
Documentation technique du projet LabXpert - l'API REST via Spring Boot (Part 2)

Réalisé par

GUELSA Mouna

EZZAOUIBI Yassine

CHERGAOUI Mohammed

2023-2024

TABLE DES MATIÈRES

Documentation	3
1 Introduction	3
2 Conduite du projet	3
2.1 Méthodologie de travail	3
2.2 Outil de collaboration	4
3 Etude et analyse fonctionnelle	4
3.1 Besoins Fonctionnels	4
3.2 Modélisation	5
4 Etude technique et technologique	8
4.1 Technologies et outils techniques	8
5 Démonstration	14
5.1 Swagger	14
5.2 Tests Manuels -Postman	15
5.3 Tests Unitaires -JUnit	17
6 Conclusion	20

TABLE DES FIGURES

1	Méthode KANBAN	3
2	Diagramme de cas d'utilisation.	5
3	Diagramme de classes.	7
4	Architecture MVC	8

1 Introduction

TechLab, laboratoire médical à la pointe de l'innovation, s'engage dans une transformation majeure avec le lancement du projet LabXpert. Conçu pour optimiser chaque aspect des opérations du laboratoire, LabXpert promet d'élever l'efficacité et la précision dans le traitement des analyses médicales. Cette solution complète couvre un large spectre, depuis l'enregistrement des échantillons jusqu'à la gestion des résultats, avec des fonctionnalités telles que le suivi en temps réel des analyses, la gestion des patients, le contrôle des stocks de réactifs, et bien plus encore. LabXpert, c'est l'avenir de TechLab, offrant une gestion intégrée qui promet d'optimiser les services pour un diagnostic médical plus rapide et plus précis.

2 Conduite du projet

Le choix de la conduite du projet est une phase d'exterminante pour accomplir le projet dans les bonnes conditions. Il faut donc bien définir le processus de développement et en déduire le planning du projet à suivre.

2.1 Méthodologie de travail

Afin de structurer les différentes phases de notre projet, et qui vont être explicitées par la suite, et afin de garantir une organisation optimale, il est nécessaire de fixer un cadre qui simplifie indéniablement le lancement du projet, sa progression et sa réussite par la suite. Pour cela Afin d'assurer le respect des délais et le bon déroulement du projet, nous avons opté pour la méthode agile KANBAN qui nous fournit un cadre d'agilité simple à comprendre. Cette méthode repose sur un système de visualisation de cartes dans un tableau. Les cartes ou étiquettes représentent les tâches à traiter qui correspondent aux demandes entrantes. Le tableau a l'avantage d'être facile à déchiffrer . Il indique également l'état d'avancement des tâches à traiter en affichant les cartes dans différentes colonnes : à faire ,en cours,à tester,terminé

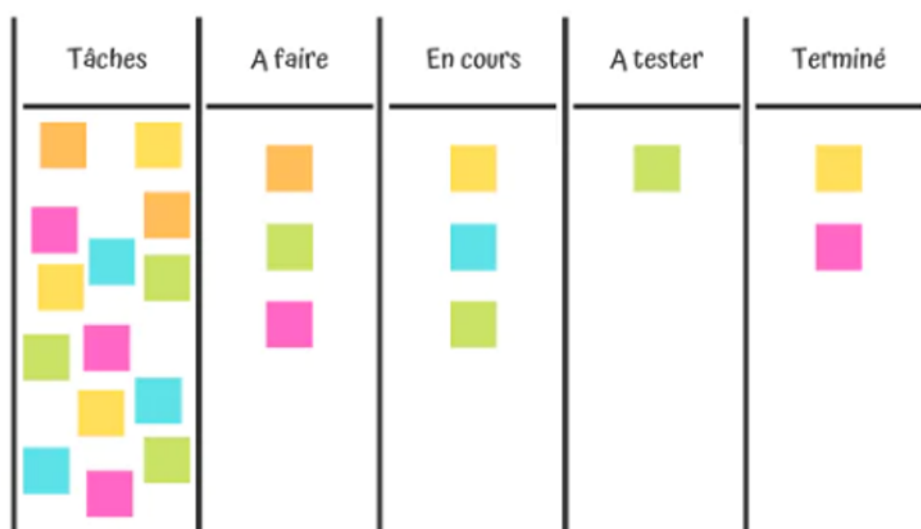


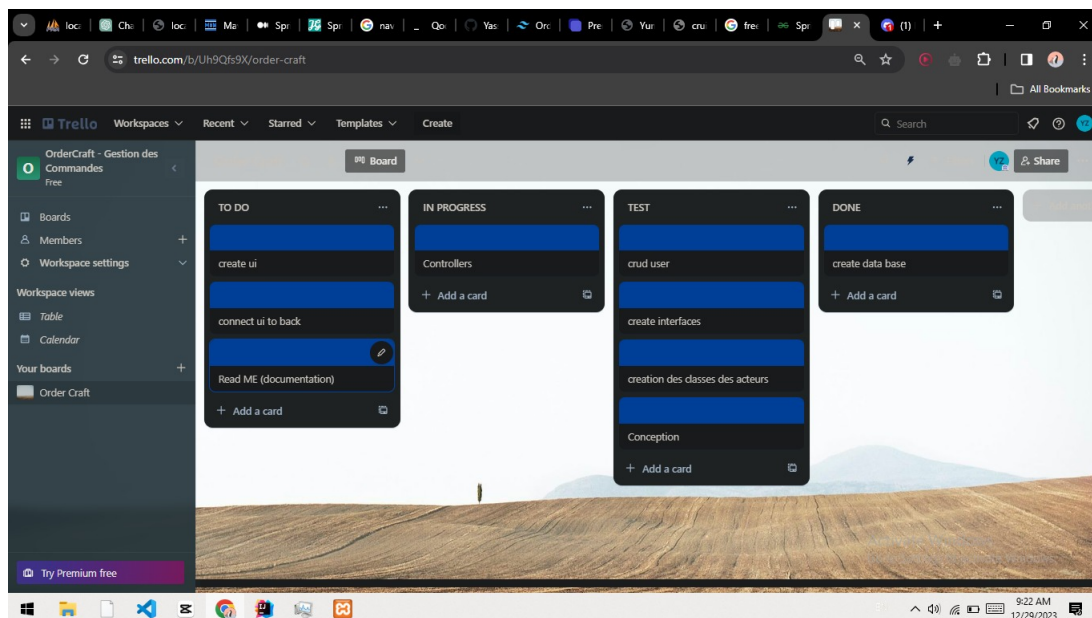
FIGURE 1 – Méthode KANBAN

2.2 Outil de collaboration

2.2.1 Trello



Trello est un outil de gestion de projet en ligne, lancé en septembre 2011 et inspiré par la méthode Kanban de Toyota. Il repose sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, traduisant leur avancement.



3 Etude et analyse fonctionnelle

3.1 Besoins Fonctionnels

Le système de gestion du laboratoire "LabXpert" offre plusieurs fonctionnalités qui répondent aux besoins fonctionnels du laboratoire TechLab. Voici une liste détaillée de ces fonctionnalités :

- **Enregistrement des échantillons** : Le système permet aux techniciens d'enregistrer de nouveaux échantillons en spécifiant les informations pertinentes telles que le patient, le type d'analyse et la date de prélèvement.
- **Suivi des analyses en cours** : Une interface conviviale permet aux techniciens et aux responsables de laboratoire de suivre en temps réel l'état d'avancement des analyses en cours, avec des détails spécifiques pour chaque échantillon.
- **Gestion des résultats** : Les résultats des analyses sont consignés de manière systématique, permettant un accès rapide aux informations et la possibilité de partager les résultats avec les professionnels de la santé concernés.

