PLAN DE TEST

Projet : PoC pour le sous-système d'intervention d'urgence en temps réel

Client : MedHead

HISTORIQUE DES RÉVISIONS

Date	Version	Commentaires
2 janvier 2022	0.01	Plan de test préliminaire.

TABLE DES MATIÈRES

INTRODUCTION

Dans ce document, nous allons décrire la portée, l'approche, les ressources, les fonctionnalités et les types de tests à effectuer dans le cadre du projet de développement d'une preuve de concept pour le sous-système d'intervention d'urgence en temps réel souhaité par le consortium *MedHead*. Dans la dernière partie de ce plan de test, vous trouverez les scénarios prenant en compte le comportement attendu des différentes fonctionnalités.

PORTÉE

Toutes les fonctionnalités des microservices développés pour la preuve de concept ont été testées. Le tableau ci-dessous rappelle leurs noms et fournit une brève description pour chacune d'entre elles.

Microservices	Id	Fonctionnalité	Description
Emergency	1	Create	Une demande de réservation de lit est publiée à partir d'une spécialisation médicale, du lieu d'incident et de l'identité du patient.
	2	List	Toutes les demandes de réservation d'un lit sont consultables sous la forme d'une liste qui nous renseigne sur le patient, l'hôpital et la spécialité.
Hospital	3	Create	Ajout d'un nouvel hôpital. Un nom, le nombre de lits disponibles et les spécialités doivent être renseignés.
	4	Read	Permet de sélectionner un ou plusieurs hôpitaux.
	5	Update	Mise à jour des informations liées à un hôpital, en particulier le nombre de lits disponibles.
	6	Delete	Suppression d'un hôpital.
Patient	7	Create	Ajout d'un nouveau patient. Un nom de famille, un prénom et un id doivent être renseignés.
	8	Read	Permet de sélectionner un ou plusieurs patients.
	9	Update	Mise à jour des informations liées à un patient.
	10	Delete	Suppression d'un patient.

Tableau 1 - Les descriptions des fonctionnalités groupées par composants

Au moment de la rédaction de ce document, la persistance des données et l'interface avec les systèmes auxiliaires sont implémentés de manière factice afin de simplifier la réalisation du projet en se concentrant principalement sur les exigences convenues dans la déclaration d'hypothèse.
Il sera toujours possible d'ajouter ces tests lorsque les fonctionnalités initiales et l'architecture cible auront été déployées.

OBJECTIFS

Les objectifs des tests sont de vérifier la conformité des nouvelles fonctionnalités avec les exigences convenues lors de la définition de l'hypothèse de travail, à savoir, *« une API RESTful tient les intervenants médicaux informés en temps réel »* et *« la POC est validée avec des tests reflétant la pyramide de test (unitaires, d'intégration et de bout en bout) »*. Par conséquent, nous avons élaboré une stratégie de test basée sur les récits des utilisateurs décrivant les interactions avec le système afin de garantir que toutes les fonctionnalités sont adaptées aux attentes de la société et qu'elles peuvent fonctionner de manière optimale dans un environnement de production.

MÉTHODOLOGIE

Afin d'élaborer les tests, nous avons pris comme point de départ le scénario donné en exemple dans l'hypothèse de travail et le principe d'architecture C4 (Favoriser une culture de "learning" avec des preuves de concept, des prototypes et des Spike). Nous avons donc eu recours au BDD (Behavior Driven Development), un processus agile, qui encourage la collaboration entre les développeurs, les testeurs et les utilisateurs de la solution. Il combine des techniques et les principes généraux du TDD (Test Driven Development) avec le DDD (Domain Driven Design) et a pour objectif de formuler les exigences des utilisateurs sous la forme de tests d'acceptation avant l'élaboration de tests de plus bas niveau. Le comportement du programme est décrit par les utilisateurs, qui n'ont pas nécessairement besoin de comprendre le jargon technique des développeurs puisque les tests sont créés à partir du langage business. D'abord, des fichiers de fonctionnalités non techniques contiennent les scénarios de test et précisent les étapes pour définir le code. Ensuite, on crée les tests fonctionnels de bout en bout qui représentent les parcours de nos utilisateurs. Puis, viennent les tests d'intégration qui permettent de tester les interactions entre différents composants. Enfin, nous arrivons au plus bas niveau avec les tests unitaires pour valider le comportement de nos classes par exemple. L'ensemble des tests est automatisé à l'aide d'outils techniques que nous allons détailler dans la suite de ce document.

