MedHead

STRATEGIE DE TESTS

MedHead

STRATEGIE DE TESTS

1. Introduction

Dans la méthode de gestion de projet, la mise en place du projet doit s'organiser en plusieurs phases successives :

1. Phase d'expression des besoins
2. Phase de conception
3. Phase de réalisation
4. Phase de tests
5. Phase de livraison

Dans ce document, nous allons nous intéresser à la phase de tests qui est un moyen de vérifier de manière fiable les fonctionnalités des différents composants du système et éventuellement de détecter des erreurs.

Les testeurs doivent avoir en leur possession un plan de tests qui décrit les différentes procédures à réaliser sous la forme de scénario de tests.

2. Objectif

Ce document de stratégie de test décrit les exigences, les scénarios de test, les objectifs métier et qualitatifs, pour pallier aux risques liés au traitement des recommandations de lits d'hôpitaux dans des situations d'intervention d'urgence.

Dans le cadre de ce projet, il convient de tester les différents composants applicatifs mais aussi l'application entièrement avec l'ensemble de ses fonctionnalités, et nous allons appliquer la méthodologie BDD (Behavior Driven Development) pour réaliser ses tests.

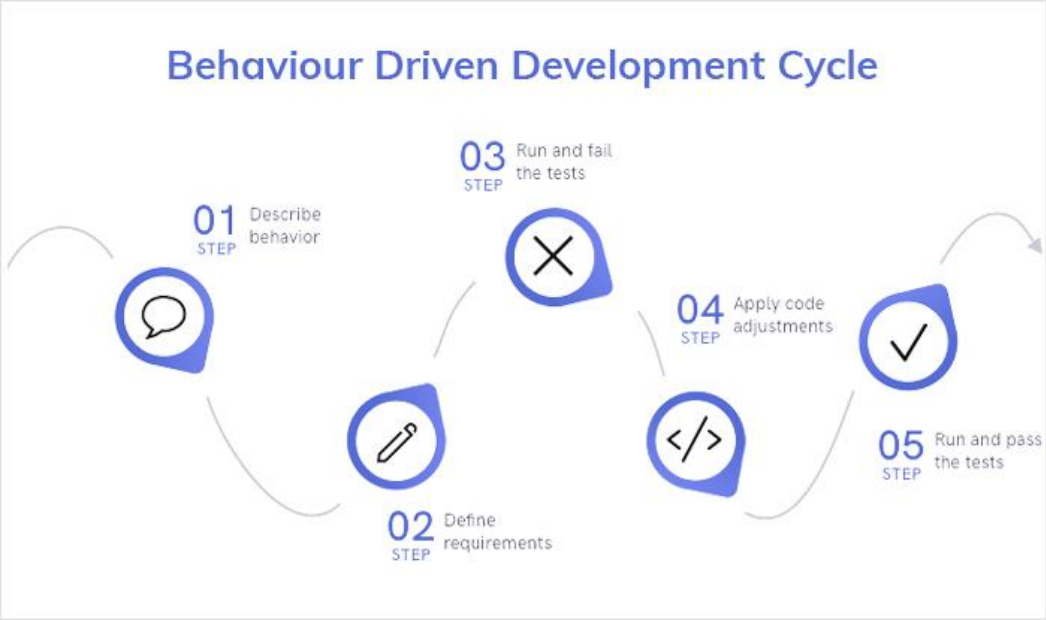
3. Les étapes de la BDD

La méthodologie BDD permet de tester pleinement les comportements de la plate-forme de streaming vidéos interactif. Il s'agit d'une méthodologie de travail, permettant d'écrire des tests compréhensibles à la fois par le client et par les développeurs et s'intégrant directement dans la base de code.

La structure hiérarchique qu'on va suivre dans le cadre du Behavior Driven Development :

1. Analyse des besoins afin de définir les tâches, les objectifs et les fonctionnalités du logiciel (déjà réalisée).
2. Après avoir identifié toutes les fonctionnalités, celles-ci sont décrites sous la forme de scénarios prédéfinis. On décrit toutes les situations possibles dans lesquelles la plate-forme de straming vidéo interactif doit réagir avec une réponse précise.
3. L'étape suivante consiste à définir les réponses attendues pour chaque scénario dans un **schéma de vérification (« étant donné-quand-alors »)**.
   « **Étant donné** » décrit la plate-forme avant le test,
   « **quand** », l'action pendant le test
   Et « **alors** » l'état du logiciel après le test.

Le BDD est matérialisé par le schéma suivant



Description du cycle de BDD :

1. **Décrire le comportement** : Cela inclut le flux et les caractéristiques du produit signifie la vision principale.
2. **Définir les besoins** : Exigences modélisées avec des règles métier pour une compréhension partagée.
3. **Exécutez et échouez les tests** : Développer et exécuter les cas de test.
4. **Appliquer la mise à jour du code** : Refactorisez-le en fonction de l'exigence.
5. **Exécuter et réussir les tests** : Exécutez le code mis à jour et réussissez les scénarios de test.

4. Les catégories de tests

Notre périmètre pour ce projet va se limiter à deux catégories de tests :

1. Tests unitaires
2. Tests d'intégration
3. Test d'acceptation
4. Test du système

Description du plan de test :

Dans le cadre de ce projet, nous allons commencer par:

1. Les tests unitaires : c'est un processus de vérification d'une unique unité de logiciel, elle-même définie comme étant la plus petite partie non divisible d'un code source.

