



DOCUMENT DE STRATEGIE DE TEST

Imamou Chaima
Architecte logiciel MedHead+

Table des matières

I.	Contexte	2
II.	Périmètre métier	2
III.	Organisation du projet	2
A.	Les étapes à suivre pour mettre en pratique les tests BDD	3
B.	Les différentes catégories de tests du projet	4
IV.	Plan de test	4
A.	La phase de test	4
B.	Test d'acceptation	5
1.	Test d'acceptation orientés scénario	5
2.	Les User story	5
C.	Test unitaire et d'intégration	6
1.	Description des fonctionnalités à tester	6
2.	Micro-service gestion des hôpitaux	6
a)	Fonctionnalité n° 1 :	6
b)	Fonctionnalité n° 2 :	7
c)	Fonctionnalité n° 3 :	7
d)	Résultat des tests	8
3.	Micro-service réservation lit	8
a)	Fonctionnalité n° 1 :	9
b)	Résultat des tests	9
V.	Les différents tests automatisés misent en place pour ce PoC	10
VI.	Les différents rapports concernant les tests	12
VII.	Outils de gestion et d'automatisation des tests	13

I. Contexte

Le groupe MedHead+ veut mettre en place une plateforme qui pallie les risques liés au traitement des recommandations de lits d'hôpitaux. Le groupe décide de se concentrer sur le système d'intervention d'urgence qui permet l'attribution en temps réel de lits d'hôpital en fonction de la pathologie.

Cependant, il a été décidé de mettre en place une preuve de concept afin de valider les différentes exigences et hypothèses du comité.

II. Périmètre métier

Les fonctionnalités qui entrent dans le périmètre du projet sont les suivantes :

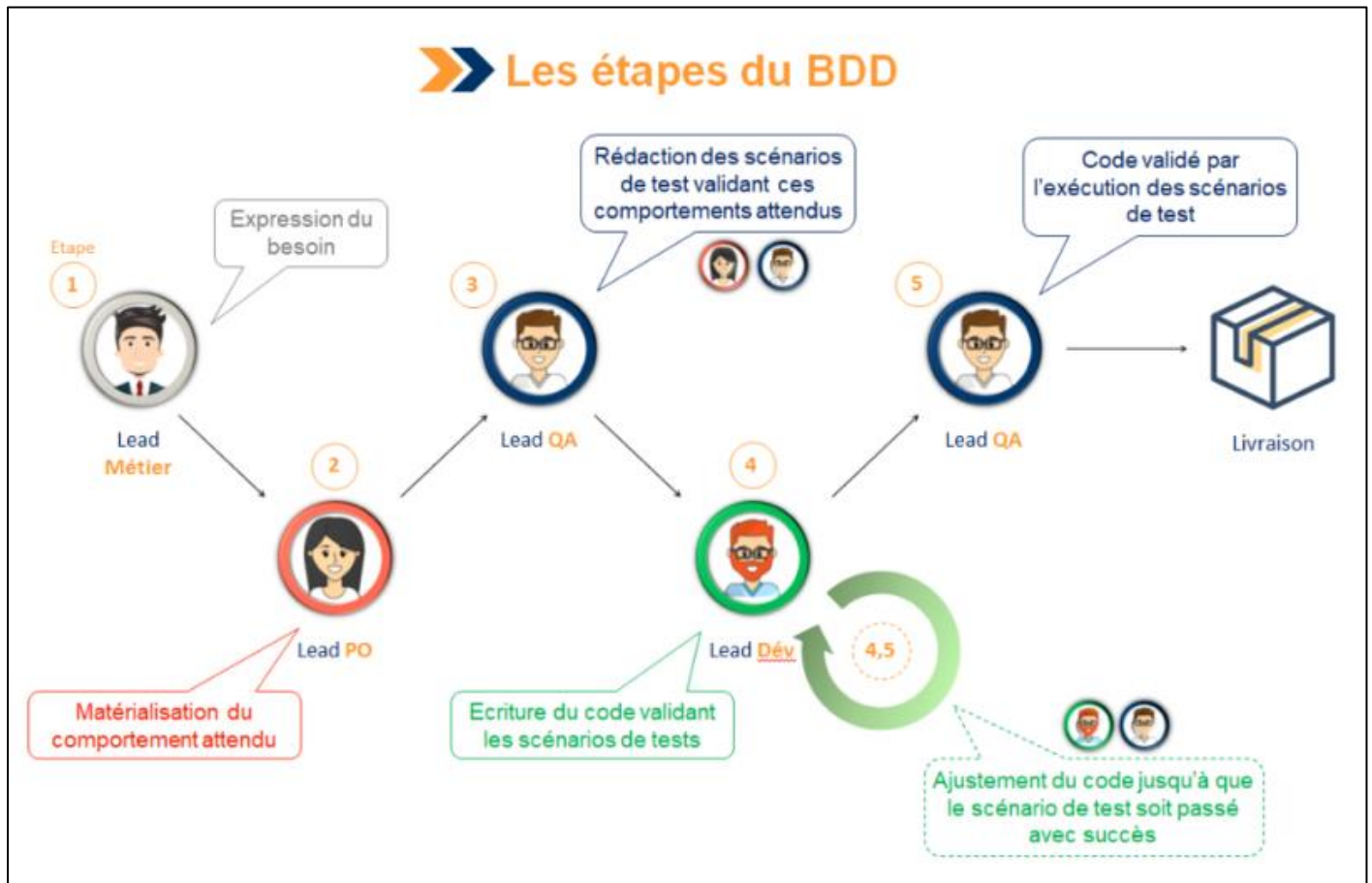
Fonctionnalité	Description
Saisir le lieu de l'incident d'urgence	Le personnel médical peut saisir le lieu de l'incident (en cas d'urgence)
Rechercher un hôpital en fonction de la spécialisation et du lieu de l'incident	En fonction de la localisation et de la spécialisation, l'hôpital le plus proche de l'incident doit être proposé (en fonction de la disponibilité des lits)
Vérifier si dans l'hôpital en question il y a un lit disponible	En fonction de la localisation et de la spécialisation, vérifier si l'hôpital possède des lits disponibles
Réservez un lit d'hôpital au nom du patient	Si un lit est disponible, un événement doit être déclenché pour réserver un lit

