



**DOCUMENT
DE STRATEGIE DE TEST**

Table des matières

1 CONTEXTE	3
2 PERIMETRE METIER	3
3 ORGANISATION DU PROJET	4
4 PLAN DE TEST.....	6
4-1 TEST D'ACCEPTATION	7
5. TEST UNITAIRE ET D'INTÉGRATION.....	9
5-1 MICRO-SERVICE GESTION DES HÔPITAUX	9
6. RESULTAT DE DIFFERENTS TEST REALISES	12
5-1 MICROSERVICE RESERVATION LIT.....	16
7. TEST AUTOMATISES	17
7-1 OUTIL D'INTEGRATION ET DEPLOIEMENT CONTINUE : GITLAB CI.....	18

1 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.

2 PERIMETRE METIER

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

Fonctionnalités	Description
Saisir le lieu de l'incident	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 selon la disponibilité	Si un lit est disponible, un événement doit être déclenché pour réserver un lit

3 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.

Et pour la réalisation des sprints pour ce projet « un proof of concept (POC) » on a mis en place des sprints sur Jira comme logiciel gratuit et performant.

Sprint 1 : Étude de faisabilité et conception initiale

- Étude des exigences et besoins du projet avec les parties prenantes.
- Définition des cas d'utilisation et des scénarios d'utilisation.
- Conception initiale de l'architecture globale de l'application.
- Mise en place de l'environnement de développement.

Sprint 2 : Gestion des hôpitaux

- Création d'une entité de modèle pour représenter les informations des hôpitaux (nom, localisation, lits disponibles, etc.).
- Mise en place d'une API REST pour la gestion des hôpitaux (CRUD).
- Intégration de l'API Google Maps pour la géolocalisation des hôpitaux.

Sprint 3 : Gestion des accidents

- Création d'une entité de modèle pour représenter les informations des accidents (localisation, gravité, etc.).
- Mise en place d'une API REST pour la gestion des accidents (CRUD).
- Intégration de l'API Google Maps pour la géolocalisation des accidents.

Sprint 4 : Attribution des urgences aux hôpitaux

- Implémentation d'un algorithme pour déterminer l'hôpital le plus proche du lieu de l'accident.
- Gestion de l'attribution des patients aux hôpitaux en fonction des lits disponibles.
- Mise à jour de l'état des lits disponibles après chaque attribution.

Sprint 5 : Authentification et sécurité

- Mise en place du système d'authentification et d'autorisation des utilisateurs.
- Gestion des rôles pour différents types d'utilisateurs (administrateur, personnel médical, etc.).
- Protection des API sensibles avec des mécanismes de sécurité appropriés.

Sprint 6 : Interface utilisateur (UI)

- Conception et développement de l'interface utilisateur (web ou mobile) pour les différentes fonctionnalités.
- Intégration des API pour la géolocalisation, la gestion des hôpitaux, et la gestion des accidents.
- Tests d'interface utilisateur et optimisation de l'expérience utilisateur.

Sprint 7 : Tests et débogage

- Réalisation de tests unitaires pour les différentes fonctionnalités.
- Tests d'intégration pour vérifier le bon fonctionnement de l'application dans son ensemble.
- Correction des bogues et des problèmes identifiés.

Sprint 8 : Documentation et préparation du POC

- Rédaction de la documentation du projet, y compris le guide d'installation et d'utilisation.
- Préparation de la démonstration du POC pour les parties prenantes.
- Revue finale du POC et validation de son bon fonctionnement.

Sprint 9 : Présentation et évaluation

- Présentation du POC aux parties prenantes et aux utilisateurs potentiels.
- Collecte des retours et des commentaires.

- Évaluation des performances et de l'utilisabilité du POC.

Sprint 10 : Conclusion et planification future

- Analyse des résultats du POC et décision concernant la poursuite du projet.
- Identification des améliorations potentielles et des fonctionnalités à ajouter.
- Planification des prochaines étapes pour le développement de l'application complète.

(Voir le document Scénarios BDD)

4 PLAN DE TEST

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

Les 3 types de tests automatisés les plus courants sont les tests unitaires, d'intégration et fonctionnels. Ces types de tests sont souvent présentés selon la pyramide des tests ci- dessous.

Les différentes étapes réalisées :

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

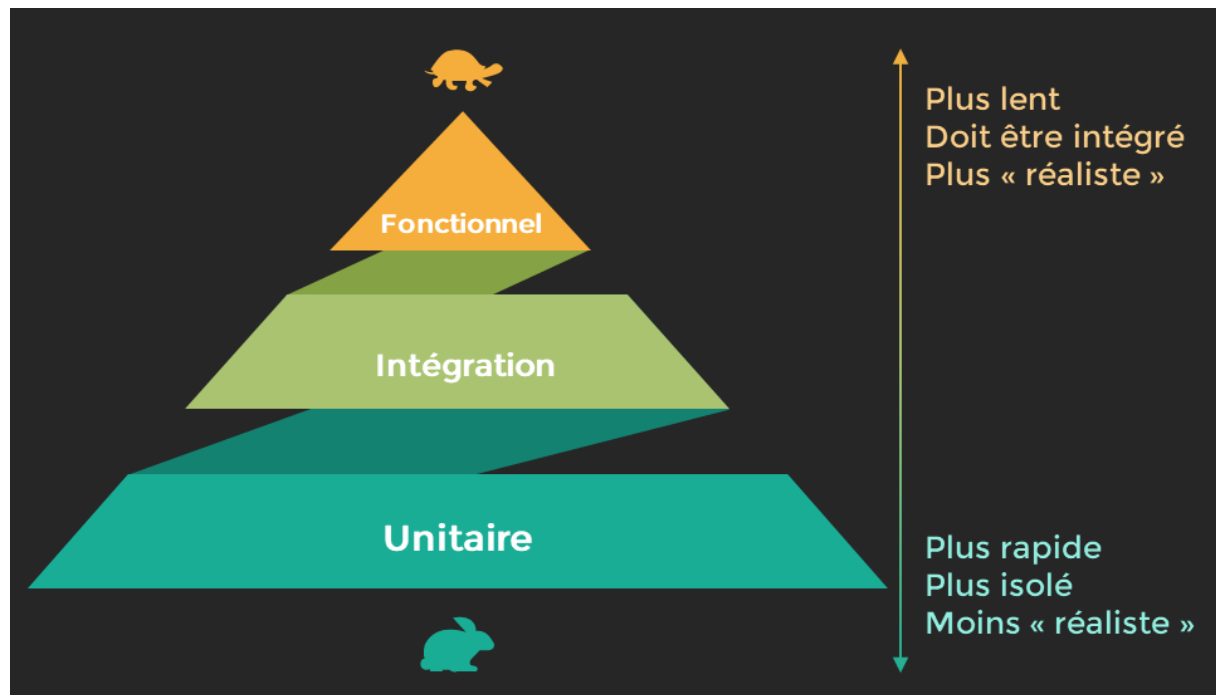


Figure N°1 représente les différents tests

4-1 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.

A-scénarios d'acceptation

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 ?

B-Les Users stories

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

(Voir le document énonces des travaux d'architecture TOGAF)

Les tests utilisateurs de bout en bout

Qui: les utilisateurs ou des testeurs

Quand: régulièrement après chaque livraison

Où: sur un environnement de test dédié

Pourquoi: S'assurer du bon fonctionnement du système.

Il est préférable de jouer les tests utilisateurs selon la complexité des scénarios, en commençant par les fonctionnalités les plus élémentaires.

5 LES TEST DE PERFORMANCE

Apache JMeter est une source ouverte outil de test sans frais de licence. Il s'agit d'une application de bureau basée sur le langage de programmation Java qui permet de tester des programmes client-serveur tels que des bases de données, des serveurs FTP, des sites Web on a fait le choix de l'utiliser pour tester notre backend.