- **Gestion des patients** : Un module dédié offre la possibilité de gérer les informations relatives aux patients, assurant une centralisation des données et une navigation facilitée.
- **Inventaire des réactifs** : Intégration d'un suivi des stocks pour garantir la disponibilité des réactifs nécessaires aux différentes analyses.
- **Gestion des utilisateurs** : Une interface d'administration permet de gérer les droits d'accès et les informations des utilisateurs, assurant une sécurité accrue des données.
- **Planification des analyses** : Possibilité de planifier les analyses en fonction de la charge de travail, optimisant ainsi l'utilisation des ressources du laboratoire.
- **Rapports statistiques** : Génération de rapports statistiques pour évaluer les performances du laboratoire, identifier les tendances et prendre des décisions basées sur les données.

Ces fonctionnalités sont conçues pour améliorer l'efficacité et la précision dans le traitement des analyses médicales, optimiser les opérations du laboratoire et fournir un service plus rapide et plus précis aux patients.

3.2 Modélisation

3.2.1 Diagramme de cas d'utilisation

Ce diagramme illustre le flux d'utilisation de l'application et détaille les accès ainsi que les fonctionnalités données à l'utilisateur.

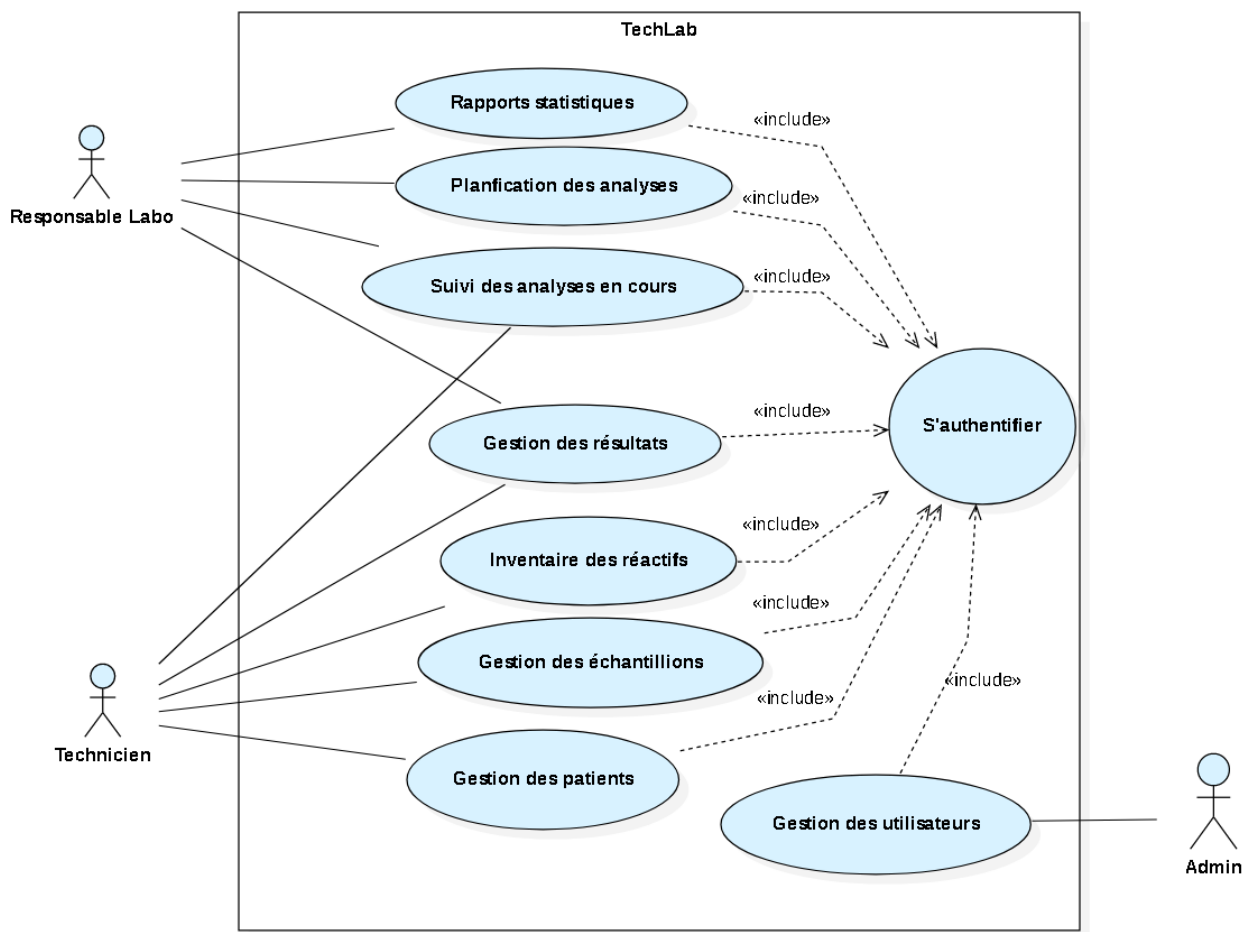


FIGURE 2 – Diagramme de cas d'utilisation.

Le diagramme de cas d'utilisation décrit les interactions entre les utilisateurs (acteurs) et le système. Voici une description textuelle des cas d'utilisation pour le système de gestion du laboratoire "LabXpert" :

- **Enregistrer Échantillon :**
 - Acteur : Technicien
 - Description : Le technicien enregistre un nouvel échantillon en fournissant les informations nécessaires telles que le patient, le type d'analyse, et la date de prélèvement.
- **Suivre Analyses en Cours :**
 - Acteur : Technicien, Responsable de laboratoire
 - Description : Les utilisateurs peuvent suivre en temps réel l'état d'avancement des analyses en cours. Ils peuvent obtenir des détails spécifiques pour chaque échantillon.
- **Consulter Résultats :**
 - Acteur : Technicien, Professionnels de la santé
 - Description : Les utilisateurs peuvent consulter les résultats des analyses enregistrés de manière systématique. Ils peuvent également partager ces résultats avec les professionnels de la santé concernés.
- **Gérer Patients :**
 - Acteur : Personnel administratif, Technicien
 - Description : Les utilisateurs peuvent gérer les informations relatives aux patients, assurant une centralisation des données et une navigation facilitée.
- **Gérer Inventaire Réactifs :**
 - Acteur : Technicien
 - Description : Le technicien peut suivre les stocks de réactifs, assurant la disponibilité des réactifs nécessaires aux différentes analyses.
- **Gérer Utilisateurs :**
 - Acteur : Administrateur
 - Description : L'administrateur peut gérer les droits d'accès et les informations des utilisateurs, assurant une sécurité accrue des données.
- **Planifier Analyses :**
 - Acteur : Responsable de laboratoire
 - Description : Le responsable de laboratoire peut planifier les analyses en fonction de la charge de travail, optimisant ainsi l'utilisation des ressources du laboratoire.
- **Générer Rapports Statistiques :**
 - Acteur : Responsable de laboratoire
 - Description : Le responsable de laboratoire peut générer des rapports statistiques pour évaluer les performances du laboratoire, identifier les tendances et prendre des décisions basées sur les données.

3.2.2 Diagramme de classes

Les tableaux suivants expliquent les relations entre les classes et détaillent leurs attributs ainsi que leurs méthodes .

* Les relations entre les classes :

1. **Relation Analyse - Echantillon :**
 - Analyse a une relation Many-to-One avec Echantillon.
 - Echantillon a une relation One-to-Many avec Analyse.
2. **Relation Analyse - Utilisateur (Technicien) :**
 - Analyse a une relation Many-to-One avec Utilisateur (Technicien).
 - Utilisateur a une relation One-to-Many avec Analyse.