III. Organisation du projet

Nous travaillons sur le modèle agile (Scrum). Cependant, nous envisageons de faire des réunions quotidiennes pour la mise en place de la totalité de l'application. Nous serons donc amenés à solliciter l'équipe métier afin de vous faire des démonstrations des applications en cours de mise en place, afin d'avoir un retour des utilisateurs.

Dans notre phase de test, nous utilisons la méthode **BDD (Behavior-Driven Development)**. Cette dernière repose sur la modélisation des besoins de l'entreprise et utilise ces modèles pour guider la conception. Il s'oriente vers le comportement que les logiciels doivent avoir.

A. Les étapes à suivre pour mettre en pratique les tests BDD



Comme annonc  pr c demment, nous avons mis en place les tests en utilisant **l'approche BDD**. Cette approche peut se faire de mani re collective favorisant ainsi les  changes entre les diff rentes entit s gr ce   un langage naturel qui permet aux personnes m tiers de se faire mieux comprendre et aux d veloppeurs de r aliser les besoins exprim s.

Les diff rentes  tapes de la BDD :

- **L'expression du besoin :**

Une  tape primordiale est de recueillir les besoins, les attentes du projet    laborer .

- **La mod lisation du comportement attendu :**

Une r union est r alis e dans le but de discuter et de soulever toutes les questions sur le fonctionnement r el de l'application, d'identifier tous les diff rents comportements couvrant l'attente du client.

- **La r daction des sc narios de test validant ces comportements attendus :**

C'est dans cette partie qu'on d crit et r dige les attentes et comportement en diff rents sc narios d'utilisation.

- **L'écriture du code**

Une fois que les besoins ont été compris, et que chaque cas d'utilisation a été illustrés en scénarios, les développeurs peuvent écrire les tests. Les différents scénarios vont être la base pour écrire les tests.

- **Validation du code par l'exécution des scénarios de test et la livraison**

On exécute le scénario de test afin de vérifier si le test est valide ou pas. S'il est valide le test passe en production. Le cas contraire le code sera ajuster jusqu'à ce que les scénarios de test soient validés avec succès.