La *Figure 1* ci-dessous schématise le processus appelé red-green-refactor et donne une vision globale de l'enchaînement des étapes décrites précédemment.

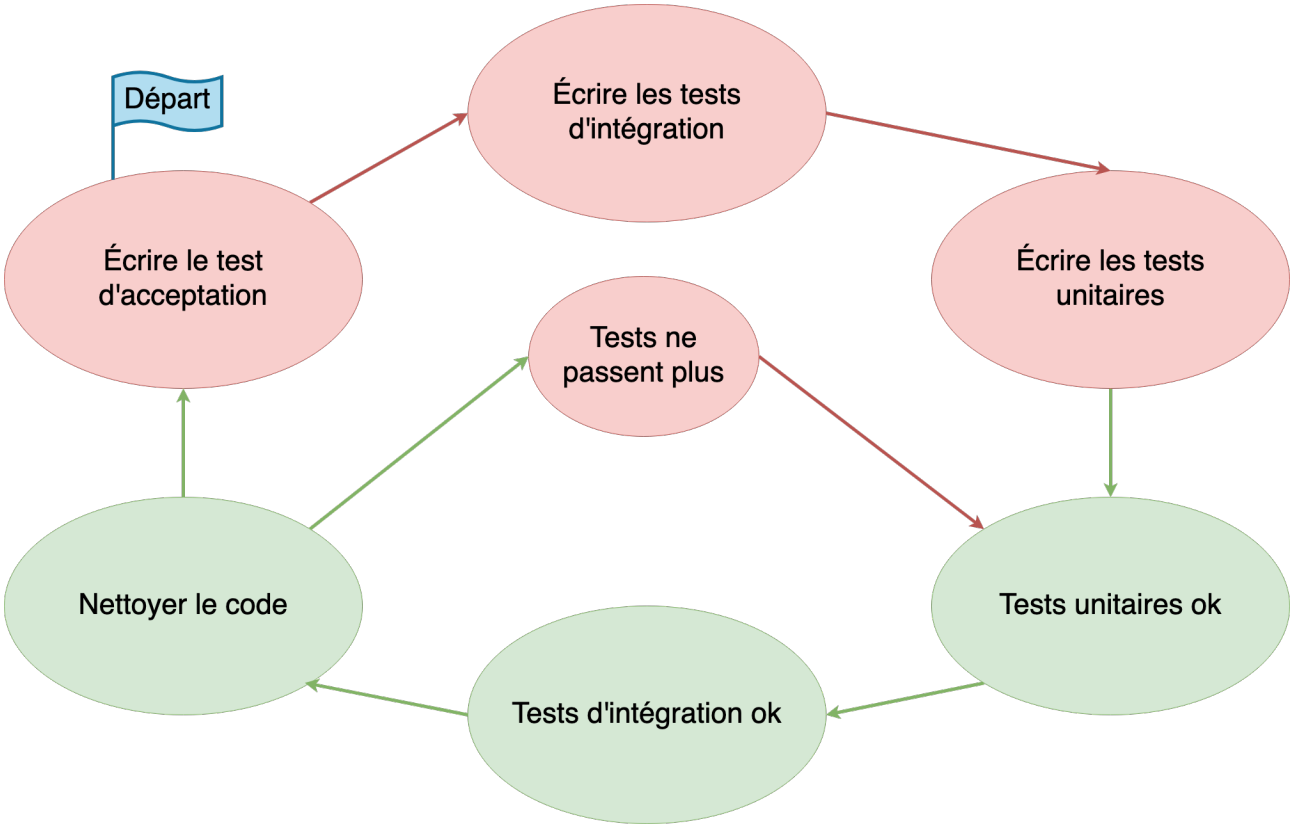

Figure 1 - L'approche red, green, refactor

Un autre aspect de la méthodologie employée pour élaborer des tests unitaires performants, est le principe F.I.R.S.T, issu lui aussi du développement agile. Vous trouverez plus de détails dans le livre Clean Code sur ce sujet. Voici une description de chacune des propriétés de ce principe :

- Fast. Des milliers de tests doivent pouvoir être exécutés en quelques secondes;
- Isolates. Quand un test échoue, il est facile de savoir pourquoi;
- Repeatable. Donne toujours le même résultat malgré l'ordre ou le moment d'exécution;
- Self-validating. Aucune évaluation manuelle n'est nécessaire, le framework de test exécute et génère des rapports de tests clairs.
- Timely. Sont écrits avant le code

Pour conclure ce chapitre, nous allons lister deux types de tests favorisant une définition adaptée qui couvrent l'ensemble des possibilités liées à notre application.

