Rapport de POC (Proof of Concept)

Grégoire Marousé Développeur Java

BUT Informatique 2e année (2B1)

IUT de Vannes - 8 Rue Michel de Montaigne, 56000 Vannes

Accompagnement

Maître d'alternance (CA-TS) : **R. Hannaoui** (rachid.hannaoui@ca-ts.fr)

Tuteur (IUT) : **JF.Kamp** (jean-francois.kamp@univ-ubs.fr)

Enseignants responsables de l'alternance à l'IUT

G.Kerbellec (goulven.kerbellec@univ-ubs.fr)

T.Godin (thibault.godin@univ-ubs.fr)

Octobre 2025

Introduction

Dans le cadre de mon alternance au Crédit Agricole Technologies & Services, je vais intégrer l'équipe en charge du développement back-end de l'application de paiement mobile. Ainsi, je vais travailler avec Java Spring Boot et des API REST. Afin de mieux comprendre l'environnement technique dans lequel je vais évoluer, j'ai réalisé, dans le cadre de la SAE 301 à l'IUT, ce Proof of Concept (POC).
Ce mini-projet s'inscrit dans la continuité de ma veille technologique. L'objectif de ce POC est donc de me familiariser concrètement avec la stack Java / Spring Boot / API REST, de comprendre son fonctionnement interne et ses conventions, et d'acquérir les réflexes nécessaires pour débuter mon alternance dans de bonnes conditions. Il ne s'agit pas ici d'un projet en lien direct avec l'application bancaire de l'entreprise, mais plutôt d'un projet d'apprentissage personnel permettant de consolider mes bases et de préparer mon arrivée en entreprise.

Mise en oeuvre

Pour mener à bien cette démarche, j'ai découpé mon apprentissage en plusieurs phases progressives.

Phase 1 – Découverte et fondamentaux de Spring Boot

J'ai commencé par suivre la formation Spring Boot Tutorial – Full Course (YouTube), afin de découvrir les bases du framework et sa philosophie. Cette première étape m'a permis de comprendre comment configurer un projet avec Maven, gérer les dépendances via le fichier pom.xml et appréhender le fonctionnement de l'injection de dépendances et de l'auto-configuration Spring.
J'ai également pu aborder le pattern MVC (Model-View-Controller), le rôle des principales annotations (@SpringBootApplication, @Component, @Autowired) ainsi que la logique de structuration d'un projet.
Dans le cadre de ce cours, j'ai développé une petite application web basée sur un contrôleur, quelques vues HTML et la gestion de formulaires simples. Cette première réalisation m'a permis de comprendre la structure et le cycle de vie d'une application Spring Boot. L'application ne figure pas dans le rendu car j'ai simplement suivi le tutoriel.

Phase 2 – Création d'API REST avec Spring Boot

La deuxième phase, plus orientée vers le cœur de mon futur travail, portait sur la création d'API REST à l'aide de Spring Boot. J'ai suivi la formation RESTful Web Services with Spring Framework – A Quick Start (Udemy), qui m'a permis d'approfondir la logique RESTful, les méthodes HTTP (GET, POST, PUT, DELETE) et les annotations spécifiques telles que @RestController, @RequestMapping, @GetMapping ou @PostMapping.

J’ai également appris à gérer les réponses HTTP à l’aide de ResponseEntity, à valider les entrées utilisateurs grâce à @Valid, @NotBlank, @Email et à mettre en place une gestion d’erreurs centralisée avec @ControllerAdvice.
L’exercice final consistait à concevoir une API de gestion d’utilisateurs comprenant plusieurs endpoints CRUD, testés avec Postman pour valider le bon fonctionnement, la conformité aux standards REST et la robustesse face aux erreurs.
Cette étape m’a permis de consolider ma compréhension du modèle REST et de la façon dont Spring Boot facilite la création de services web modernes, structurés et maintenables. Le projet ne figure pas non plus dans le rendu. L'auteur de la formation met à disposition la version finale du projet qui est développé au fur et à mesure des cours sur ce github.

Phase 3 – Projet TODO API - Application pratique

Pour mettre en œuvre l’ensemble des connaissances acquises lors des deux premières phases, j’ai réalisé un mini-projet : une API REST de gestion de tâches ou “todo liste”. L’objectif de cette phase était de consolider ma compréhension du développement d’API REST avec Spring Boot, tout en respectant les conventions professionnelles de structuration du code et de séparation des responsabilités.
Afin de garder le projet simple et pédagogique, j’ai choisi d’utiliser un stockage en mémoire plutôt qu’une base de données. Ce choix m’a permis de me concentrer sur les principes clés du développement d’API sans complexité supplémentaire liée à la persistance. Le stockage est assuré à l’aide d’une simple liste Java.
J’ai structuré le projet selon une architecture en deux couches, inspirée des bonnes pratiques vues en formation :

- La couche Controller gère les requêtes HTTP, les routes et les codes de retour.
- La couche Service contient la logique métier ainsi que la gestion du stockage en mémoire.
- Une classe Model (Task) représente les données manipulées.
- Un gestionnaire global d’exceptions (GlobalExceptionHandler) permet de renvoyer des messages d’erreur clairs et structurés en cas de problème.