Objectifs	valeur à obtenir	Résultat obtenu après le test
Acheminement vers l'hôpital compétent le plus proche.	Pouvoir acheminer 90% des demandes vers les services appropriés	90%
Temps moyen de traitement d'une urgence	< 12 min	9,5 min
Temps de réponse	< 200 ms	188 ms
Taux de couverture des Tests unitaires	>80 %	86%
Réalisation des Tests de performance	OUI	OUI
Réalisation des tests Utilisateurs	75%	75%

Qui: les testeurs ou les développeurs

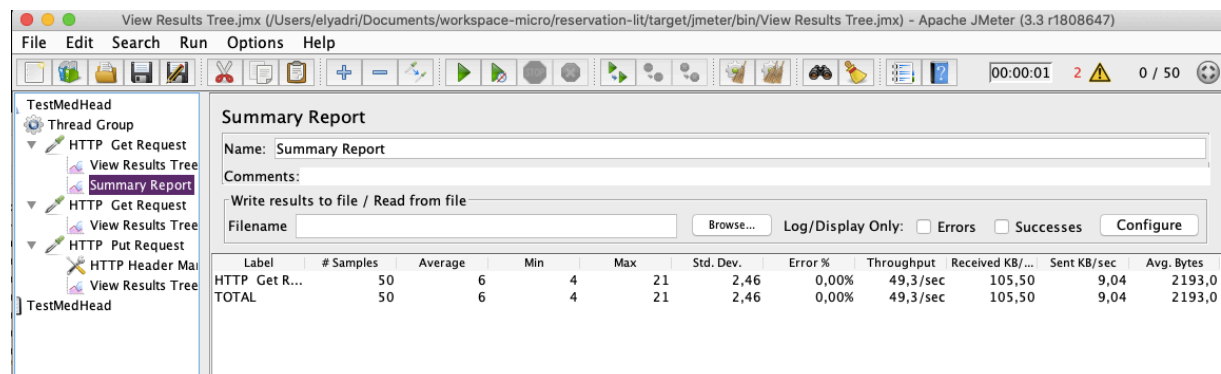
Quand: régulièrement après chaque livraison, ou avant chaque mise en production. Durant les périodes de fort trafic sur l'application.

Où: sur un environnement de test dédié

Outils: JMeter

Pourquoi: S'assurer que les performances sont maintenues lors de la mise en production.

Exemple de résultat de test de charge :



The screenshot shows the Apache JMeter 3.3 interface. On the left, a tree view shows the test structure: 'TestMedHead' (Thread Group) containing three 'HTTP Get Request' elements, each with a 'Summary Report' child. The main window displays the 'Summary Report' for the selected 'HTTP Get Request'. The report includes a table with the following data:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Get R...	50	6	4	21	2,46	0,00%	49,3/sec	105,50	9,04	2193,0
TOTAL	50	6	4	21	2,46	0,00%	49,3/sec	105,50	9,04	2193,0

Voir les fichiers en utilisant générés dans le dossier de microservice réservation lit et microservice gestion hôpital.

6 TEST UNITAIRE ET D'INTÉGRATION

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

Nous avons utilisé Junit 5 pour les tests unitaires et les tests d'intégrations.

6-1 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, ...)

Fonctionnalité 1 : Rechercher un lit

Fonctionnalisées	Description	Microservice	Test	Impact
Saisir le lieu de l'incident	Le personnel médical peut saisir le lieu de l'incident (en cas d'urgence)	Gestion des Hôpitaux	Si le test est un succès, on affiche l'hôpital et ces informations Sinon on affiche un message d'erreur « La spécialisation est introuvable »	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 et qui ont plus proche de l'adresse saisie. Le résultat fournit est l'hôpital avec toutes ces informations (nom, adresse, lits disponible, ...)

Fonctionnalité 2 : Afficher la liste de toutes les spécialisations des hôpitaux

Fonctionnalités	Description	Microservice	Test	Impact
-----------------	-------------	--------------	------	--------

Saisir le lieu de l'incident	Le personnel médical peut saisir le lieu de l'incident (en cas d'urgence)	Gestion des Hôpitaux	Si le test est un succès, on affiche l'hôpital et ces informations Sinon on affiche un message d'erreur « La spécialisation est introuvable »	Une liste de spécialisation doit être fournis
------------------------------	---	----------------------	--	---

Fonctionnalité 3 : Réduire le nombre de lit

Fonctionnalités	Description	Microservice	Test	Impact
Réduire le nombre de lit	Le personnel médical peut saisir le lieu de l'incident (en cas d'urgence)	Gestion des Hopitaux	Si le test est un succès, on affiche réduit le nombre de lit, Sinon on affiche un message d'erreur soit : * « Il n'y a pas de spécialisation * trouvée pour cette hôpital » * « L'hôpital est introuvable »	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. (Voir capture d'écran)