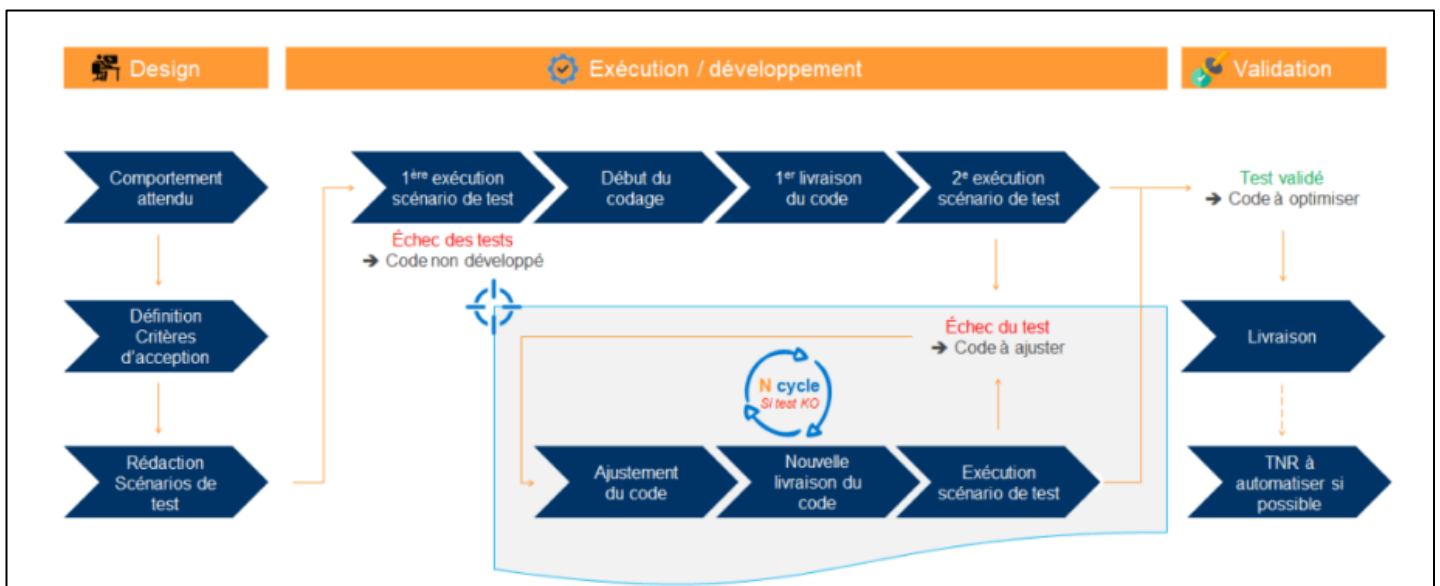
B. Les différentes catégories de tests du projet

Ce plan de test va couvrir ces catégories de tests :

- **Les tests unitaires et d'intégration** : nous avons utilisé JUnit et Mockito pour la réalisation de ces tests.
- **les tests d'acceptation (TA)** : qu'on appelle également tests de validation ou de conformité. Ils permettent de garantir la conformité entre le produit livré et les règles fonctionnelles définies au préalable par le client.

IV. Plan de test

A. La phase de test



Nous avons automatisé tous les tests qui sont développés dans ce projet.

Les différentes étapes réalisées :

- L'écriture de scénario
- Développement des tests
- Exécution des tests
- Livraison

B. Test d'acceptation

Nous allons nous baser sur **les tests d'acceptance** (ou test d'acceptation) sur ce projet. Pour savoir si le produit livré correspond aux besoins, il lui suffira de s'assurer que les cas d'utilisation sont respectés. Ces cas d'utilisation, garants du produit, sont regroupés dans les tests d'acceptation.

1. Test d'acceptation orientés scénario

Sur ce projet, nous avons utilisé des scénarios pour décrire les fonctionnalités et les attentes. Les scénarios se présentent sous une forme standardisée (sous le nom de type *Given / When / Then* (GWT)). Il comporte un titre et des instructions :

1. **Etant donné (Given)** : je raconte le contexte, la situation initiale, l'écran sur lequel je me trouve...
2. **Quand (When)** : je parle de l'action utilisateur ou système (un clic, un changement de valeur...)
3. **Alors (Then)** : qu'est-ce que j'ai en retour ? comment le système a été modifié ?

