جامعة سيدي محمد بن عبد الله بفاس
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES

كلية العلوم والتقنيات فاس
FACULTÉ DES SCIENCES ET TECHNIQUES DE FÈS

Projet de Fin d'Etudes

Licence Sciences et Techniques Génie Informatique

Département Informatique

Développement d'APIs d'une plateforme digitale pour le secteur touristique

Lieu de stage : NAWRA TECHNOLOGY

Réalisé par :

- **MELLOUKY Mohamed**

Encadré par :

- **Pr. OUZARF Mohamed**
- **Mr. KAARAR Mohamed**

Soutenu le 4 /7 /2022 devant le jury composé de :

- **Pr. R. BEN ABBOU**
- **Pr. S. NAJAH**
- **Pr. M. OUZARF**

Année Universitaire 2021-2022

Remerciement

Je suis heureux d'écrire ces lignes de remerciement à toutes les personnes qui m'ont aidé pendant ma vie et ma carrière académique, Notamment mes parents qui ont faits le maximum possible pour que je puisse continuer mes études. Je tiens à adresser mes remerciements les plus sincères à tout le corps professoral et administratif de la Faculté des Sciences et des Techniques de Fès. À tous les professeurs qui m'ont formé en tronc commun et en licence génie informatique, particulièrement monsieur A. ZAHI, chef du département informatique à la faculté des sciences et des techniques, Fès, et madame F. MRABTI, responsable de la licence génie informatique à la FST Fès, je la remercie pour son effort qu'elle fait pour communiquer avec les étudiants, pour qu'ils puissent profiter d'une formation complète, dans un climat de confiance, et convivialité.

Je tiens à remercier monsieur Mohamed OUZARF**,** qui m'a encadré tout au long de ce stage. Sa disponibilité, ses conseils, son aide et ses réponses à mes questions ont été utiles pour avancer dans le stage.

Je tiens à remercier également tout le corps de NAWRA TECHNOLOGY pour l'accueil qui m'a réservé. Je remercie précisément monsieur Mohamed KAARAR, pour son encadrement, ses informations qui a partagé avec les stagiaires, tout le long de la période du stage. Je le remercie encore une autre fois pour les points de suivi qu'on a fait chaque semaine, c'est là où il était toujours disponible à répondre à mes questions.

Je remercie aussi les membres de jury, Pr. R. BEN ABBOU et Pr. S. NAJAH, pour avoir accepté d'être parmi les membres de jury de mon projet de fin d'étude pour la licence en informatique.

Résumé

Dans le cadre de mon stage de fin d'étude à la faculté des sciences et techniques à Fès, j'ai eu l'honneur de travailler sur le projet Atlas au sein de l'entreprise NAWRA TECHNOLOGY.

L'objectif du projet est la mise en place d'une plate-forme qui permet aux touristes de louer des biens proposés par des individus et des entreprises spécialisées. Les entreprises spécialisées peuvent proposer des voyages organisés et leurs services. L'entreprise a adopté l'architecture micro-service pour mettre en place la plate-forme.

Pendant mon stage, j'ai réalisé l'API *communicationApi*. Cette API est responsable de gérer les communications entre le système et le client et entre les utilisateurs inscrits dans la plate-forme.

D'abord j'ai commencé le développement par la mise en place des opérations CRUD en respectant le contrat d'interface de l'API, alors que l'envoie des e-mails est encore en étude technique.

Toutes les APIs du projet ont été développées par l'écosystème JAVA, particulièrement Framework spring.

Abstract

As part of the bachelor degree program at the faculty of sciences and technologies at Fes, I had the honor to work on Atlas project as an intern at Nawra Technology.

Atlas project aims to build a platform in order to help tourists coming to Morocco from all over the world find rentals made available by individuals subscribed in this platform, also it makes possible to companies to offer trips and different activities in the platform. Nawra technology has chosen to implement the project using micro service architecture.

During my intern, I have worked in an API called *communicationApi*. It has to establish a communication between the system and the client, also, between the clients subscribed in the platform.

First, I have started to develop the CRUD operations of the API in order to persist data as well as making API responses respect the API documentation provided by the company, while sending e-mails is still under technical study.

All APIs are developed using JAVA ecosystem, in particular JAVA framework spring.

Sommaire

Liste des Figures

Liste des acronymes

API	Application programming interface
SMS	Short message system
JSON	JavaScript Object Notation
http	HyperText Transfer Protocol
SGBD	Système de gestion des bases de données
URI	Uniform ressource Identifier
URL	Uniform Ressource Locator
REST	Representational state transfer
CRUD	CREATE-READ-UPDATE-DELETE
IHM	Interface homme machine

Introduction générale

L'informatique est la science de résoudre des problèmes du monde réel par des solutions informatiques. Elle se base sur l'ensemble des connaissances théoriques du domaine et les prérequis scientifiques pour concevoir et développer des systèmes et des logiciels qui répondent aux exigences du client.

Le stage de deux mois au sein de NAWRA TECHNOLOGY est une expérience qui m'a permis de sentir le sens de résoudre des problèmes du monde réel par des solutions informatiques. En fait, j'ai travaillé sur un micro-service d'un système qui facilite aux touristes de trouver des locations au Maroc.

Dans ce rapport, je présente les différentes étapes menant à atteindre l'objectif du projet.

En fait, le premier chapitre présente le lieu du stage, une description du projet en général et une description de l'API que j'ai développée.

Je présente dans le deuxième chapitre l'analyse et la conception du micro-service.

Le troisième chapitre présente les différentes technologies utilisées, en plus, une présentation qui montre le fonctionnement de l'API qui établit une communication entre les différents micro-services du système.

Finalement, j'ai mentionné les différentes compétences acquis dans cette expérience.

Chapitre 1 : Contexte général du projet

1. Organisme d'accueil

1.1. Présentation de l'entreprise

Figure 1: Logo de Nawra Technology

Nawra Technology est une société marocaine, basée à Fès, créée par des développeurs intéressés par le développement des nouvelles solutions informatiques dans le secteur touristique au royaume du Maroc.

La mission de NAWRA TECHNOLOGY est de trouver et implémenter des solutions informatiques à des problèmes du monde réel pour ses clients. L'une de ces solutions est celle d'informatiser le secteur touristique au Maroc, en créant une plate-forme qui facilite aux touristes de trouver des auberges et des hôtels à réserver, des locations, des excursions et des voyages organisés par des établissements spécialisées dans le domaine.

1.2. Gestion de projet

L'entreprise utilise une méthode Agile comme méthodologie de gestion de projet. Les valeurs de la méthode Agile permet aux équipes de développement de collaborer dans un projet, comme il permet une forte implication du client dans le cycle de vie de développement du logiciel, et peut répondre facilement à des modifications. La méthode Agile consiste à décomposer le projet en des petites tâches, réaliser ses tâches d'une manière incrémental itérative implique la réalisation du projet.

Les tâches que nous avons fait dans l'entreprise passe par quatre phases :

- À faire
- En cours
- En revue
- Terminé

En premier temps la tâche est considérée comme une tâche à faire, en commençant à la réaliser, cette tâche passe à la phase *en cours*, une fois qu'elle est effectuée elle devient une tâche *en revue,* mon encadrant d'entreprise fait une révision, il me suggère des corrections, s'il n'y a pas de modifications, la tâche passe à l'état *terminé*.

Figure 2: Logo Trello

Puisque l'entreprise exécute ses activités complétement à distance, elle utilise l'application web **Trello** pour visualiser les différentes tâches, à qui elles sont affectées et dans quelle phase elles sont à un moment donné.

Figure 3: Kanban board

1.3. Planification du projet

Le diagramme de Gantt suivant montre les différentes tâches prévues pour le stage, en plus l'intervalle de temps alloué pour chaque tâche.

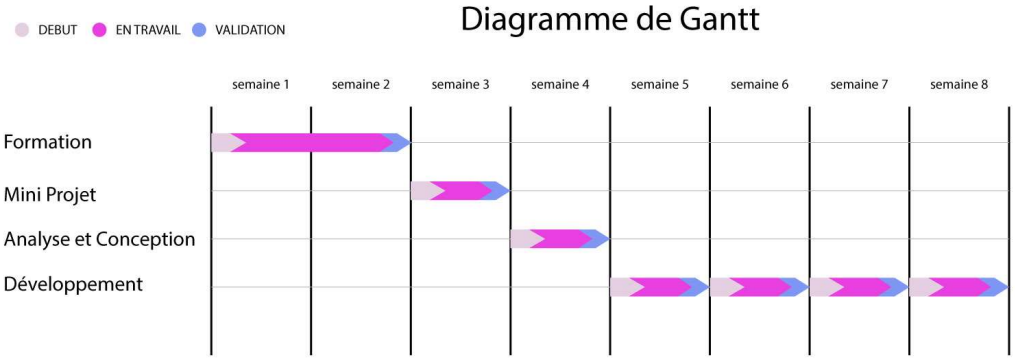


Figure 4: Diagramme de Gantt

2. Etude de l'existant

NAWRA TECHNOLOGY est une nouvelle entreprise dans le monde du TECH, elle n'a pas encore des infrastructures informatiques déjà déployées.

L'API *communicationApi,* comme toutes les autres APIs de l'entreprise, elle est encore dans la phase de développement. Elle n'est pas encore fonctionnelle, en fait, elle ne répond à aucune requête http. La figure ci-dessus représente le projet dans l'état initial.

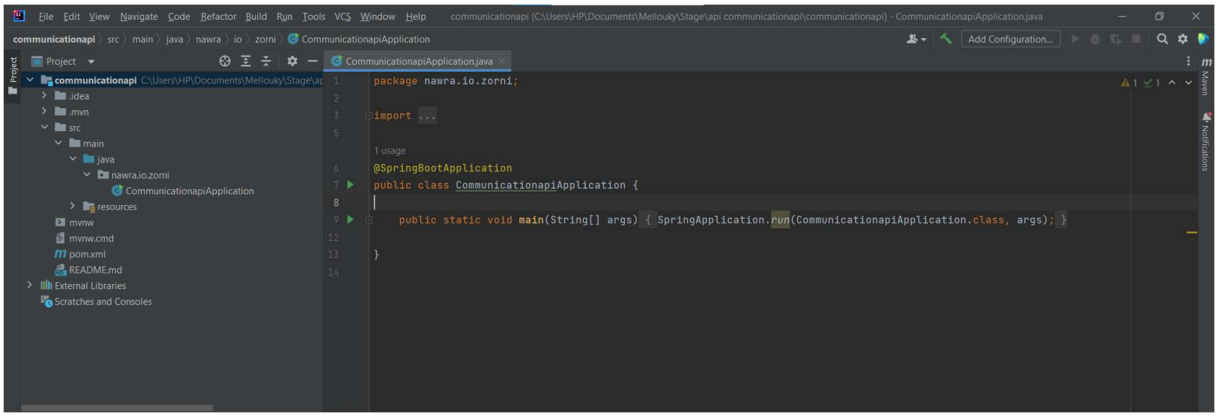


Figure 5: Structure du projet lors du début du stage

Alors, il faut poser une structure du projet qui répond aux exigences suivantes :

- Rendre le projet lisible
- Séparer les données des traitements.

En plus, il faut implémenter les exigences de l'API en respectant le contrat d'interface.

Le contrat d'interface est une documentation de l'API, il indique les données qu'une API a besoin pour réaliser ses fonctionnalités, comment on peut consommer cette API et les différentes réponses données par cette API.

3. Description du projet

3.1. Vue générale

Le projet **ATLAS** consiste à réaliser une plate-forme qui permet aux touristes (en général, aux utilisateurs) de trouver des locations au Maroc, comme il permet aux individus de louer leurs biens, en plus, des entreprises spécialisées peuvent proposer des voyages organisés.

Pour réaliser cette plate-forme, l'entreprise a adopté une architecture micro-service, au lieu de créer une application monolithique qui répond à tous les besoins fonctionnels du projet, on crée plusieurs micro-services, chaque service est responsable d'une fonctionnalité bien précisée, ses micro-services communiquent entre eux par des APIs.

La figure ci-dessous représente l'architecture du système :

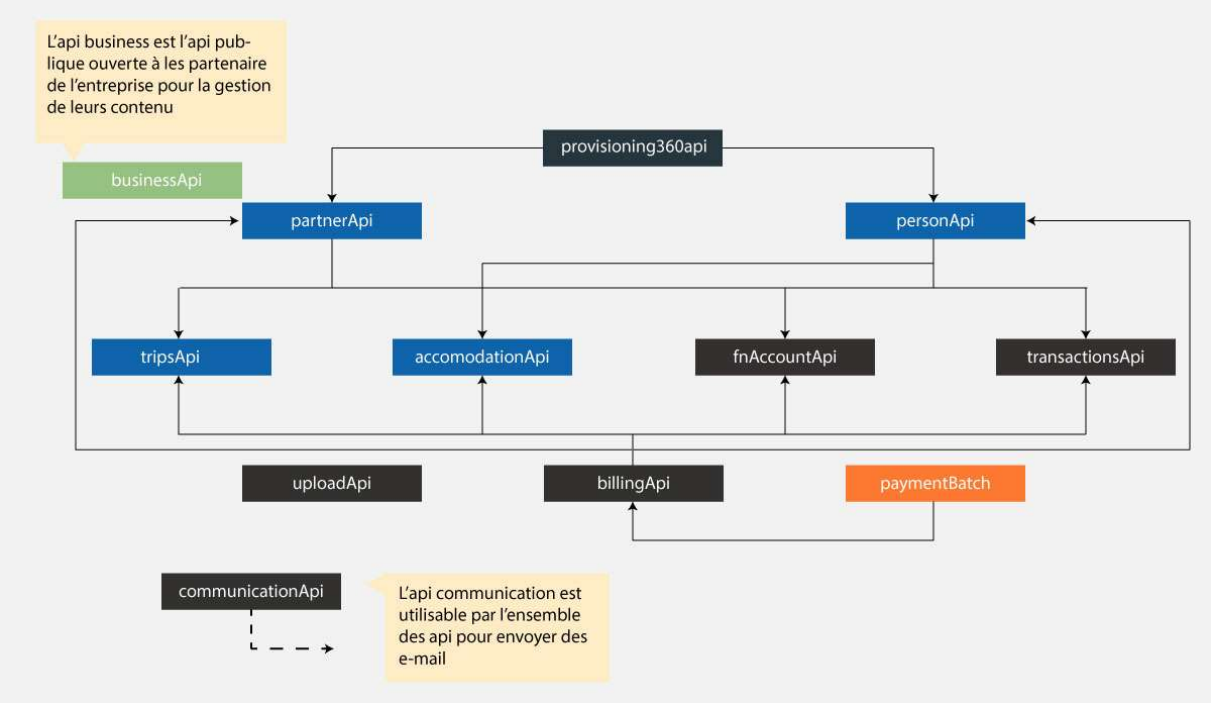


Figure 6: L'architecture micro service du projet Atlas

3.2. Description des APIs

Le système est composé de plusieurs APIs, chacune a son périmètre fonctionnel.

3.2.1. Provisioning360Api

ProvisioningApi est l'API responsable de la gestion des comptes des utilisateurs, elle s'occupe de leurs création, modification et la suppression.

3.2.2. PartnerApi

PersonApi est l'API responsable de la gestion et la persistance des comptes des partenaires enregistrés dans la plate-forme. Les partenaires peuvent être des hôtels, des auberges, en général sont des entreprises spécialisées qui offrent leurs services dans la plate-forme.

3.2.3. PersonApi

PersonApi est l'API responsable de la persistance des comptes des utilisateurs enregistrés dans la plate-forme.

3.2.4. AccomodationApi

AccomodationApi est responsable de la gestion des biens. Il permet aux utilisateurs de publier leurs biens dans la plate-forme pour qu'ils puissent être réservés par d'autre utilisateurs.

Un micro-service du système lorsqu'il veut envoyer un email, il consomme le service exposé par l'API *communicationApi.*

3.3. CommunicationApi

L'API que j'ai développée est *communicationApi.* Elle est utilisée par les autres micro-services du système. Le rôle de cette API est d'envoyer des e-mails comme les e-mails de confirmation de la création du compte ou les

e-mails de la récupération des mots de passe, elle permet aussi l'envoie des messages, et gérer le formulaire de *contact* dans la partie interface.

3.3.1. Les acteurs

Les acteurs qui utilisent l'API *communicationApi* sont des acteurs système qui bénéficient de la fonctionnalité fournie par cette API.

L'identification des acteurs est détaillée dans le chapitre 2.

3.3.2. Les fonctionnalités

L'API *communicationApi* fournit les fonctionnalités suivantes :

- Permettre au client applicatif d'envoyer des e-mails.
- Permettre l'envoie des messages entre les utilisateurs de la plate-forme.
- Gérer les données reçues du formulaire de *contact* dans l'interface de la plate-forme.

3.4. Qu'est-ce qu'une API

L'API est l'abréviation du mot anglais *Application Programming Interface*, c'est une brique applicative responsable sur un périmètre fonctionnel bien précis.

Les APIs permettent d'établir une communication entre deux ou plusieurs systèmes. Alors, un système peut consommer le service fourni par un autre système, comme ils peuvent échanger les données, d'où une API permet l'interopérabilité entre les différents systèmes.

Les APIs REST, - développées dans ce projet – sont les APIs qui respectent l'architecture REST.

L'architecture REST est un ensemble de convention et de bonne pratique à respecter, il ne s'agit pas d'une technologie. L'architecture REST s'appuie principalement sur le protocole http.