7 RESULTAT DE DIFFERENTS TEST REALISES

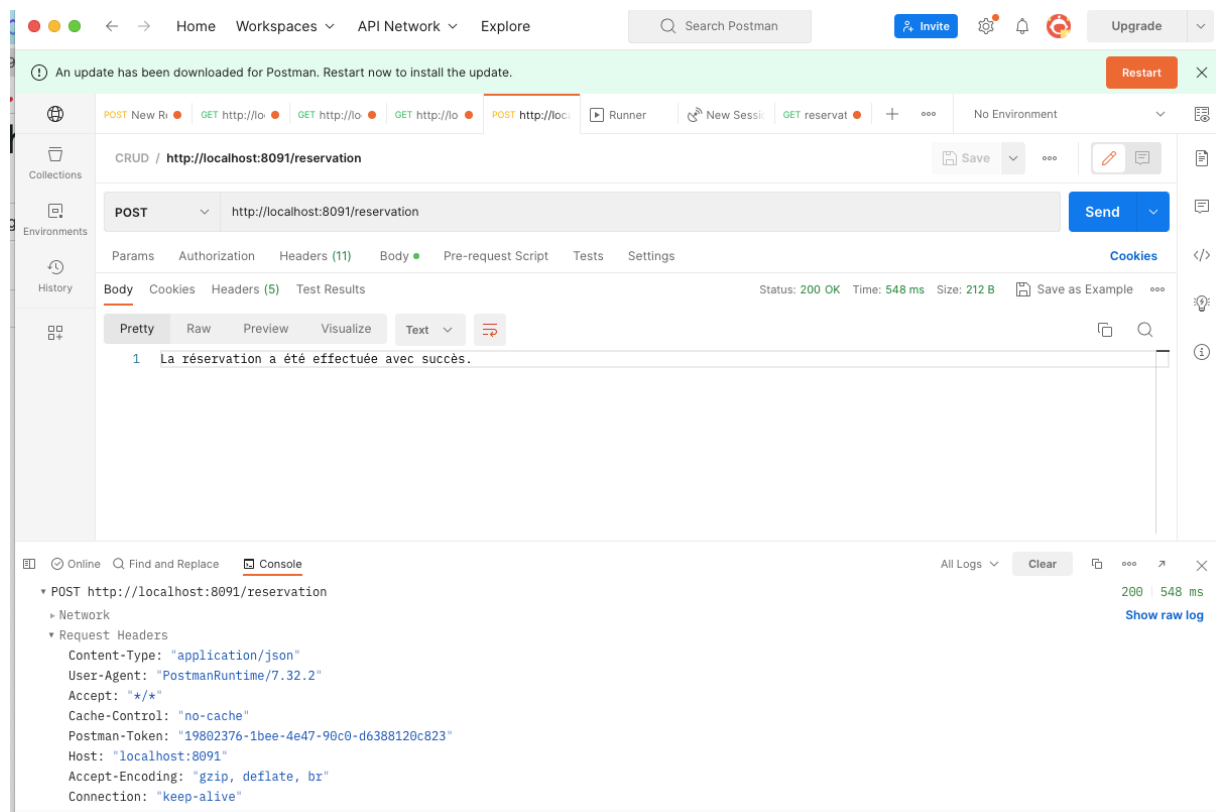


Figure N°1 représente les tests réalisés sur Postman

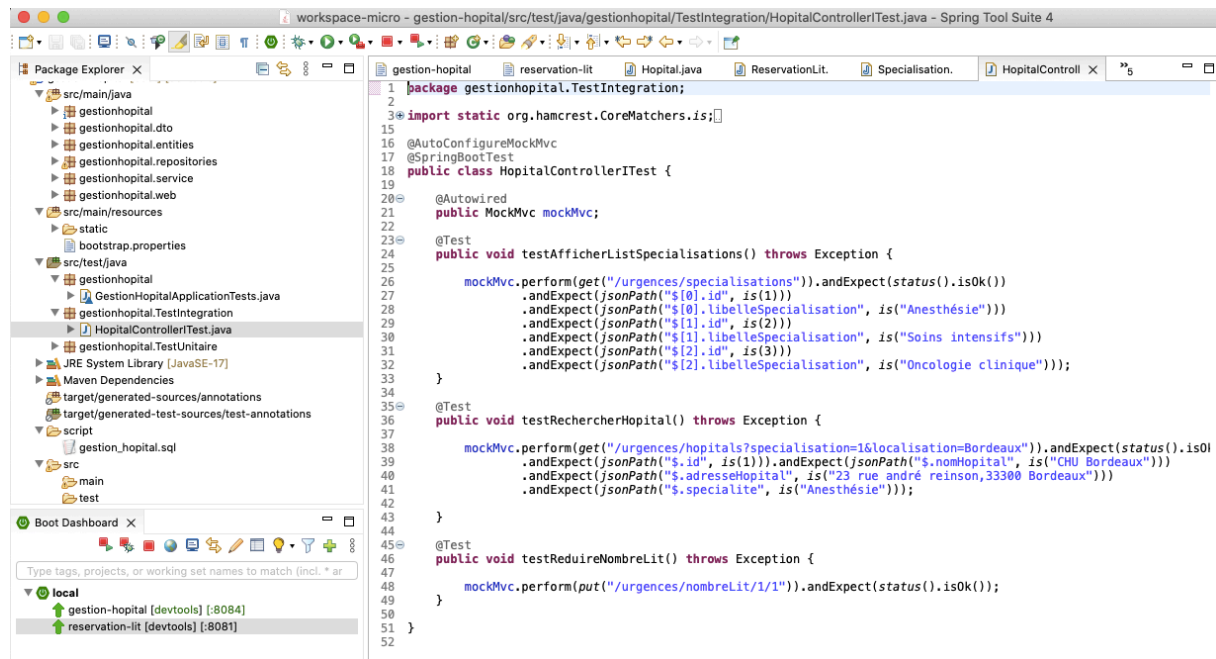


Figure N°2 représente les tests réalisés sur JUnit5

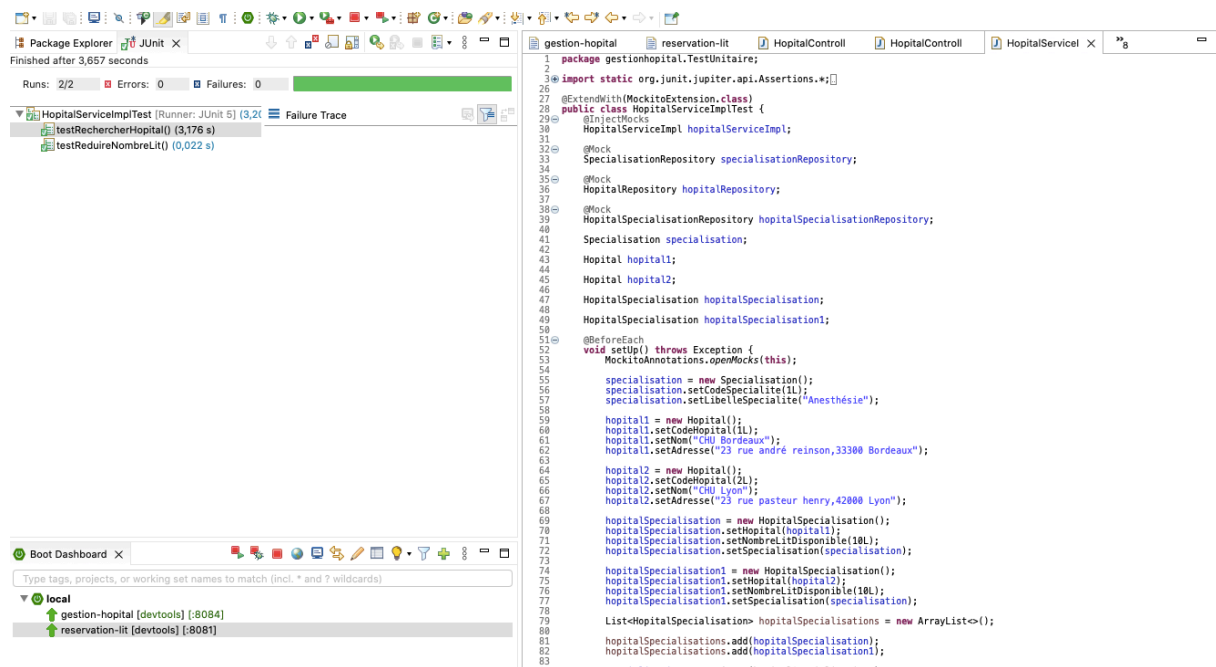


Figure N°3 représente les tests réalisés sur JUnit5

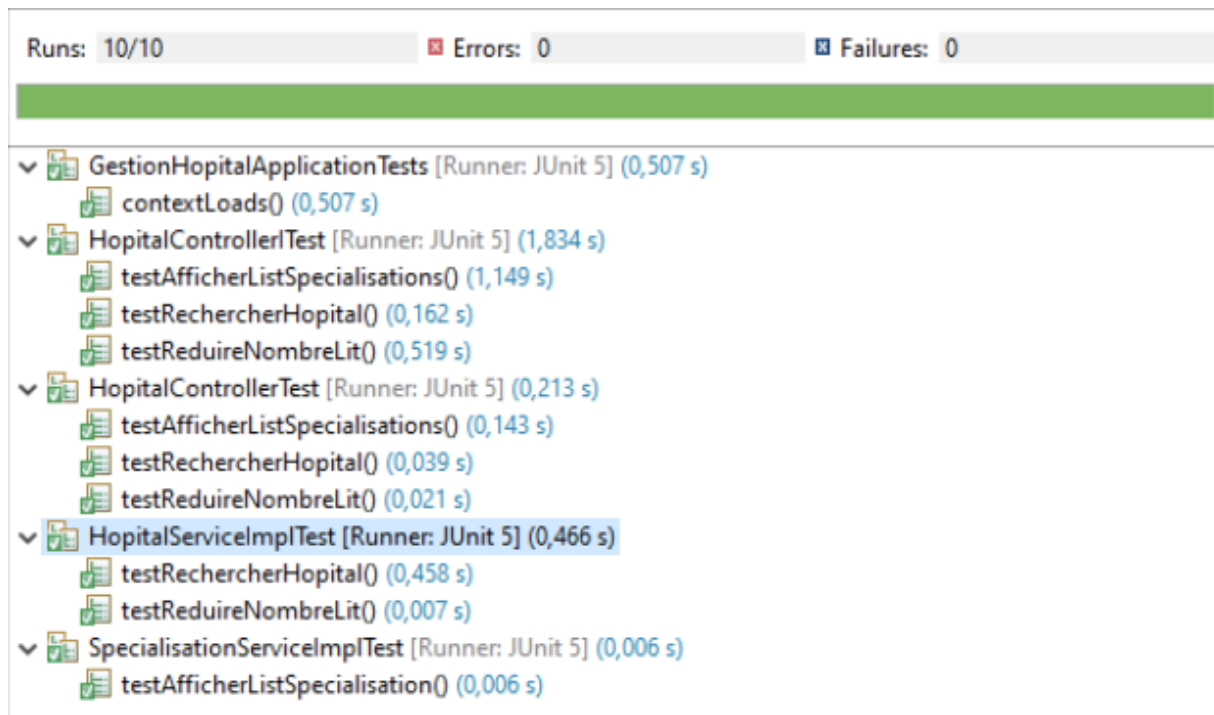


Figure N°4 représente les tests réalisés sur Junit5

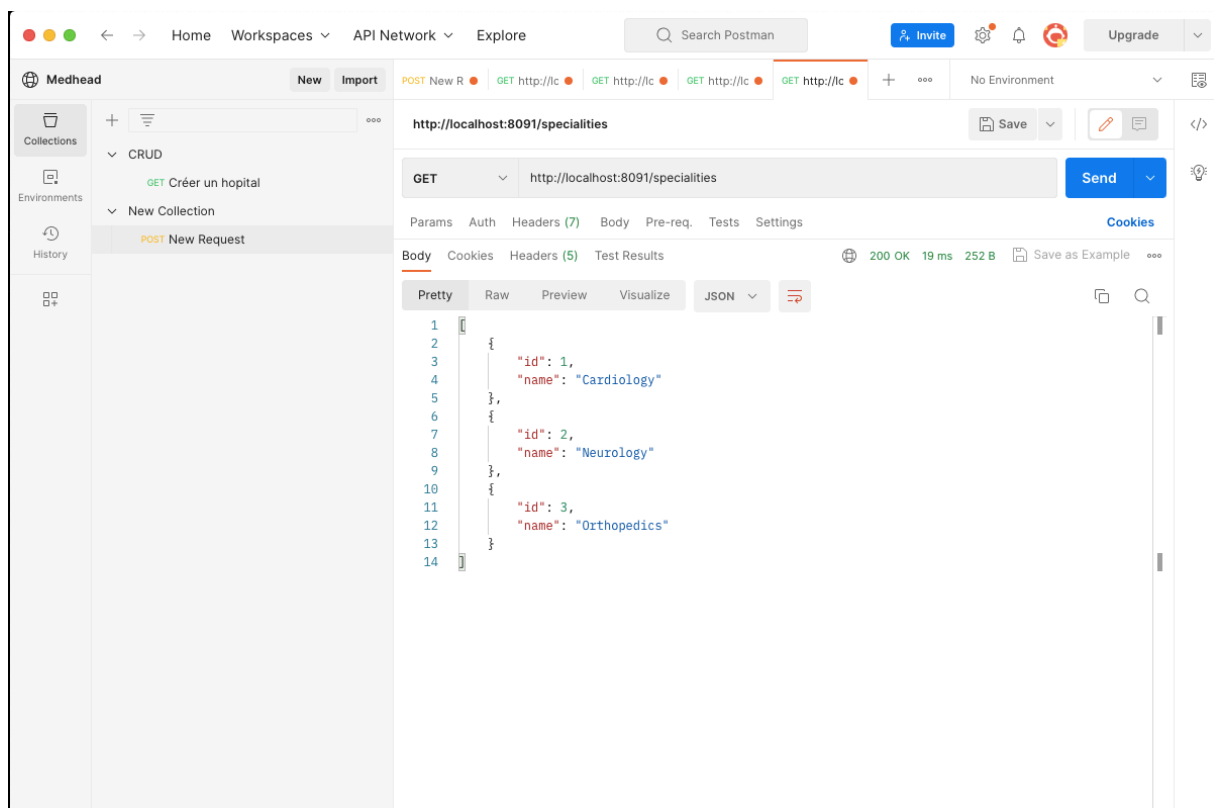


Figure N° 5 représente les tests réalisés sur Postman

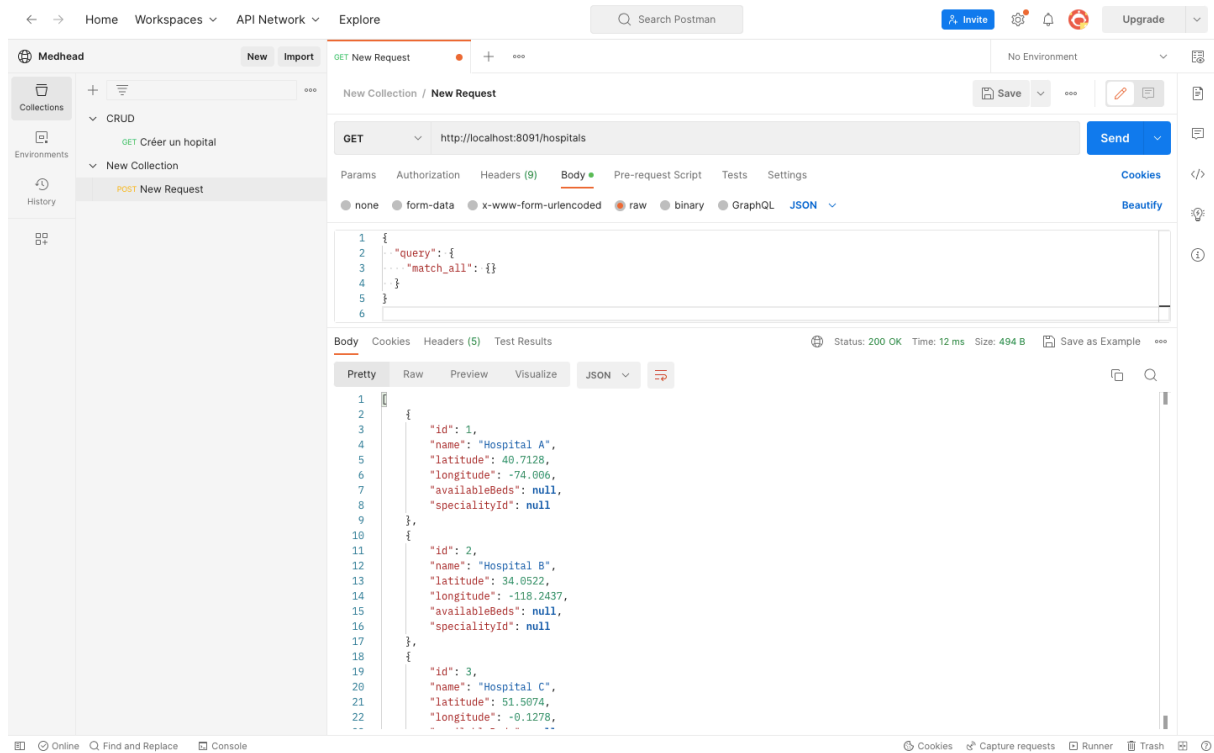


Figure N°6 représente les tests réalisés sur Postman

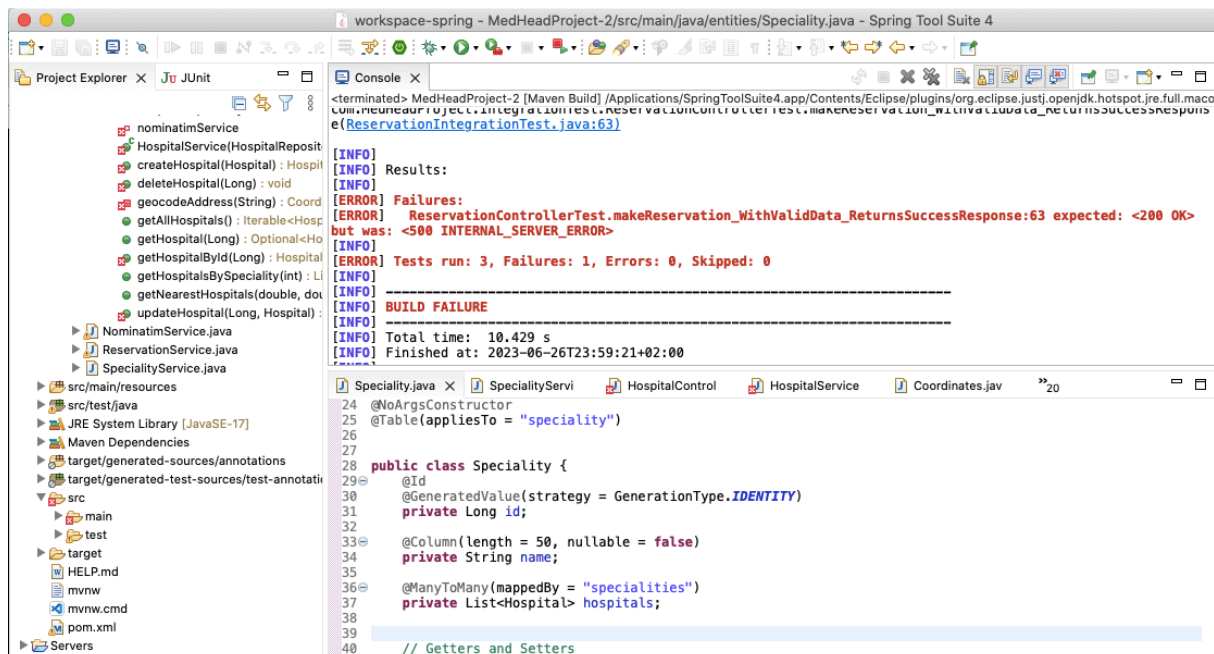


Figure N°7 représente les tests échoués avant les corrections

Documentation de Jacoco pour les tests : le lien :

<file:///Users/elyadri/Documents/workspace-micro/reservation-lit/target/site/jacoco/index.html>

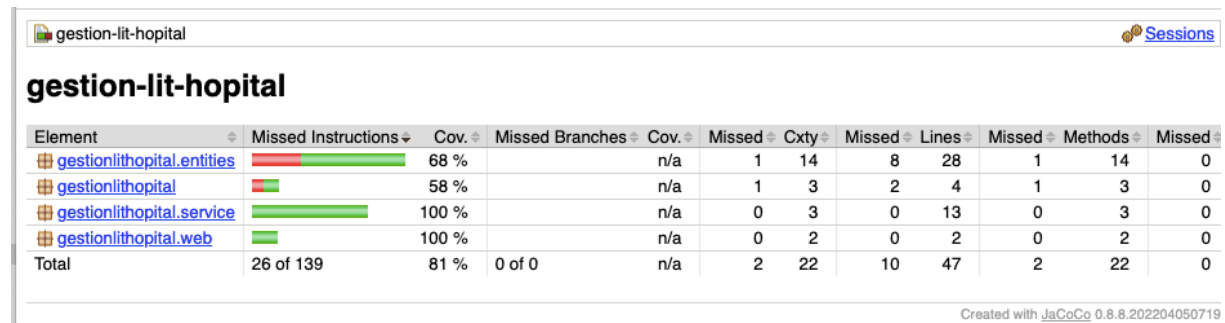


Figure N°8 représente les tests réalisés « documentation Jacoco »

7-1 MICROSERVICE RESERVATION 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 cardiologie, anesthésie, ...) et du microservice gestion des hôpitaux (API Rest).

Fonctionnalité 1 : Réserver un lit

