Stratégie de test

Comme déclaré dans le document “Hypothèse de validation de principe”, le sous-système d'intervention d'urgence en temps réel, ERS (Emergency Responder System) est destiné à recevoir une ou plusieurs spécialités médicales et une banque de données d'informations récentes sur les hôpitaux afin de suggérer l'hôpital le plus proche offrant un lit disponible, associé à une ou plusieurs spécialisations correspondantes. Le lieu de l'incident d'urgence doit également être fourni. Dans ce même document nous avons établi des exigences et des kpi extrêmement élevés, c’est pourquoi notre stratégie de test doit être complète et fiable.

Project Name:	ERS, Emergency Responder System (allocation de lits d'hôpital pour les urgences)		
Prepared By:	Damien Senechal	**Document Version No:**	0.1
Title:	Stratégie de test	**Document Version Date:**	
Reviewed By:		**Review Date:**	15/06/22

Table des matières

Objectifs et tâches

Objectifs

L'objectif de ce document est de couvrir l'intégralité des tests et de les commenter. Nous pouvons nous appuyer sur les documents déjà fournis :

- Hypothèse de validation de principe
- Statement of Architecture Work
- Principes de l'architecture
- Solution Building Blocks

Tâches

Pour nous assurer que notre solution est entièrement validée par des tests automatisés, l'ensemble des tâches doit être opéré à chaque implémentation de fonctionnalité :

- Tests unitaires
- Test du système et de l'intégration
- Tests de performances et de stress
- Test d'acceptation par l'utilisateur
- Tests End To End (E2E)

A cela il faut ajouter les rapports

Stratégie de test

Test unitaire

Définition : En s'alignant sur la méthode Test-Driven Development (TDD), un test unitaire doit correspondre aux spécifications de l'application, il faut donc écrire les tests en premier puis les faire passer par la suite plutôt que d'écrire le code avant et de prendre le risque d'être influencé par celui-ci lors de la rédaction des tests. Voici un modèle simple pour l'écriture des tests unitaires :

1. Écrire une fonction de test qui doit obtenir un résultat défini dans les spécifications. Ce code appelant un code qui n'existe pas encore, celui-ci doit échouer. Ceci a pour but de définir une fonction qui teste « quelque chose ».
2. Écrire le code (le minimum de « quelque chose ») pour faire réussir le test.
3. Une fois le test en succès, rajouter un autre test pour obtenir un résultat légèrement différent, en faisant varier les entrées par exemple. Ce nouveau test fera faillir le code principal.
4. Modifier le code principal pour faire réussir les tests.

5. Recommencer, en éliminant et refactorisant les éventuelles redondances dans le code des tests. On refactorise en même temps le code principal que le code des tests.
6. Un test unitaire doit tester une caractéristique et une seule. On ne définit pas un « scénario » de test complexe dans un test unitaire.
7. Il est déconseillé de tester les détails d'implémentation tels que les fonctions privées d'une classe, on se concentrera à tester les fonctions publiques, c'est-à-dire les interfaces avec lesquelles les acteurs extérieurs interagissent. Ainsi, on découpe les tests de l'implémentation et on se concentre sur la vérification du comportement attendu tout en gardant une flexibilité sur la manière d'arriver au résultat souhaité.

Participants : Les responsables des tests unitaires sont directement les développeurs eux-même, cela signifie qu'à chaque fonctionnalité, un ou plusieurs tests seront écrits.

Méthodologie :