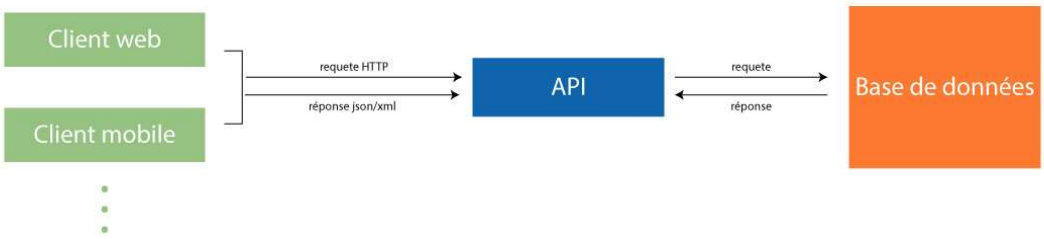


Figure 7: API REST

Chapitre 2 : Analyse et conception

1. Outils de conception

1.1. Langage de modélisation UML

Figure 8: Logo UML

UML, c'est une abréviation de *unified modeling language*, c'est un langage graphique de modélisation comportant plusieurs diagrammes et symboles. Les symboles sont unifiés et portent une sémantique bien connue.

UML est devenu une norme de l'OMG –Object Management Group - en 1997. Dans le génie logiciel, UML est utilisé pour l'analyse et la conception des logiciels et des systèmes.

2. Spécifications des besoins

L'API *communicationApi* doit répondre à deux types de besoins :

- Besoins fonctionnels
- Besoins techniques

2.1. Besoins fonctionnels

L'API doit répondre aux besoins fonctionnels suivants :

- L'envoi d'un e-mail.
- L'envoi des messages entre les utilisateurs de la plate-forme.

- Gérer le formulaire de *contact* dans l'interface.
- Chercher un message spécifié.
- Chercher un contact spécifié.
- Informer un client du nombre des entités stockées dans la base de données et le nombre de pages nécessaires pour les afficher dans l'interface.

2.2. Besoins techniques

Le système doit répondre aux besoins techniques suivants :

- Utilisation d'un mécanisme de communication entre les micro-services du système.
- Le mécanisme de communication utilisé doit suivre une architecture REST.

3. Analyse fonctionnelle

3.1. Identification des acteurs

Un acteur est une entité externe qui communique et interagit directement avec le système. Il peut être un acteur humain, un autre système qui offre un service ou un autre système qui utilise le service.

Les acteurs de l'API *communicationApi* sont des acteurs applicatifs, autrement dit, ils sont les autres APIs du système qui utilisent le service fourni par *communicationApi*.

Les acteurs de communicationApi sont :

- provisioning360Api. API responsable de la création des comptes.
- billingApi : API responsable des payements.
- L'IHM : L'interface graphique de la plate-forme.

3.1.1. Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation permet de représenter les fonctionnalités du système. En plus, il modélise l'interaction des acteurs avec le système.

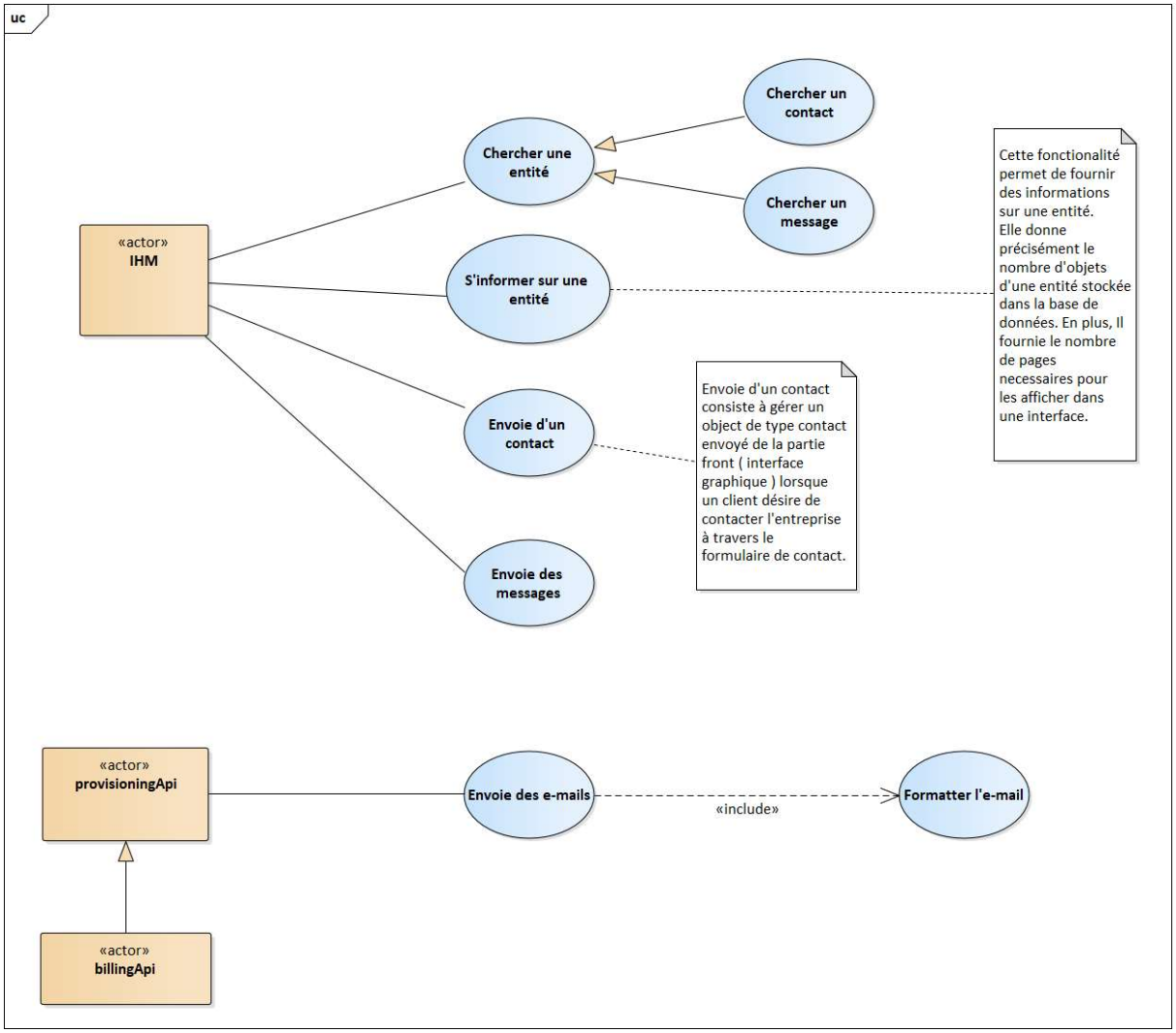


Figure 9: Diagramme de cas d'utilisation - communicationApi

S'informer sur une entité est la fonctionnalité de l'API qui permet de renvoyer à la partie front (interface graphique) le nombre d'objets (message, contact) stockés dans la base de données ainsi que le nombre de pages nécessaires pour les afficher.

La généralisation entre l'acteur système **billingApi** et **provisioning360Api** veut dire que les deux acteurs consomment le même service exposé par l'API.

3.1.2. Description textuelle du cas d'utilisation : Envoie des e-mails

Nom du cas d'utilisation	Envoie des e-mails
Les acteurs	provision360Api, billingApi
Description	Cas d'utilisation qui permet de stocker les données nécessaires à l'envoie de l'e-mail, ensuite il l'envoie.
Scénario nominal	▪ Le client fait appel à l'Api en envoyant une requête http de méthode POST à travers l'endpoint : http://domain.com/communication/mail ▪ Le système valide l'objet reçu dans le corps de la requête http. ▪ Le système sauvegarde l'objet dans la base de données ▪ Le système génère une réponse avec le code 201 CREATED ▪ Le système envoie la réponse au client
Scénario alternatif L'objet n'est pas valide	▪ Le client fait appel à l'Api en envoyant une requête http de méthode POST à travers l'endpoint : http://domain.com/communication/mail ▪ Le système valide l'objet reçu dans le corps de requête http.

	▪ Le système génère une réponse avec le code 400 BAD REQUEST ▪ Le système envoi la réponse au client

Tableau 1: Description textuelle du cas d'utilisation : Envoie des e-mails

3.2. Diagramme de classe

Le diagramme de classe exprime la structure statique du système en terme de classes et de relations entre ces classes.

Le diagramme de classe du système est le suivant :

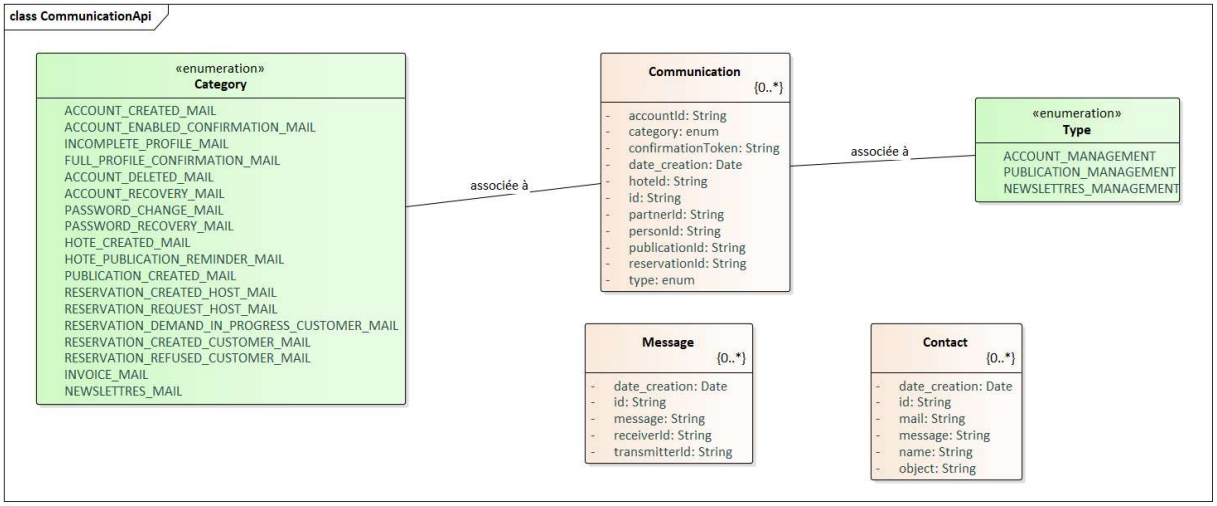


Figure 10: Diagramme de classes – communicationApi

La classe *communication* représente les objets de type communication, elle contient les attributs nécessaires pour l'envoi d'un email.

La classe *message* représente les objets manipulés lors de l'envoi d'un message dans la plate-forme entre deux utilisateurs, en fait parmi les attributs de cette classe on a *receiverId* pour identifier le récepteur du message, *transmitterId* pour identifier l'émetteur du message en plus d'une chaine de caractère qui représente le texte du message à envoyer.

La classe *contact* représente les objets reçus de la partie front lorsqu'un client remplit le formulaire de contact.

3.3. Diagramme de séquence

Le diagramme de séquence illustre les interactions entre les acteurs et le système dans le temps. Les acteurs envoient des messages au système, ce message peut provoquer le déclanchement d'une réaction chez le récepteur, tel que l'exécution d'une opération, la création ou la destruction d'une instance.

Les messages peuvent être synchrones. L'émetteur de ce message reste bloqué jusqu'à la réponse du récepteur. Comme ils peuvent être asynchrones, dans ce cas l'émetteur ne reste pas bloqué durant l'intervalle de temps où le récepteur traite le message.

3.3.1. L'endpoint sendMail

Le diagramme de séquence dans la page suivante représente le comportement de système lorsque le client consomme la ressource *communication/mail*, c'est la ressource consommée lorsqu'un client applicatif désire envoyer un e-mail. (Le diagramme est dans la page suivante pour raison de lisibilité)

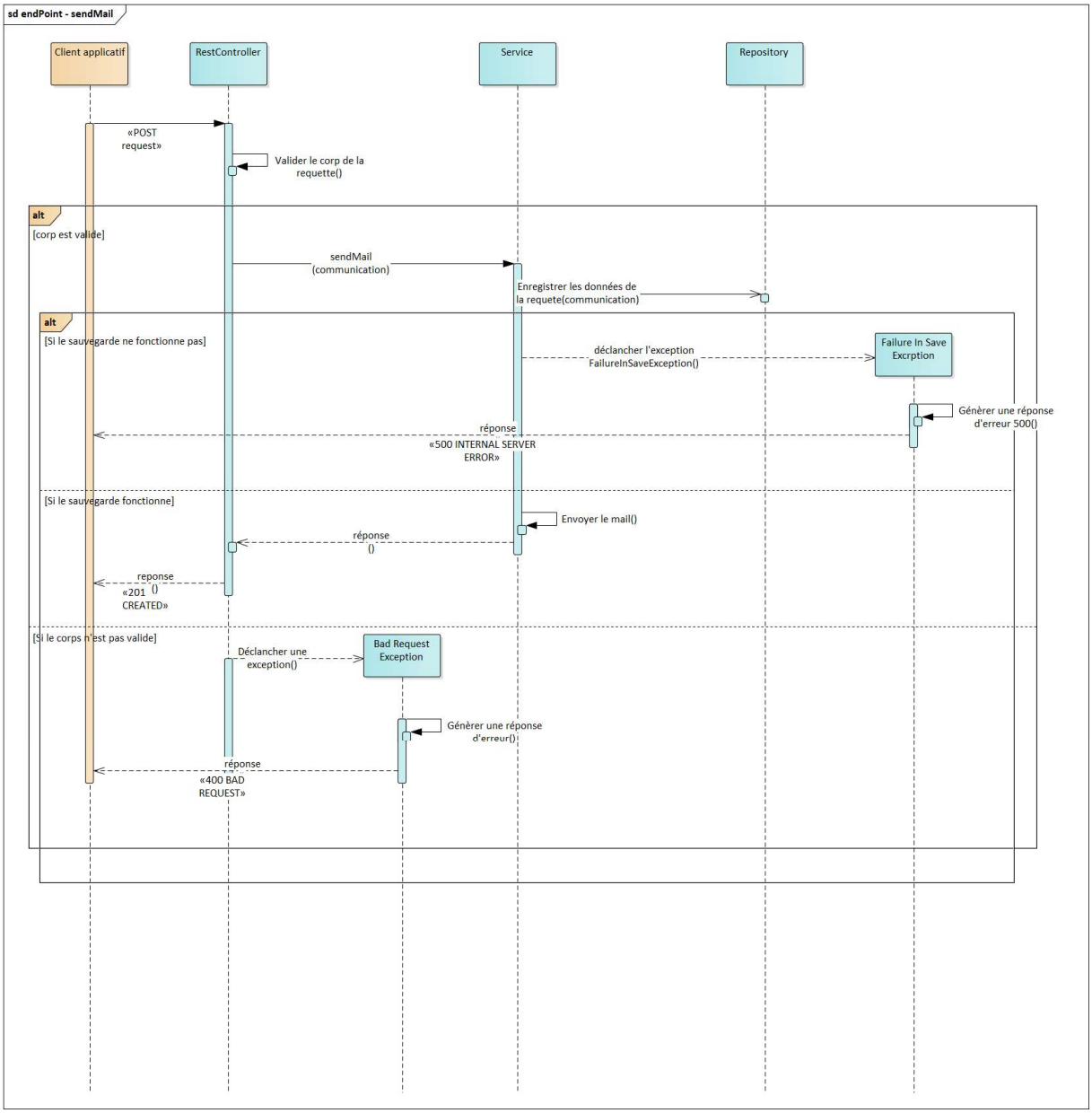


Figure 11: Diagramme de séquence - endpoint sendMail

3.3.2. L'endpoint sendContact

Lorsqu'un utilisateur – humain – remplies le formulaire de contact dans la plateforme, l'IHM envoi les données remplies à l'API *communicationApi*.

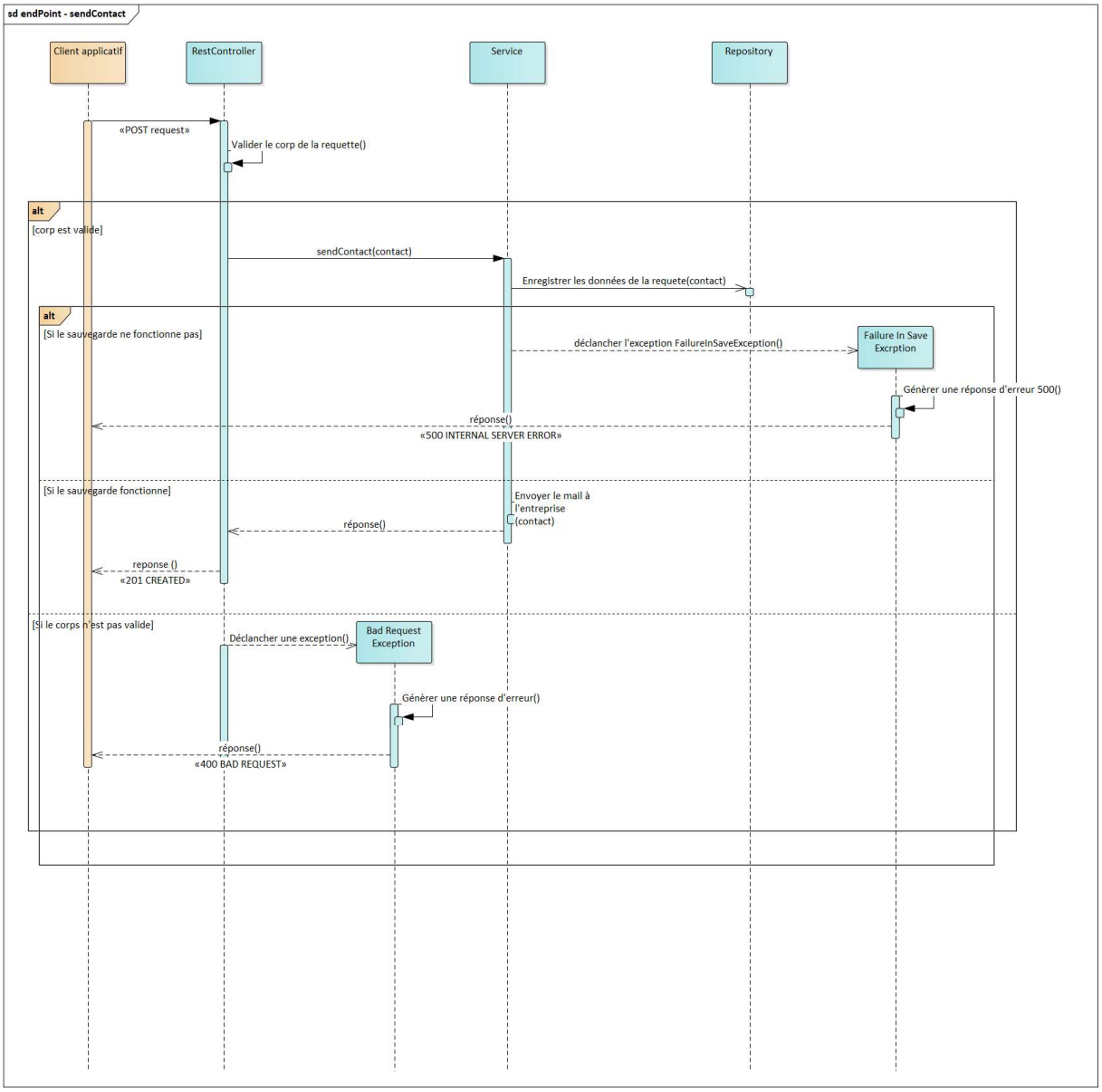


Figure 12: Diagramme de séquence - sendContact

3.3.3. L'endpoint getContact

Cet endpoint demande au client un identifiant et répond avec un objet de type contact qui correspond à cet objet. Si l'identifiant ne correspond à aucun objet stocké dans la base de données, l'API répond avec le code erreur 404 Not Found.

