MedHead

PROJET DU SYSTÈME D'INTERVENTION D'URGENCE

-

STRATÉGIE DE TEST

STRATÉGIE DE TESTS

MedHead

TABLE DES MATIÈRES

STRATÉGIE DE TESTS

MedHead

HISTORIQUE

Date	Version	Commentaires
04.2023	1.0	Version initiale

INTRODUCTION

Ce plan de test concerne le projet de système d'intervention d'urgence. Il couvre l'ensemble des besoins métiers pris en charge par la PoC du projet.

OBJECTIFS

L'objectif de ce plan de test est de définir les tâches nécessaires à la réussite du projet de nouveau système d'intervention d'urgence.

Les tâches permettant d'atteindre cet objectif sont :

- définir la stratégie de tests
- lister les fonctionnalités à tester
- Identifier les risques inhérents et les hypothèses de prévention

PÉRIMÈTRE

Le périmètre fonctionnel de ce plan de test se limite aux fonctionnalités couvertes par la PoC..

STRATÉGIE DE TEST

Stratégie adoptée

La stratégie de test adoptée pour ce projet est d'utiliser la méthode du développement piloté par le comportement plus communément appelée BDD (Business Driven Development ou Behaviour Driven Development en anglais).

Cette méthode consiste à définir des comportements attendus d'un applicatif logiciel ou matériel et de les traduire en un ensemble de cas / scenarii de test. Cette méthode, comme son nom l'indique, est une méthode de développement avant tout et peut être accompagnée des principes du développement piloté par les tests (ou TDD - Tests Driven Development) qui consistent à écrire les tests avant d'écrire le code. Le TDD est l'étape suivante naturelle du BDD.

Le BDD est basé sur le langage Gherkin qui utilise des mots tels que "Étant donné", "Quand", "Alors", et éventuellement "Et", qui vont décrire le comportement de votre fonctionnalité. On retrouve souvent la formulation anglaise "Given/When/Then/And".

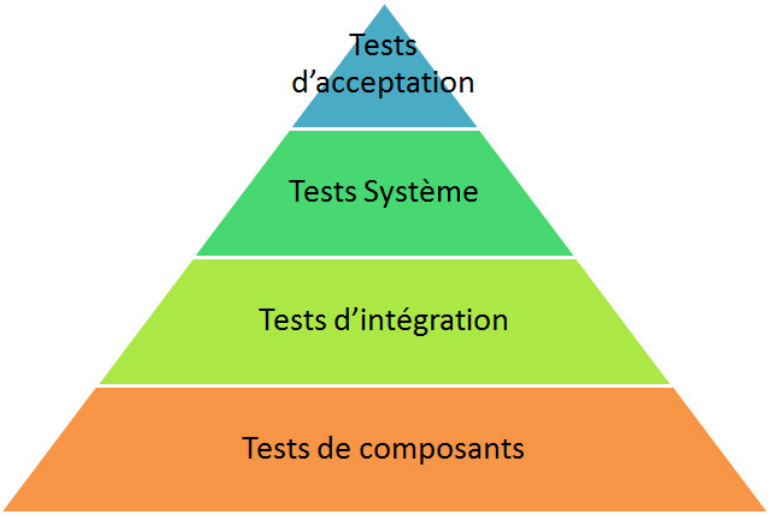
L'utilisation du BDD pour notre plan de tests s'appuiera sur l'ensemble des fonctionnalités d'interactivité décrites dans le cahier des charges du projet auxquelles viendront s'ajouter l'intégralité des fonctionnalités d'origine de l'ancienne architecture du fait de la refonte de cette architecte (détaillée dans le document de définition d'architecture du projet).

En plus de la méthode BDD, des tests de charge/performance devront être effectués pour contrôler les composants techniques assurant la scalabilité de l'architecture.

L'approche globale de ce projet de PoC fera grâce à l'utilisation de l'intégration continue (CI/CD).

Généralités sur l'organisation d'un plan de test

D'une manière générale, et selon l'ISTQB, on trouve 4 niveaux de test.