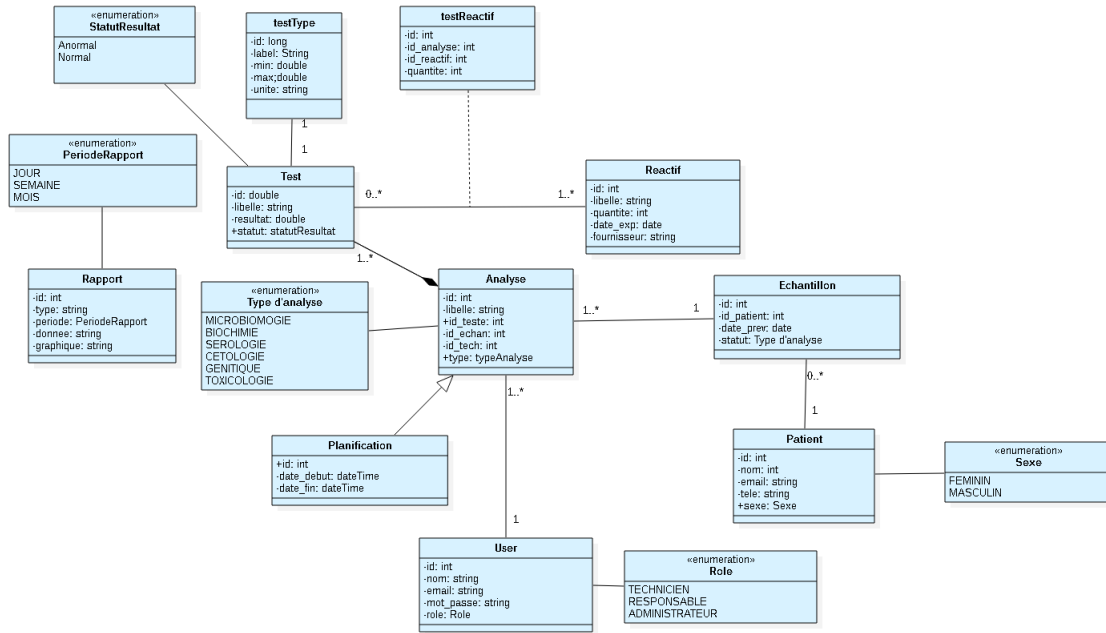


FIGURE 3 – Diagramme de classes.

3. Relation Analyse - Test :

- Analyse a une relation One-to-Many avec Test.
- Test a une relation Many-to-One avec Analyse.

4. Relation Echantillon - Patient :

- Echantillon a une relation Many-to-One avec Patient.
- Patient a une relation One-to-Many avec Echantillon.

5. Relation Echantillon - Analyse :

- Echantillon a une relation One-to-Many avec Analyse.
- Analyse a une relation Many-to-One avec Echantillon.

6. Relation Echantillon - Type de Test :

- Echantillon a une relation Many-to-One avec TestType.
- TestType a une relation One-to-Many avec Echantillon.

7. Relation Patient - Analyse :

- Patient a une relation One-to-Many avec Analyse.
- Analyse a une relation Many-to-One avec Patient.

8. Relation Planification - Analyse :

- Planification étend Analyse, héritant de ses relations.
- Relation supplémentaire : Planification a une relation Many-to-One avec Utilisateur (Technicien).

9. Relation Rapport - Analyse :

- Rapport a une relation Many-to-One avec Analyse.

10. Relation Reactif - TestReactif :

- Reactif a une relation One-to-Many avec TestReactif.
- TestReactif a une relation Many-to-One avec Reactif.

11. Relation Test - TestReactif :

- Test a une relation One-to-Many avec TestReactif.
- TestReactif a une relation Many-to-One avec Test.

12. **Relation TestType - Test :**

- TestType a une relation One-to-Many avec Test.
- Test a une relation Many-to-One avec TestType.

13. **Relation Utilisateur - Analyse :**

- Utilisateur a une relation One-to-Many avec Analyse.
- Analyse a une relation Many-to-One avec Utilisateur (Technicien).

14. **Relation Utilisateur - Planification :**

- Utilisateur a une relation One-to-Many avec Planification.
- Planification a une relation Many-to-One avec Utilisateur (Technicien).

4 Etude technique et technologique

4.1 Technologies et outils techniques

4.1.1 Concepts DevOps

Intégration continue

L'intégration continue (CI) est la pratique qui permet d'automatiser l'intégration des modifications du code provenant de plusieurs contributeurs dans un même projet. Il s'agit d'une des principales bonnes pratiques DevOps, qui permet aux développeurs de fusionner fréquemment les modifications de code dans un répertoire central où les builds et les tests sont ensuite exécutés. Des outils automatisés sont utilisés pour garantir la conformité du nouveau code avant son intégration. Un système de contrôle de la version du code source est l'élément central du processus de CI. Le système de contrôle de version est également complété par d'autres contrôles tels que les tests de qualité du code, les outils de révision du style syntaxique, etc. On a utilisé GITHUB.

Jenkins

Jenkins est un outil d'automatisation open-source écrit en Java avec des plugins construits pour les besoins de la livraison continue. Jenkins est utilisé pour builder et tester les projets logiciels en temps réel, ce qui permet aux développeurs d'intégrer plus facilement les modifications apportées au projet et aux utilisateurs d'obtenir plus facilement une nouvelle version.

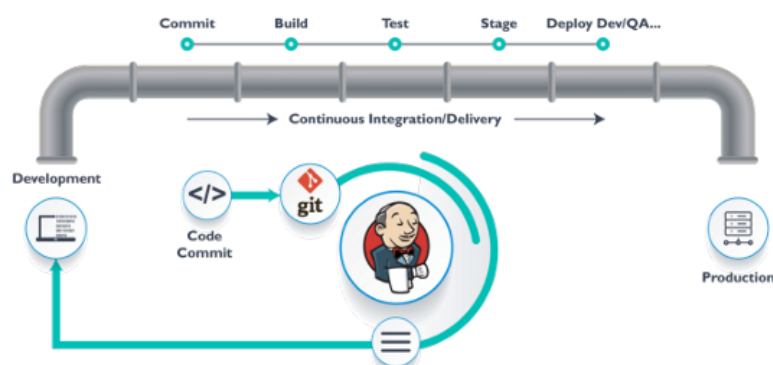


FIGURE 4 – Architecture MVC .

4.1.2 Outils

* Spring Framework :



Le Spring Framework est très largement utilisé dans la communauté Java. Il permet d'accélérer le développement d'applications d'entreprise (notamment le développement d'applications Web et d'API Web). Mais on trouve des applications fondées sur le Spring Framework dans d'autres domaines. Les fonctionnalités de base de Spring Framework peuvent être utilisées pour développer n'importe quelle application Java, mais il existe des extensions pour créer des applications Web sur la plate-forme Java EE. Le framework Spring vise à rendre le développement J2EE plus facile à utiliser et promeut les bonnes pratiques de programmation en activant un modèle de programmation basé sur POJO.

Injection de dépendance :

L'inversion de contrôle (IoC) est un concept général et peut être exprimé de différentes manières. L'Injection de Dépendance n'est qu'un exemple concret d'Inversion de Contrôle. Lors de l'écriture d'une application Java complexe, les classes d'application doivent être aussi indépendantes que possible des autres classes Java afin d'accroître la possibilité de réutiliser ces classes et de les tester indépendamment des autres classes lors des tests unitaires. L'injection de dépendance aide à coller ces classes ensemble et en même temps à les maintenir indépendantes. Par exemple, la classe A dépend de la classe B. Voyons maintenant la deuxième partie, l'injection. Cela signifie simplement que la classe B sera injectée dans la classe A par l'IoC. L'injection de dépendance peut se produire en passant des paramètres au constructeur ou en post-construction à l'aide de méthodes setter.