- Les tests de cas limites (Edge cases en anglais) sont des tests conçus pour vérifier l'inattendu aux limites du système et des données.
- Les tests de cas pathologiques (Corner cases en anglais) testent des situations improbables dans lesquelles notre application pourrait se retrouver.

LIVRABLES

Dans cette partie, nous allons lister les livrables attendus pendant la mise en oeuvre du plan de test. Nous précisons que les différents journaux et rapports générés pendant et après l'exécution des tests, correspondent à la manière dont les données des tests vont être collectées.

Avant la phase de test :

- Plan de test
- Scénarios de tests
- Mesures des tests

Pendant les tests :

- Journaux d'exécution
- Journaux d'erreurs

Après l'exécution des tests :

- Rapports d'exécution des tests
- Rapports sur la qualité du code
- Rapports sur la couverture du code par les tests

OUTILS

Tests unitaires

Nous utilisons le framework *JUnit5* et la bibliothèque *AssertJ* pour tester les applications *Spring Boot* dans le but de créer un code lisible, proche du langage naturel, et simple d'utilisation pour les développeurs. L'ajout de la bibliothèque *Mockito* permet la création d'acteur de remplacement, appelé doublure de test, afin de simuler le comportement des dépendances et de s'assurer que nos tests respectent le principe d'isolation. Grâce à la classe *BDDMockito* nous avons organisé nos tests en suivant une structure à trois niveaux (given, when, then) semblable à celle décrivant les scénarios.

Tests d'intégration

Tout d'abord, nous précisons que les tests d'intégration développés pour la PoC ne sont pas encore inclus dans le pipeline CI/CD et font l'objet d'une follow-up issue. Dans notre environnement de développement, nous employons *Postman* pour tester les APIs, et, en complément, nous avons recours au framework *Cypress* que nous utilisons pour automatiser l'exécution de nos tests.

Tests fonctionnels de bout en bout

Pour effectuer cette tâche, nous utilisons aussi le framework *Cypress*. Ce framework JavaScript nous permet de tester les parcours utilisateurs et différents éléments de l'interface réalisée avec la bibliothèque JavaScript *React*. Ces tests doivent eux aussi être intégrés dans le pipeline CI/CD.

Tests d'acceptation

En nous basant sur le scénario partagé dans l'hypothèse de travail et dans le but de compléter cet exemple en suggérant d'autres cas d'utilisation du système d'intervention d'urgence, nous avons utilisé le langage *Gherkin*. Cela nous offre un moyen d'écrire des tests que tout le monde peut comprendre, dans la langue de notre choix. Nous avons rédigés les scénarios ou les tests d'acceptation qui décrivent le comportement du système, du point de vue des utilisateurs, en exploitant un ensemble commun de mots-clés (Feature, Given, When, Then, etc). Vous trouverez une liste des tests d'acceptation pour les fonctionnalités de ce projet dans le chapitre *Scénarios* de ce document.

Autres

L'étude de la couverture du code par les tests permet de vérifier si nos tests parcourent l'ensemble de notre code. Nous utilisons *JaCoCo* (Java Code Coverage) pour nos applications Java.
L'audit automatisé de code permet de vérifier si notre code est bien construit, qu'il ne produit pas de bugs connus ou certaines failles de sécurité. Pour tester la qualité du code nous utilisons *Code Climate* et son plugin *SonarJava,* un analyseur de code pour les projets Java, qui effectue cet audit et affiche le résultat sous forme de rapports détaillés et interactifs.

Afin de conclure cette partie, nous partageons d'autres outils de test que nous n'avons pas installés, faute de spécifications de cas d'utilisation réels d'une part, mais surtout par manque d'accès à un volume de données large et distribué justifiant la pertinence de ces tests.
Les tests de performance peuvent être mis en place pour vérifier la rapidité d'exécution de notre application grâce à *JMeter*.
Pour s'assurer que l'application reste fonctionnelle lors d'un usage intensif réalisé par un grand nombre d'utilisateurs, nous pouvons utiliser l'outil Open-Source *Gatling*.
Une fois l'architecture microservice déployée, l'ingénierie du chaos et les outils associés à cette pratique, comme *Chaos Monkey*, créé par *Netflix,* visent à identifier les défaillances du système avant qu'elles ne deviennent des pannes afin de construire des systèmes plus résistants et peuvent être utilisés dans l'architecture distribuée retenue pour la plateforme d'allocation de lits d'hôpital pour les urgences.

