou plusieur evType et un Statut