Puis, on va faire les tests systèmes dans l'ordre suivant :

2. Test d'intégration : Les tests d'intégration sont effectués pour tester les modules / composants lorsqu'ils sont intégrés pour vérifier qu'ils fonctionnent comme prévu, c'est-à-dire pour tester les modules qui fonctionnent correctement individuellement ne pose pas de problèmes lorsqu'ils sont intégrés

3. Test d'acceptation : Les tests d'acceptation sont un type de test de logiciel qui se concentre sur les exigences du client et si le système ou le logiciel les remplit. Le but des tests d'acceptation est de s'assurer que le logiciel fonctionne bien pour les besoins de l'utilisateur.

4. Tests du système : Le but de ces test est de s'assurer que l'application continue de fonctionner normalement en temps de stress ou d montée en charge.

5. Détails de la stratégie de test

5.1. Tests unitaires

Un nombre croissant de tests unitaires ont été réalisés dans le cadre de la réalisation de ce POC dans le but de valider chaque portion de code de manière séparée.

5.2. Tests d'intégrations

Les endpoints de chaque microservice sont testé par la réalisation de ces tests. Ils nous permettent de valider le retour de chacun de nos endepoints.

5.3. Test d'acceptation

Les fonctionnalités de l'application, à savoir les besoins clients sont testés par la réalisation des tests d'acceptation.

5.3.1.Fonctionnalités

Dans le tableau ci-dessous sont listées les fonctionnalités à tester:

Microservices	Fonctionnalités
Hopital-service	- Rechercher un hopital en fonction du lieu d'incident et de la spécialité
Reservation-service	- Réserver un hopital

5.3.2.User stories

Dans cette section nous allons définir un ensemble de user stories permettant de définir et tester le projet avec clarté.

Comme indiqué au début du document, pour écrire les user stories nous allons appliquer le principe de la BDD qui consiste à écrire selon le paradigme :

1. **Etant que**
2. **Je souhaite**
3. **Afin que**

MedHead

STRATEGIE DE TESTS

Le tableau ci-dessous liste des user stories tenant compte des fonctionnalités du projet. C'est un tableau à 3 colonnes dont «Fonctionnalités», «User Stories», «Critères d'acceptation».

Fonctionnalités	User Stories	Critères d'acceptation
Rechercher un hopital avec un lit disponible	**En tant que** intervenant, **je souhaite** rechercher l'hopital le plus proche du lieu d'incident ayant au moins un lit disponible pour une spécialité donnée. **afin que** je puisse faire une réservation	**Étant donné que** j'accède à la plate-forme, **lorsque** je je renseigne le lieu de l'incident et la spécialité (qui existe), **alors** je dois avoir l'hôpital le plus proche ayant un lit disponible tenant compte de la spécialité demandée. **Étant donné que** j'accède à la plate-forme, **lorsque** je je renseigne le lieu de l'incident et la spécialité (qui n'existe pas), **alors** je dois être informé du problème par un message d'erreur et aucun hôpital ne doit être retourné.
Réserver l'hopital	**En tant que** intervenant, **je souhaite** réserver un lit dans un hôpital pour la spécialité demandée, suite à ma recherche. **afin que** je puisse avoir une réservation de lit enregistré	**Étant donné que** je reçois l'hôpital le plus proche ayant un lit disponible selon la spécialité que j'ai renseignée, **lorsque** je saisis mon nom d'intervenant et que je réserve l'hôpital, **alors** Une réservation de lit doit être enregistrée en mon nom dans cet hôpital et pour la spécialité demandée. **Étant donné que** je reçois l'hôpital le plus proche ayant un lit disponible selon la spécialité que j'ai renseignée, **lorsque** je saisis mon nom d'intervenant et que je réserve l'hôpital, si entre-temps l'hopital n'a plus de disponibilité **alors** Un message doit m'informer de l'indisponibilité de l'hôpital et la réservation de lit ne doit pas être

		enregistrée.

5.3.3.Outils utilisés

Le tableau ci-dessous nous présente l'outil qui est utilisé pour l'automatisation de ces tests d'acceptation par l'utilisateur :

Nom	Description
Jenkins	CI/CD pipelines (intégration continue)
Maven	Gestion des dépendances (facilite)
Jacoco	Réalisation de la Couverture de code testé
JUnit	Réalisation des test
JMeter	Tests de montée en charges

KPI - Indicateur clé de performance

Dans le tableau ci-dessous sont listés les KPI mais aussi la description de chacun d'eux y compris des valeurs d'acceptation.

KPI	Description
Montée en charge	Supporter 800 requêtes par seconde et a une réponse en 200 milisecondes.
Acheminement vers l'hôpital le plus proche	Dans 90% des cas, le patient doit être acheminé vers l'hôpital le plus proche

5.4. Tests du système

5.4.1. Tests de montée en charge

Le test de montée en charge permet d'évaluer la capacité de l'application à supporter de plus en plus d'usagers tout en maintenant une expérience utilisateurs optimale et un fonctionnement correspondant aux cahier des charges.

Le test de montée en charge doit permettre de comprendre :

1. Le nombre d'utilisateurs que l'application/site peut prendre en charge simultanément ;
2. La capacité opérationnelle maximale de l'application ;
3. Si l'infrastructure actuelle peut permettre aux utilisateurs de profiter de l'application de façon optimale ;
4. La durabilité de l'application quand elle est soumise à la charge de pointe des utilisateurs.

5.4.2. Outils utilisés

Le tableau ci-dessous nous présente l'outil qui sera utilisé pour l'automatisation des tests de performance :

Nom	Avantages
Jmeter	1. Facile d'utilisation et son langage est très intuitif. Une simple connaissance de Java suffit pour s'en servir. 2. Permet de réaliser des scénarios (appels de différentes fonctions dans un ordre et avec des données précises). 3. Open-source