* Spring Framework :

Spring Boot est un framework qui permet la mise en place d'application Spring rapidement



et facilement. Il se base sur le Framework Spring et permet de s'affranchir de la plupart des configurations de celui-ci à mettre en place pour créer une application. Les principales fonctions :

- Une gestion des dépendances Spring simplifiée
- Undéploiement facilité
- Intégrez directement Tomcat, Jetty ou Undertow (inutile de déployer des fichiers WAR (Web Application Resource))
- La configuration automatique de bibliothèques Spring et autres
- La configuration des propriétés externes plus lisible
- Facilités pour créer des repositories- Des possibilités de déclarer des sorties JSON (JavaScript Objet Notation) multiples.
- L'exposition des ressources par REST juste avec une annotation

- Aucune génération de code et aucune exigence pour la configuration XML (Extensible Markup Language)

*** Junit :**



JUnit est un framework open source pour le développement et l'exécution de tests unitaires automatisables. Le principal intérêt est de s'assurer que le code répond toujours aux besoins même après d'éventuelles modifications. Plus généralement, ce type de tests est appelé tests unitaires de non-régression. Le but est d'automatiser les tests. Ceux-ci sont exprimés dans des classes sous la forme de cas de tests avec leurs résultats attendus. JUnit exécute ces tests et les compare avec ces résultats. Cela permet de séparer le code de la classe, du code qui permet de la tester. Souvent pour tester une classe, il est facile de créer une méthode `main()` qui va contenir les traitements de tests. L'inconvénient est que ce code "superflu" est inclus dans la classe. De plus, son exécution doit se faire manuellement.

Avec JUnit, l'unité de test est une classe dédiée qui regroupe des cas de tests. Ces cas de tests exécutent les tâches suivantes :

- création d'une instance de la classe et de tout autre objet nécessaire aux tests
- Appel de la méthode à tester avec les paramètres du cas de tests
- comparaison du résultat attendu avec le résultat obtenu : en cas d'échec, une exception est levée

*** Swagger :**



Swagger est un langage de description d'interface permettant de décrire des APIs RESTful exprimées à l'aide de JSON. Swagger est utilisé avec toute une série d'outils logiciels open source pour concevoir, créer, documenter et utiliser des services Web RESTful.

*** Postman :**



Postman est officiellement présentée comme une plateforme API pour la création et l'utilisation d'API. D'une manière générale, Postman est une plateforme qui permet de simplifier chaque étape du cycle de vie des API et de rationaliser la collaboration, afin de créer, plus facilement et plus rapidement, de meilleures API. Il permet de tester manuellement des requêtes HTTP.

*** Mockito :**



Mockito est un framework de test open source pour Java, spécialisé dans la création et la gestion de mocks, des objets fictifs qui simulent le comportement d'objets réels. Il offre des fonctionnalités puissantes de configuration des mocks, de vérification des appels de méthodes, et d'annotations facilitant le processus de test. En permettant aux développeurs de simuler le comportement des dépendances, Mockito favorise la création de tests unitaires isolés, facilitant ainsi l'identification et la résolution des problèmes. Son intégration transparente avec JUnit en fait un choix populaire pour les développeurs Java soucieux de garantir la qualité et la robustesse de leur code.

*** H2 :**



La base de données H2 est un système de gestion de base de données relationnelle (SGBDR) open-source, écrit en Java. Il a été conçu pour être léger, rapide et intégrable dans des applications Java. Voici quelques caractéristiques et informations importantes sur la base de données H2 :

- **Type de base de données :** H2 prend en charge les bases de données relationnelles. Il prend en charge le langage SQL standard avec des fonctionnalités telles que les transactions ACID (Atomicité, Cohérence, Isolation, Durabilité).

- **Langage de programmation** : H2 est écrit en Java et peut être intégré dans des applications Java. Il peut être utilisé dans des applications Java SE (Standard Edition) ainsi que dans des applications Java EE (Enterprise Edition).
 - **Caractéristiques principales** :
 - **Embarqué** : H2 peut être utilisé comme une base de données embarquée dans une application Java, ce qui signifie qu'il n'y a pas besoin de démarrer un processus de base de données distinct.
 - **Mode serveur** : Il peut également fonctionner en mode serveur, permettant à plusieurs clients d'accéder à la base de données simultanément via le réseau.
 - **Compatibilité avec les modes mémoire** : H2 prend en charge les bases de données en mémoire, ce qui signifie que les données peuvent être stockées en RAM plutôt que sur le disque pour une performance maximale.
 - **Interface utilisateur** : H2 fournit une interface utilisateur basée sur le web appelée H2 Console, qui permet aux utilisateurs d'interagir avec la base de données via un navigateur web.
 - **Compatibilité JDBC** : H2 est compatible avec JDBC, ce qui le rend facile à intégrer avec des applications Java utilisant cette API standard pour interagir avec des bases de données.
 - **Multiplateforme** : Comme il est écrit en Java, H2 est multiplateforme et peut être utilisé sur divers systèmes d'exploitation.
 - **Licence** : H2 est distribué sous la licence open-source EPL (Eclipse Public License).
- * **PostgreSQL** :



PostgreSQL, souvent abrégé en "Postgres," est un système de gestion de base de données relationnelle (SGBDR) open source et puissant. Il est reconnu pour sa conformité aux normes SQL, sa fiabilité, et sa flexibilité. PostgreSQL offre des fonctionnalités avancées de gestion des données, de traitement transactionnel, et prend en charge la modélisation complexe des données.

Voici quelques points clés à propos de PostgreSQL :

- **Open Source** : PostgreSQL est distribué sous licence open source, ce qui signifie que son code source est accessible et modifiable par la communauté. Cela favorise la collaboration et permet aux utilisateurs de contribuer à son développement.
- **Conformité aux Normes SQL** : PostgreSQL est conforme aux normes SQL ANSI et supporte un large éventail de fonctionnalités SQL avancées. Il propose également des extensions spécifiques qui étendent ses capacités.
- **Fiabilité et Robustesse** : PostgreSQL est réputé pour sa stabilité et sa robustesse. Il offre des mécanismes de sauvegarde, de restauration, et de récupération après incident (point-in-time recovery) pour garantir la sécurité des données.
- **Extensibilité** : Il est possible d'étendre les fonctionnalités de PostgreSQL grâce à des extensions, des langages de programmation intégrés (tels que PL/pgSQL, PL/Python, etc.), et des types de données personnalisés.
- **Support des Fonctions Géospatiales** : PostgreSQL intègre des capacités géospatiales, ce qui le rend adapté aux applications nécessitant des fonctionnalités de géolocalisation et de cartographie.

- **Communauté Active :** La communauté PostgreSQL est dynamique et active. Cela se traduit par des mises à jour régulières, un support en ligne, et une documentation détaillée.
- **Multiplateforme :** PostgreSQL est compatible avec plusieurs plates-formes, ce qui signifie qu'il peut être installé et exécuté sur différents systèmes d'exploitation, y compris Linux, Windows, et macOS.