2. Les User story

Nous nous sommes appuyés sur les différentes fonctionnalités des micro-services pour illustrer les différents scénarios de tests d'acceptation :

En effet, nous avons deux cas possibles pour chaque scénario :

- Scénario positif de couleur verte (le cas où l'action marche)
- Scénario négatif de couleurs noire (le cas où l'action marche)

Fonctionnalités	Descriptions
Chercher et suggérer l'hôpital le plus proche de l'incident	<i>Etant donné qu'il y a un incident</i> <i>Quand le lieu de l'incident est renseigné</i> <i>Et que la pathologie est sélectionnée</i> <i>Alors le système m'affiche l'hôpital de la ville sélectionné</i>

	<i>Etant donné qu'il y a un incident</i> <i>Quand le lieu de l'incident est renseigné</i> <i>Et que la pathologie n'est pas sélectionnée</i> <i>Alors un message d'erreur s'affiche</i>
Réserver un lit d'hôpital selon la disponibilité	<i>Etant donné qu'il y a un incident</i> <i>Quand il y a un hôpital qui est proposé</i> <i>Et qu'on a saisi le nom et prénom du patient</i> <i>Alors le système déclenche un événement pour réserver un lit en fonction du nom du patient. Un message de succès s'affiche</i>
	<i>Etant donné qu'il y a un incident</i> <i>Quand il y a un hôpital qui est proposé</i> <i>Et qu'il n'y a pas de place disponible sur les hôpitaux les plus proches de l'incident</i> <i>Alors un message en rouge s'affiche « il n'y a pas de lit disponible dans cet hôpital »</i>

C. Test unitaire et d'intégration

1. Description des fonctionnalités à tester

Pour tester les différentes fonctionnalités, nous avons développé des tests unitaires et d'intégration.

Nous avons utilisé Junit et les *mock* (avec Mockito). Les *mocks* sont des objets simulés qui reproduisent le comportement d'objets réels de manière contrôlée.

2. Micro-service gestion des hôpitaux

Pour cette partie, nous aurons besoin de l'objet Hôpital (contient les hôpitaux et ces données (exemple le nom, le nombre de lit, l'adresse, ...)) et l'objet spécialisation (contient les différentes spécialisations des hôpitaux par l'exemple gynécologie, anesthésie, ...)

a) Fonctionnalité n° 1 :

Micro-service	Gestion des hôpitaux
Objectifs	Quand on saisit la localisation et le code de la spécialisation, nous devons obtenir un objet hôpital avec toutes ces informations selon la disponibilité des lits.
Fonctionnalité	Rechercher un hôpital

Impacts	En fonction du code de la spécialisation et de la localisation saisie par l'utilisateur nous allons chercher les hôpitaux. On va ensuite comparer les hôpitaux qui ont la localisation saisie, et qui dispose d'un lit disponible . Le résultat fournit est l'hôpital avec toutes ces informations (nom, adresse, lits disponible, ...)
Test	Si le test est un succès, on affiche l'hôpital et ces informations Sinon on affiche un message d'erreur « La spécialisation est introuvable »

b) Fonctionnalité n° 2 :