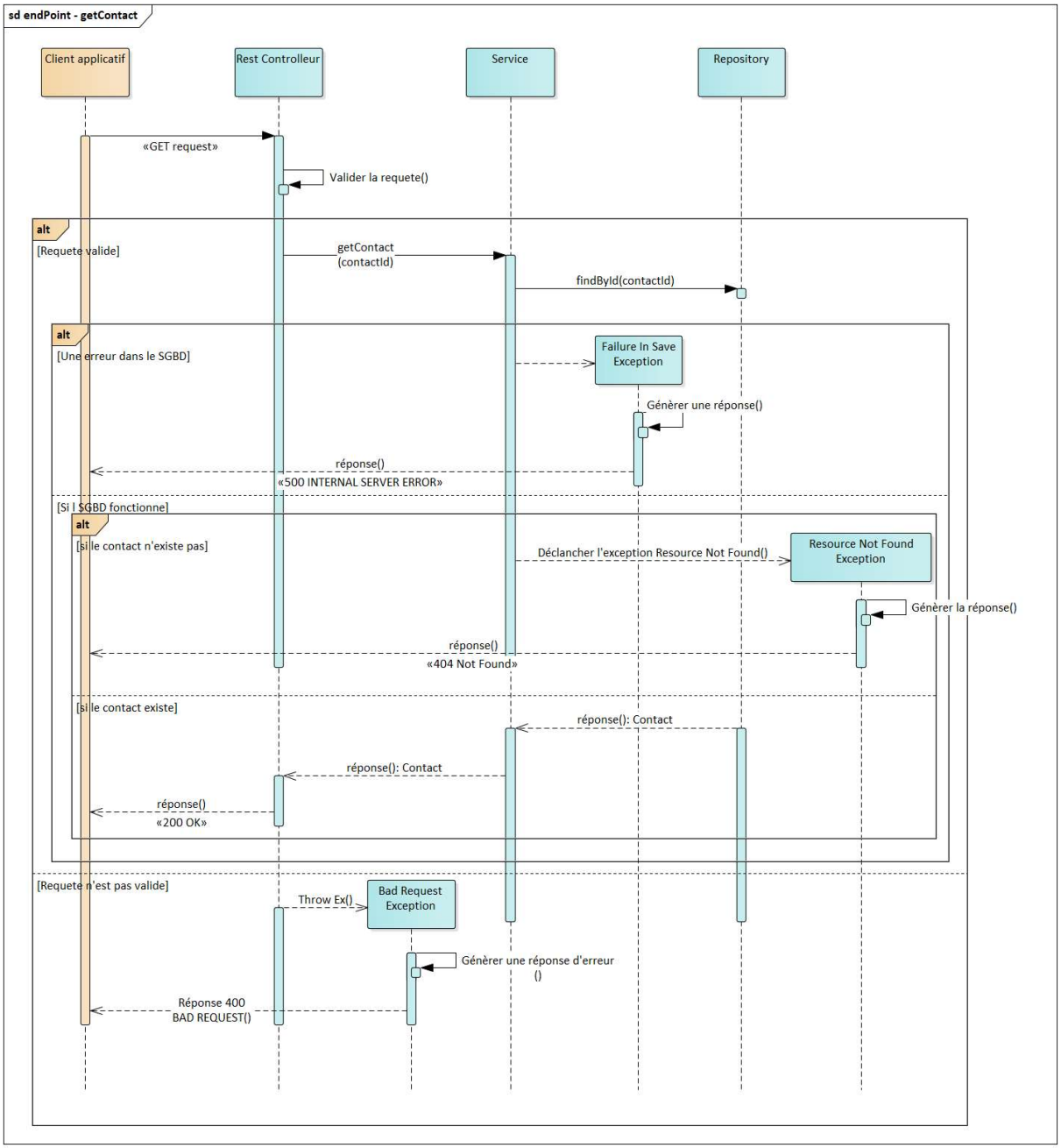


Figure 13: Diagramme de séquence getContact

3.3.4. L'endpoint listContact

ListContact est l'endpoint qui fournit à la partie front le nombre de pages nécessaires pour afficher les objets de type contact ainsi que le nombre total d'objets de type contact qui sont stockés dans la base de données

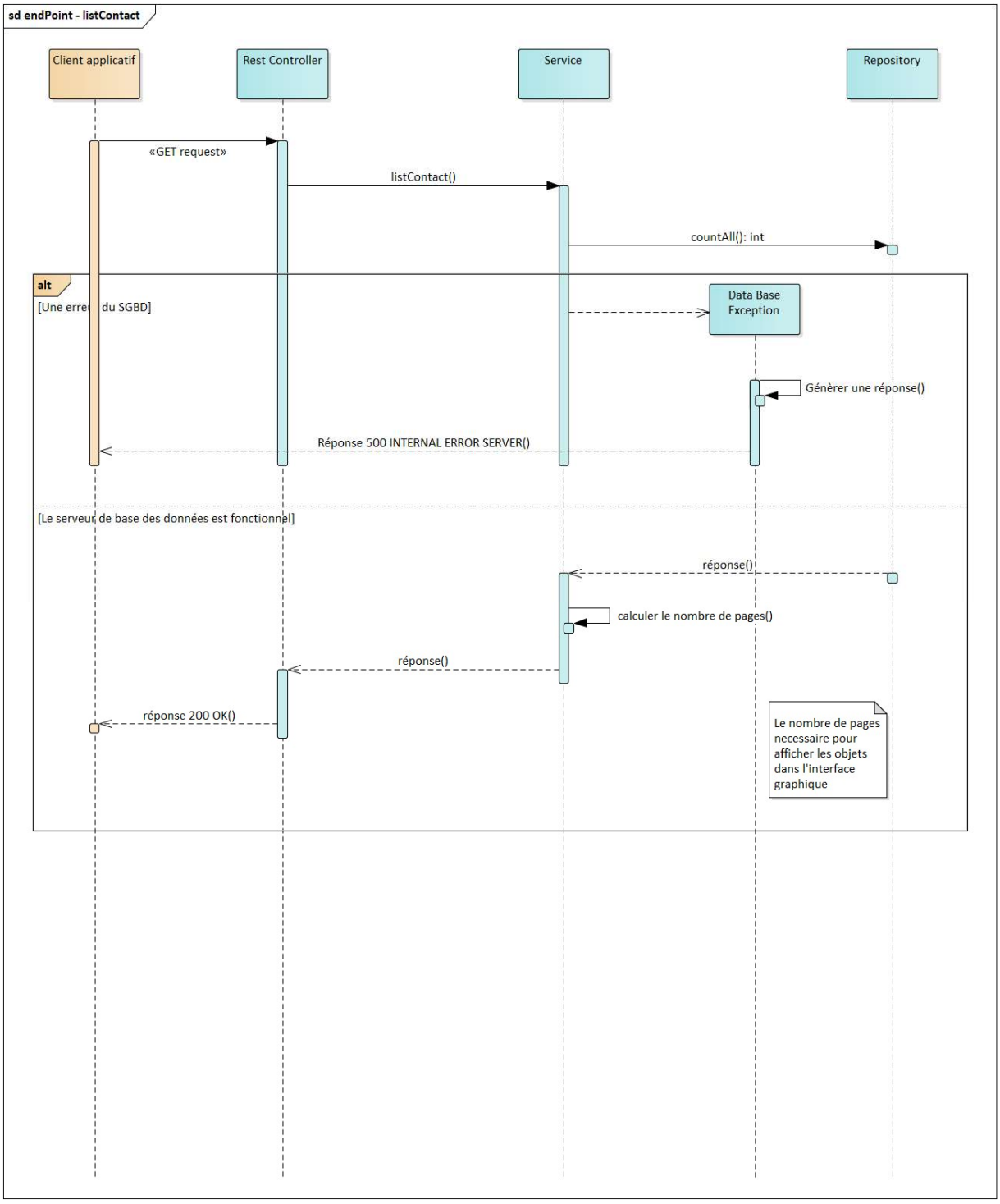


Figure 14: Diagramme de séquence : listContact

3.3.5. L’endpoint sendMessage

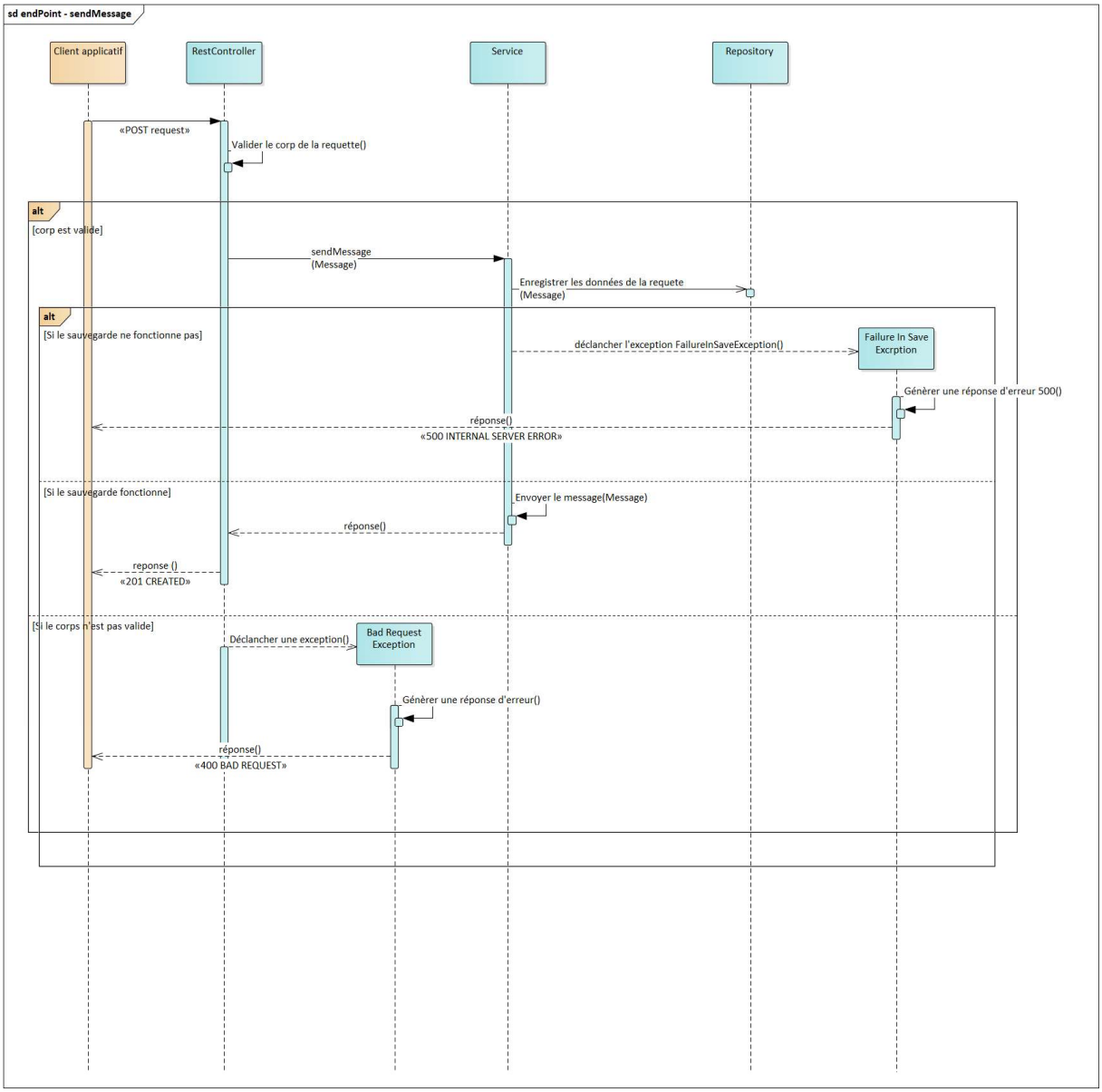
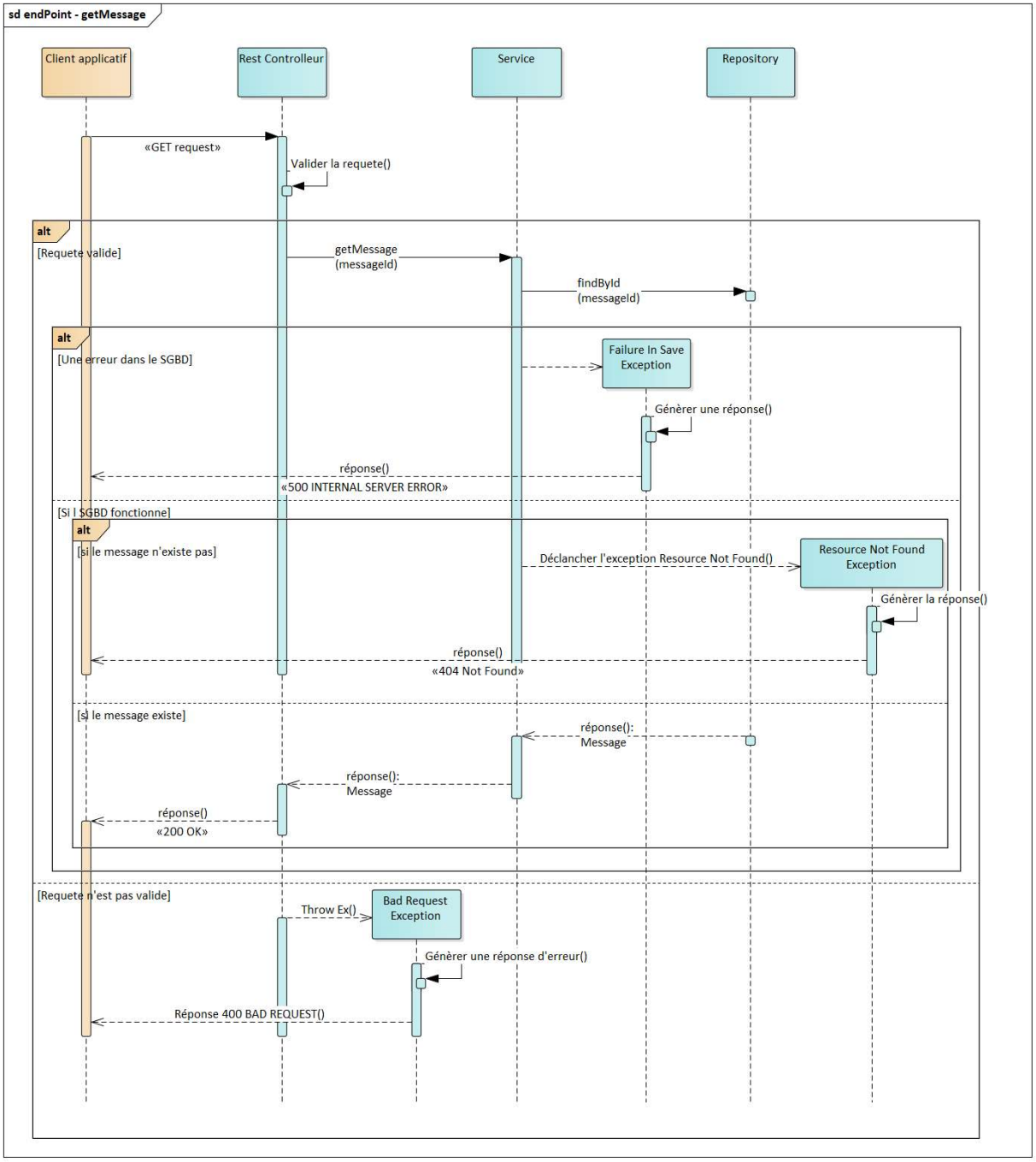
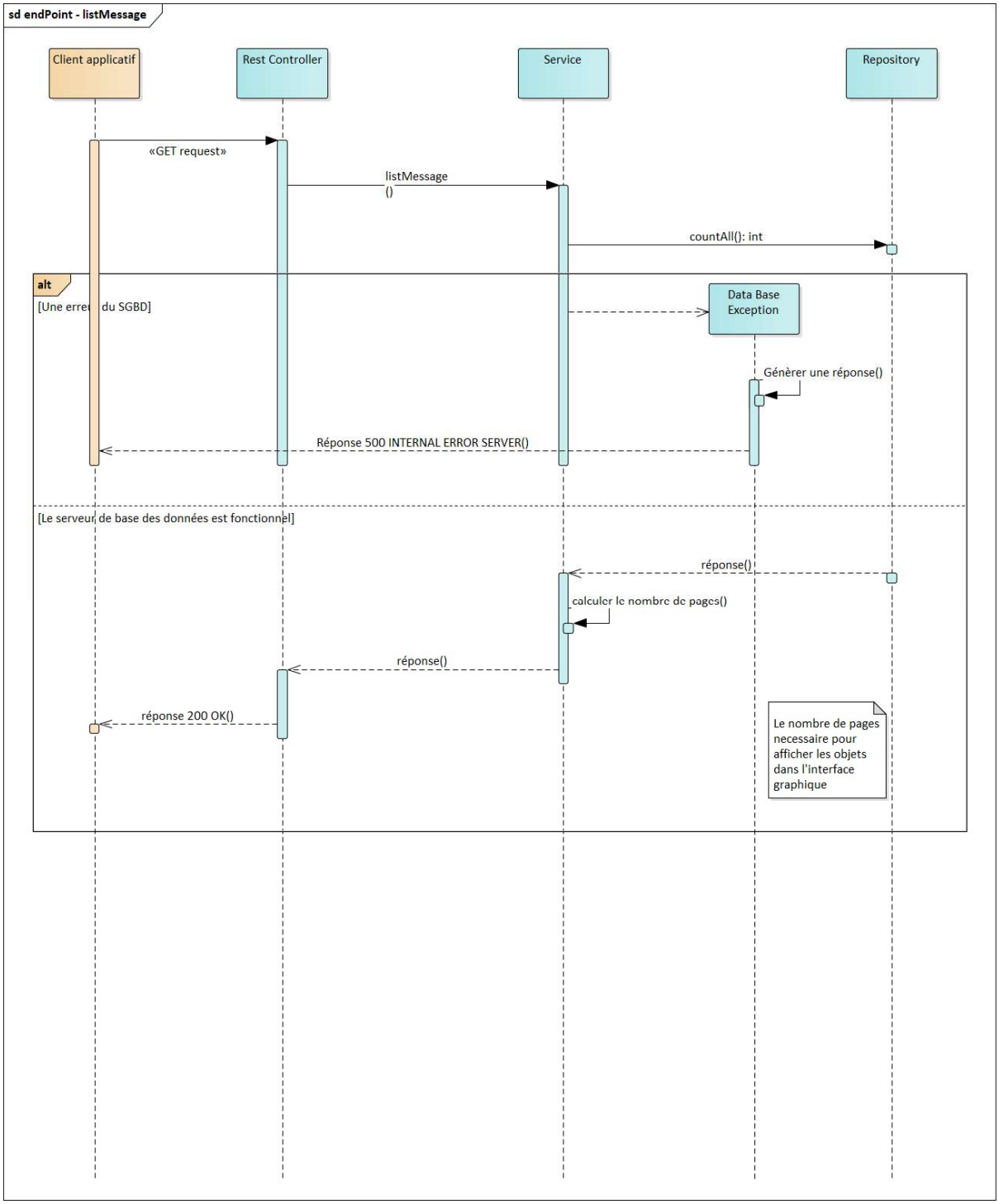