Et enfin, la mise en place de tests de sécurité, vérifiant que notre application ne peut pas être exploitée par un tiers, à l'aide de l'outil *ZAP* (Zed Attack Proxy) maintenu par l'*OWASP* (Open Web Application Security Project).

ENVIRONNEMENT

La *Figure 2* ci-dessous représente l'intégration des outils listés dans le chapitre précédent avec le logiciel Open-Source *GitLab*. Nous avons adopté cette plateforme DevOps pour mettre en place un pipeline d'intégration continue. Cela présente l'avantage de travailler dans un environnement permettant d'automatiser l'exécution des tests. Ainsi, à chaque nouveau push des développeurs, nous nous assurons que les nouvelles fonctionnalités ne créent pas de régression dans le code. Des rapports sont générés à chaque exécution et présentés de manière lisible à travers une interface web, ce qui est un avantage pour la communication entre les acteurs de différents domaines impliqués dans le projet. Le livrable intitulé « Description du Pipeline » contient tous les détails sur le travail effectué lors de cette mise en place.

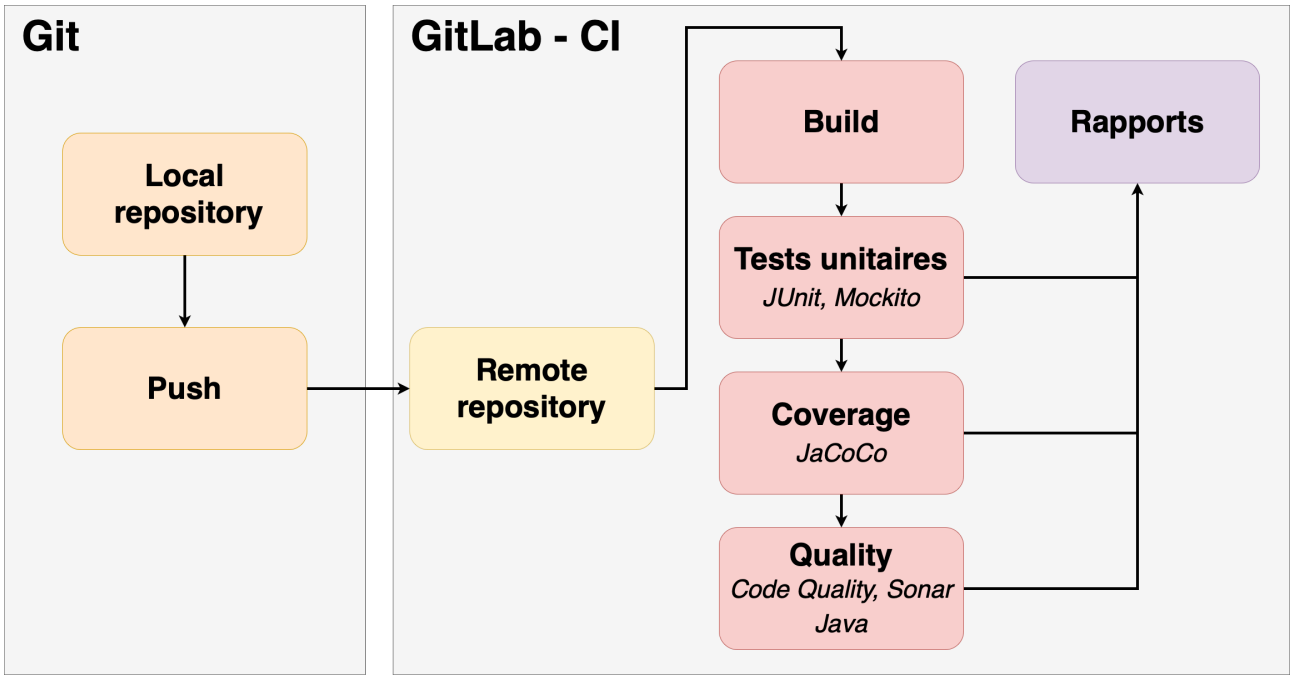

Figure 2 - Vue simplifiée de l'intégration des outils avec Gitlab

SCÉNARIOS

Dans ce chapitre nous proposons la rédaction des tests d'acceptation pour chacune des fonctionnalités attendues afin de fournir des contrôles pour vérifier la conformité de notre projet. Nous utilisons les mots-clés de la syntaxe *Gherkin* et avons choisi l'utilisation du français comme langue de référence.