Fonctionnalités	Description	Microservice	Test	Impact
Réserver un lit	Le personnel médical peut saisir le lieu de l'incident (en cas d'urgence)	Gestion des réservations	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	En fonction du code de la spécialisation, du code de l'hôpital et des informations saisi par l'utilisateur (nom et numéro de telephone, nombre des patients) , nous devons

			réservation n'a pas abouti »	réduire le nombre de lit et réserver un lit)
--	--	--	---------------------------------	---

8 TEST AUTOMATISES

L'approche CI/CD permet d'augmenter la fréquence de distribution des applications grâce à l'introduction de l'automatisation au niveau des étapes de développement des applications. Les principaux concepts liés à l'approche CI/CD sont l'intégration continue, la distribution continue et le déploiement continu. L'approche CI/CD représente une solution aux problèmes posés par l'intégration de nouveaux segments de code pour les équipes de développement et d'exploitation (ce qu'on appelle en anglais « integration hell », ou l'enfer de l'intégration).

Plus précisément, l'approche CI/CD garantit une automatisation et une surveillance continues tout au long du cycle de vie des applications, des phases d'intégration et de test jusqu'à la distribution et au déploiement. Ensemble, ces pratiques sont souvent désignées par l'expression « pipeline CI/CD » et elles reposent sur une collaboration agile entre les équipes de développement et d'exploitation, que ce soit dans le cadre d'une approche DevOps

L'objectif de l'intégration et du déploiement continus (CI/CD) est de permettre aux équipes de développement de livrer fréquemment des logiciels fonctionnels aux utilisateurs, ce qui leur permet à la fois de fournir une valeur ajoutée et d'obtenir un retour d'information utile sur la façon dont leur produit est utilisé dans le monde réel.

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.



8-1 OUTIL D'INTEGRATION ET DEPLOIEMENT CONTINUE : GITLAB CI

GitLab a été classé n°1 au classement Forrester CI WaveTM

Prise en main rapide

GitLab utilise un fichier de configuration écrit en YAML que n'importe quel développeur peut comprendre et prendre en main.

Trois en Un

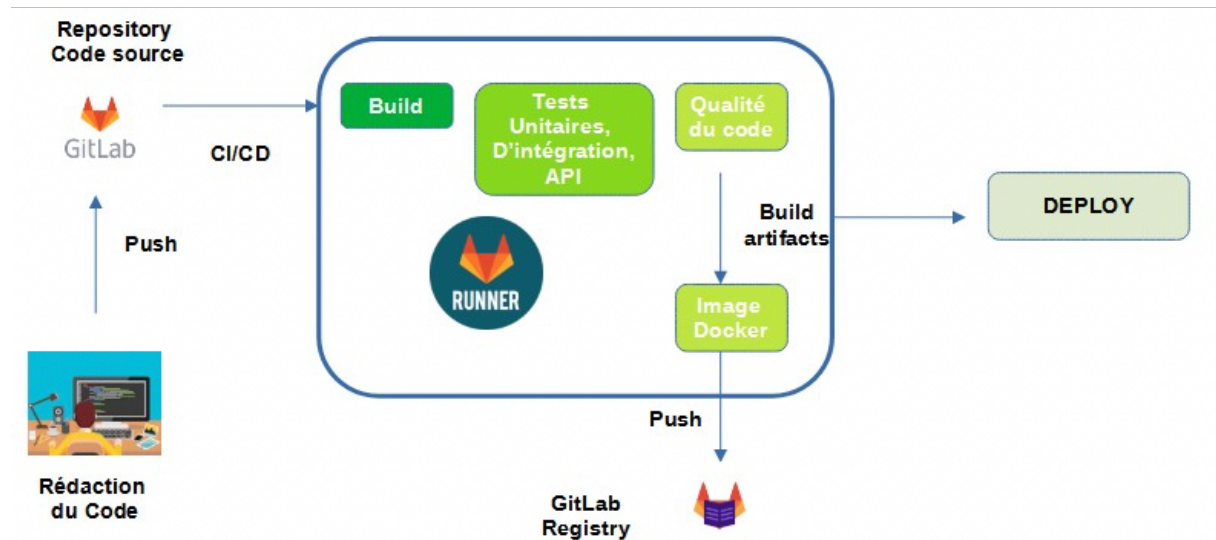
La plateforme tout-en-un de GitLab permet

- de planifier des projets
- de gérer le code source
- de gérer l'intégration et déploiement continue.