Figure 15: Diagramme de séquence getMessage

3.3.6. L’endpoint getMessage



3.3.7. L’endpoint listMessage



Chapitre 3 : Réalisation

1. Outils de développement

Pour implémenter les fonctionnalités désirées par l'API, j'ai utilisé un ensemble d'outils, certains utilisés pour collaborer avec les autres développeurs de l'entreprise, d'autres pour traduire ces fonctionnalités en code.

Mes contributions doivent être revues avant de les fusionner avec la branche principale du projet. Si une contribution provoque un bogue, l'équipe de développement doit avoir la possibilité de revenir vers la dernière version correcte.

Pour répondre à ce besoin, l'entreprise utilise **GIT.**

Figure 16: Logo git

GIT, est un système de contrôle de version distribué développé en 2005 par Linus Torvalds, l'ingénieur derrière le noyau de linux. Un système de contrôle de version permet de suivre l'historique des fichiers d'un projet et les changements effectués sur ces fichiers par rapport au temps. Le fait de suivre l'historique du projet permet à l'équipe de développement de revenir à la version qui ne contient pas d'erreurs.

Au sein de l'entreprise, on travaille avec un fournisseur des serveurs GIT, en particulier GITLAB.

Pour répondre aux exigences de l'API, j'ai utilisé l'écosystème JAVA, précisément le Framework spring.

Spring est un Framework modulable, il fournit des services pour développer, tester et valider une application.

Spring Framework se compose de plusieurs modules, ils peuvent être utilisés selon les besoins du développeur, c'est-à-dire, le développeur n'a pas besoins de charger tous les modules pour utiliser seulement quelques modules.

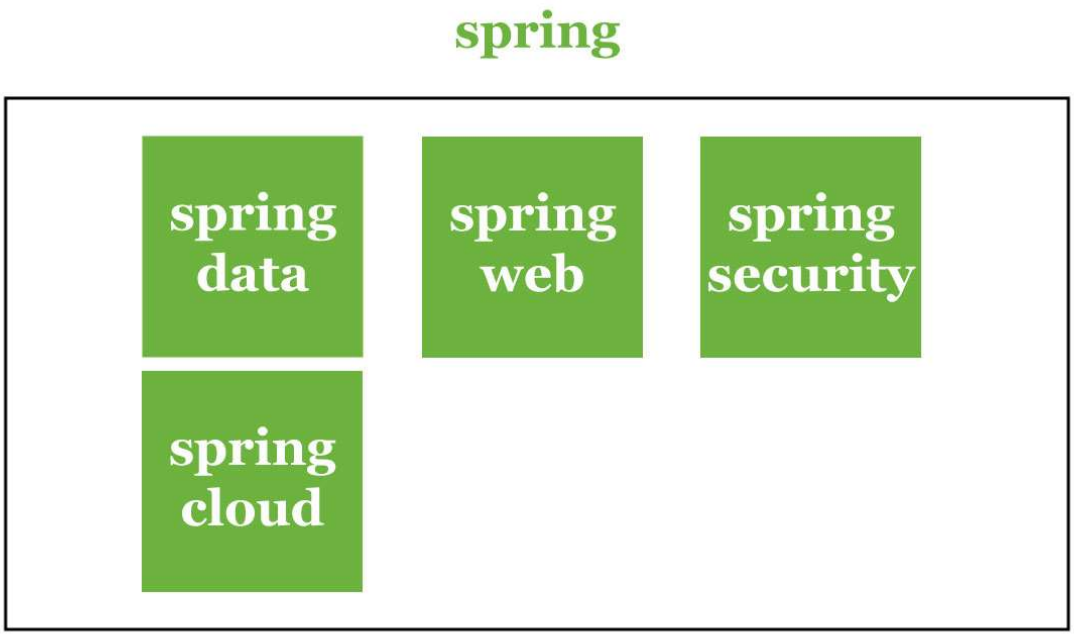


Figure 17: Le Framework spring

Spring boot, est le point d'entrée vers les autres composants. C'est lui qui charge les bonnes versions et donne une configuration par défaut pour démarrer le projet rapidement, cette configuration peut être personnalisée.

Spring data est un module qui ajoute une couche d'abstraction en permettant le développeur de manipuler les données sous forme d'objet JAVA.

Spring web est le module qui implémente l'architecture REST, il s'occupe de sérialiser les objets JAVA en format JSON, ou l'inverse.

Gérer ces modules manuellement est une tâche complexe, pour cela j'ai utilisé un gestionnaire de dépendance, l'entreprise utilise **MAVEN.**

Figure 18: Logo maven

MAVEN est un outil de gestion des projets, notamment réalisés avec le langage de programmation JAVA. Il permet de créer les fichiers JAR de nos projets, les tester et gérer leurs dépendances.

Pour stocker les données d'une manière persistante. J'ai utilisé le système de gestion de bases de données **PostgreSql.**

Figure 19: Logo Postgresql

PostgreSql est un système de gestion de bases de données relationnels objets, il est le successeur de INGRES, le SGBD implémenté en 1975-1977

à l'université de Californie, d'où vient le nom POSTGRESQL. Il est créé pour supporter des types de données complexe, défini par le développeur. PostgreSQL est multiplateforme, il peut être installé et exécuté sur les différents systèmes d'exploitations, notamment Windows, Linux, Mac os.

Le code qui met en place l'API que j'ai développée est composé de plusieurs méthodes qui appartiennent à des différentes classes déjà prédéfinies par d'autres développeurs ou organismes. Elles sont nombreuses, il faut avoir un outil pour lister les méthodes disponibles dans chaque classe. En outre, le code doit être comité et poussé à une repository distante, pour cela il est nécessaire d'utiliser GIT en parallèle avec la phase de développement. De plus, j'ai besoin de gérer les dépendances du projet, c'est le rôle de Maven.

Pour répondre à ces exigences, j'ai travaillé avec un environnement de développement intégré qui regroupe tous ces outils, en particulier, j'ai codé sur **INTELLIJ IDEA**.

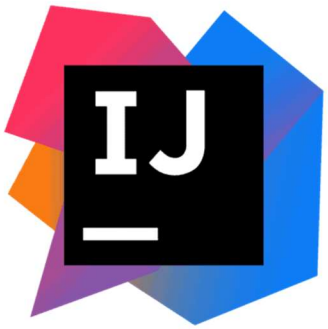


Figure 20: Logo de l'IDE Intellij

INTELLIJ IDEA est un IDE qui est disponible en plusieurs éditions, j'ai choisis de travailler avec l'édition community pour deux raisons :

- C’est une édition open-source et gratuite.
- C’est l’édition utilisé au sein de l’entreprise.

L’API réalisée répond essentiellement à des requêtes http. Pour tester les réponses de l’API, j’ai utilisé le client **Postman.**

Figure 21: Logo Postman

Postman est un client REST qui permet de tester des APIs, en envoyant des requêtes http au serveur et en présentant la réponse de l’API d’une manière lisible.

Comment utiliser POSTMAN ?

Postman nous met à disposition une application web et une application desktop. Les deux ont une interface graphique dans laquelle on peut préciser :

- La méthode http
- L’identifiant de la ressource (URI)

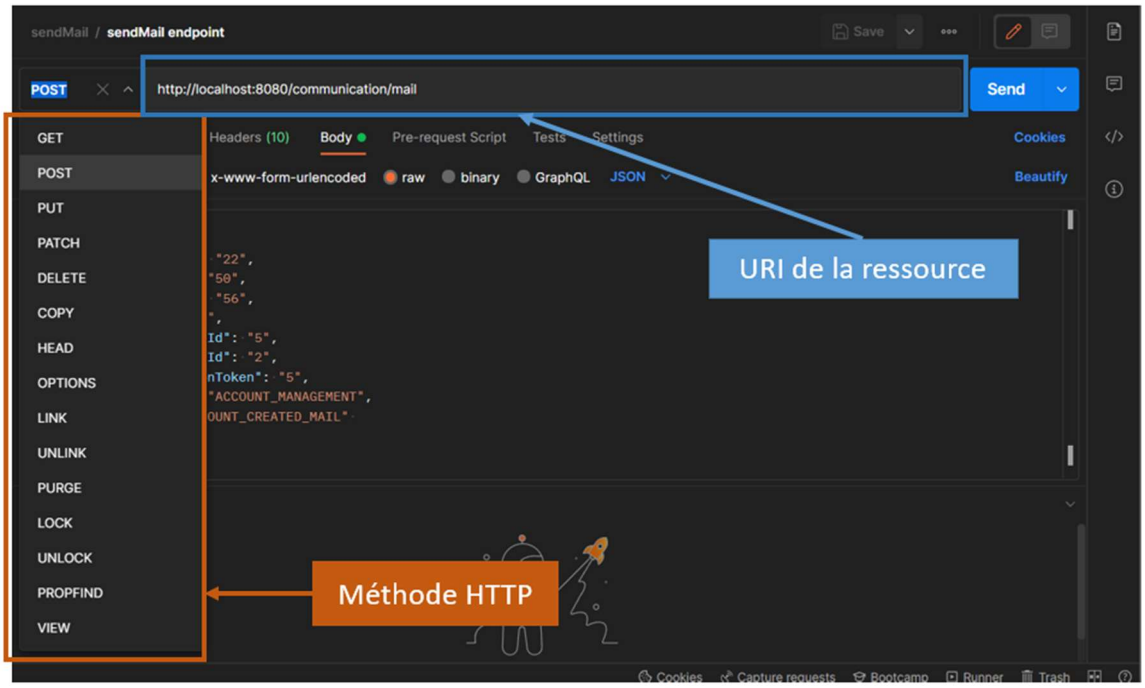


Figure 22: Interface de POSTMAN – méthode http, URI

- S'il s'agit d'une requête http PUT ou POST, on peut spécifier sous quel format on veut envoyer les données à l'API (JSON, Text, HTML, XML), en plus d'une TextArea où on met les données.

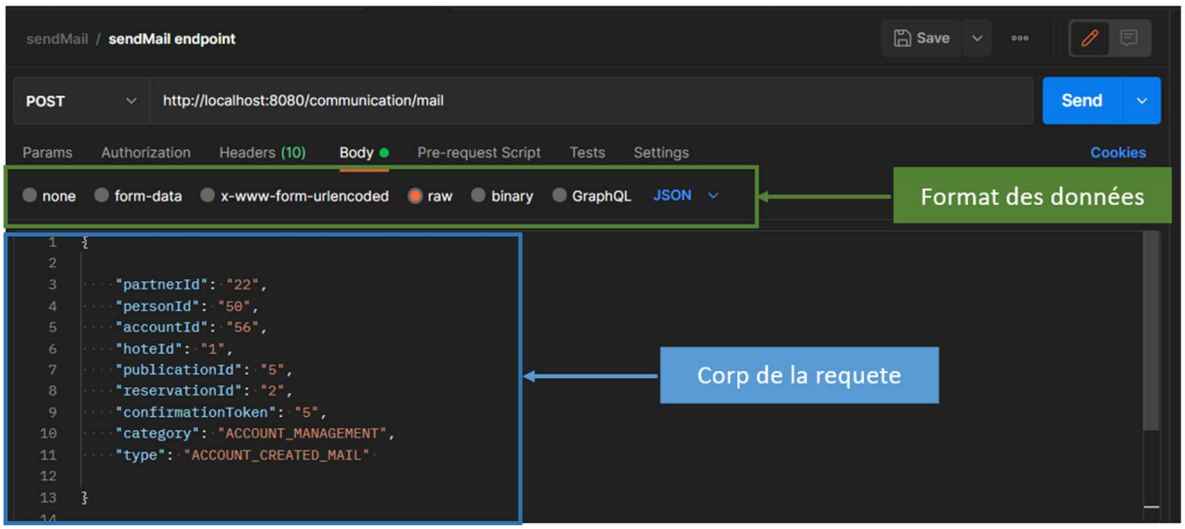


Figure 23: Interface de POSTMAN - Format des données, Corps de la requête

Le clique sur le bouton **send** envoie une requête http au serveur.

POSTMAN présente la réponse de l'API. Il fournit l'objet que l'API retourne au client, en plus du code http correspondant.

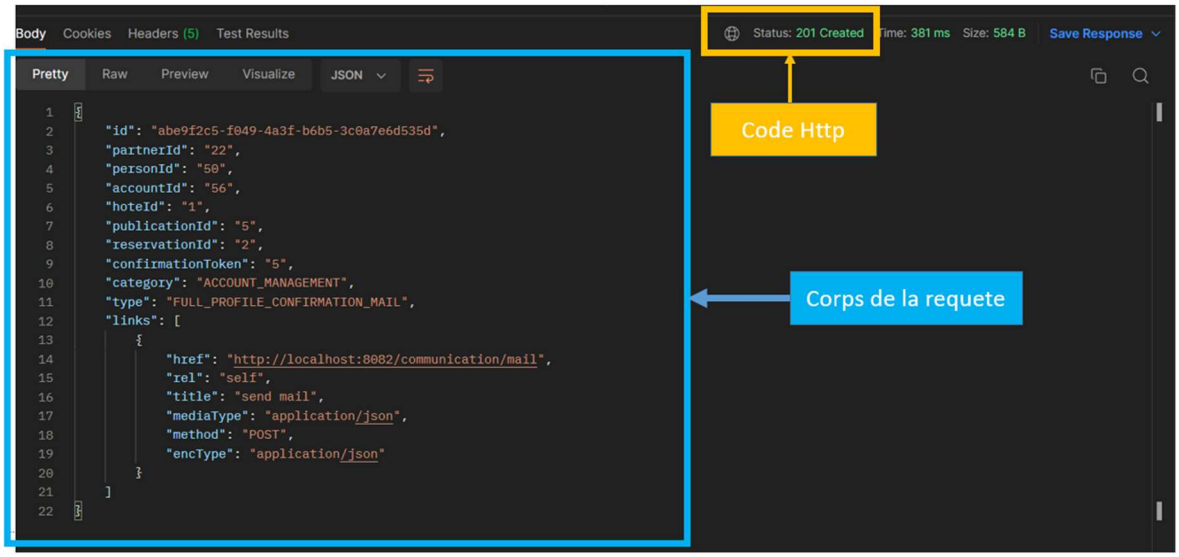


Figure 24: Interface POSTMAN - représentation réponse API

La figure ci-dessus représente le corps de la requête http reçu par le serveur. On peut visualiser aussi l'en-tête de la requête dans l'onglet **Header.**

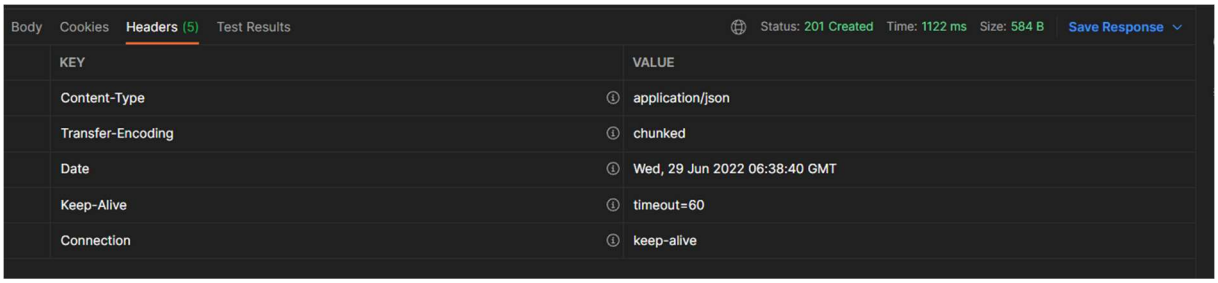


Figure 25: La réponse de l'api dans POSTMAN - visualiser l'en-tête

Quelles sont les alternatives de POSTMAN ?

