École Polytechnique de Montréal
Département de génie informatique et génie logiciel

LOG3430 - Méthodes de test et de validation du logiciel

Travail pratique #1 - Tests unitaires
Automne 2018

1. Mise en contexte théorique

L'objectif des tests unitaires est de valider que chaque unité d'un logiciel sous test fonctionne comme prévu. Par unité on désigne la plus petite composante testable d'un logiciel. Généralement, cette composante possède une ou plusieurs entrées et une seule sortie.
En programmation procédurale, une unité peut être un petit programme, une fonction, une procédure, etc. En programmation orientée objet, la plus petite unité est une méthode qui peut appartenir à une classe de base ou à une classe dérivée.
Les tests unitaires ne doivent jamais interagir avec des éléments d'une base de données ou des objets spécifiques à certains environnements ou à des systèmes externes. Si un test échoue sur une machine parce qu'il requiert une configuration appropriée, alors il ne s'agit pas d'un test unitaire.
Comme présenté dans les notes de cours, les frameworks de tests unitaires, les pilotes **"*Drivers*"**, les ***Stubs, Spies*** et les faux objets **"*Mocks*"** sont utilisés pour faciliter les tests unitaires.
Il existe plusieurs *Framework* de tests unitaires spécifiques à chaque langage. Parmi les framework les plus populaires on trouve `JUnit` pour `Java`, `unittest` pour `Python`, `RSpec` pour `Ruby`, **`Karma, Jasmine, Mocha, Chai`** et **`Sinon`** pour **`JavaScript`**, etc.
Étant donné que le projet fourni pour ce laboratoire est en JavaScript, les tests unitaires seront exécutés dans un environnement Node.js en utilisant les outils mentionnés précédemment.

2. Objectifs

- Mettre en pratique les connaissances théoriques acquises sur les tests unitaires.
- Apprendre à rédiger des cas de test appropriés pour chaque unité testée.
- Se familiariser avec les tests unitaires impliquant des requêtes HTTP.

3. Mise en contexte pratique

`SharedBox` est inspiré du `SendSecure SDK` disponible en plusieurs langages de programmation et développé par XMedius, compagnie montréalaise spécialisée dans l'échange de fichiers sécuritaire entre les entreprises. `SharedBox` est un SDK développé seulement pour des fins de tests. Il permet de créer des boites virtuelles pour échanger des fichiers entre les individus (très similaire à `Dropbox`). Afin de pouvoir utiliser le SDK, l'utilisateur doit s'inscrire au service et obtenir un `token` et un `id` qui seront utilisés par

toutes les opérations disponible dans le SDK. Ensuite, il sera possible de créer des `SharedBox`, ajouter des destinataires et fermer une `SharedBox` existante. Le projet et la charge de travail ont été adaptés aux requis de ce laboratoire.

Toute la documentation nécessaire est incluse à la fin de cet énoncé. De plus, des exemples de données vous ont été fournis afin de vous simplifier la réalisation de vos cas de tests.

4. Travail à effectuer

4.1. Helpers

Ce module regroupe les objets/entités utilisés par la classe `Client`. Dans le cadre de ce TP, on vous demande de tester les classes `Sharedbox` et `Recipient`. La construction des objets ainsi que les méthodes disponibles dans chaque classe doivent être testées.

4.2. JsonClient

La classe `JsonClient` est responsable de l'interaction avec le API de l'application SharedBox en envoyant des requêtes HTTP. On vous demande de tester les méthodes suivantes :

- `initializeSharedbox`
- `submitSharedbox`
- `addRecipient`
- `closeSharedBox`

Note : Veuillez vous référer aux exemples de JSON fournis pour réaliser vos tests.

4.3. Client

La classe `Client` est une classe de haut niveau et responsable de récupérer les réponses des requêtes HTTP effectuées par le `JsonClient` en manipulant des objets du module `Helpers` pour simplifier l'utilisation de la librairie. Pour cette classe on vous demande de tester les méthodes suivantes :

- `initializeSharedbox`
- `submitSharedbox`
- `uploadAttachment`
- `addRecipient`
- `closeSharedBox`

Note : Veuillez vous référer aux exemples de JSON fournis pour réaliser vos tests.

5. Questions

1. Quelle méthode avez vous choisi pour empêcher la classe `JsonClient` d'exécuter les vrais appels API durant vos tests unitaires?

2. Quelle méthode avez vous choisi pour empêcher la classe `Client` d'exécuter les vrais appels API contenus dans les méthodes de la classe `JsonClient` durant vos tests unitaires?
3. Est-ce que vos tests unitaires ont découvert une ou plusieurs défaillances? Si oui, expliquez ce que vous avez trouvé?
4. Le projet implémente un type d'exceptions personnalisé `SharedBoxException` qui étend le type standard `Error`. Lorsque vous avez testé les exceptions qui peuvent être lancées par la classe `Client`, était-il possible de vérifier le type de l'exception? Si non, expliquez pourquoi et proposez une solution.

6. Livrable à remettre, procédure de remise et retard

1. Le dossier test contenant tous vos ***Spec.js**.
2. Un rapport de quatre pages (PDF) où vous expliquez vos choix de Mocks, Stubs et/ou Spies ainsi que votre démarche suivi pour réaliser vos cas de tests. Le rapport doit contenir les réponses aux questions aussi.

Veuillez envoyer vos fichiers dans une archive de type ***.zip** (et seulement zip, pas de rar, 7z, etc) qui portera le nom : **LOG3430_lab1_MatriculeX_MatriculeY.zip** tel que : MatriculeX < MatriculeY.

Date limite pour la remise :
Groupe B1 : Mardi 2 octobre à 7:00h du matin.
Groupe B2 : Lundi 24 septembre à 7:00h du matin.

7. Barème de correction

Rapport de 4 pages	**1.5 points**

- Pertinence des jeux de tests
- Pertinence des explications et des analyses
- Réponses aux questions
- Respect du nombre de pages

Tests	**3.5 points**

- Respect des consignes
- Tests exécutables
- Qualité du code

Les travaux en retard seront pénalisés de 20% par jour de retard. Aucun travail ne sera accepté après 4 jours de retard. Si votre dépôt ne respecte pas la nomenclature définie ci-dessus, 0.5 point de pénalité sera appliqué.

8. Documentation

Lien pour télécharger NodeJs : https://nodejs.org/en/
Assertions avec Chai **:** http://www.chaijs.com/api/assert/

Configuration Sinon : https://sinonjs.org/releases/v6.1.5/general-setup/
Mocks : https://sinonjs.org/releases/v6.1.5/mocks/
Stubs : https://sinonjs.org/releases/v6.1.5/stubs/
Spies: https://sinonjs.org/releases/v6.1.5/spies/
Assertions avec Sinon : https://sinonjs.org/releases/v6.1.5/assertions/

Pour exécuter les tests dans les laboratoires :

```
cd TP1-code
npm install --no-cache
npm test --no-cache
```

Si vous utilisez vos ordinateurs personnels :

```
cd TP1-code
npm install
npm test
```