L’API expose plusieurs endpoints pour gérer les tâches : création, lecture, modification et suppression (CRUD complet). J’ai ensuite testé l’ensemble des fonctionnalités à l’aide de Postman, en vérifiant les retours HTTP, le bon format des réponses JSON et la gestion des erreurs côté serveur.
Cette phase a marqué une étape importante dans ma progression : elle m’a permis de passer de l’apprentissage théorique à une mise en pratique concrète et autonome.

Documentation de l'API TODO

Cette documentation a pour but de présenter clairement les différents endpoints, leurs rôles, les codes de réponse attendus, ainsi que les principaux cas de test effectués.

Pour lancer le projet :

1. Se déplacer dans le dossier du projet
2. Utiliser la commande : **./mvnw clean install**
3. Démarrer avec : **./mvnw spring-boot:run**
4. Dans le terminal attendre l'apparition de la ligne "Ready"

L'API TODO permet de gérer une collection de tâches à travers un ensemble d'opérations conformes aux standards RESTful. Chaque endpoint correspond à une action CRUD bien définie (création, lecture, mise à jour, suppression), en utilisant les méthodes HTTP appropriées :

- GET pour la lecture des ressources
- POST pour la création
- PUT pour la mise à jour complète
- DELETE pour la suppression

Les données échangées sont au format JSON, tant pour les requêtes que pour les réponses. Chaque appel a été testé et validé à l'aide du logiciel Postman.

Lors des tests, j'ai vérifié plusieurs éléments essentiels :

- La conformité des codes de statut HTTP (200 OK, 201 Created, 404 Not Found, 400 Bad Request, 500 Internal Server Error) selon le résultat attendu.
- La validité du contenu JSON retourné (structure, types, cohérence des champs).
- La validation des données entrantes, notamment les cas de champs manquants, incorrects ou mal formatés.
- La gestion centralisée des erreurs, assurée par mon gestionnaire global d'exceptions (GlobalExceptionHandler), permettant d'obtenir des messages d'erreur clairs et normalisés.

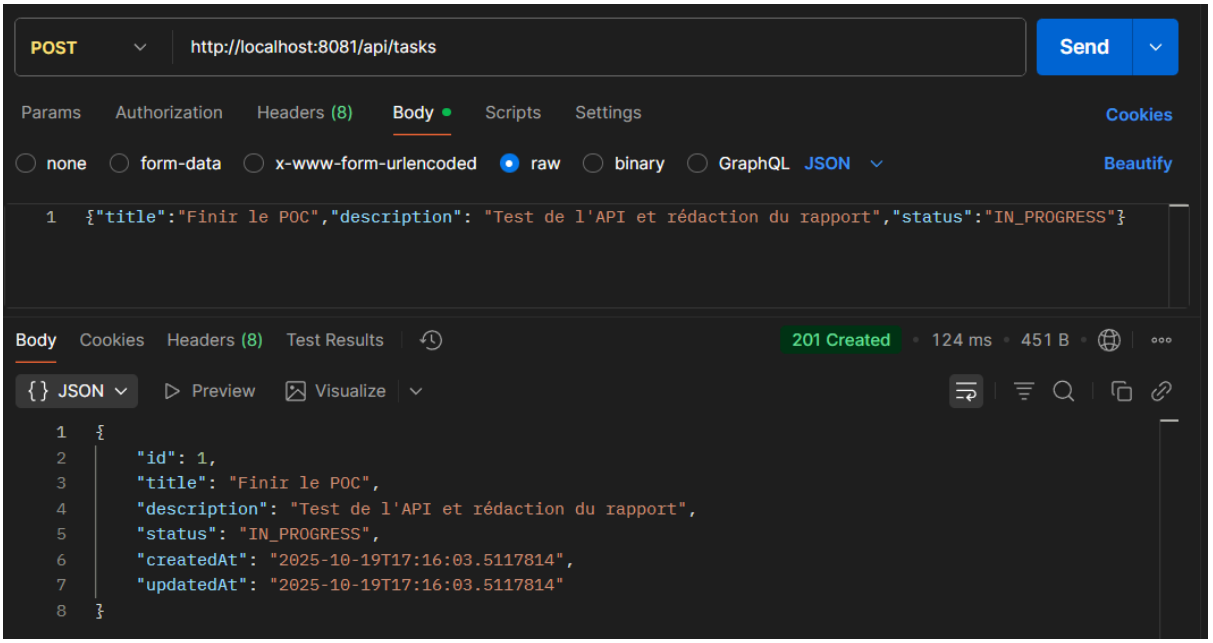
La documentation s'appuie sur ces tests pour présenter, pour chaque endpoint, les éléments suivants :