Les tests de composants

Les tests de composants ont pour but de tester les différents composants du logiciel séparément afin de s'assurer que chaque élément fonctionne comme spécifié. Ces tests sont aussi appelés tests unitaires ou encore alpha tests.

Ces tests sont généralement écrits et exécutés par le développeur qui a écrit le code du composant. Ces tests peuvent être manuels ou automatisés.

Les tests d'intégration

Les tests d'intégrations sont des tests effectués entre les composants afin de s'assurer du fonctionnement des interactions et de l'interface entre les différents composants.

Ces tests sont également réalisés, en général, par des développeurs. Ces tests peuvent être manuels ou automatisés.

Les tests système

Les tests système vérifient la conception, le comportement et les attentes présumées du client. Ils sont également destinés à réaliser des tests aux limites définies dans les spécifications du logiciel ou du matériel mais également par-delà ces limites

Ces tests vérifient que les différents aspects d'un système logiciel correspondent bien aux spécifications et constituent le premier niveau des tests dits « boite noire », c'est-à-dire qu'ils sont définis sur la base des spécifications du produit (règles métier, exigences de performance etc...) sans aucun accès au code source.

Ces tests sont le plus souvent réalisés de manière automatique.

Les tests d’acceptation

Les tests d'acceptation ont pour but de confirmer que le produit final correspond bien aux besoins des utilisateurs finaux.

Ces tests sont réalisés par le métier ou les utilisateurs finaux (par exemple avec un bêta test). Ces tests sont des tests manuels.

NB : Une application peut parfaitement répondre aux spécifications sans pour autant totalement répondre aux besoins des utilisateurs (spécifications non conforme aux processus métiers, problèmes d’ergonomie/cosmétiques, …).

Application des généralités à la méthodologie BDD

Intégration continue

L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier de manière automatique à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

L'approche BDD (couplée aux principes TDD) se combine parfaitement avec l'intégration continue et permet d'accélérer significativement les temps de traitement des tests et la qualité du code produit.

Les niveaux de test, tels que définis par l'ISTQB, couverts grâce à l'intégration continue sont :

- Les tests unitaires
- Les tests d'intégration

Test d'acceptation

La méthode BDD étant particulièrement appropriée aux méthodes agiles, les tests d'acceptation seront intégrés au processus de développement (sans pour autant être automatisés).

Ces tests seront réalisés par des testeurs faisant partie des équipes de développement sur la base des critères d'acceptation inclus dans les users stories et ceci à la fin de chaque cycle de développement agile (appelé "sprint").

Les utilisateurs effectueront eux-aussi des tests d'acceptation à la fin des jalons de livraison de fonctionnalités définis en amont du projet. Ces tests s'appuieront à la fois sur les spécifications fonctionnelles et sur l'expérience des utilisateurs afin de contrôler l'utilisabilité des fonctionnalités mises en place (ergonomie, pertinence fonctionnelle, ...).

Le test sera fait manuellement par les utilisateurs concernés par les fonctionnalités métier impactées dans le sprint.

Ces tests, s'ils révèlent des résultats négatifs, devront suivre les procédures de changement de périmètre et de corrections.

Tests système

Il existe un grand nombre de tests système. Pour les besoins spécifiques aux composants et fonctionnalités nécessaires à l'architecture mise en place pour ce projet, les tests système suivants seront réalisés :

- test de charge
- Test de stress ou de résistance

Ces types de tests sont réalisés de manière automatique par un des outils préconisés

Test de charge

Le test de charge est un type de test de performance où l'application est testée pour ses performances en utilisation normale et maximale. Les performances d'une application sont vérifiées par rapport à sa réponse à la demande de l'utilisateur et sa capacité à répondre de manière cohérente dans une tolérance acceptée sur différentes charges d'utilisateurs.

Test de stress ou de résistance

Les tests de résistance sont utilisés pour trouver des moyens de mettre en défaut le système. Ils fournissent également la plage de charge maximale que le système peut supporter.