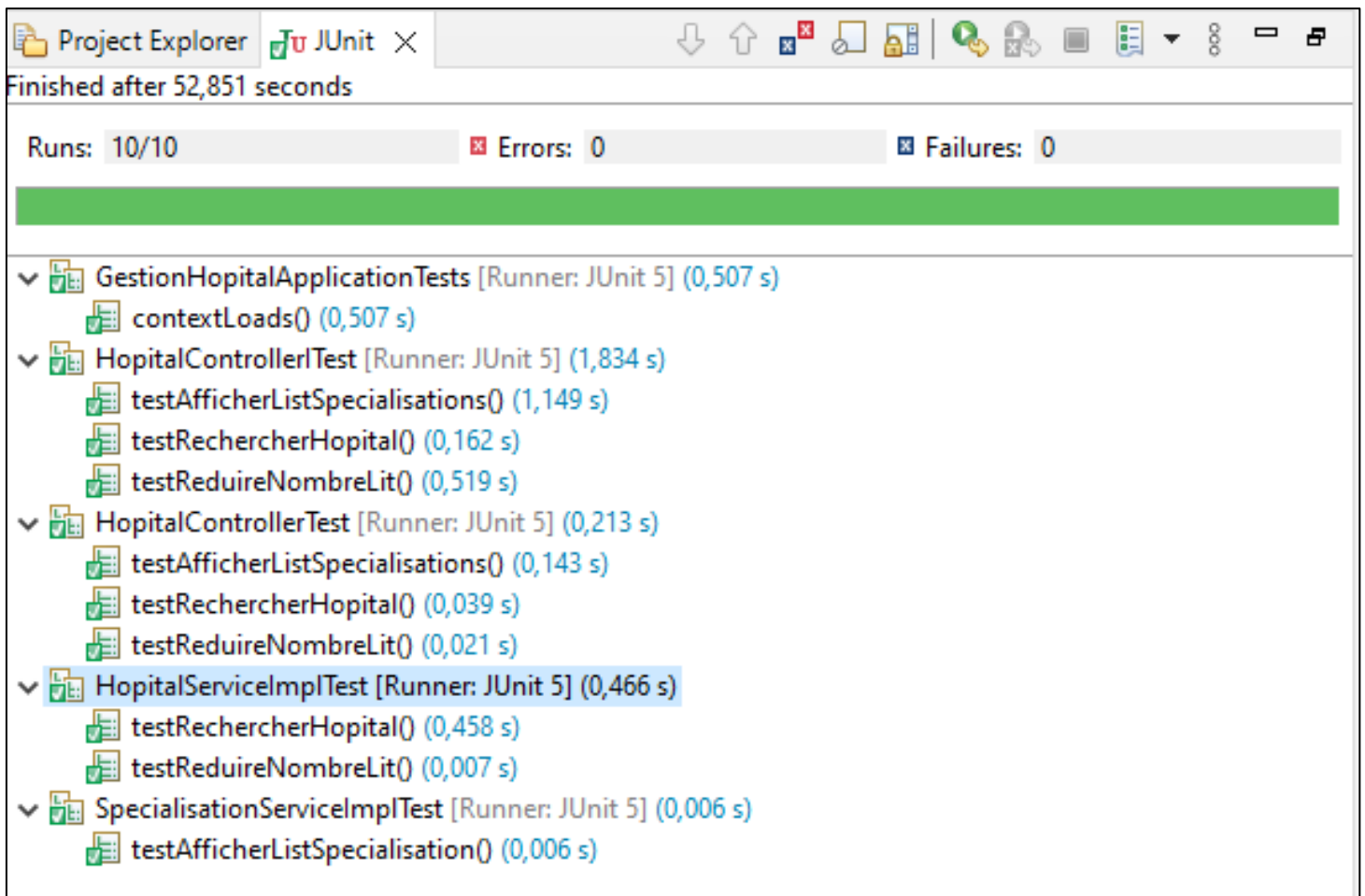
Micro-service	Gestion des hôpitaux
Objectifs	Afficher la liste de toutes les spécialisations des hôpitaux
Fonctionnalité	Afficher une liste de spécialisation
Impacts	Une liste de spécialisation doit être fournis
Test	Si le test est un succès, on affiche l'hôpital et ces informations Sinon on affiche un message d'erreur « La spécialisation est introuvable »

c) Fonctionnalité n° 3 :

Micro-service	Gestion des hôpitaux
Objectifs	Nous devons avec le code de l'hôpital et le code de spécialisation, pouvoir réduire le nombre de lit de l'hôpital.
Fonctionnalité	Réduire le nombre de lit
Impacts	En fonction du code de la spécialisation et du code de l'hôpital, nous devons être en mesure de réduire le nombre de lit de l'hôpital.
Test	Si le test est un succès, on affiche réduit le nombre de lit, Sinon on affiche un message d'erreur soit : <ul style="list-style-type: none"> - « Il n'y a pas de spécialisation trouvée pour cette hôpital » - « L'hôpital est introuvable »

d) *Résultat des tests*

Extrait de la validation des tests :