5 Démonstration

5.1 Swagger

***PatientController - AnalyseController**

analyse-controller Analyse Controller		▼
GET	/api/analyses	obtenirAnalyses
POST	/api/analyses	ajouterAnalyse
GET	/api/analyses/{idAnalyse}	obtenirAnalyseParId
PUT	/api/analyses/{idAnalyse}	modifierAnalyse
DELETE	/api/analyses/{idAnalyse}	supprimerAnalyse
GET	/api/analyses/{idAnalyse}/resultats	obtenirResultatAnalyse
GET	/api/analyses/en-cours	obtenirAnalysesEnCours
echantillon-controller Echantillon Controller		▼
GET	/api/echantillons	obtenirEchantillons
POST	/api/echantillons	ajouterEchantillon
GET	/api/echantillons/{idEchantillon}	obtenirEchantillonParId
PUT	/api/echantillons/{idEchantillon}	modifierEchantillon
DELETE	/api/echantillons/{idEchantillon}	supprimerEchantillon

***ReactifController TestController**

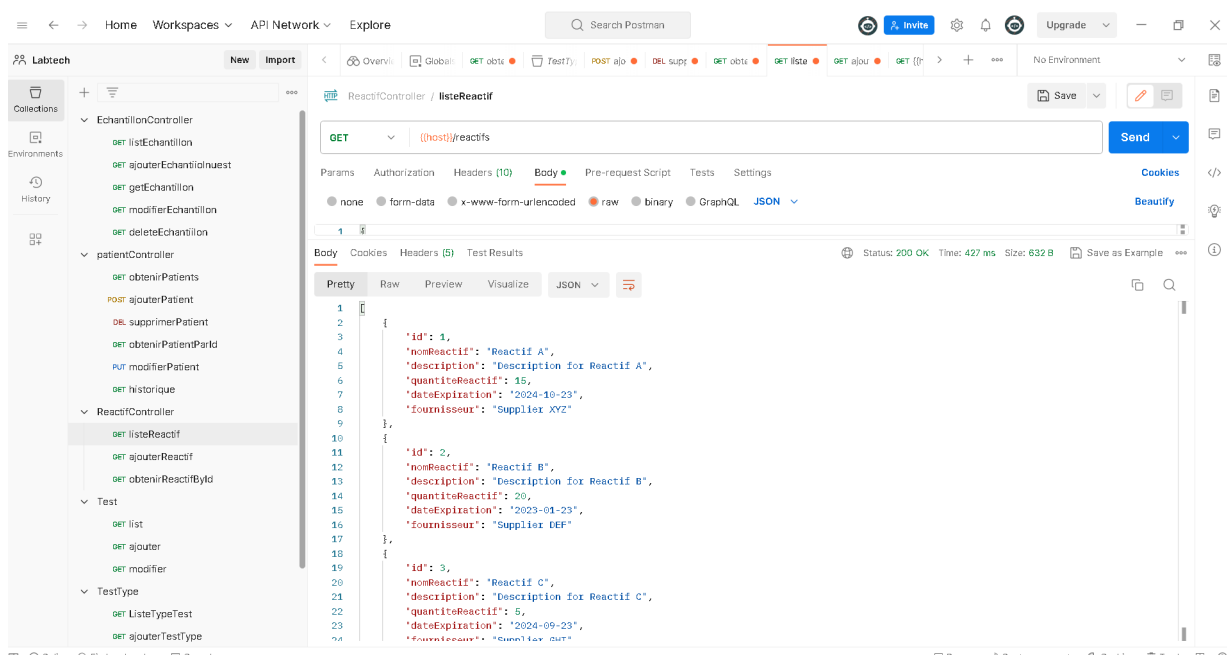
reactif-controller Reactif Controller		▼
GET	/api/reactifs	listeReactifs
POST	/api/reactifs	ajouterReactif
GET	/api/reactifs/{idReactif}	obtenirReactifParId
PUT	/api/reactifs/{idReactif}	modifierReactif
DELETE	/api/reactifs/{idReactif}	supprimerReactif
GET	/api/reactifs/reactifExpire	reactifExpire
GET	/api/reactifs/reactifReptureStock	reactifReptureStock
test-controller Test Controller		▼
GET	/api/tests	listeTest
POST	/api/tests	ajouterTestType
GET	/api/tests/{idTest}	obtenirTestParId
PUT	/api/tests/{idTest}	modifierTest
DELETE	/api/tests/{idTest}	supprimerTest
test-type-controller Test Type Controller		▼
GET	/api/testType	listeTestTypes
POST	/api/testType	ajouterTestType
GET	/api/testType/{idTestType}	obtenirTestTypeParId