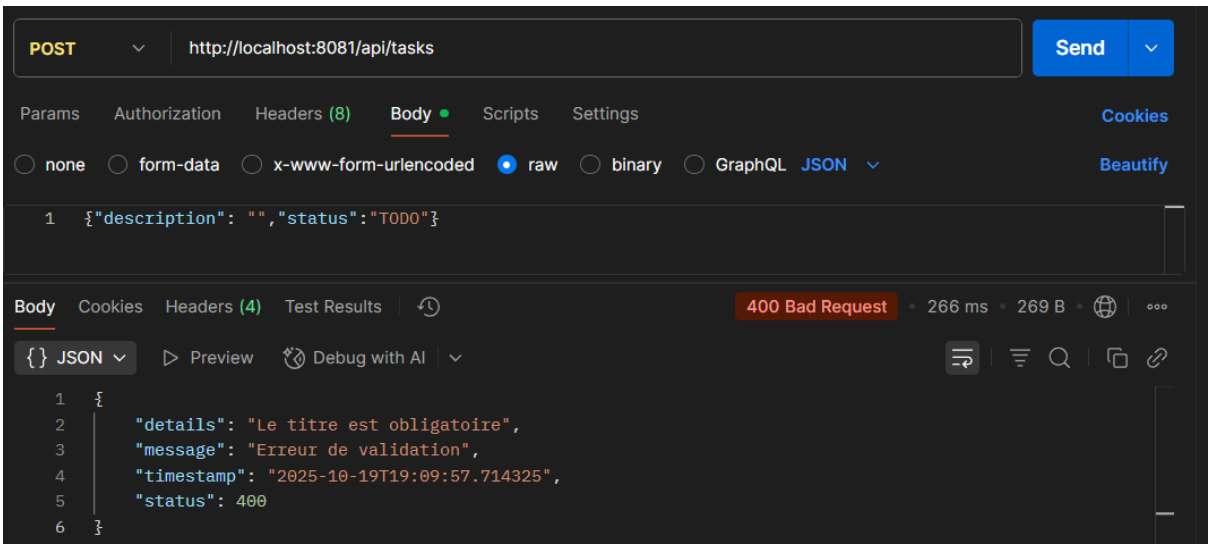
- L'URL et la méthode HTTP utilisée
- Le corps de requête (Request Body) attendu lorsqu'il y en a un
- Un exemple de réponse réussie (200/201)
- Les cas d'erreur (400, 404, etc.) illustrés par des captures d'écran issues de Postman

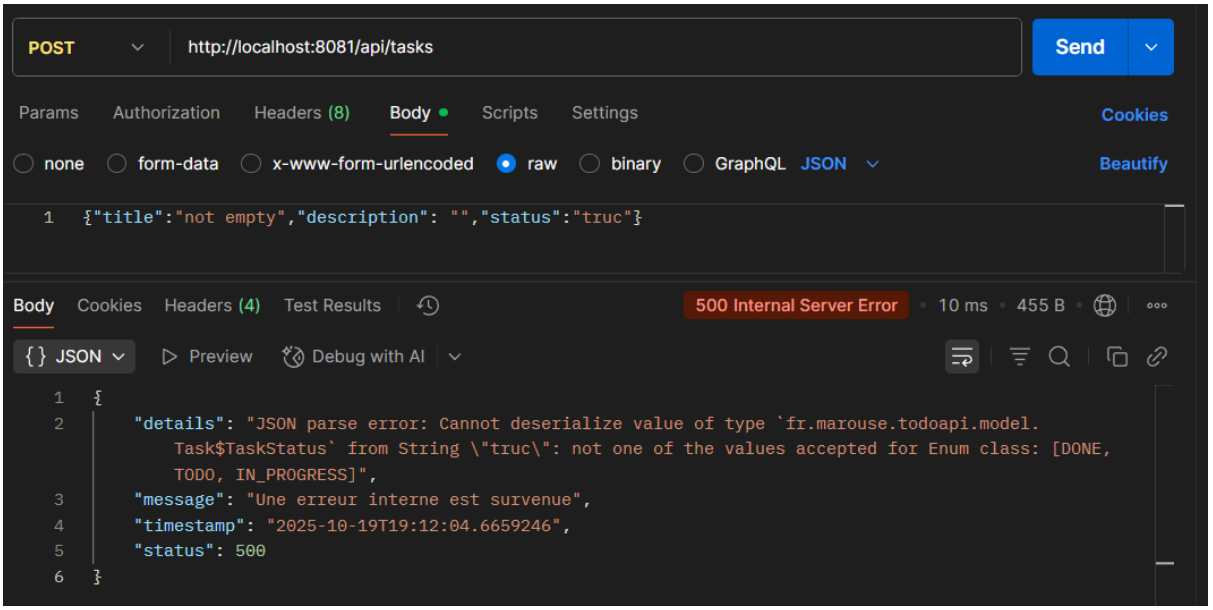
L'ensemble des résultats de test est présenté ci-dessous sous forme de captures Postman, illustrant les différents scénarios possibles pour chaque endpoint de l'API TODO.