3. Micro-service réservation lit

Pour cette partie, nous aurons besoin de l'objet Hôpital (contient les hôpitaux et ces données exemple le nom, le nombre de lit, l'adresse, ...) et l'objet spécialisation (contient les différentes spécialisations des hôpitaux par exemple gynécologie, anesthésie, ...) et du micro-service gestion des hôpitaux (API Rest).

a) *Fonctionnalité n° 1 :*


Micro-service	Réservation de lit
Objectifs	Nous devons avec le code hôpital et le nom et prénom saisie, pouvoir réduire le nombre de lit de l'hôpital et réserver un lit
Fonctionnalité	Réserver un lit
Impacts	En fonction du code de la spécialisation, du code de l'hôpital et des informations saisi par l'utilisateur (nom et prénom) , nous devons réduire le nombre de lit et réserver un lit.
Test	Si le test est un succès, on enregistre la réservation et un message s'affiche « la réservation a été réalisé avec succès Sinon on affiche un message d'erreur « la réservation n'a pas abouti »









b) *Résultat des tests*

Extrait de la validation des tests :

Finished after 51,092 seconds

Runs: 4/4 ❌ Errors: 0 ❌ Failures: 0



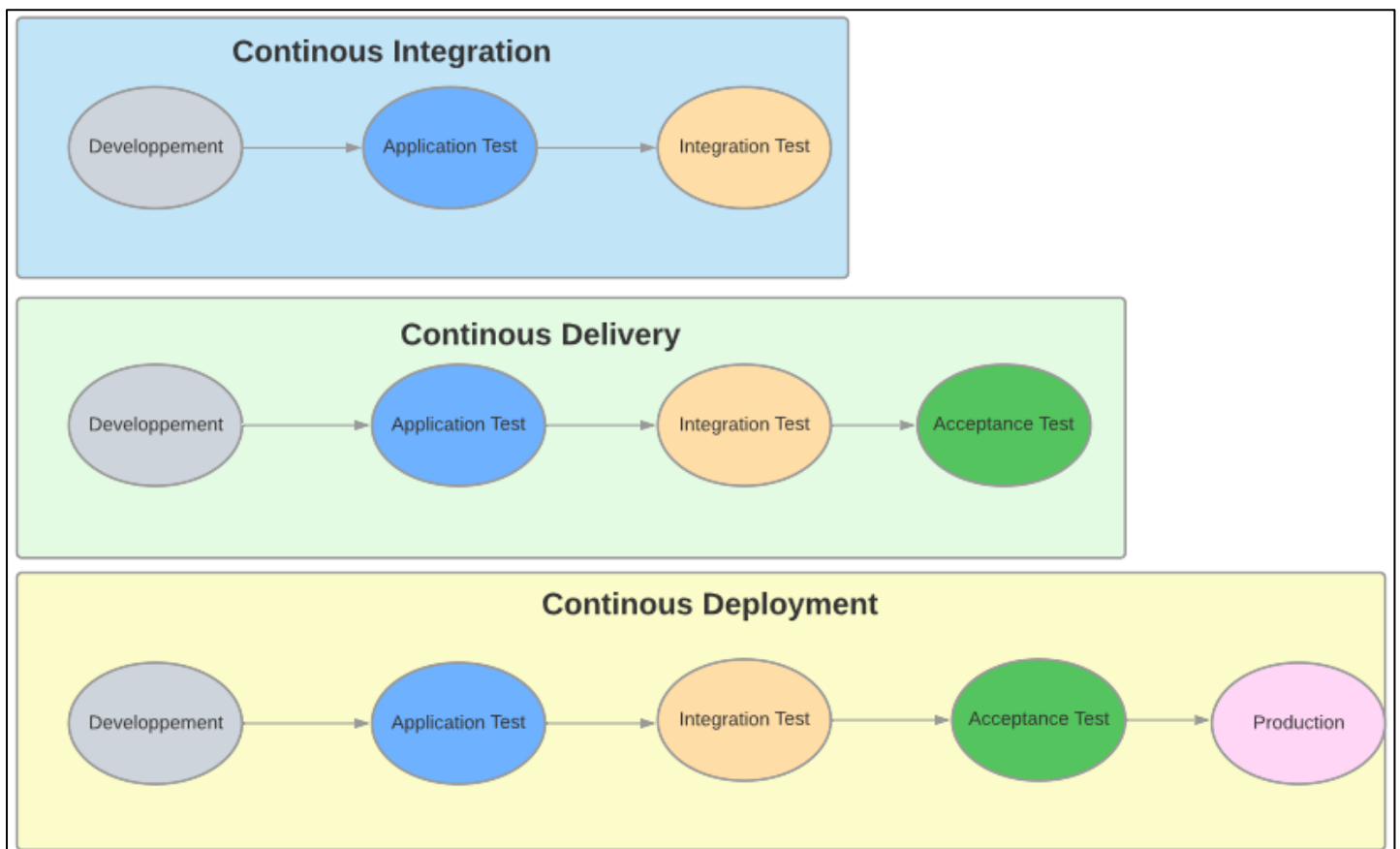
- ▼  GestionLitHopitalApplicationTests [Runner: JUnit 5] (0,469 s)
 -  contextLoads() (0,469 s)
- ▼  ReservationLitServiceImplTest [Runner: JUnit 5] (0,870 s)
 -  testReserverLit() (0,870 s)
- ▼  ReservationLitControllerTest [Runner: JUnit 5] (0,331 s)
 -  testReservationLit() (0,331 s)
- ▼  ReservationLitControllerITest [Runner: JUnit 5] (0,979 s)
 -  testReservationLit() (0,979 s)