Open Source

GitLab est un logiciel open source, qui peut être installé en local ou accessible via internet.

Pipeline CI/CD : GitLab CI



GitLab est un logiciel permettant de servir de repository, ainsi les différents modules développés pourront être déposés sur GitLab.

GitLab CI/CD utilise un certain nombre de concepts pour décrire et exécuter le build et déploiement.

A l'aide des runners de GitLab, lors du Push de son code dans le repository approprié par le développeur, des jobs seront automatiquement déclenchés, les uns à la suite des autres.

Aux validations des tests (unitaire, intégration et API), un artefact est produit, il permettra de construire une image de l'API en question et permettre par la suite le lancement d'un container Docker. GitLab met à disposition une Registry permettant de stocker les différentes images produites.

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

🔍 Search or go to...

med-gestion-hopital

Project overview

Pinned

Issues 0

Merge requests 0

Manage

Plan

Code

Merge requests 0

Repository

Branches

Commits

Tags

Repository graph

Compare revisions

Help

liam liam > med-gestion-hopital > Repository

```

18
19 stages:           # List of stages for jobs, and their order of execution
20   - build
21   - test
22   - deploy
23
24 build-job:        # This job runs in the build stage, which runs first.
25   stage: build
26   script:
27     - echo "Compiling the code"
28     - echo "Building the hospital management microservice"
29     - echo "Compile complete"
30
31 unit-test-job:    # This job runs in the test stage.
32   stage: test     # It only starts when the job in the build stage completes successfully.
33   script:
34     - echo "Running unit tests for the microservice This will take about 60 seconds."
35     - sleep 60
36     - echo "Code coverage is 98%"
37
38 lint-test-job:    # This job also runs in the test stage.
39   stage: test     # It can run at the same time as unit-test-job (in parallel).
40   script:
41     - echo "Linting code,This will take about 10 seconds."
42     - sleep 10
43     - echo "No lint issues found."
44
45 deploy-job:       # This job runs in the deploy stage.
46   stage: deploy  # It only runs when *both* jobs in the test stage complete successfully.
47   environment: production
48   script:
49     - echo "Deploying application medhead"
50     - echo "Application successfully deployed."
51

```

🔍 Search or go to...

med-gestion-hopital

Project overview

Pinned

Issues 0

Merge requests 0

Manage

Plan

Code

Build

Pipelines

Jobs

Pipeline editor

Pipeline schedules

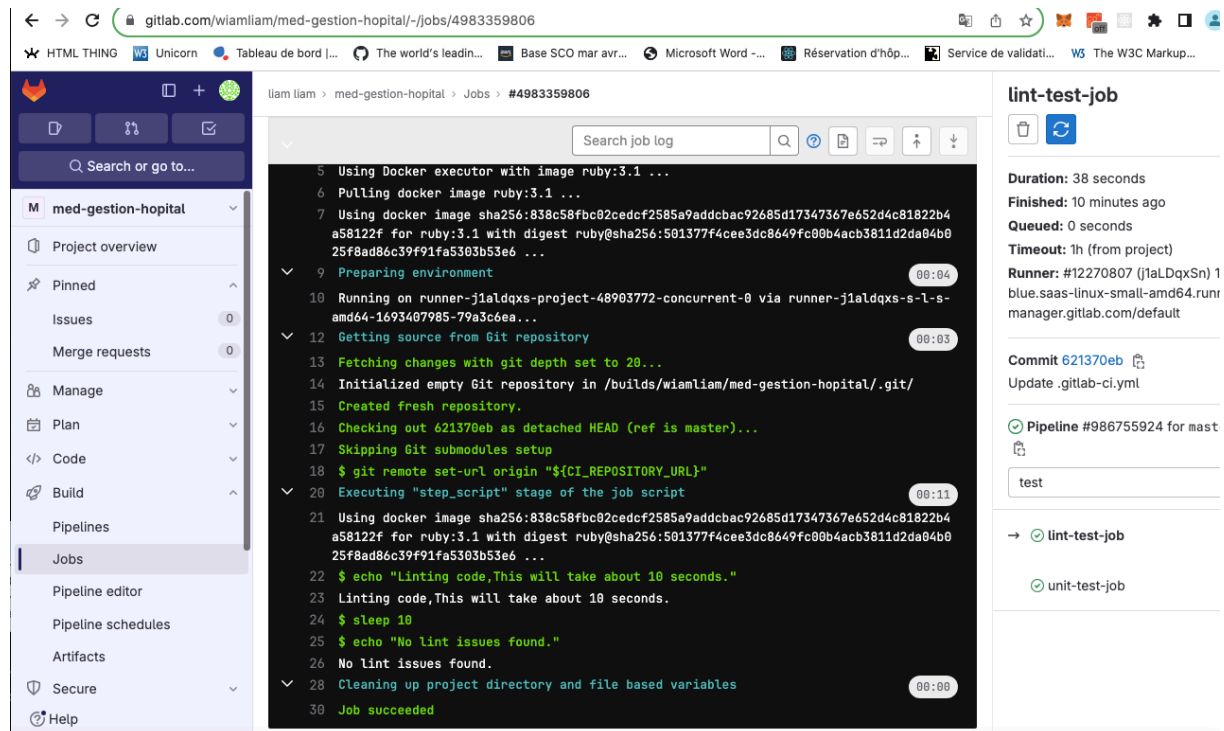
Artifacts

Secure

Help

liam liam > med-gestion-hopital > Jobs

Status	Job	Pipeline	Stage	Name	Duration	Coverage
passed	#4983359810 master 621370eb	#986755... created by	deploy	deploy-job	00:00:28 6 minutes ago	
passed	#4983359806 master 621370eb	#986755... created by	test	lint-test-job	00:00:38 7 minutes ago	
passed	#4983359803 master 621370eb	#986755... created by	test	unit-test-job	00:01:28 6 minutes ago	
passed	#4983359799 master 621370eb	#986755... created by	build	build-job	00:00:28 8 minutes ago	
passed	#4983271743 master 65b403b9	#986741... created by	deploy	deploy-job	00:00:29 13 minutes ago	
passed	#4983271740 master 65b403b9	#986741... created by	test	lint-test-job	00:00:43 14 minutes ago	
passed	#4983271734 master 65b403b9	#986741... created by	test	unit-test-job	00:01:28 13 minutes ago	