*Models

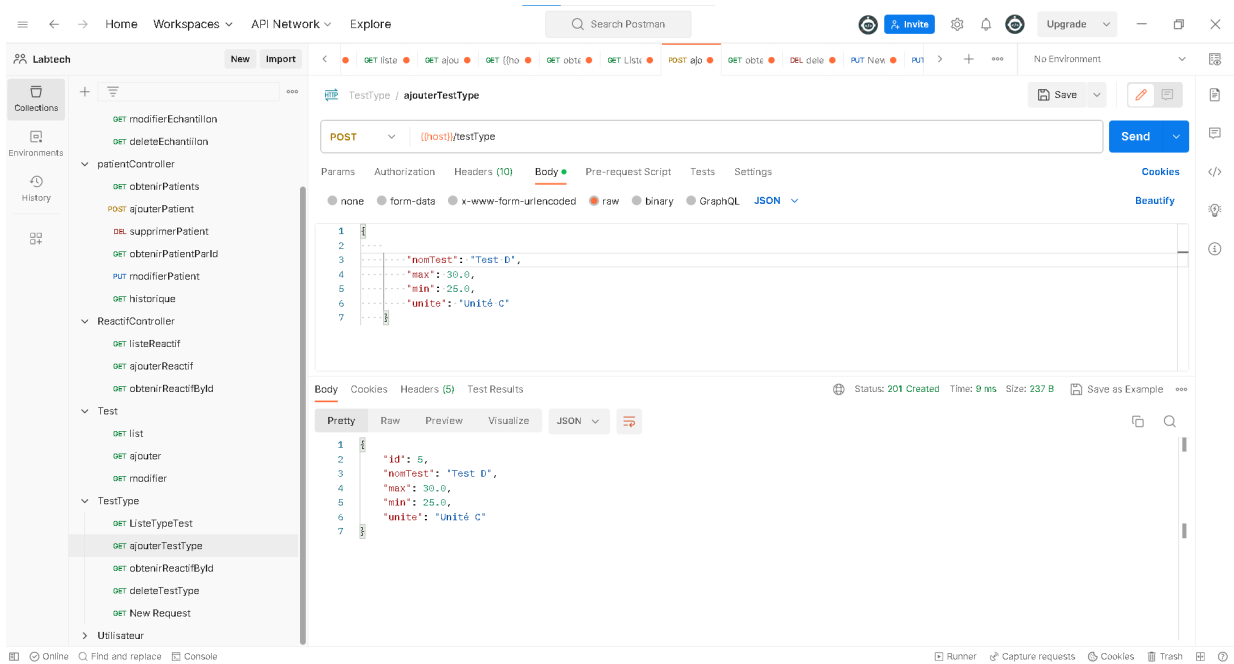


5.2 Tests Manuels -Postman

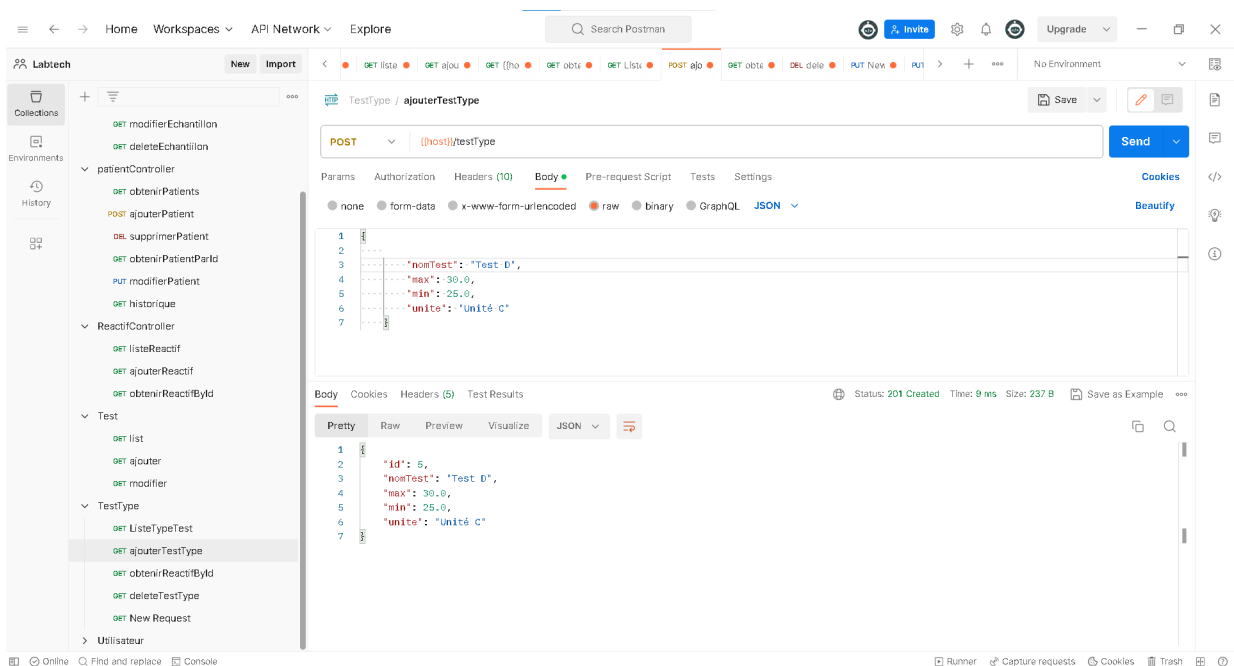
*La liste des réactifs :



*Ajouter un test :



*La liste des échantillons :



5.3 Tests Unitaires -JUnit

* PatientRepositoryTest :

```

Run: TechLabApplication x EchantillonRepositoryTest x
Tests passed: 4 of 4 tests - 548 ms

PatientRepositoryTest (com.example.tec 548 ms)
  ✓ Test get patient By Id 501 ms
  ✓ Test save patient 17 ms
  ✓ Test delete patient 19 ms
  ✓ Test get all patient 11 ms

"C:\Program Files\Java\jdk-17\bin\java.exe" ...
16:23:04.375 [main] DEBUG org.springframework.test.context.BootstrapU
16:23:04.386 [main] DEBUG org.springframework.test.context.BootstrapU
16:23:04.421 [main] DEBUG org.springframework.test.context.BootstrapU
16:23:04.442 [main] DEBUG org.springframework.test.context.support.Abs
16:23:04.442 [main] DEBUG org.springframework.test.context.support.Abs
16:23:04.442 [main] INFO org.springframework.test.context.support.Abs
16:23:04.481 [main] DEBUG org.springframework.test.context.support.Ac
16:23:04.611 [main] DEBUG org.springframework.boot.test.context.Spring
16:23:04.611 [main] INFO org.springframework.boot.test.context.Spring
16:23:04.627 [main] INFO org.springframework.boot.test.context.Spring
16:23:04.629 [main] DEBUG org.springframework.test.context.support.Abs

```

* EchantillonRepositoryTest :

```

Run: TechLabApplication x EchantillonRepositoryTest x
Tests passed: 4 of 4 tests - 541 ms

EchantillonRepositoryTest (com.exempl 541 ms)
  ✓ Test get echantillon By Id 482 ms
  ✓ Test save echantillon 23 ms
  ✓ Test delete echantillon 21 ms
  ✓ Test get all echantillon 15 ms

"C:\Program Files\Java\jdk-17\bin\java.exe" ...
16:26:58.124 [main] DEBUG org.springframework.test.context.BootstrapU
16:26:58.132 [main] DEBUG org.springframework.test.context.BootstrapU
16:26:58.163 [main] DEBUG org.springframework.test.context.BootstrapU
16:26:58.180 [main] DEBUG org.springframework.test.context.support.Abs
16:26:58.181 [main] DEBUG org.springframework.test.context.support.Abs
16:26:58.181 [main] INFO org.springframework.test.context.support.Abs
16:26:58.213 [main] DEBUG org.springframework.test.context.support.Act
16:26:58.335 [main] DEBUG org.springframework.boot.test.context.Spring
16:26:58.336 [main] INFO org.springframework.boot.test.context.SpringB
16:26:58.352 [main] INFO org.springframework.boot.test.context.SpringB
16:26:58.354 [main] DEBUG org.springframework.test.context.support.Abs

```

* UtilisateurRepositoryTest :

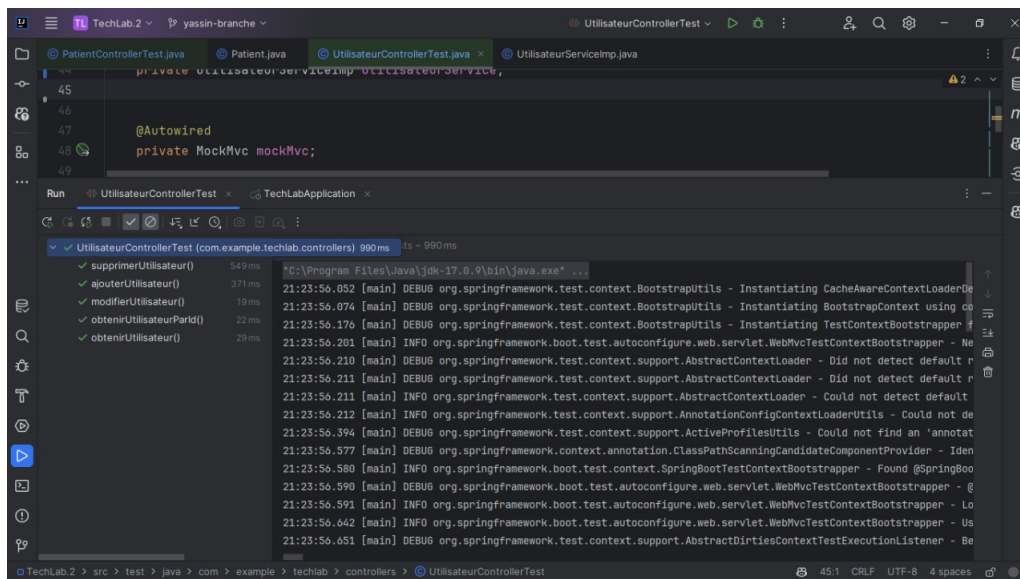
```

Run: TechLabApplication x UtilisateurRepositoryTest x
Tests passed: 4 of 4 tests - 498 ms

UtilisateurRepositoryTest (com.example 498 ms)
  ✓ Test get user By Id 455 ms
  ✓ Test save user 12 ms
  ✓ Test delete user 16 ms
  ✓ Test get all users 15 ms

"C:\Program Files\Java\jdk-17\bin\java.exe" ...
16:29:51.433 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating CacheAwareContextLoaderDelegate from 4
16:29:51.450 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating BootstrapContext using constructor [p
16:29:51.522 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContextBootstrapper for test class
16:29:51.567 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location
16:29:51.568 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location
16:29:51.570 [main] INFO org.springframework.test.context.support.AbstractContextLoader - Could not detect default resource location
16:29:51.699 [main] DEBUG org.springframework.test.context.support.ActiveProfilesUtils - Could not find an 'annotation declaring
16:29:51.817 [main] DEBUG org.springframework.boot.test.context.SpringBootTestContextBootstrapper - @TestExecutionListeners is 4
16:29:51.818 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Loaded default TestExecution
16:29:51.837 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Using TestExecutionListeners
16:29:51.842 [main] DEBUG org.springframework.test.context.support.AbstractDirtyContextTestExecutionListener - Before test class