- **Curl**

Curl est l'abréviation de client URL, c'est un utilitaire en ligne de commande, il nous permet de demander ou envoyer des ressources d'un serveur.

La figure ci-dessous présente un exemple d'une requête http de méthode POST. On reçoit la réponse toujours dans la ligne de commande.

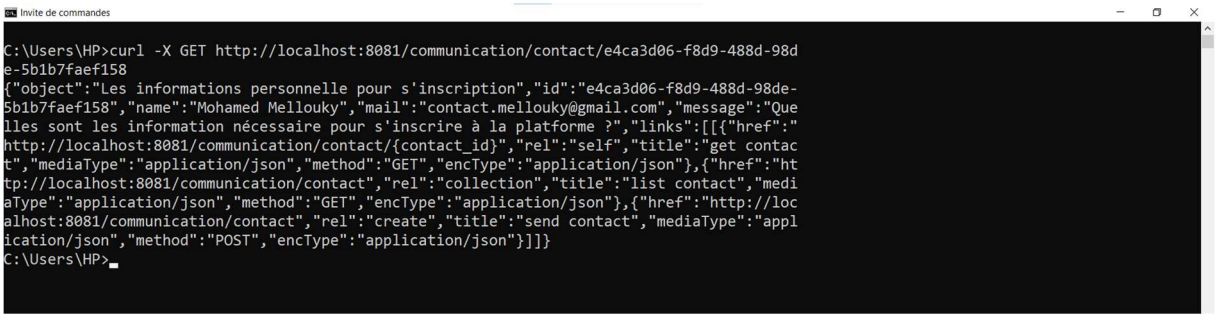


Figure 26: Curl dans la ligne de commande

2. Présentation de l'API

Tous les endpoints développés doivent respecter un contrat d'interface. En fait, ce contrat d'interface fournie les deux informations suivantes :

- Dans le cas des requête POST et PUT, le format d'objet que l'API attend du client.
- Comment l'API doit répondre en cas de succès et d'échec.

2.1. L'endpoint sendMail

L'endpoint sendMail est consommé par les clients applicatifs par le URI : communication/mail, et la méthode http POST.

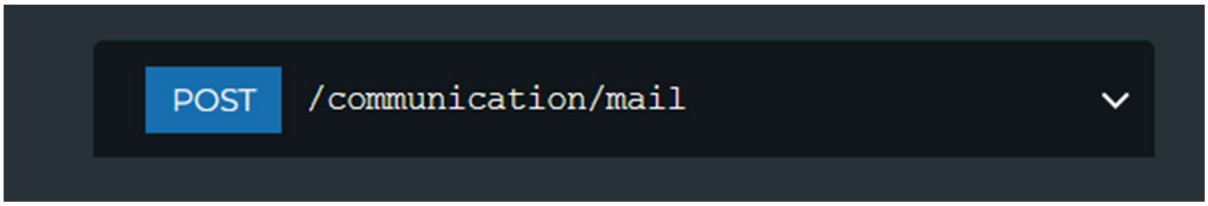


Figure 27: URI sendMail

Le corps de la requête contient un objet de type communication sous forme JSON.

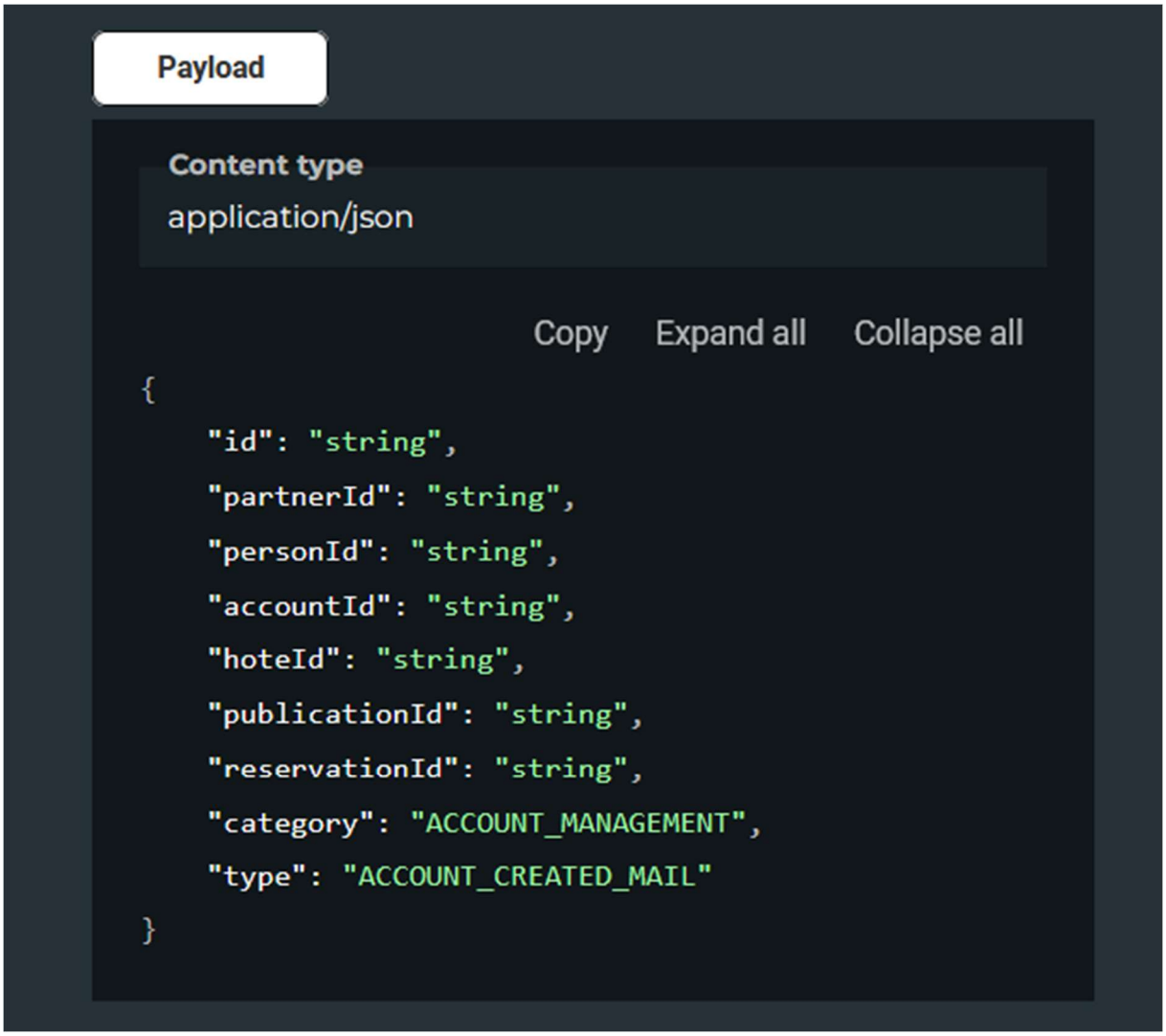


Figure 28: Objet Communication à envoyer dans la requête POST

L'API répond avec le status code 201 CREATED si aucune erreur n'apparait. Si le client ne respecte pas le contrat d'interface, l'API répond avec 400 Bad Request. Si une erreur apparait dans le serveur, l'API répond avec le code erreur 500 Internal Server Error.

2.1.1. Status code : 201 CREATED

Dans la figure ci-dessous, on précise l'identifiant de la ressource, la méthode http et l'objet de type communication qu'on désire envoyer.

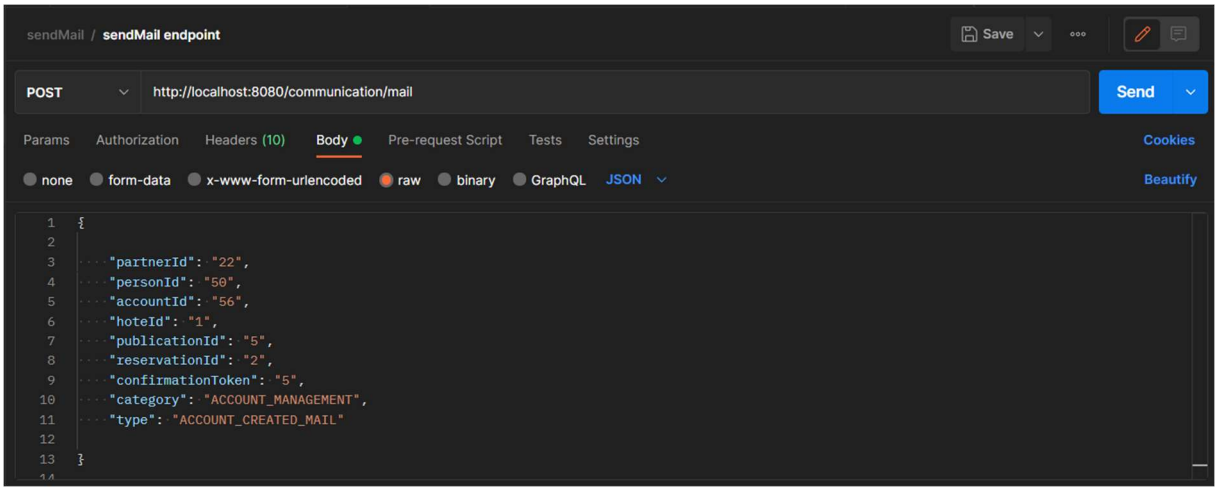


Figure 29: POST request – sendMail

La réponse de l'API doit respecter ce contrat :

Figure 30: Réponse de l'API avec le code 201 Created – sendMail

La réponse de l'API dans le cas de succès est la suivante :

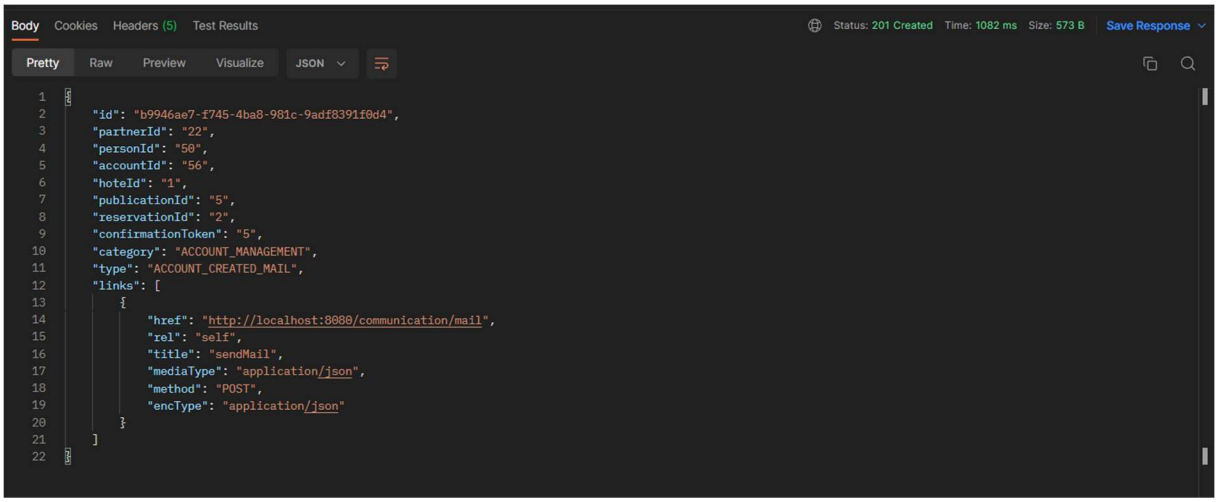


Figure 31: POST request réponse 201 CREATED

On note que *status code* est 201 CREATED, qui informe le client que sa requête a été traitée avec succès. En plus, la réponse de l'API respecte le contrat d'interface.

2.1.2. Status code : 400 BAD REQUEST

L'attribue **category** et **type** sont obligatoire, c'est-à-dire, lorsque le client ne précise pas la valeur d'une de ces deux attribues, l'API doit répondre avec un status code 400 BAD REQUEST.

D'abord, on ignore l'attribue **type,** la figure ci-dessous illustre qu'il est sans valeur.

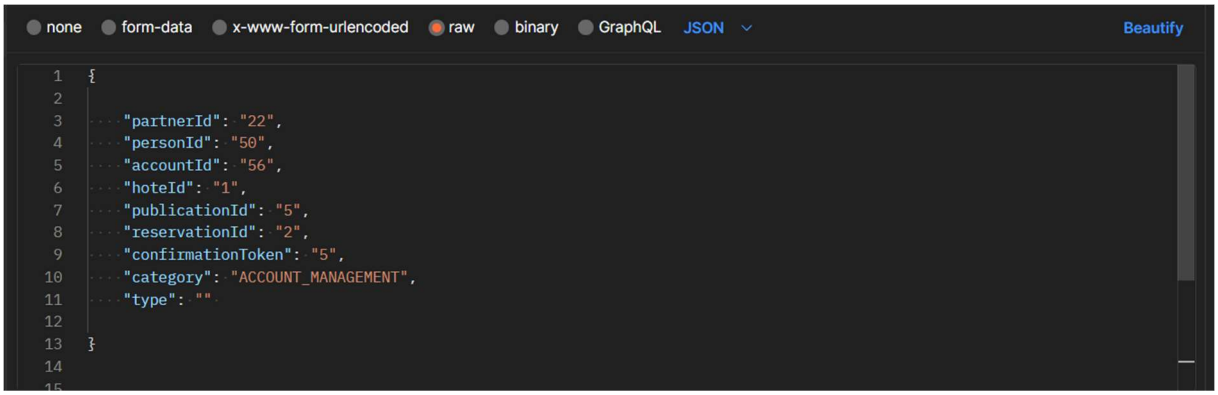


Figure 32: POST BAD REQUEST sendMail

La réponse de l'API doit répondre avec un objet de type erreur qui est sous la forme suivante (C'est l'objet renvoyé par l'API dans tous les cas d'erreur) :

Figure 33: Réponse de l'API avec le code 400 Bad Request – sendMail

La réponse de l'API est la suivante :

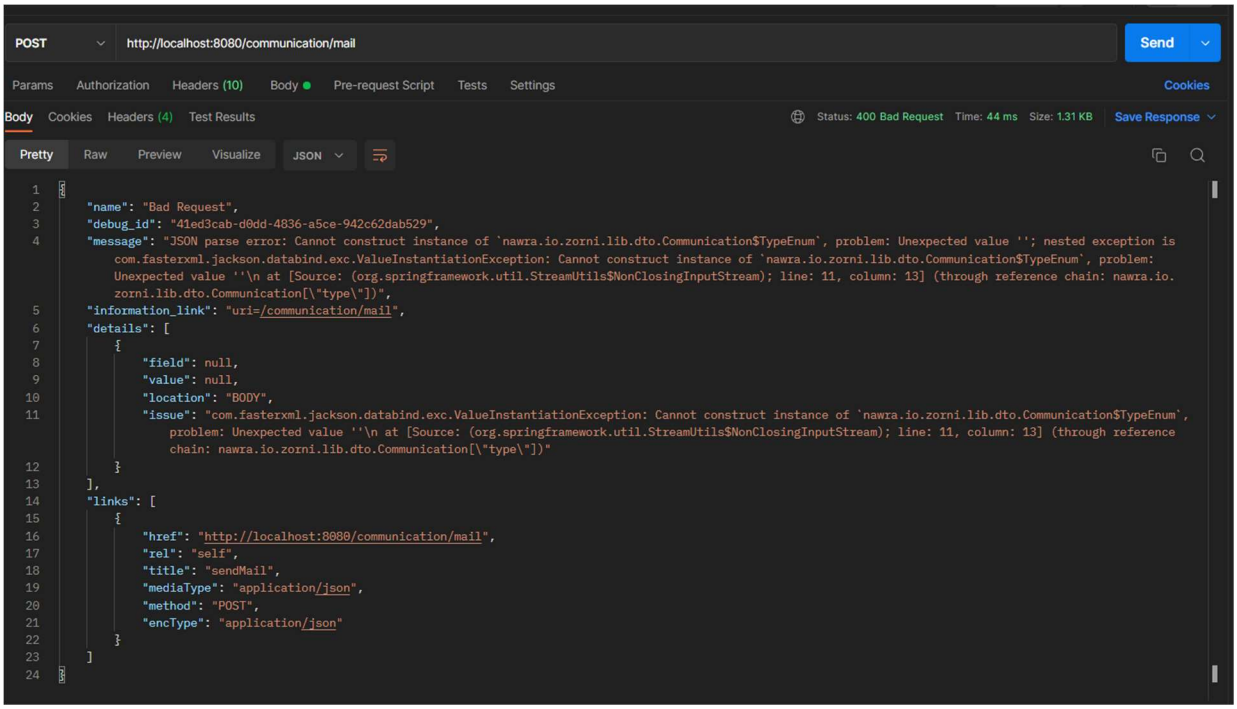


Figure 34: La réponse 400 BAD REQUEST

Le code erreur est 400 BAD REQUEST, en plus, la réponse est un objet de type erreur qui porte le nom de Bad Request et un message explicatif de l'erreur.