En règle générale, les tests de résistance ont une approche incrémentielle où la charge est augmentée progressivement. Le test démarre avec une charge pour laquelle l'application a déjà été testée. Ensuite, plus de charge est ajoutée lentement pour stresser le système. Le point auquel nous commençons à voir des serveurs ne répondant pas aux demandes est considéré comme le point de rupture.

COLLECTE DES DONNÉES

Test unitaires

Les cas de tests unitaire seront exécutés par les développeurs et ne généreront pas de données. Les tests en échec bloquant la validation du code produit, le code devra être modifié jusqu'à ce que celui-ci soit validé.

Tests d'intégration

Tests d'acceptation

Les cas de tests d'acceptation produisent des données résultant de ces tests. Ces données sont :

- Le résultat du test
- L'analyse éventuelle du résultat (découlant du résultat)
- La classification du test (découlant du résultat)

Les résultats des tests d'acceptation seront consignés dans un outil adapté.

Les tests en échec devront être consignés dans un outil de gestion de projet en suivant la procédure de suivi des demandes de correction.

Tests système

Les résultats des tests système sont consignés dans l'outil permettant leur exécution qui généra les rapports d'exécution nécessaires à l'analyse des résultats.

COMPOSANTS À TESTER

Les composants fonctionnels devant faire l'objet de tests seront :

- le service de recherche d'hôpital disponible '(/search/nearest)
- le service de réservation de lits (/bed/booking)
- le service d'annulation de réservation de lits (/bed/booking/cancel)

Les composants techniques devant faire l'objet de tests seront :

- le pipeline d'intégration continue Jenkins
- les conteneurs Docker

FONCTIONNALITÉS À TESTER

Test unitaires

Endpoint de recherche d'un hôpital disponible pour un localisation et une spécialité donnée

Cas positif

Il existe plusieurs hôpitaux pour la spécialité ayant des lits disponibles, la requête retourne l'hôpital le plus proche de la localisation recherchée.

La requête retourne le json de réponse pour l'hôpital trouvé ainsi que le code HTTP 200.

NB : Le test est créé en double en utilisant des spécialités différentes.

Cas négatif 1

Il n'existe pas d'hôpital où la spécialité est pratiquée.

La requête retourne le json de réponse sans hôpital avec un message indiquant la non disponibilité ("No nearest/available hospital found for specialty '<specialty_name>' @ <latitude>,<longitude>") ainsi que le code HTTP 404.

Cas négatif 2

Il n'y a pas de lits disponibles parmi les hôpitaux où la spécialité recherchée est pratiquée.

La requête retourne le json de réponse sans hôpital avec un message indiquant la non disponibilité ("No nearest/available hospital found for specialty '<specialty_name>' @ <latitude>,<longitude>") ainsi que le code HTTP 404.

Cas négatif 3

La requête est mal formulée (il manque un paramètre).

La requête retourne le code HTTP 400.

Endpoint de réservation d'un lit dans un hôpital donnée

Cas positif

Un lit est disponible dans l'hôpital donné.

Le nombre de lits est décrémenté (-1) dans la base de données.

La requête retourne le json de réponse pour l'hôpital concerné avec le nombre de lits disponibles mis à jour et incluant le message "Bed booked successfully in hospital <hospital_id>" ainsi que le code HTTP 200.

Cas négatif 1

Aucun lit n'est disponible dans l'hôpital donné.

Le nombre de lits n'est pas décrémenté dans la base de données.

La requête retourne le json de réponse pour l'hôpital concerné incluant le message "No bed available in Hospital <hospital_id>" ainsi que le code HTTP 409.

Cas négatif 2

L'hôpital n'est pas trouvé (n'existe pas).

Aucune action faite sur la base de données.

La requête retourne le json de réponse vide incluant le message "Hospital <hospital_id> not found" ainsi que le code HTTP 404.

Endpoint d'annulation d'une réservation d'un lit dans un hôpital donnée

Cas positif

L'hôpital est trouvé.