V. Les différents tests automatisés misent en place pour ce PoC

Les tests automatisés seront à réaliser à chaque évolution de l'outil. Ils seront maintenus et lancés dans le cadre de notre organisation CI/CD à chaque mise à jour du code.

Ce plan de test va couvrir ces catégories de tests :

- **L'intégration continue** : consiste à intégrer des modifications de code dans un dépôt plusieurs fois par jour.
- **Le déploiement continu** : la **livraison** continue automatise les intégrations de code, tandis que le **déploiement** continu publie automatiquement les versions finales aux utilisateurs finaux.



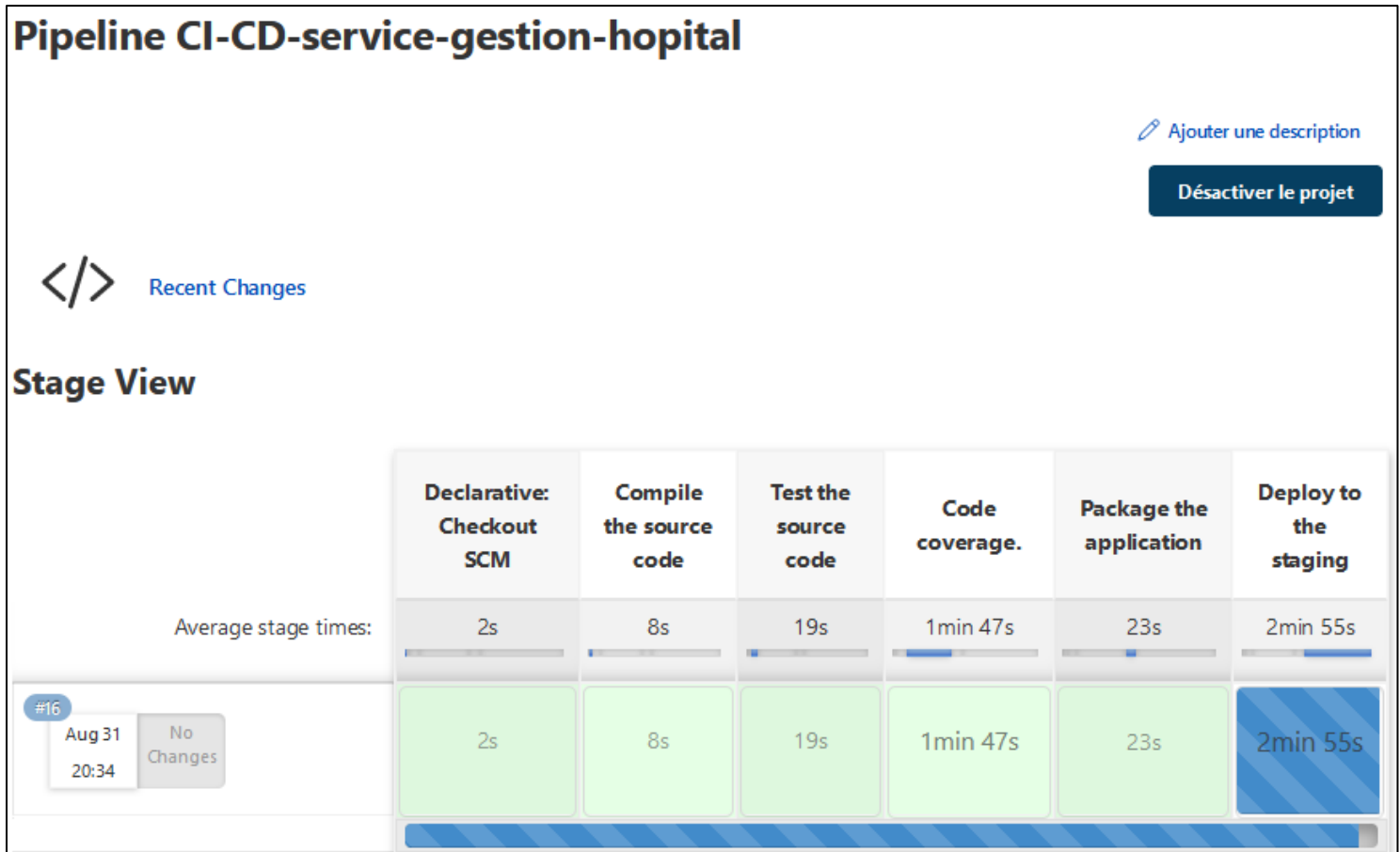
Extrait d'un pipeline CI/CD sur Jenkins du microservice gestion des hôpitaux :

```
gestion-hopital / Jenkinsfile in main

<> Edit file Preview changes

7     stage("Compile the source code")    {
8         steps    {
9             bat "mvn compile"
10        }
11    }
12    stage("Test the source code")  {
13        steps    {
14            bat "mvn test"
15        }
16    }
17    stage("Code coverage.")        {
18        steps    {
19            bat "mvn jacoco:report"
20                publishHTML (target:    [
21                                    reportDir:    'target/site/jacoco',
22                                    reportFiles:    'index.html',
23                                    reportName:    "CodeCoverageReport"
24                                ])
25            bat "mvn clean verify"
26        }