9 SECURITE

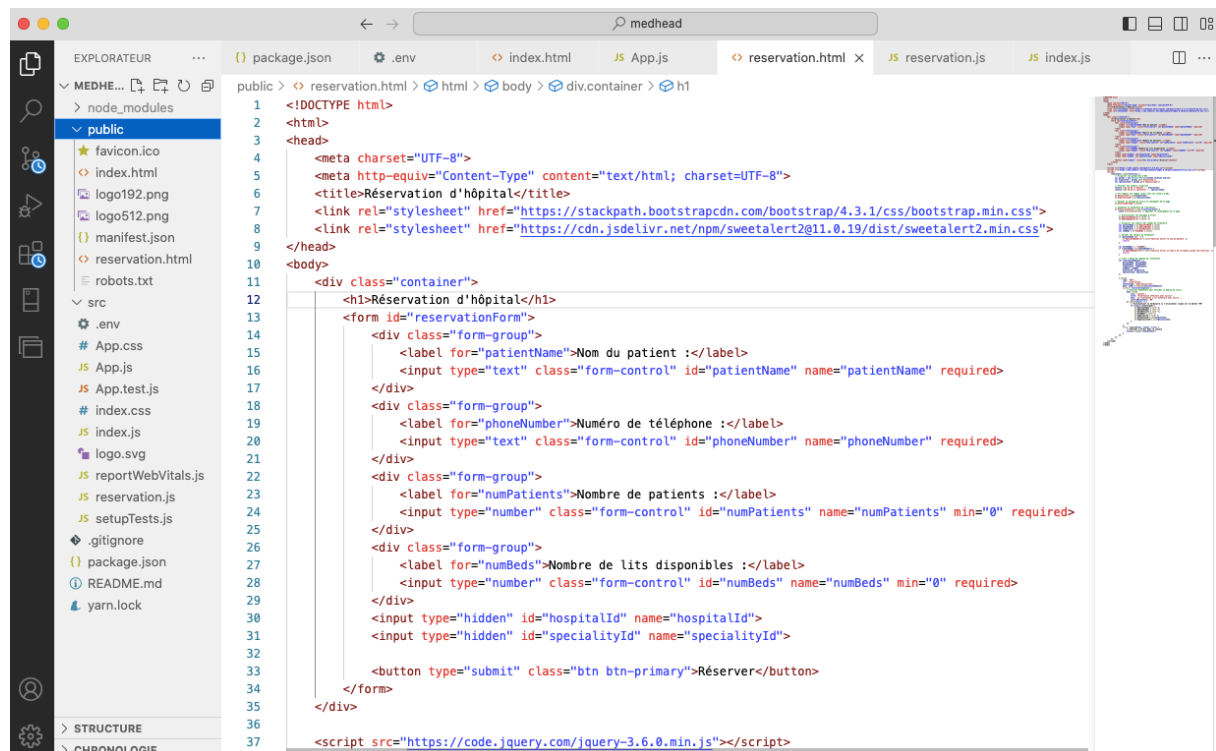
Dans le cadre de notre Proof of Concept (PoC) pour l'application de gestion des urgences, nous avons accordé une attention particulière au respect des normes et des principes afin d'assurer la conformité et la sécurité des données. Voici comment nous avons respecté le RGPD et l'architecture microservices :

Respect du RGPD :

Le Règlement général sur la protection des données (RGPD) est une priorité majeure dans notre PoC, car il concerne la protection et la confidentialité des données personnelles des utilisateurs. Pour respecter le RGPD, nous avons pris les mesures suivantes :

Minimisation des données : Nous collectons uniquement les données nécessaires pour le fonctionnement de notre application de gestion des urgences. Les informations sensibles sont minimisées, et nous veillons à ne pas stocker d'informations excessives.

Exemple d'extrait de code sur React.Js coté Client :



Validation côté client (React.js) :

Nous Effectuons une validation des données côté client avant d'envoyer le formulaire au serveur.

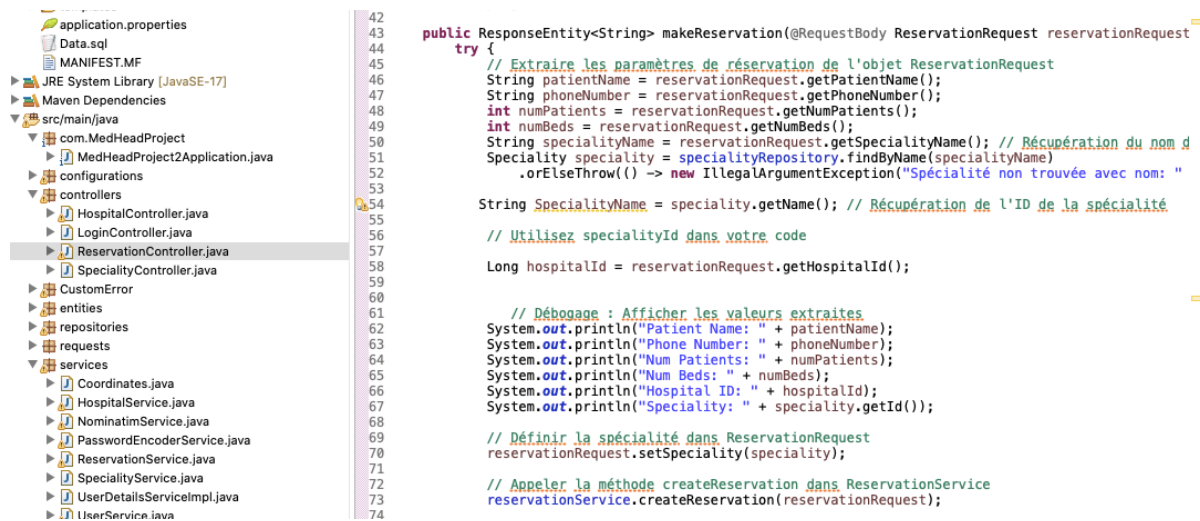
Protection contre les attaques XSS (Cross-Site Scripting) :

Échapper les données avant de les afficher sur le frontend pour éviter les attaques XSS. Utilisez des bibliothèques telles que DOMPurify pour filtrer les données HTML et empêcher l'exécution de scripts malveillants.

Transfert sécurisé des données (HTTPS) :

Nous assurons que toutes les communications entre le frontend et le backend sont sécurisées en utilisant le protocole HTTPS. Cela garantit le chiffrement des données lors de leur transmission sur le réseau.

Nous avons implémenté la fonction **makeReservation** dans **ReservationController**. Cette fonction est responsable de traiter les demandes de réservation provenant du frontend (React.js) et de sécuriser les données transmises.



CORS (Cross-Origin Resource Sharing) :

Nous avons configuré les en-têtes CORS dans l'annotation **@CrossOrigin** pour permettre les requêtes depuis les domaines spécifiés (elwiam.github.io et <http://localhost:3000>). Cela permet à notre frontend de React.js d'accéder au backend en toute sécurité.

Validation des données :

Nous avons extrait les paramètres de réservation de l'objet **ReservationRequest** et effectué une validation de ces données.

Nous vérifions si la spécialité spécifiée dans la demande existe dans la base de données en utilisant **specialityRepository.findByName(specialityName)** et nous levons une exception si elle n'existe pas.

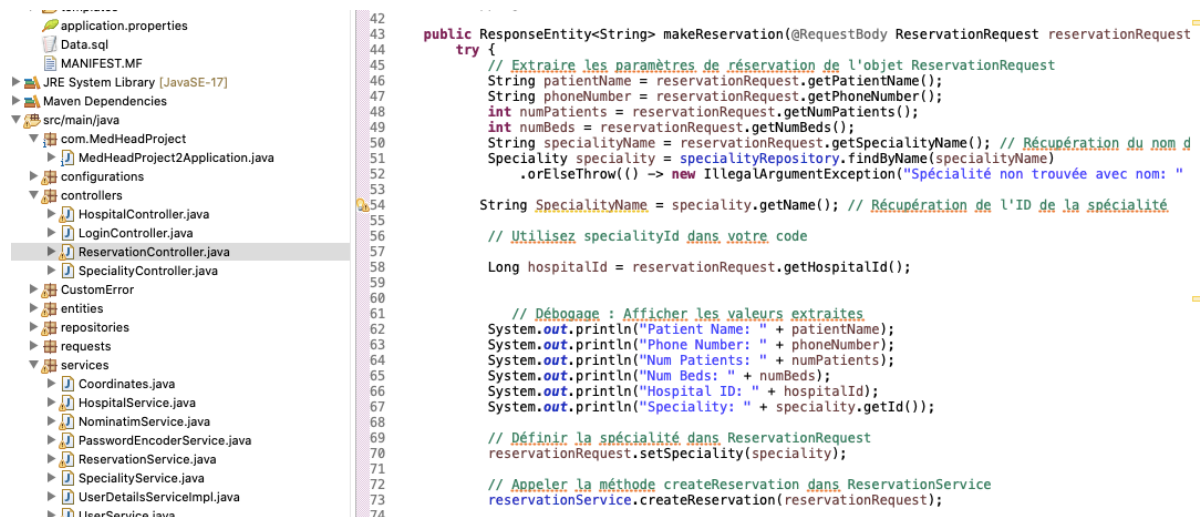
Prévention des attaques par injection :

Nous utilisons des mécanismes de sécurité fournis par Spring Data JPA, tels que **specialityRepository.findByName(specialityName)**, pour éviter les attaques par injection de SQL.