Le nombre de lits est incrémenté (+1) dans la base de données.

La requête retourne le json de réponse pour l'hôpital concerné avec le nombre de lits disponibles mis à jour et incluant le message "Booking cancelled successfully in hospital <hospital_id>" ainsi que le code HTTP 200.

Cas négatif 1

L'hôpital n'est pas trouvé (n'existe pas).

Aucune action faite sur la base de données.

La requête retourne le json de réponse vide incluant le message "Hospital <hospital_id> not found" ainsi que le code HTTP 404.

Cas négatif 2

L'hôpital est trouvé mais la mise à jour ne peut-être faite.

La requête retourne le json de réponse pour l'hôpital concerné avec le nombre de lits disponibles mis à jour et incluant le message "Unable to cancel booking in Hospital <hospital_id>" ainsi que le code HTTP 409.

Test d'intégration

Les tests d'intégration reprennent l'ensemble des tests unitaires de manière automatisée via les outils nécessaires à l'exécution du processus CI/CD.

Tests d'acceptation

Les cas de tests d'acceptation reprendront les caractéristiques et besoins métiers et utilisateurs définies dans les cahier de tests (qui seront à rédiger lors d'une étape ultérieure du projet).

Tests système

Test de charge

Le test de charge se basera sur la capacité de notre API à fournir une réponse en moins de 200 millisecondes tout en supportant une charge de travail de 800 requêtes par seconde (par instance de service).

Test de stress ou de résistance

Le test de stress se basera sur les caractéristiques du test de charge en y appliquant une caractère incrémental afin de mettre le système en défaut.

L'incrément s'appliquera sur le nombre de requêtes par seconde et se fera par tranches de 800 requêtes.

RISQUES & HYPOTHÈSES

Les risques suivants ont été identifiés pour ce projet :

Risque	Probabilité	Impact	Gravité	Stratégie d'atténuation
Le système d'intervention d'urgence en temps réel n'est pas adapté aux incidents	Inconnue	Élevé	Élevé	Test de performance précoce d'une preuve de concept représentative
Le système d'intervention d'urgence en temps réel gère la latence concernant la disponibilité des lits des hôpitaux du réseau	Inconnue	Élevé	Élevé	Test de performance précoce d'une preuve de concept représentative
Le système d'intervention d'urgence en temps réel n'offre pas de solution lorsqu'il n'y a pas de lits d'hôpital disponibles pour la spécialisation requise	Inconnue	Élevé	Élevé	Attribution à l'hôpital le plus proche disposant de lits
Le système d'intervention d'urgence en temps réel ne répond pas dans les 200 millisecondes à la demande de lits	Inconnue	Élevé	Élevé	Validation de principe pour vérifier qu'un paramètre d'urgence est en mesure de fournir au système de réponse le nom d'un hôpital disposant d'un lit en moins de 200 nanosecondes, pendant un pic d'activité
Le système d'intervention d'urgence ne peut pas être interfacé par d'autres systèmes	Moyen	Moyen	Faible	Utiliser OpenAPI pour définir les contrats de service. Inclure cela dans la première preuve de concept - personnaliser par la suite en tant que solution building blocks

OUTILS

Outils pour les tests unitaires

Les outils préconisés sont :

- Junit pour l'exécution des tests unitaires
- Jacoco pour la mesure de la couverture de code

Outils pour gérer les tests d'intégration

Les outils préconisés sont :

- Jenkins pour gérer les opérations DevOps

Outils pour gérer les tests système

Les outils préconisés sont :

- Jmeter, Flood.io ou LoadView pour les tests de charge (outils couplés à un outil d'intégration)

Outils pour les tests d'acceptation

Les outils préconisés sont :

- FitNesse pour les tests d'acceptation réalisés par les testeurs
- Cahier de tests pour les tests d'acceptation réalisés par les utilisateurs

APPROBATIONS

Nom & fonction	Date	Signature
Nicolas Oger (Architecte logiciel)		