```

*** UtilisateurControlleurTest :**

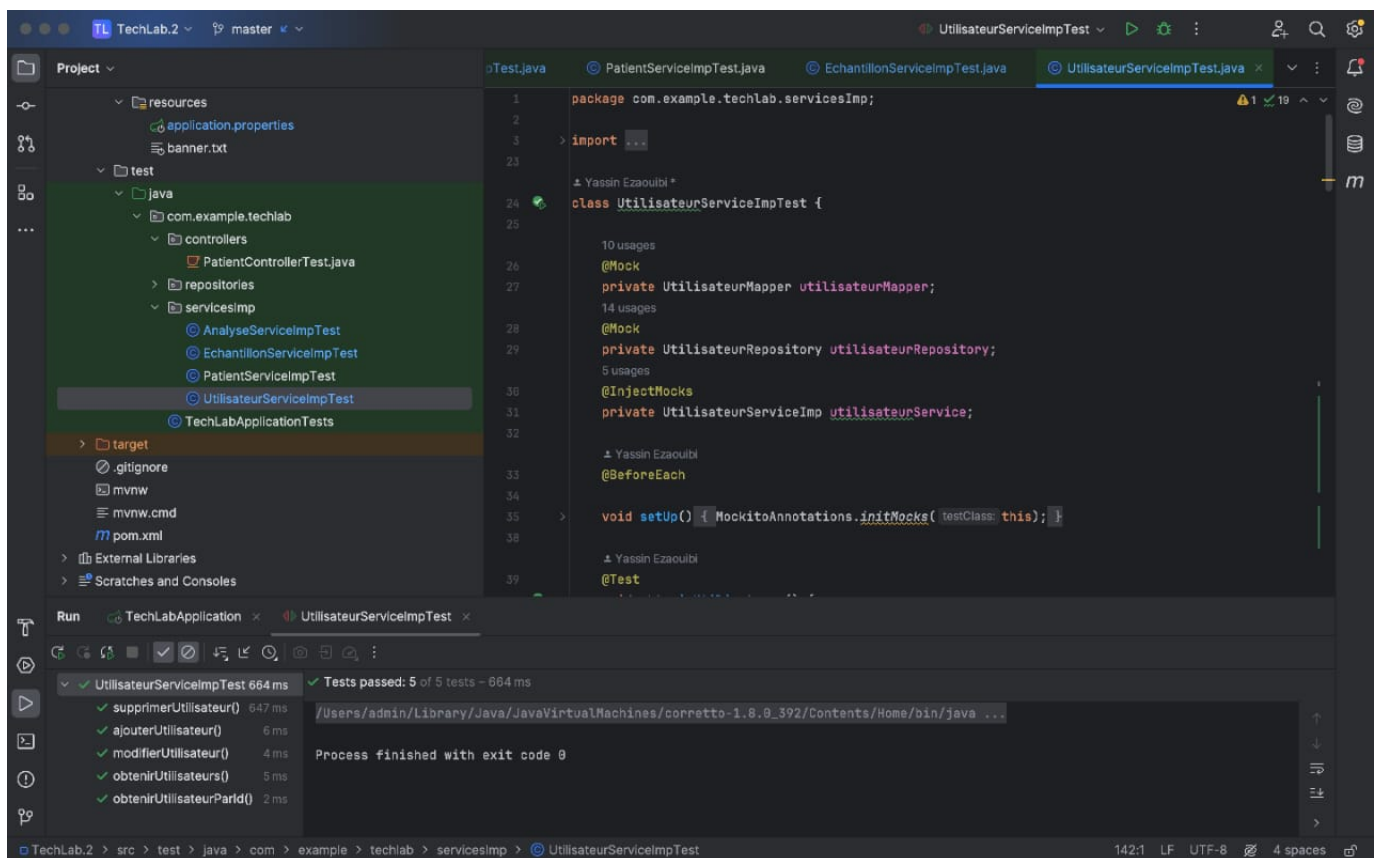
```
private UtilisateurServiceImp utilisateurService;

@Autowired
private MockMvc mockMvc;

Run UtilisateurControlleurTest x TechLabApplication x

UtilisateurControlleurTest (com.example.techlab.controllers) 990 ms
  ✓ supprimerUtilisateur() 549 ms
  ✓ ajouterUtilisateur() 371 ms
  ✓ modifierUtilisateur() 19 ms
  ✓ obtenirUtilisateurParId() 22 ms
  ✓ obtenirUtilisateur() 29 ms

21:23:56.052 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating CacheAwareContextLoaderDe
21:23:56.074 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating BootstrapContext using co
21:23:56.176 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContextBootstrapper f
21:23:56.201 [main] INFO org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTestContextBootstrapper - Ne
21:23:56.210 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default r
21:23:56.211 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default r
21:23:56.211 [main] INFO org.springframework.test.context.support.AbstractContextLoader - Could not detect default
21:23:56.212 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils - Could not de
21:23:56.394 [main] DEBUG org.springframework.test.context.support.ActiveProfilesUtils - Could not find an 'annotat
21:23:56.577 [main] DEBUG org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider - Iden
21:23:56.580 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Found @SpringBoo
21:23:56.590 [main] DEBUG org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTestContextBootstrapper - @
21:23:56.591 [main] INFO org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTestContextBootstrapper - Lo
21:23:56.642 [main] INFO org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTestContextBootstrapper - Us
21:23:56.651 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - Be
```