27    }
28
29    stage("Package the application")    {
30        steps    {
31            bat "mvn clean package -DskipTests"
32        }
33    }
34
35    stage("Deploy to the staging")  {
36        steps    {
37            bat "mvn spring-boot:run"
38        }
39    }
40 }
41 }
```

Extrait du lancement de build sur Jenkins :












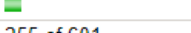


VI. Les différents rapports concernant les tests

- Rapport de la couverture de test avec Jacoco









Extrait du rapport Jacoco :

- Micro-service gestion des hôpitaux

gestion-hopital											
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes			
 gestionhopital.entities		29 %		0 %	27 49	50 86	13 35	0	3		
 gestionhopital.service		82 %		55 %	8 14	7 35	0 5	0	2		
 gestionhopital.dto		82 %		n/a	2 20	9 45	2 20	0	2		
 gestionhopital		37 %		n/a	1 2	2 3	1 2	0	1		
 gestionhopital.web		100 %		n/a	0 4	0 7	0 4	0	1		
Total	255 of 601	57 %	36 of 46	21 %	38 89	68 176	16 66	0	9		

- Micro-service gestion des lits

gestion-lit-hopital

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 gestionlithopital.entities		68 %		n/a	1	14	8	28	1	14	0	1
 gestionlithopital		58 %		n/a	1	3	2	4	1	3	0	1
 gestionlithopital.service		100 %		n/a	0	3	0	13	0	3	0	1
 gestionlithopital.web		100 %		n/a	0	2	0	2	0	2	0	1
Total	26 of 139	81 %	0 of 0	n/a	2	22	10	47	2	22	0	4

VII. Outils de gestion et d'automatisation des tests

	Outils et dépendances
Test du PoC	JUnit
	Jenkins
Documentations	Jacoco (rapport des tests)