Endpoints implémentés

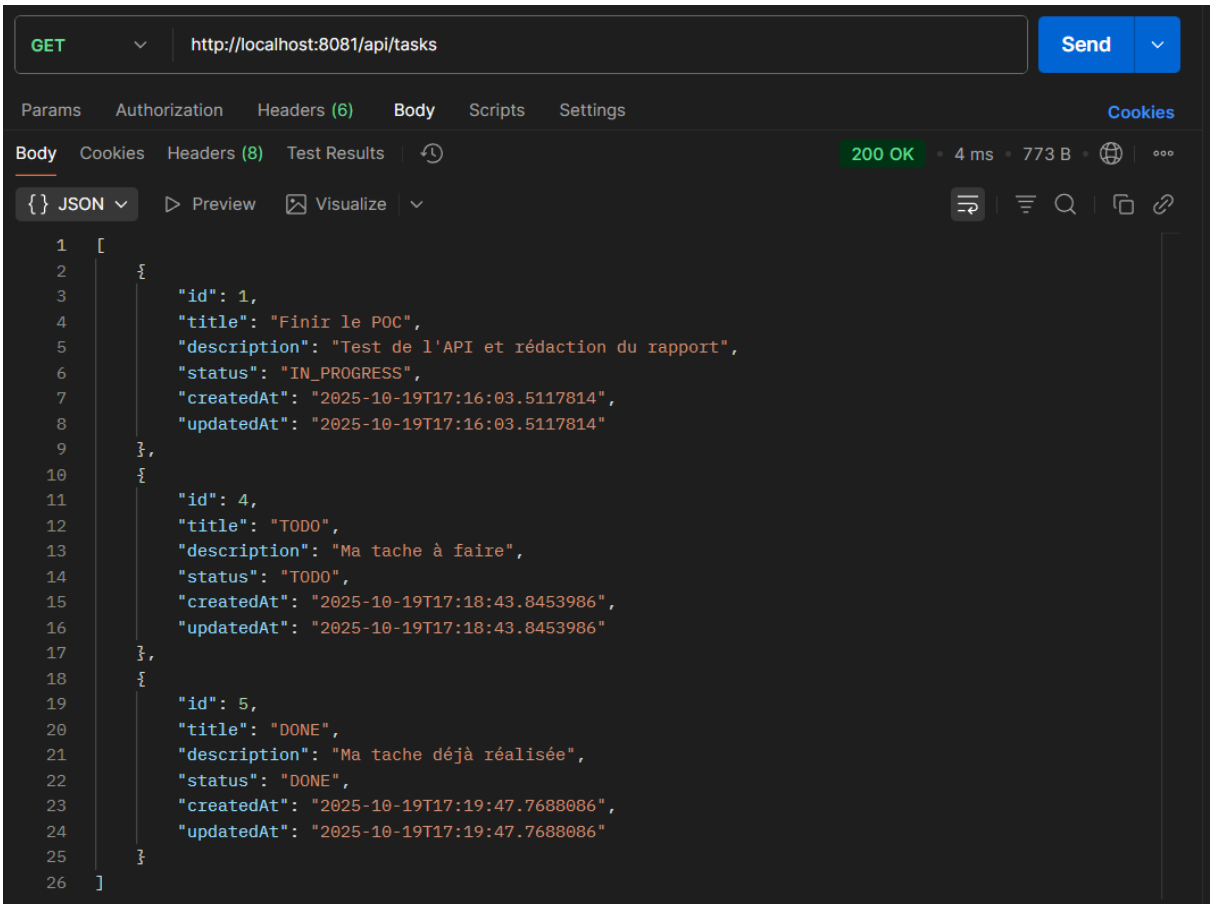
1. Créer une tâche





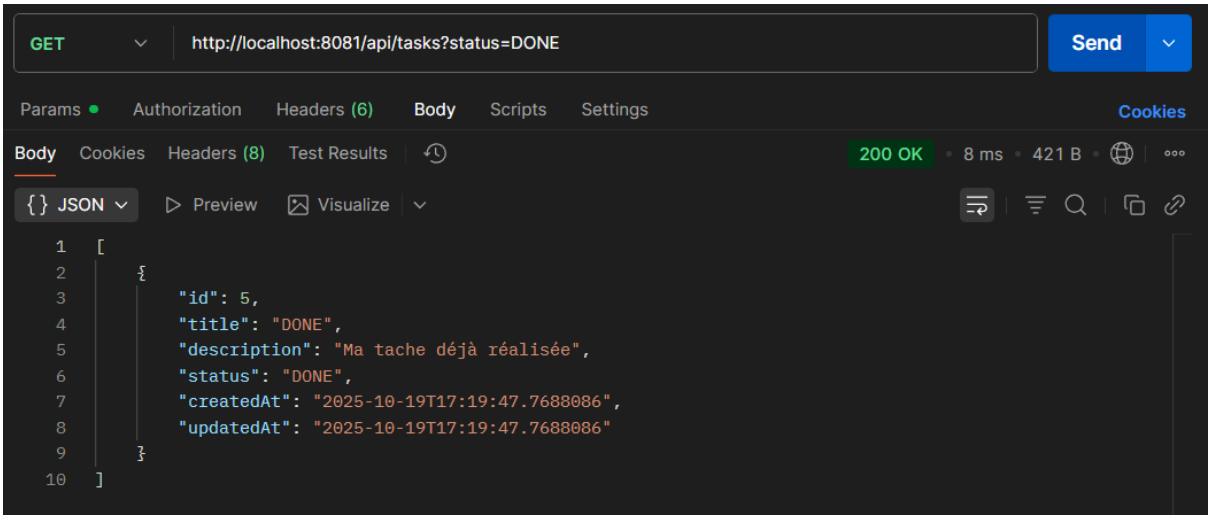


2. Lister toutes les tâches