*** PatientServiceTest :**

```
package com.example.techlab.servicesImp;

import ...

class UtilisateurServiceImpTest {

    10 usages
    @Mock
    private UtilisateurMapper utilisateurMapper;
    14 usages
    @Mock
    private UtilisateurRepository utilisateurRepository;
    5 usages
    @InjectMocks
    private UtilisateurServiceImp utilisateurService;

    Yassin Ezaouli
    @BeforeEach

    void setUp() { MockitoAnnotations.initMocks( testClass: this); }

    Yassin Ezaouli
    @Test

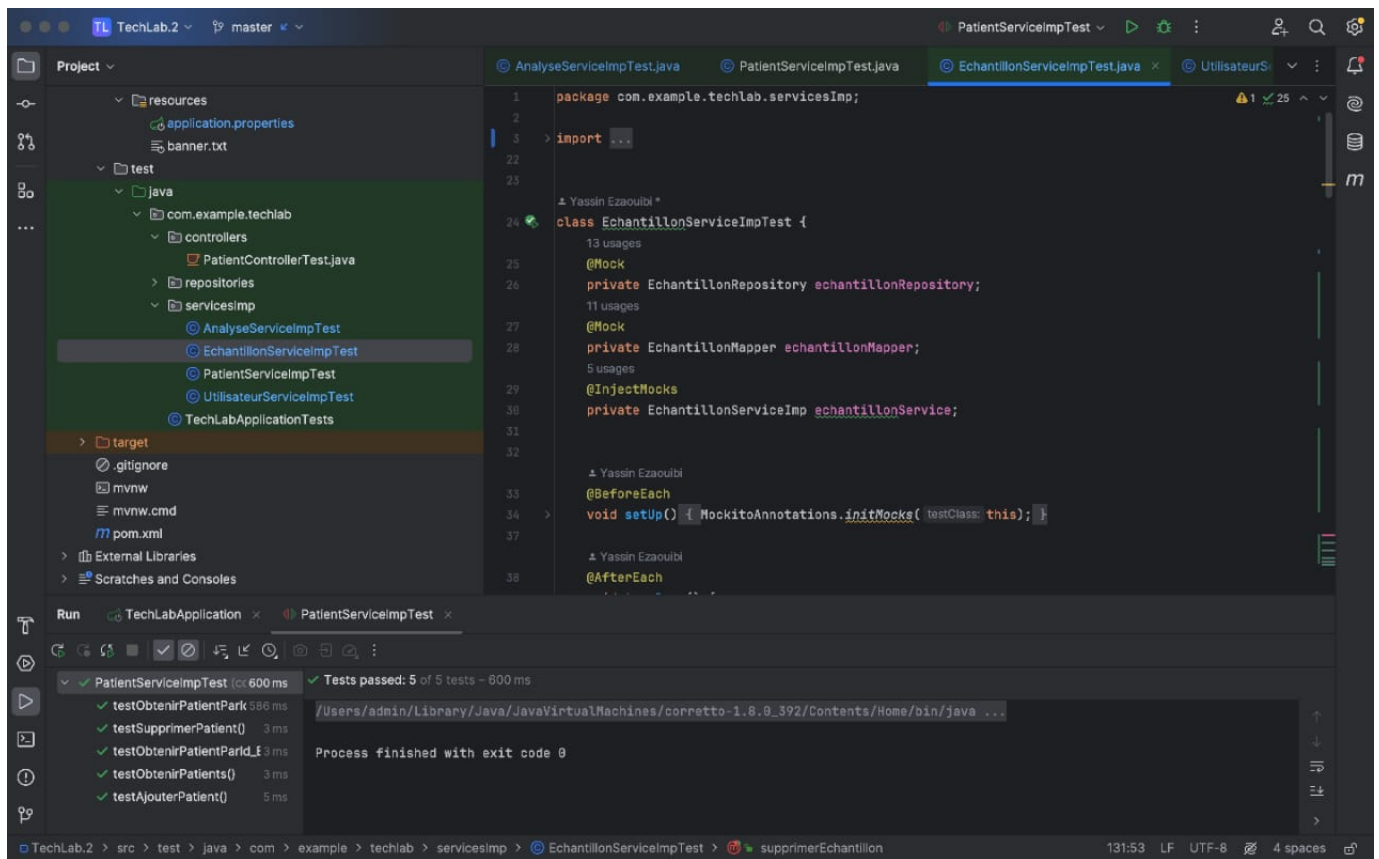
Run TechLabApplication x UtilisateurServiceImpTest x

UtilisateurServiceImpTest 664 ms
  ✓ supprimerUtilisateur() 647 ms
  ✓ ajouterUtilisateur() 6 ms
  ✓ modifierUtilisateur() 4 ms
  ✓ obtenirUtilisateurs() 5 ms
  ✓ obtenirUtilisateurParId() 2 ms

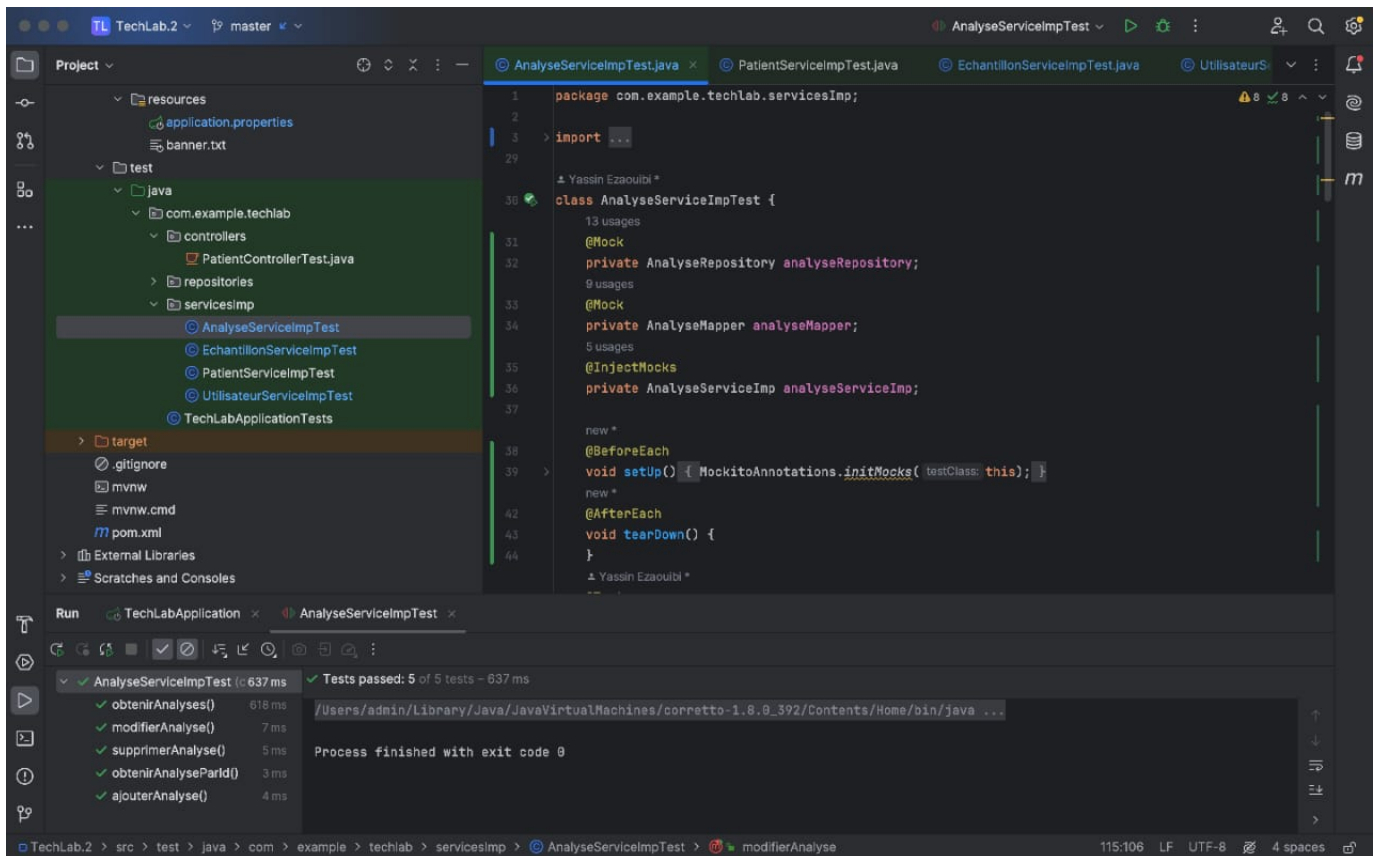
Tests passed: 5 of 5 tests - 664 ms

/Users/admin/Library/Java/JavaVirtualMachines/corretto-1.8.0_392/Contents/Home/bin/java ...

Process finished with exit code 0
```

*** UtilisateurServiceTest :**

* AnalyseServiceTest :



6 Conclusion

En somme, la mise en place des services et des API REST pour notre application de gestion de laboratoire constitue une démarche essentielle pour garantir une communication fluide, une flexibilité accrue, et une gestion sécurisée des données. En adoptant cette approche, nous créons une base solide pour l'évolutivité de notre système, favorisant ainsi une intégration efficace, une maintenance simplifiée, et une réponse agile aux besoins changeants du laboratoire. En résumé, les services REST constituent un pilier stratégique, contribuant à la modernisation, à la robustesse et à la sécurité de notre application.