Protection contre les erreurs imprévues :

Nous capturons les exceptions pour gérer les erreurs éventuelles et retourner une réponse appropriée en cas d'erreur lors de la réservation.

Exemple d'extrait de code sur SpringBoot coté Serveur :



Dans le code **ReservationService**, la sécurité est prise en compte pour assurer que les réservations sont créées de manière sécurisée et que les modifications apportées aux entités respectent les droits d'accès.

Vérification de l'existence de l'hôpital :

Avant de créer une réservation, vous récupérez l'hôpital correspondant à l'ID spécifié dans la demande de réservation à l'aide de `hospitalRepository.findById`. Si l'hôpital n'est pas trouvé, une exception est levée, ce qui permet d'éviter la création de réservations pour un hôpital inexistant.

Définition de la date et de l'heure actuelles :

Lorsque nous créons une nouvelle réservation, nous définissons la date et l'heure actuelles comme `reservationDate` en utilisant `LocalDateTime.now()`. Cela garantit que la date et l'heure sont correctement définies sans dépendre des données fournies par l'utilisateur.

Mise à jour du nombre de lits disponibles :

Après la création de la réservation, nous mettons à jour le nombre de lits disponibles dans l'hôpital en utilisant la formule `availableBeds - numBeds` pour soustraire le nombre de lits réservés du nombre de lits disponibles. Cela est essentiel pour s'assurer que le nombre de lits disponibles est correctement mis à jour après chaque réservation.

Utilisation des objets entité pour la gestion des relations :

Nous utilisons les objets entité (par exemple `Speciality`, `Hospital`, etc.) pour gérer les relations entre les réservations et les autres entités telles que les spécialités et les hôpitaux. Cela permet de s'assurer que seules les relations existantes sont utilisées pour éviter toute association incorrecte ou non autorisée.

```
LoginController  ReservationRepo  HospitalControl  ReservationCont  *ReservationSer  SpecialityContr  24
8 import entities.Reservation;
9 import repositories.HospitalRepository;
10 import repositories.ReservationRepository;
11 import repositories.SpecialityRepository;
12 import requests.ReservationRequest;
13
14 @Service
15
16 public class ReservationService {
17
18     @Autowired
19     private ReservationRepository reservationRepository;
20
21     @Autowired
22     private SpecialityRepository specialityRepository;
23
24     @Autowired
25     private HospitalRepository hospitalRepository;
26
27
28     public void createReservation(ReservationRequest reservationRequest) {
29         // Autres vérifications et validations...
30
31         Speciality speciality = reservationRequest.getSpeciality();
32         Hospital hospital = hospitalRepository.findById(reservationRequest.getHospitalId())
33             .orElseThrow(() -> new IllegalArgumentException("Hôpital non trouvé avec ID: " + reservationRequest.getHospitalId()));
34
35         Reservation reservation = new Reservation();
36         reservation.setPatientName(reservationRequest.getPatientName());
37         reservation.setPhoneNumber(reservationRequest.getPhoneNumber());
38         reservation.setNumPatients(reservationRequest.getNumPatients());
39         reservation.setNumBeds(reservationRequest.getNumBeds());
40         reservation.setHospital(hospital);
41         reservation.setSpeciality(speciality);
42
43         // Définir la date et l'heure actuelles comme reservationDate
44         reservation.setReservationDate(LocalDate.now());
45
46         reservationRepository.save(reservation);
47         // Mettre à jour le nombre de lits disponibles dans l'hôpital
48         int numBeds = reservationRequest.getNumBeds();
49         int availableBeds = hospital.getAvailableBeds();
50         hospital.setAvailableBeds(availableBeds - numBeds);
51         hospitalRepository.save(hospital);
52     }
53 }
54
55
```

Validation côté serveur (Spring Boot) :

Nous effectuons une validation supplémentaire côté serveur pour nous assurer que les données reçues sont valides et ne contiennent pas de données malveillantes. Utilisez les mécanismes de validation de Spring Boot, tels que les annotations de validation dans les classes de modèle.

Exemple :

```
@Entity
@Table(name = "Reservation")
```

Chiffrement des données sensibles :

Lorsque nous stockons des données sensibles dans la base de données, telles que les mots de passe, on les chiffre à l'aide d'algorithmes de hachage sécurisés, tels que BCrypt.

Et on ne stocke jamais les mots de passe en clair.

Gestion des droits d'accès :

Nous mettons en place un système d'autorisation approprié pour contrôler les droits d'accès des utilisateurs aux ressources et aux fonctionnalités de l'application.

Protéger les endpoints :

Ensuite, nous pouvons protéger nos endpoints en utilisant les annotations fournies par Spring Security, telles que `@PreAuthorize` ou `@Secured`. Nous pouvons spécifier les rôles requis pour accéder à chaque endpoint.

Par exemple :

```

63 @PreAuthorize("hasRole('ADMIN')")
64 @PostMapping
65 public ResponseEntity<Hospital> createHospital(@RequestBody Hospital hospital) {
66     Hospital createdHospital = hospitalService.createHospital(hospital);
67     return new ResponseEntity<>(createdHospital, HttpStatus.CREATED);
68 }
69 @PreAuthorize("hasRole('ADMIN')")
70 @PutMapping("/{id}")
71 public ResponseEntity<Hospital> updateHospital(@PathVariable Long id, @RequestBody Hospital hospital) {
72     Hospital existingHospital = hospitalService.getHospitalById(id);
73     if (existingHospital != null) {
74         hospital.setId(id);
75         Hospital updatedHospital = hospitalService.updateHospital(id, hospital);
76         return new ResponseEntity<>(updatedHospital, HttpStatus.OK);
77     } else {
78         return new ResponseEntity<>(HttpStatus.NOT_FOUND);
79     }
80 }
81 @PreAuthorize("hasRole('ADMIN')")
82 @DeleteMapping("/{id}")
83 public ResponseEntity<Void> deleteHospital(@PathVariable Long id) {
84     Hospital existingHospital = hospitalService.getHospitalById(id);
85     if (existingHospital != null) {
86         hospitalService.deleteHospital(id);
87         return new ResponseEntity<>(HttpStatus.NO_CONTENT);
88     } else {
89         return new ResponseEntity<>(HttpStatus.NOT_FOUND);
90     }
91 }
92
93 @PreAuthorize("hasRole('ADMIN')")
94 @PutMapping("/{hospitalId}/specialities/{specialityId}")
95 public ResponseEntity<Hospital> addSpecialityToHospital(
96     @PathVariable Long hospitalId,
97     @PathVariable Long specialityId

```

Avec cette configuration, seuls les utilisateurs ayant le rôle "ADMIN" pourront accéder aux opérations CRUD sur les hôpitaux, tandis que les autres utilisateurs auront un accès restreint ou interdit.

Gestion des sessions :

Nous utilisons des sessions sécurisées et des mécanismes d'expiration pour gérer les connexions utilisateur et les sessions actives.

Contrôle des erreurs :

Nous limitons les informations d'erreur divulguées aux utilisateurs en cas de problème. Nous personnalisons les messages d'erreur pour ne pas révéler d'informations sensibles.