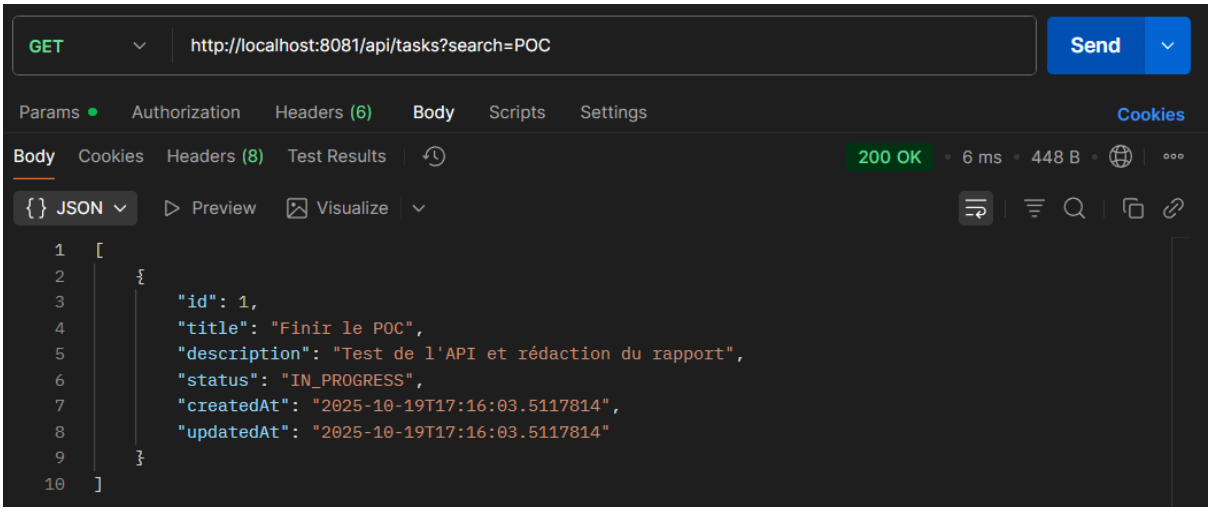


3. Filtrer les tâches par statut

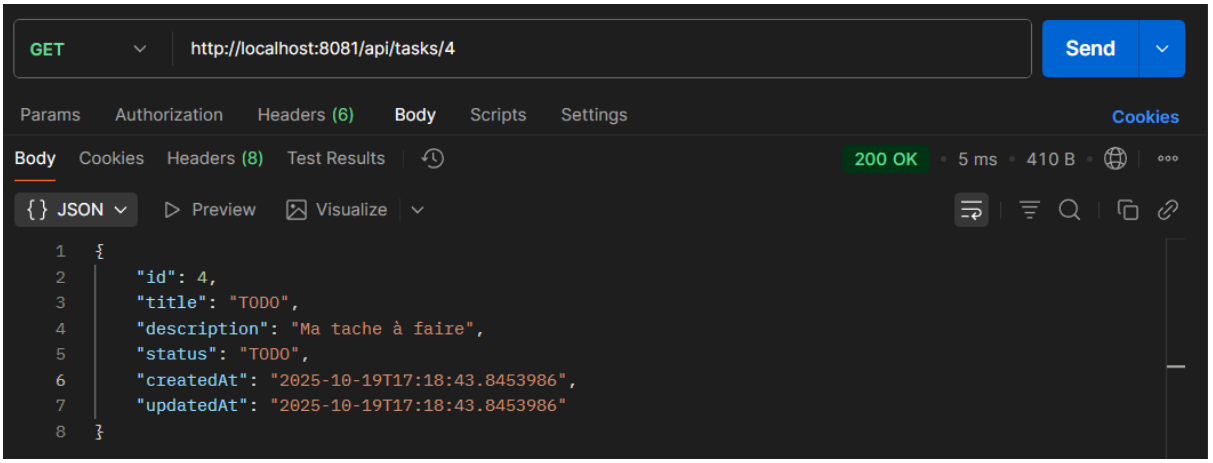
Statuts disponibles : `TODO`, `IN_PROGRESS`, `DONE`