Maintenant, on ignore la valeur de **category.**

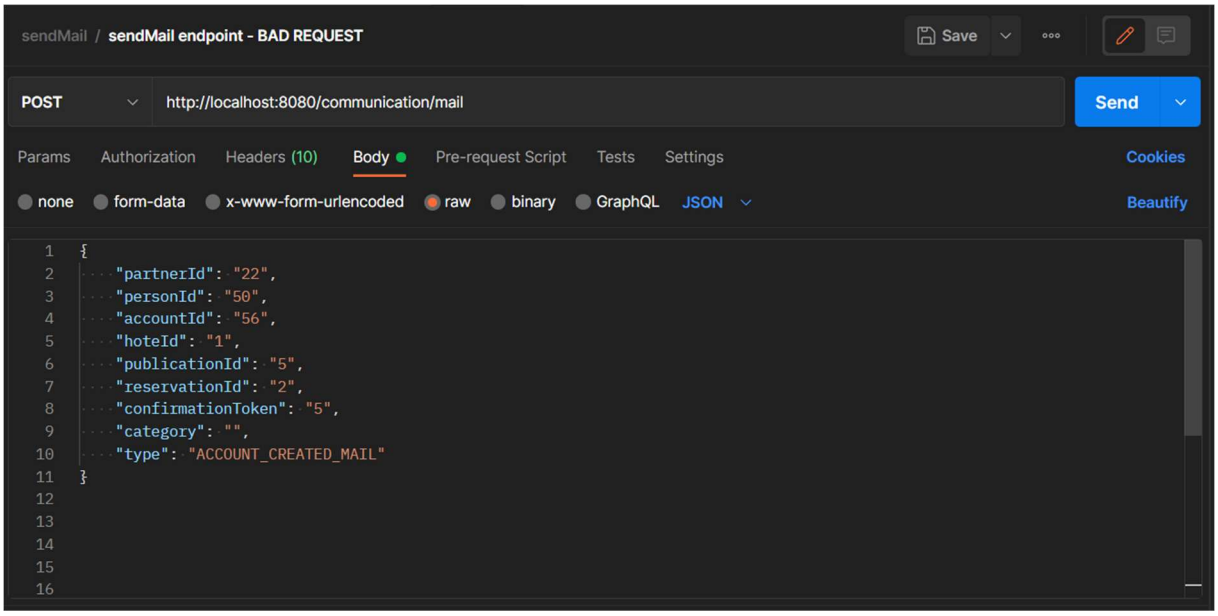


Figure 35: POST BAD REQUEST sendMail (deuxième cas)

La réponse de l'API est la suivante :

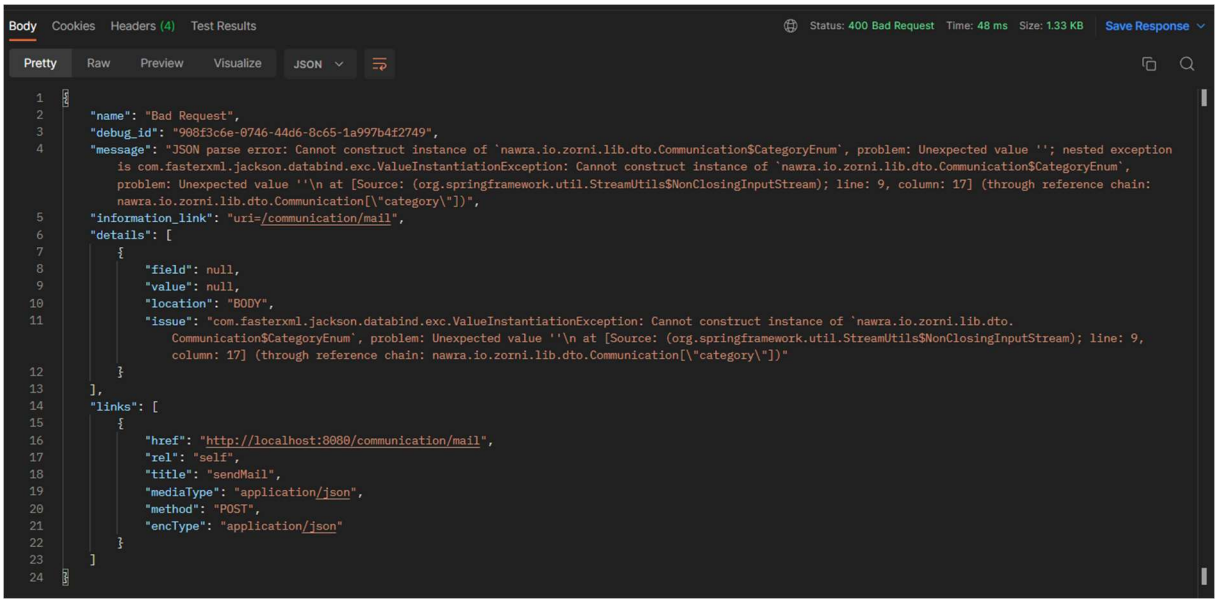


Figure 36: La réponse 400 BAD REQUEST (deuxième cas)

2.1.3. Status code : 500 INTERNAL SERVER ERROR

Si une erreur interne au serveur s'est produite, l'API répond avec le status code 500 INTERNAL SERVER ERROR. Elle indique au client applicatif que le serveur a rencontré une erreur, en plus, le champ renvoyé contient un attribue message qui donne plus de détails sur l'erreur.

Pour illustrer cela, je provoque un INTERNAL SERVER ERROR par l'envoie d'une requête http avec un objet nul.

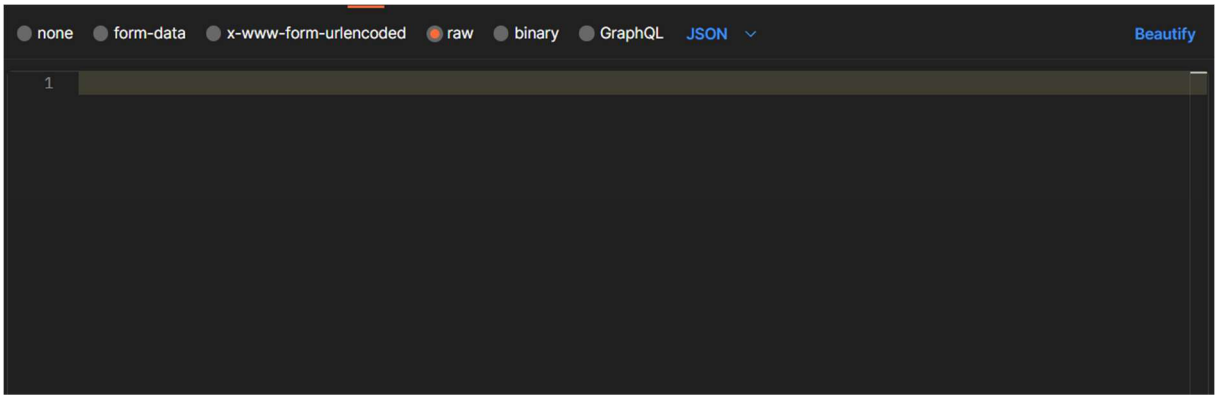


Figure 37: POST request avec un corps nul

La réponse de l'API est la suivante :

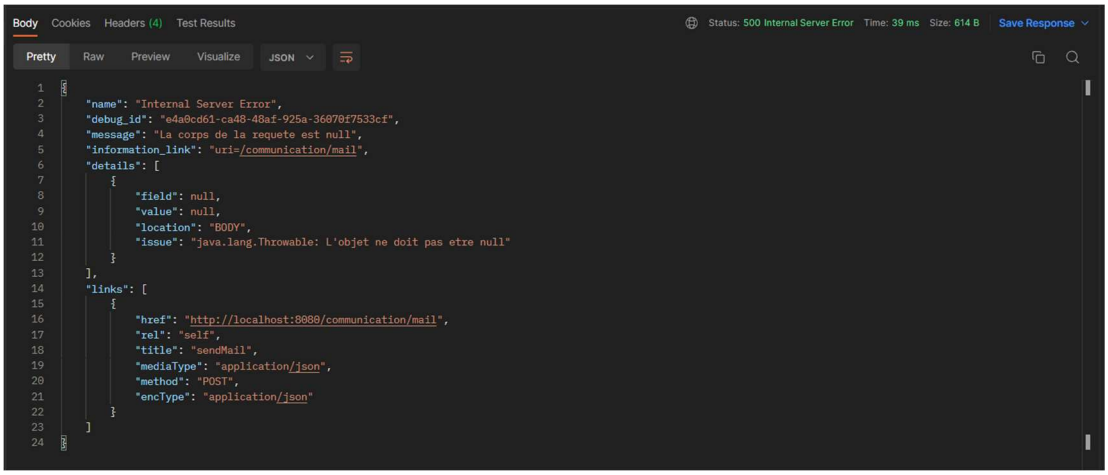


Figure 38: Réponse de l'api avec le code 500 INTERNAL SERVER ERROR

L'API répond avec 500 INTERNAL SERVER ERROR au lieu de 400 BAD REQUEST, parce que le serveur essaiera de manipuler un objet nul, ce qui va causer une exception de type nullPointerException.

On note aussi que l'API respecte le contrat d'interface dans le cas d'une erreur, mentionné dans la figure 32.

2.2. L'endpoint sendContact

2.2.1. Status code : 201 CREATED

L'endpoint sendContact peut être consommer par les clients applicatifs à travers une requête http POST via l'URI suivant :

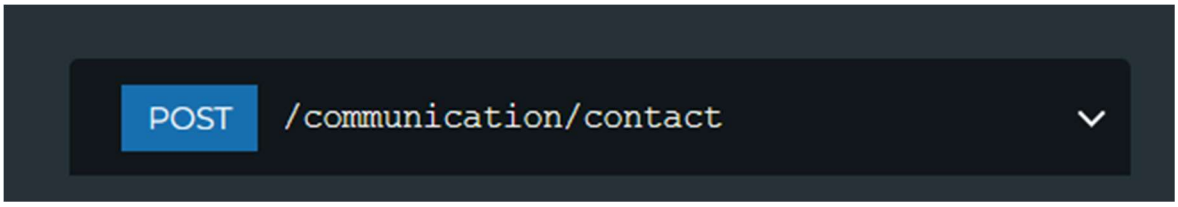


Figure 39: URI sendContact

Le client envoie un objet de type contact, sous format JSON.

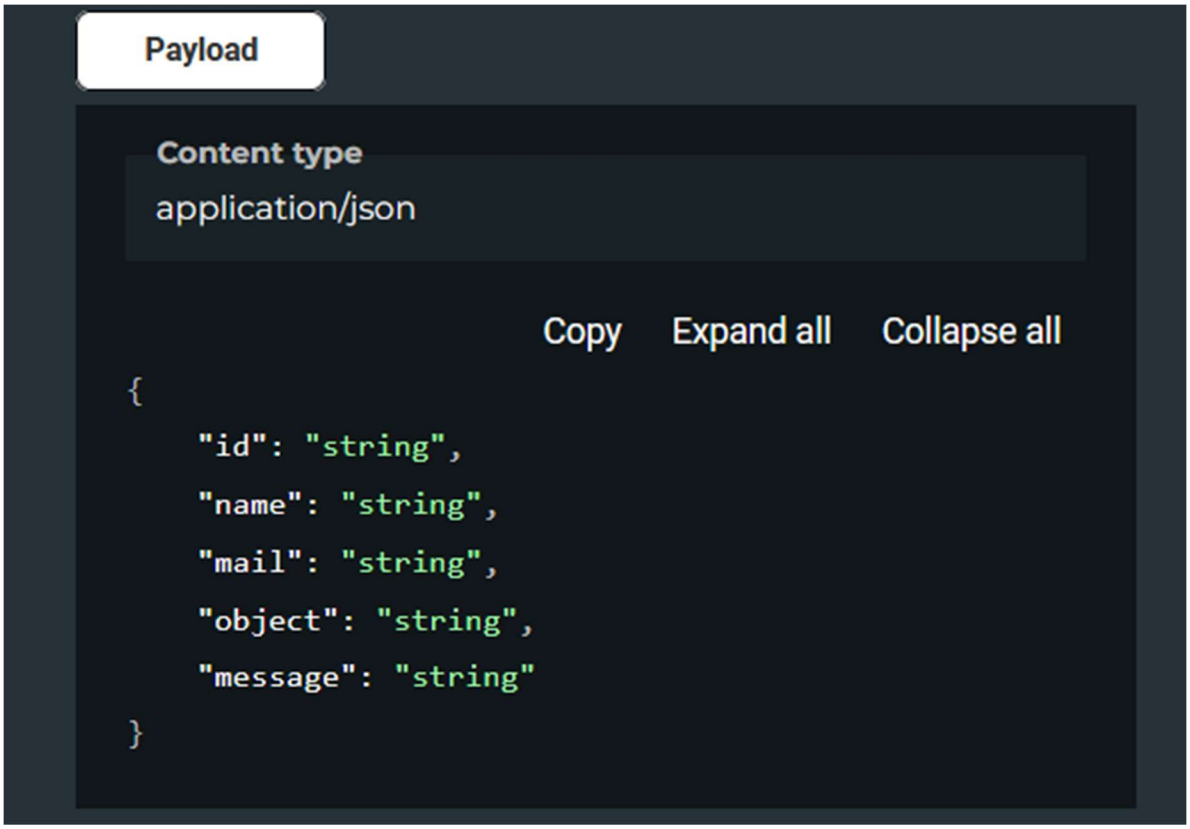


Figure 40: Objet Contact à envoyer dans la requête POST

Voici un exemple du client Postman.

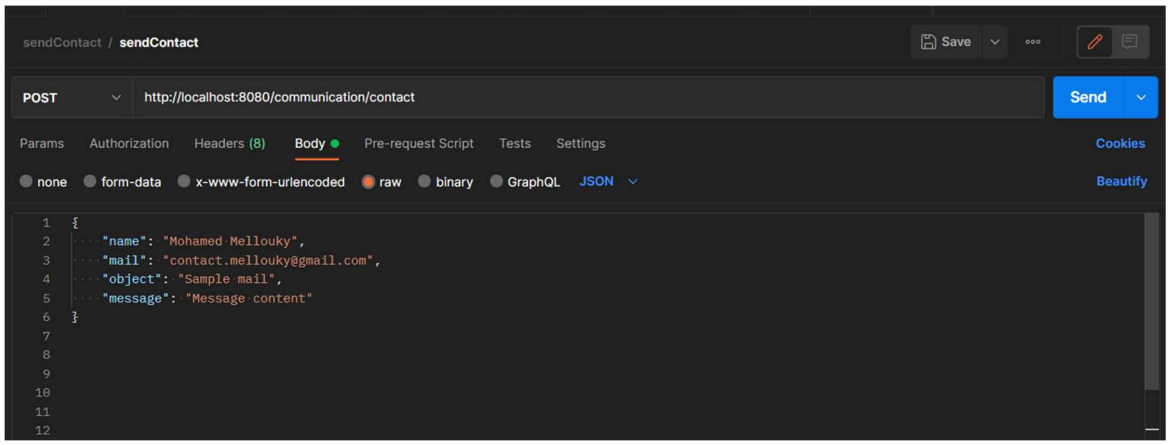


Figure 41: Requête POST – sendContact

La réponse de l'API dans le cas de succès selon la documentation de l'API est la suivante :

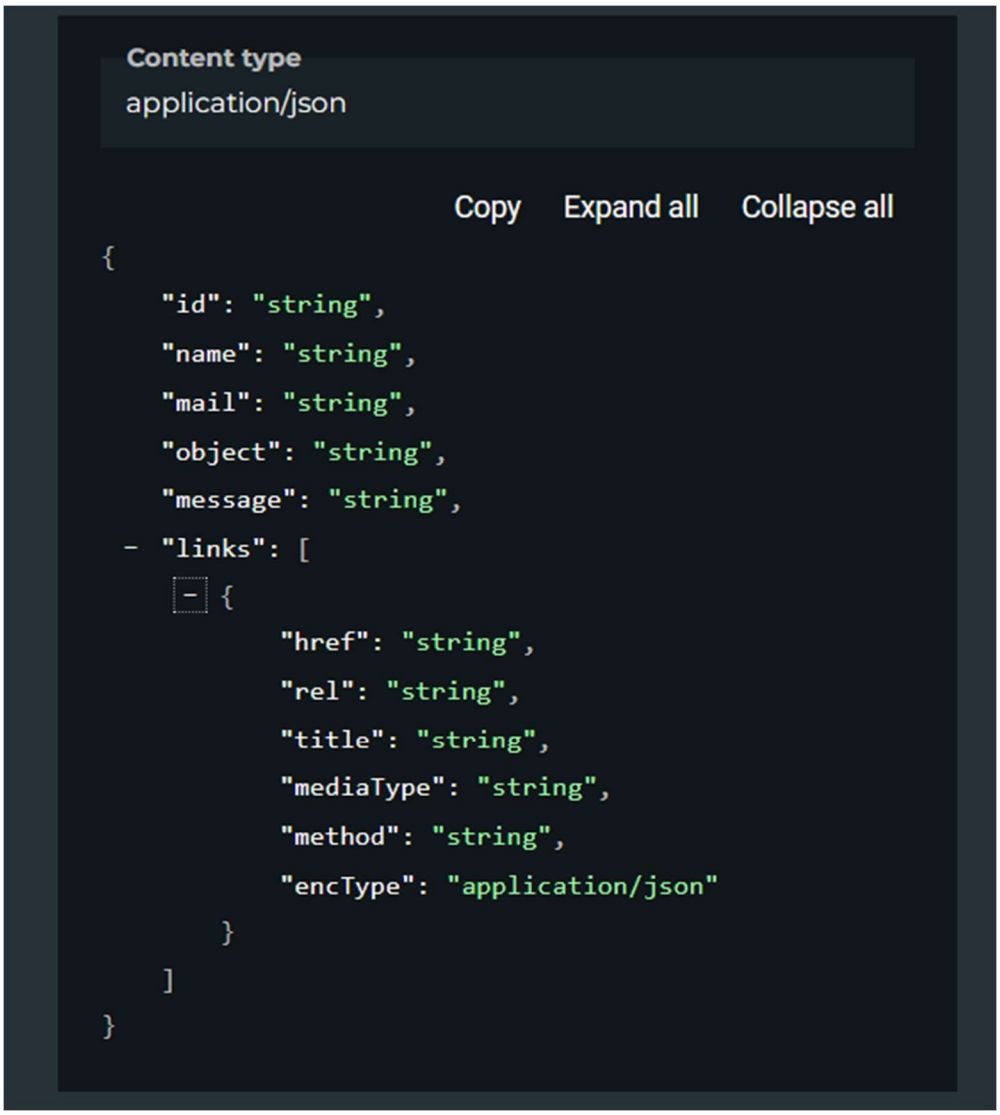


Figure 42: Réponse de l'API avec le code 201 Created – sendContact

En fait, la réponse de l'API est la suivante :