Fonctionnalité: Create Emergency (id 1)

Scénario : Un utilisateur fait une demande de réservation de lit en urgence en spécifiant un lieu, une spécialité et l'id du patient.

Sachant que le formulaire de création d'une urgence est affiché
Lorsque l'utilisateur renseigne le champ location avec Fred Brooks
Et le champ speciality avec cardiologie
Et le champ id avec 1
Et qu'il clique sur le bouton submit
Alors une urgence est créée avec un nouvel id, l'id (1) le nom (Brooks) le prénom (Fred) du patient et l'id (1) le nom de l'hôpital (Fred Brooks) et la spécialité (cardiologie) et une option lit disponible (true)

Scénario : Un utilisateur fait une demande de réservation de lit en urgence en spécifiant un lieu, une spécialité.

Sachant que le formulaire de création d'une urgence est affiché
Lorsque l'utilisateur renseigne le champ location avec Beverly Bashir
Et le champ speciality avec immunologie
Et qu'il clique sur le bouton submit
Alors une urgence est créée avec un nouvel id, l'id (999) le nom (UNKNOWN) le prénom (PATIENT) du patient et l'id (3) le nom de l'hôpital (Beverly Bashir) et la spécialité (immunologie) et une option lit disponible (true)

Scénario : Un utilisateur fait une demande de réservation de lit en urgence en spécifiant un lieu.

Sachant que le formulaire de création d'une urgence est affiché
Lorsque l'utilisateur renseigne le champ location avec Beverly Bashir
Et qu'il clique sur le bouton submit
Alors une urgence est créée avec un nouvel id, l'id (999) le nom (UNKNOWN) le prénom (PATIENT) du patient et l'id (3) le nom de l'hôpital (Beverly Bashir), la spécialité n'est pas renseignée et une option lit disponible (true)

Scénario : Un utilisateur fait une demande de réservation de lit en urgence en spécifiant une spécialité.

Sachant que le formulaire de création d'une urgence est affiché
Lorsque l'utilisateur renseigne le champ spécialité avec immunologie
Et qu'il clique sur le bouton submit
Alors une urgence est créée avec un nouvel id, l'id (999) le nom (UNKNOWN) le prénom (PATIENT) du patient et l'id (3) le nom de l'hôpital (Beverly Bashir), la spécialité (immunologie) et une option lit disponible (true)

Scénario : Un utilisateur fait une demande de réservation de lit en urgence en spécifiant une spécialité et l'id du patient.

Sachant que le formulaire de création d'une urgence est affiché
Lorsque l'utilisateur renseigne le champ speciality avec neuropathologie diagnostique
Et le champ id avec 2
Et qu'il clique sur le bouton submit
Alors une urgence est créée avec un nouvel id, l'id (2) le nom (Crusher) le prénom (Julia) du patient et l'id (3) le nom de l'hôpital (Beverly Bashir), la spécialité (neuropathologie diagnostique) et une option lit disponible (true)

Scénario : Un utilisateur fait une demande de réservation de lit en urgence en spécifiant un lieu et l'id du patient.

Sachant que le formulaire de création d'une urgence est affiché
Lorsque l'utilisateur renseigne le champ location avec Beverly Bashir
Et le champ id avec 3
Et qu'il clique sur le bouton submit
Alors une urgence est créée avec un nouvel id, l'id (3) le nom (Bashir) le prénom (Beverly) du patient et l'id (3) le nom de l'hôpital (Beverly Bashir), la spécialité n'est pas renseignée et une option lit disponible (true)

Fonctionnalité: List Emergencies (id 2)

Scénario : Un utilisateur consulte la liste des réservations de lit en temps réel.

Sachant que la page d'accueil est affichée
Lorsque l'on fait défiler la page vers le bas
Alors une liste des réservations de lit est consultable et affiche toutes les informations concernant le patient, l'hôpital, la spécialité et si un lit est disponible

Fonctionnalité: Create hospital (id 3)

Scénario : Un utilisateur fait une demande de création d'un nouvel hôpital.

Sachant que le formulaire de création d'hôpital est affiché
Lorsque l'utilisateur renseigne le champ nom avec Beverly Bashir
Et le champ nombre de lits avec 23
Et le champ spécialités avec cardiologie et immunologie
Alors un nouvel hôpital est crée avec un id, un nom (Beverly Bashir), le nombre de lits (23) et les spécialités médicales disponibles (cardiologie et immunologie)

Fonctionnalité: Read hospital (id 4)

Scénario : Un utilisateur consulte les informations d'un hôpital en spécifiant son nom.