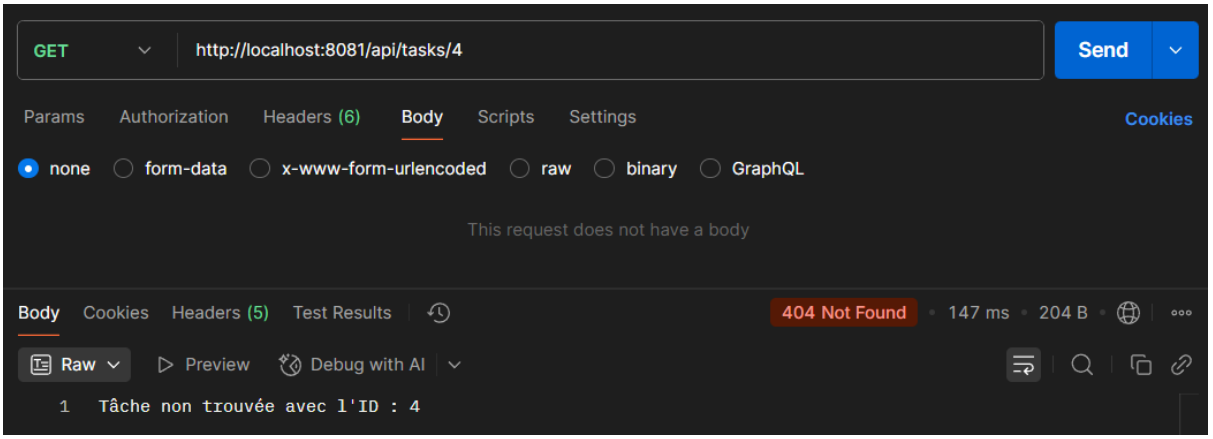


4. Rechercher par titre

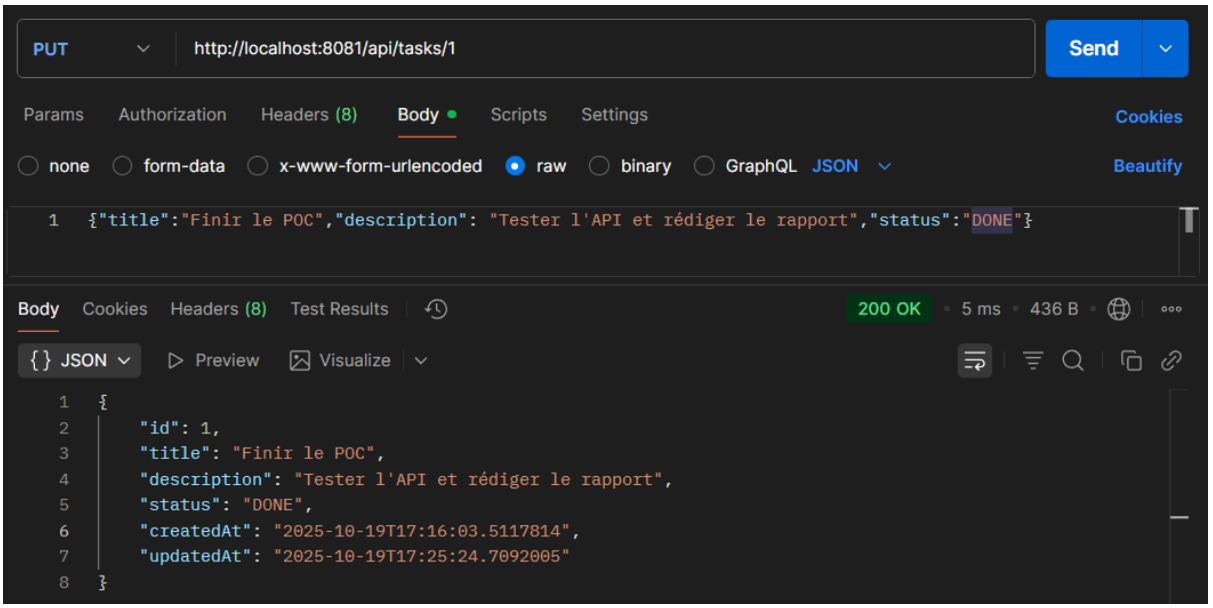


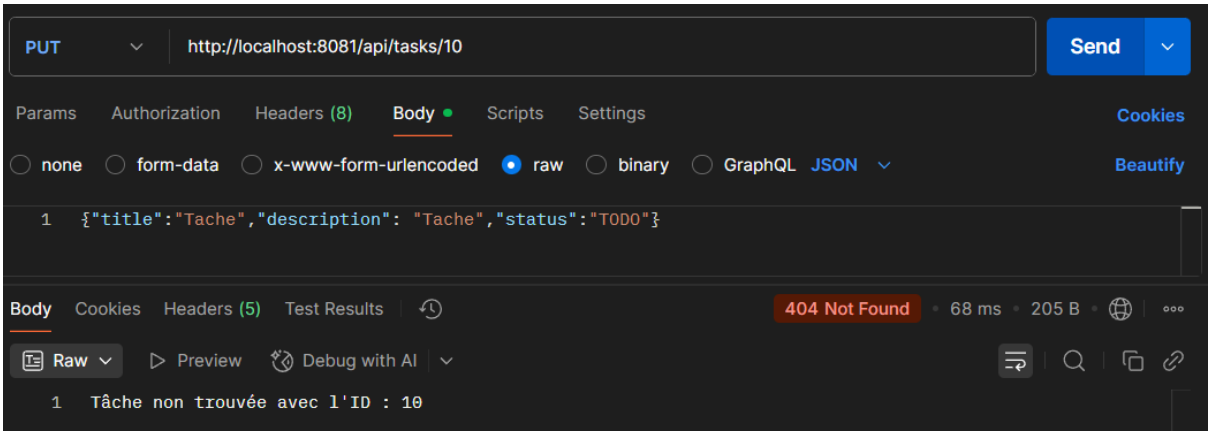
5. Récupérer une tâche par ID

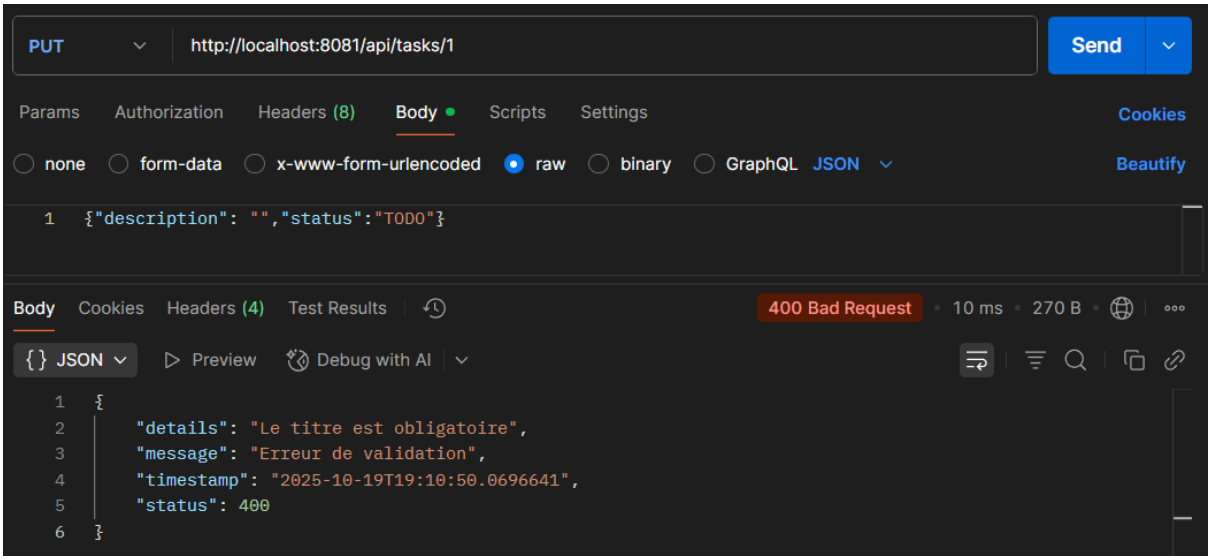




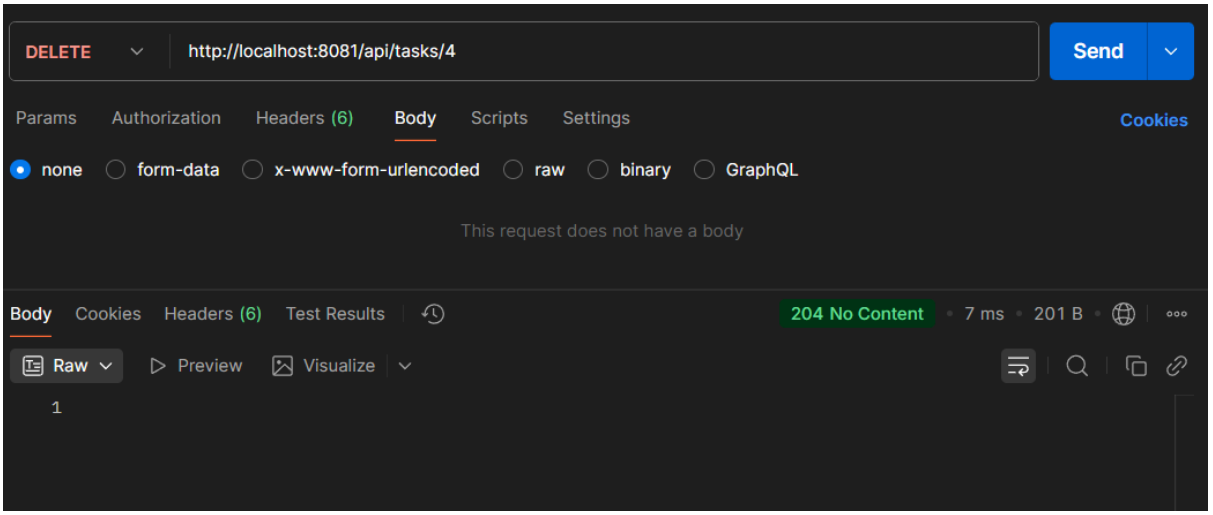
6. Modifier une tâche

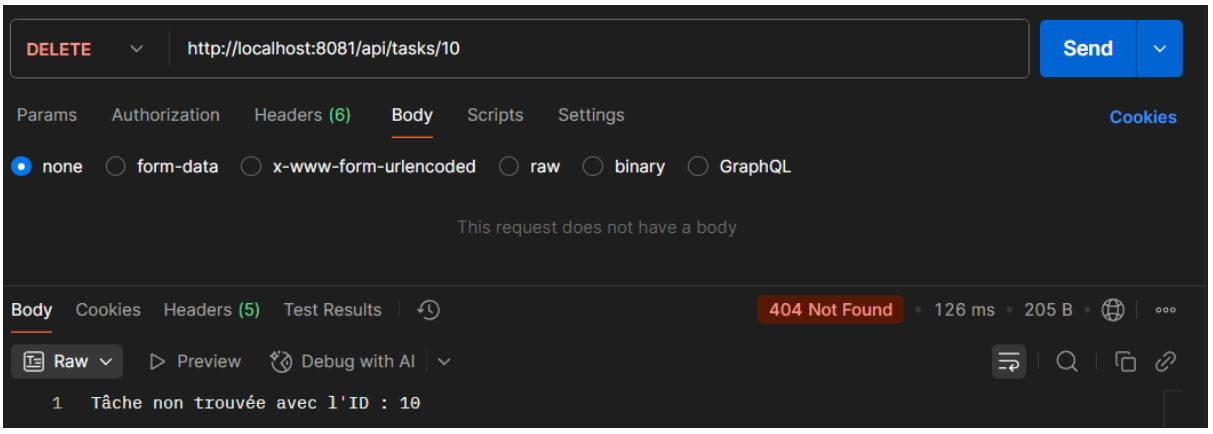






7. Supprimer une tâche





Conclusion

La réalisation de ce Proof of Concept m'a permis d'aborder concrètement les fondements de Spring Boot et des API REST, deux technologies centrales dans mon futur environnement de travail. Ce projet, mené de manière progressive, m'a offert une compréhension claire de la structure et du fonctionnement d'une application back-end (Java Spring Boot), depuis la configuration initiale jusqu'à la gestion complète du cycle de vie d'une ressource via des endpoints RESTful.

Le mini-projet développé, bien que élémentaire, intègre déjà les bonnes pratiques de développement professionnel : architecture en couches, validation des données, gestion centralisée des erreurs et respect des conventions REST.

Les compétences acquises sont directement transposables au contexte de mon alternance au Crédit Agricole Technologies & Services, notamment pour le développement et la maintenance d'API de transactions dans le cadre de l'application de paiement mobile.

Enfin, ce POC a également mis en évidence les axes à approfondir pour consolider mes compétences : intégration d'une base de données, ajout d'un système de sécurité ou encore mise en place de tests unitaires.