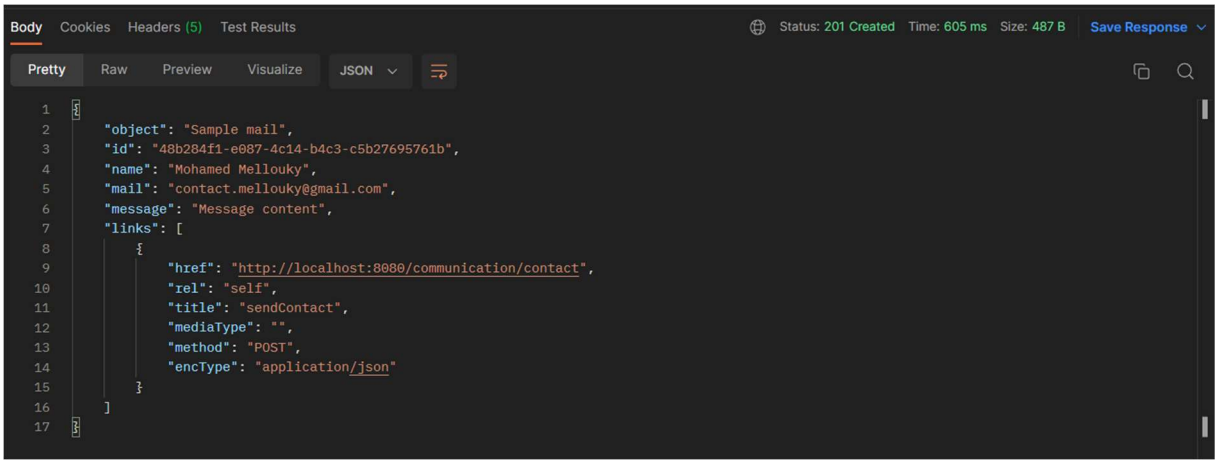


Figure 43: Réponse de l'api à la requête POST – sendContact

2.2.2. Status code : 500 INTERNAL SERVER ERROR

Comme l'endpoint précédent, je provoque un erreur 500 par l'envoie d'un objet nul. Dans cet exemple, je change la valeur de la variable d'en-tête **content-length**, je mets content-length = 0.

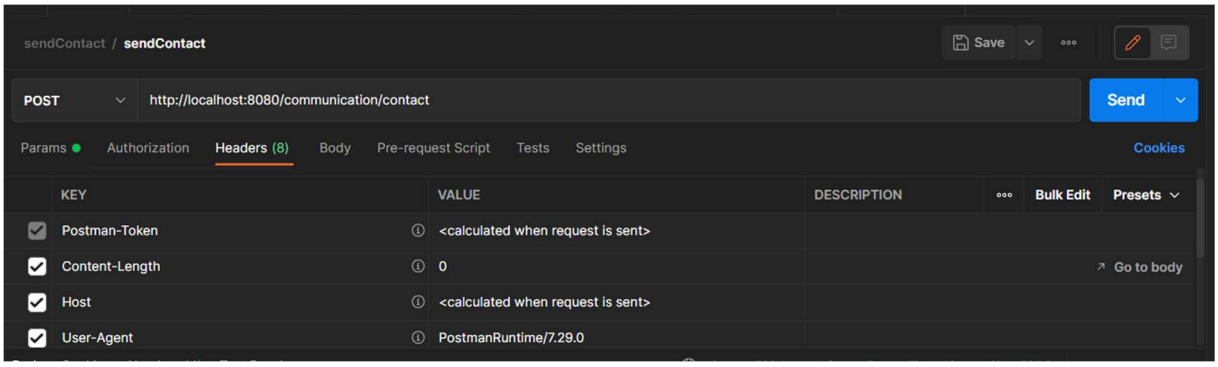


Figure 44: content-length = 0

L'API répond avec un erreur 500 INTERNAL SERVER ERROR, avec un message qui indique que l'objet envoyé ne doit pas être nul.

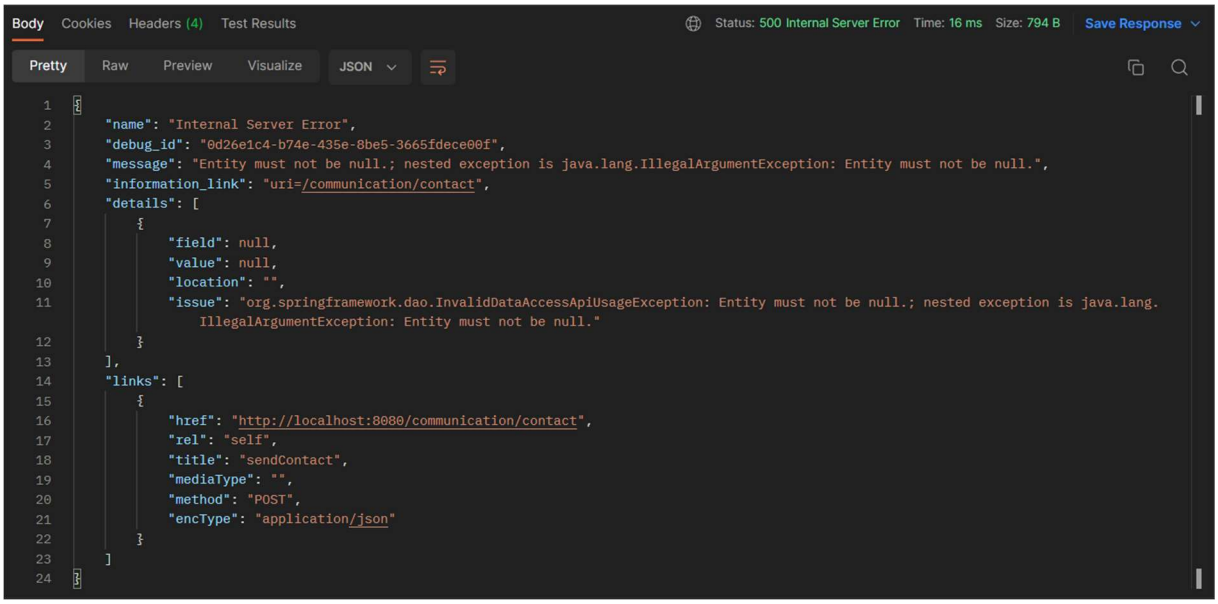


Figure 45: content-length = 0 réponse 500

2.3. L'endpoint getContact

2.3.1. Status code : 200 OK

Consommer l'endpoint getContact permet de récupérer un objet de type Contact de la base de données par son identifiant. Alors, le client doit fournir l'id de l'objet comme paramètre dans le chemin de l'URL.

Figure 46: URI getContact

La figure ci-dessous présente une requête http GET pour récupérer l'objet de type Contact avec l'id : dab729db-bbb9-4ec5-a643-85558b0f9076

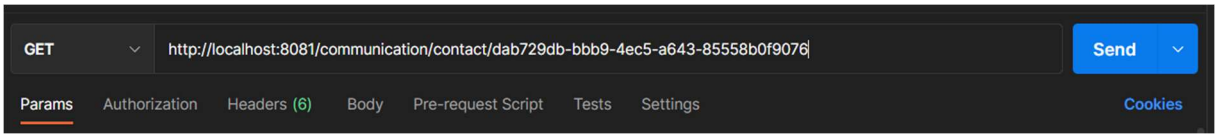
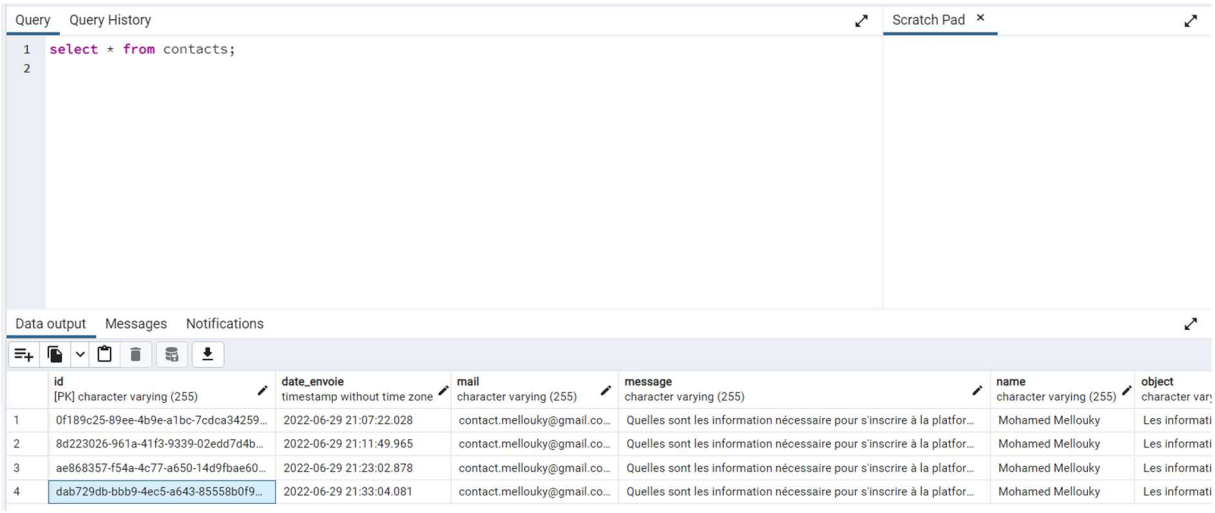


Figure 47: GET request – getContact

Avant d'entamer la réponse de l'API, la figure suivante vérifie que l'objet mentionné ci-dessus est dans la base de donnée :



	id [PK] character varying (255)	date_envoie timestamp without time zone	mail character varying (255)	message character varying (255)	name character varying (255)	object character vary
1	0f189c25-89ee-4b9e-a1bc-7cdca34259...	2022-06-29 21:07:22.028	contact.mellouky@gmail.co...	Quelles sont les information nécessaire pour s'inscrire à la platfor...	Mohamed Mellouky	Les informati
2	8d223026-961a-41f3-9339-02edd7d4b...	2022-06-29 21:11:49.965	contact.mellouky@gmail.co...	Quelles sont les information nécessaire pour s'inscrire à la platfor...	Mohamed Mellouky	Les informati
3	ae868357-f54a-4c77-a650-14d9fbae60...	2022-06-29 21:23:02.878	contact.mellouky@gmail.co...	Quelles sont les information nécessaire pour s'inscrire à la platfor...	Mohamed Mellouky	Les informati
4	dab729db-bbb9-4ec5-a643-85558b0f9...	2022-06-29 21:33:04.081	contact.mellouky@gmail.co...	Quelles sont les information nécessaire pour s'inscrire à la platfor...	Mohamed Mellouky	Les informati

Figure 48: Vérifier que l'objet avec l'id dab729db-bbb9-4ec5-a643-85558b0f90 existe dans la base de donnée

La réponse de l'API doit être sous la forme suivante :

Figure 49: Réponse de l'API avec le code 201 Created – getContact

Voici la réponse de l'API :

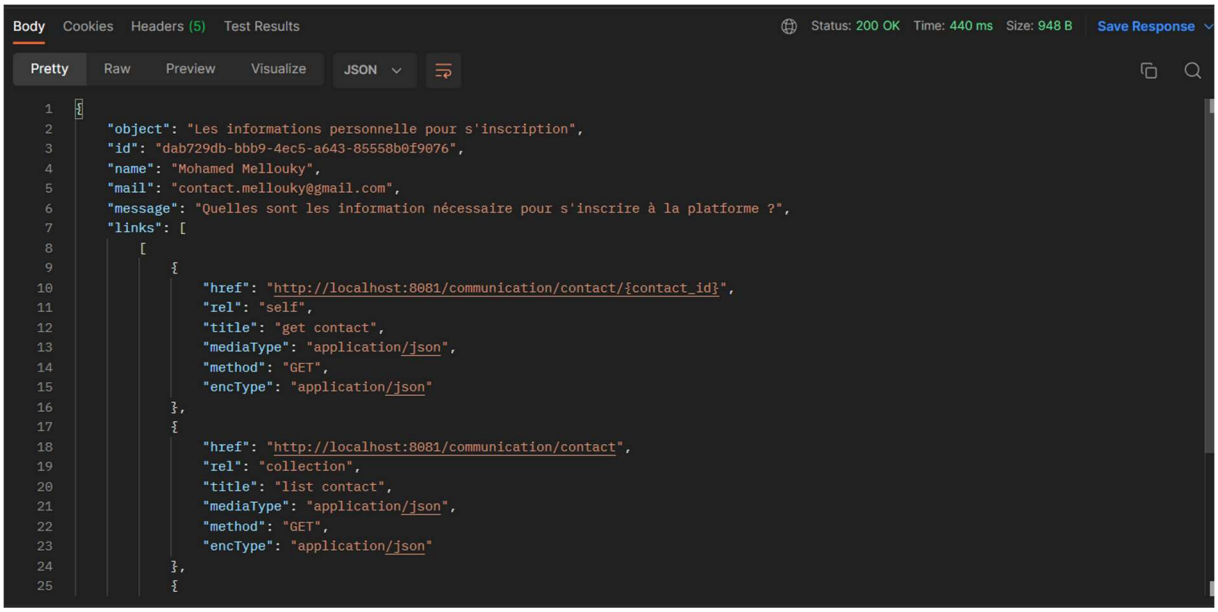


Figure 50: réponse de l'API avec le code 200 OK – getContact

2.3.2. Status code : 404 Not Found

Il est fort probable que le client cherchera une entité qui n'existe pas dans la base de donnée, dans ce cas, l'API répond au client avec un code approprié, c'est le 404 Not Found.

Pour présenter ce cas, je donne dans l'URI un identifiant qui n'est associé à aucun objet.

Exemple : xd9006c8-a8e4-4aa0-9aab-6104c271c42x

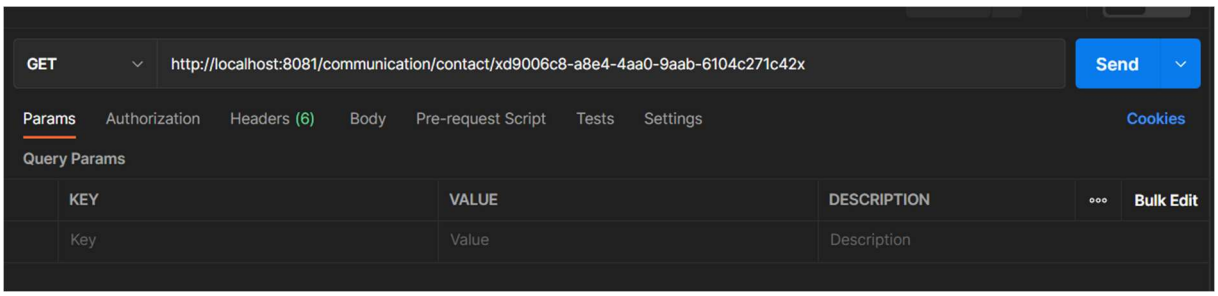


Figure 51: GET request - getClient - cas d'un contact inexistant

La réponse de l'API est la suivante :

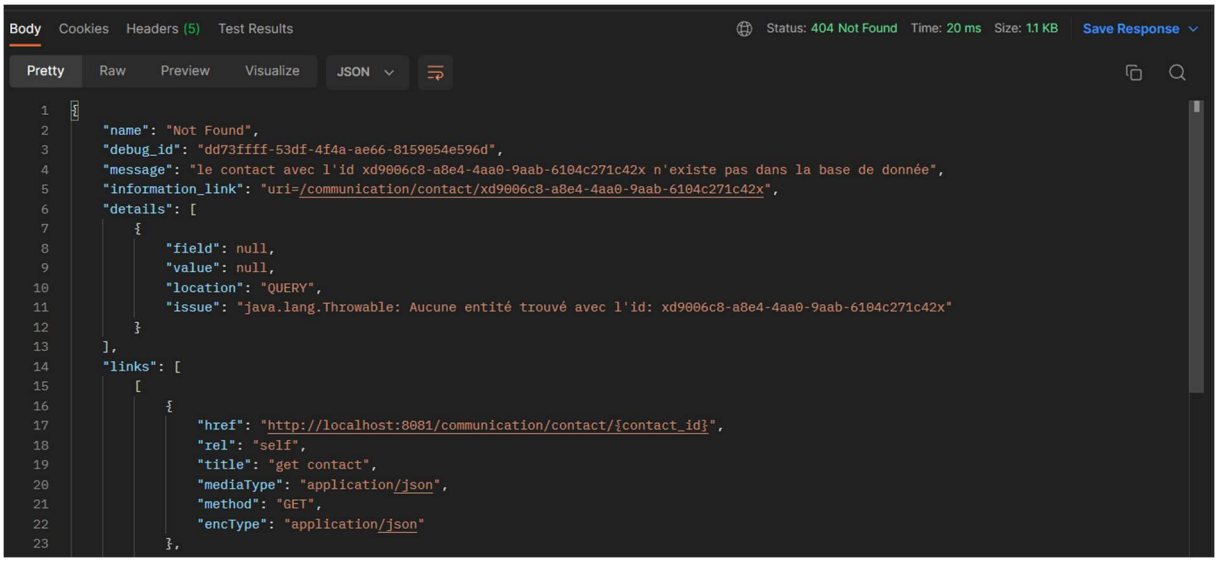


Figure 52: Réponse de l'api par le code erreur 404 Not Found

2.4. L'endpoint listContact

L'endpoint listContact est utilisé pour informer le front (la personne responsable à créer les interface graphique) combien d'objets sont persistés dans la base de données et combien de pages nécessaires pour les afficher.