Sachant que le formulaire de consultation des hôpitaux est affiché
Lorsque l'utilisateur renseigne le champ nom avec Fred Brooks
Alors les informations de l'hôpital Fred Brooks s'affichent à l'écran

Scénario : Un utilisateur consulte la liste de tous les hôpitaux.

Sachant que la page d'accueil des hôpitaux est affichée
Lorsque l'utilisateur demande la liste des hôpitaux
Alors une liste contenant tous les hôpitaux est affichée à l'écran

Scénario : Un utilisateur récupère une liste des hôpitaux avec au moins un lit disponible.

Sachant que la liste des hôpitaux est affichée
Lorsque l'utilisateur sélectionne l'option avec des lits disponibles
Alors une nouvelle liste filtrée avec l'option lit disponible est présentée

Scénario : Un utilisateur récupère le nombre de lits disponibles d'un hôpital.

Sachant que la formulaire de recherche des hôpitaux est affiché
Lorsque l'utilisateur renseigne le champ nom avec Fred Brooks
Et coche l'option lit disponible
Alors une réponse est retournée à l'utilisateur indiquant si oui ou non un lit est disponible

Scénario : Un utilisateur récupère une liste d'hôpitaux pratiquant une spécialité médicale.

Sachant que la formulaire de recherche des hôpitaux est affiché
Lorsque l'utilisateur renseigne le champ spécialité avec cardiologie
Alors s'affiche une liste des hôpitaux pratiquant cette spécialité

Scénario : Un utilisateur récupère une liste d'hôpitaux avec au moins un lit disponible et pratiquant une spécialité médicale.

Sachant que la formulaire de recherche des hôpitaux est affiché
Lorsque l'utilisateur renseigne le champ spécialité avec cardiologie
Et qu'il coche l'option lit disponible
Alors s'affiche une liste des hôpitaux avec au moins un lit disponible et pratiquant cette spécialité

Fonctionnalité: Update hospital (id 5)

Scénario : Un utilisateur veut modifier une information concernant un hôpital.

Sachant que le formulaire de modification est affiché
Lorsque l'utilisateur change les informations contenues dans un champ
Et qu'il valide les changements
Alors les informations de l'hôpital sont mises à jour

Scénario : Un utilisateur peut soustraire un lit au total des lits disponibles d'un hôpital.

Sachant que les informations d'un hôpital sont affichées
Lorsque l'utilisateur interagit avec le bouton moins
Alors le nombre de lits disponibles est diminué de un

Fonctionnalité: Delete hospital (id 6)

Scénario : Un utilisateur peut supprimer un hôpital.

Sachant que la liste des hôpitaux est affichée
Lorsque l'utilisateur clique sur le bouton supprimer
Et qu'il confirme son choix
Alors l'hôpital sélectionné est supprimé

Fonctionnalité: Create patient (id 7)

Scénario : Un utilisateur fait une demande de création d'un nouveau patient.

Sachant que le formulaire de création de patient est affiché
Lorsque l'utilisateur renseigne le champ nom avec Keller
Et le champ prénom avec Helen
Alors un nouveau patient est crée avec un id, un nom (Keller) et un prénom (Helen)

Fonctionnalité: Read patient (id 8)

Scénario : Un utilisateur récupérer les informations d'un patient avec son id

Sachant que le formulaire de recherche de patient est affiché
Lorsque l'utilisateur renseigne le champ id avec 1
Alors le nom Brooks et le prénom Fred sont affichés

Scénario : Un utilisateur consulte la liste de tous les patients.

Sachant que la page d'accueil des patients est affichée
Lorsque l'utilisateur demande la liste des patients
Alors une liste contenant tous les patients est affichée à l'écran

Scénario : Un utilisateur consulte les informations d'un patient en spécifiant son nom.

Sachant que le formulaire de recherche des patients est affiché
Lorsque l'utilisateur renseigne le champ nom avec Brooks
Alors les informations du patient Fred Brooks s'affichent à l'écran

Fonctionnalité: Update patient (id 9)

Scénario : Un utilisateur veut modifier une information concernant un patient.

Sachant que le formulaire de modification est affiché
Lorsque l'utilisateur change les informations contenues dans un champ
Et qu'il valide les changements
Alors les informations du patient sont mises à jour

Fonctionnalité: Delete patient (id 10)

Scénario : Un utilisateur peut supprimer un patient.

Sachant que la liste des patients est affichée
Lorsque l'utilisateur clique sur le bouton supprimer
Et qu'il confirme son choix
Alors le patient sélectionné est supprimé