En fait, l'API retourne deux informations :

- totalItems : représente le nombre total d'objets persistés dans la base de données.
- totalPages : représente le nombre total de pages nécessaires pour les afficher.

Voici le contrat d'interface correspondante :

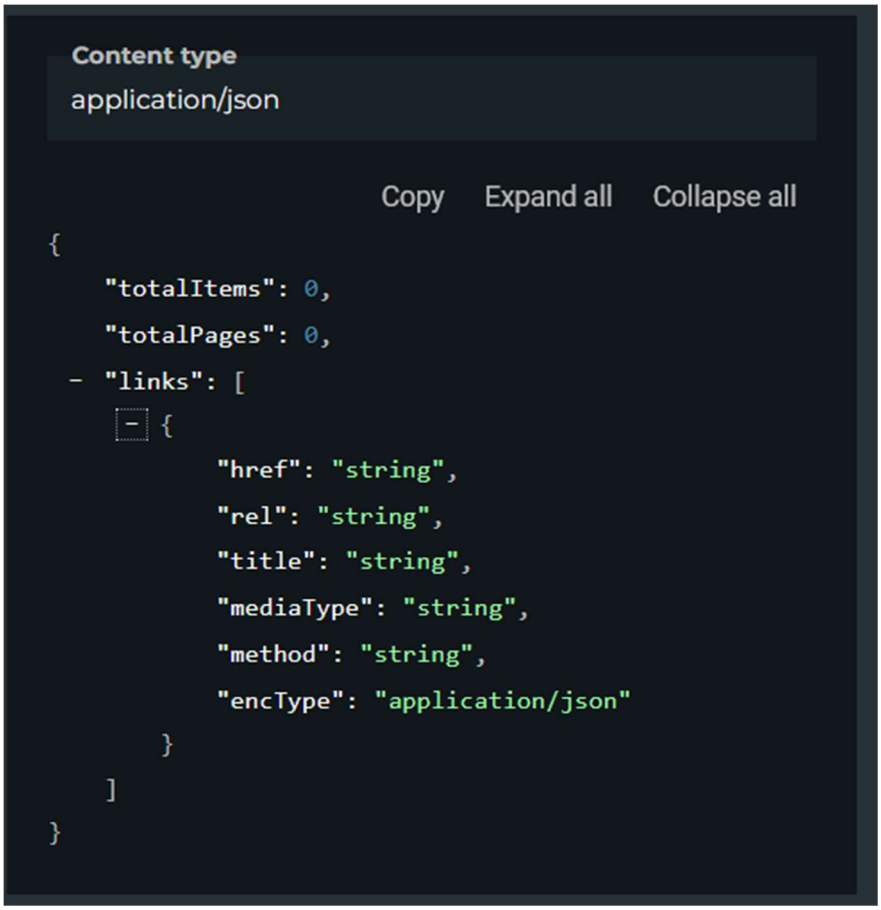


Figure 53: : Réponse de l'API avec le code 200 OK– listContact

2.4.1. Status code : 200 OK

L'endpoint listContact est consommé par une requête http GET.

Figure 54: URI listContact

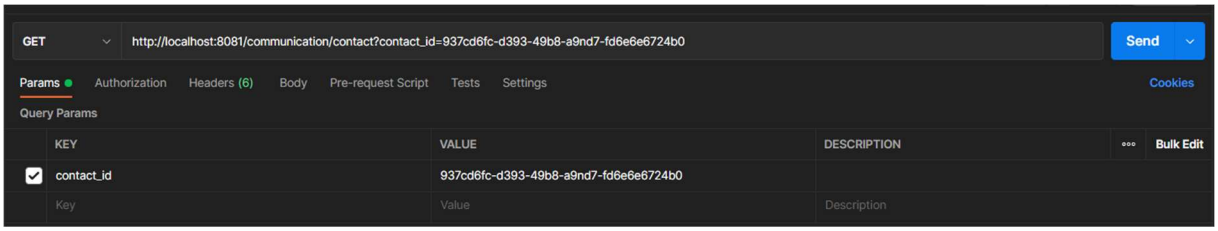


Figure 55: GET request – listContact

La réponse de l'API est la suivante :

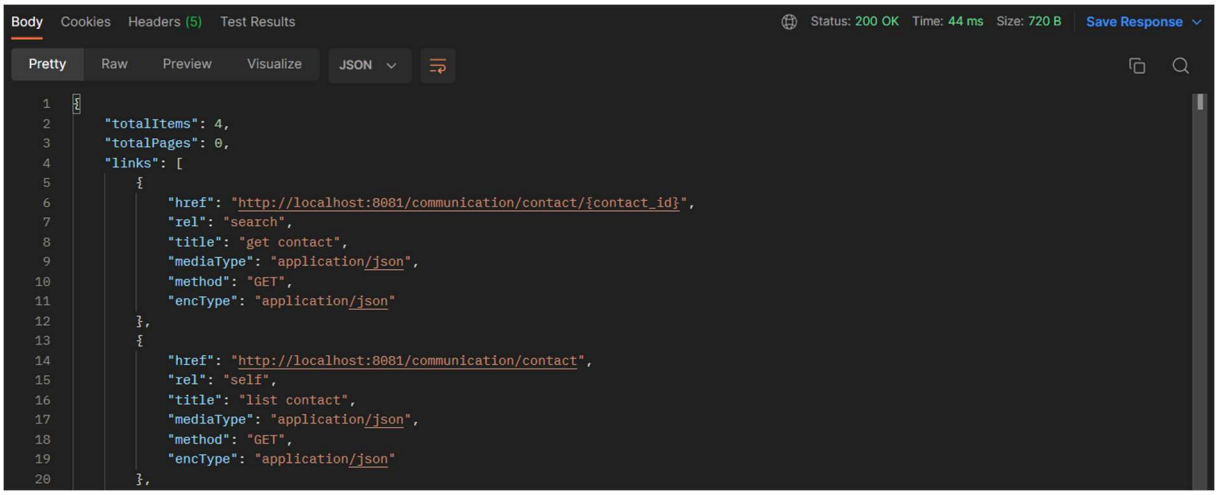


Figure 56: réponse de l'api par le code 200 Ok – listContact

On note que la réponse de l'API respecte la documentation.

2.5. L'endpoint sendMessage

2.5.1. Status code : 201 CREATED

L'endpoint sendMessage peut être consommé par une requête http POST.

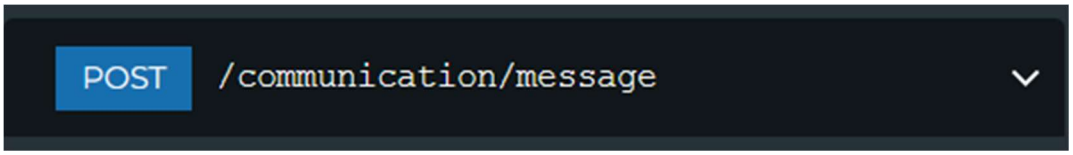


Figure 57: URI sendMessage

La requête http contient dans le corps un objet de type message, reçu sous format JSON

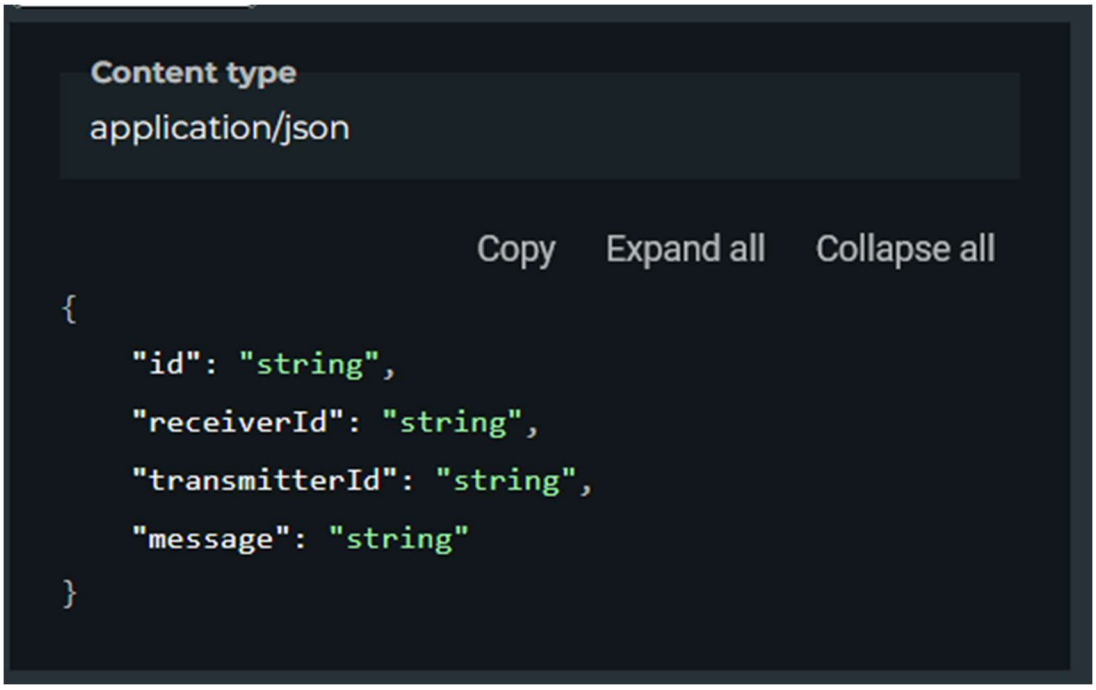


Figure 58: Objet Message à envoyer dans la requête POST

La figure ci-dessous représente une requête http par Postman :

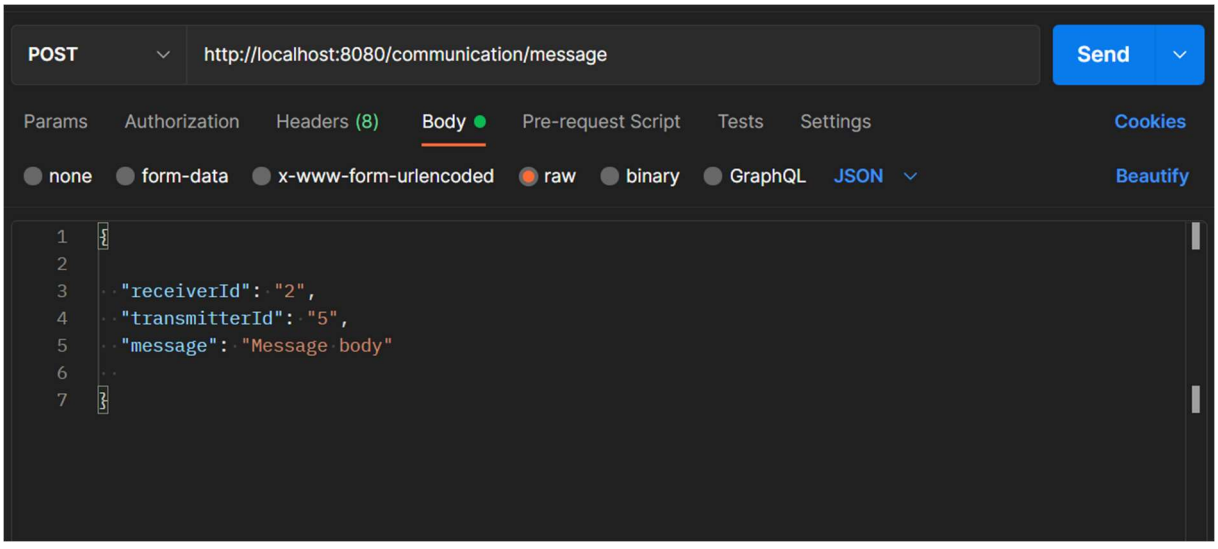


Figure 59: POST request sendMessage

La réponse de l'API dans le cas de succès est la suivante :

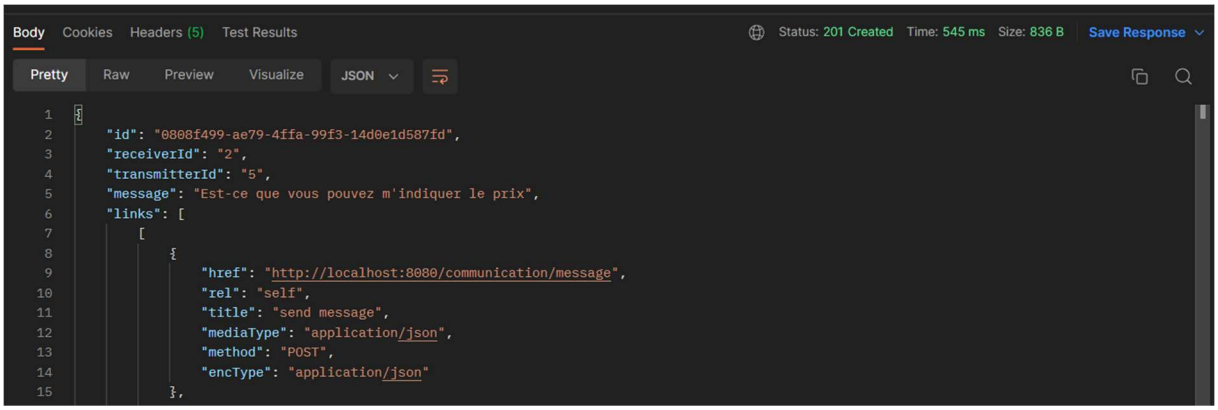


Figure 60: Réponse de l'api à la requête POST sendMessage

Voici la documentation, donc l'API répond correctement.

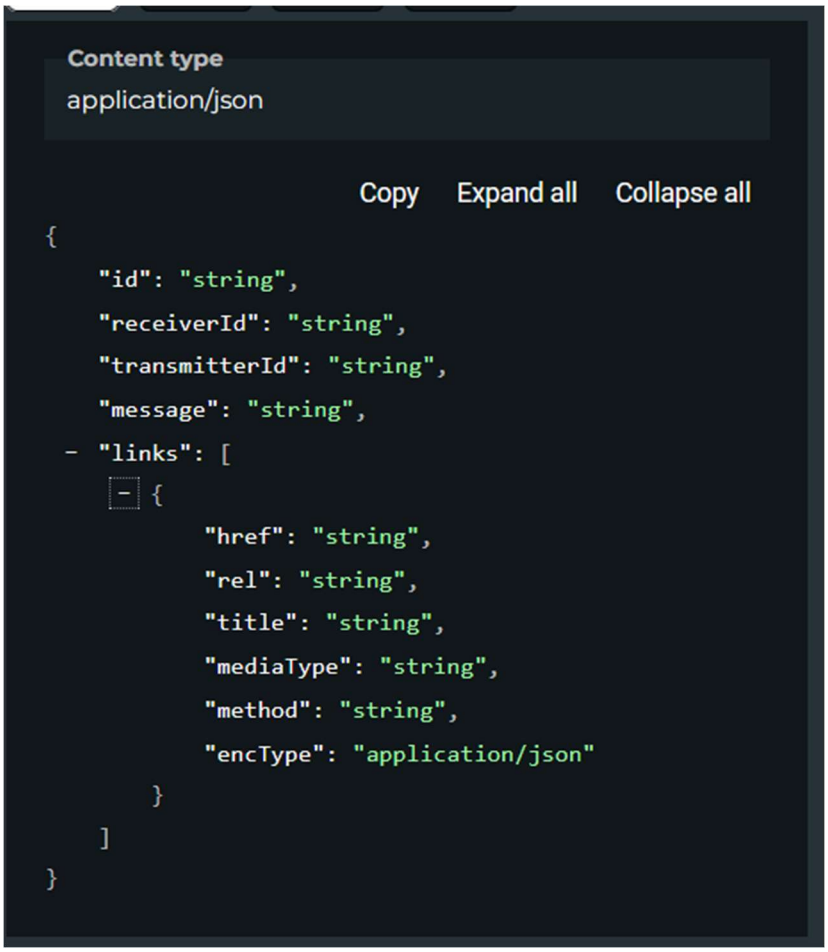


Figure 61: Réponse de l'API avec le code 201 Created – sendMessage

2.5.2. Status code : 500 INTERNAL SERVER ERROR

Pour présenter l'erreur 500, cette fois je simule une erreur dans le SGBD. Précisément je supprime une table dans laquelle je persiste les objets de

type *message*, dans ce cas, le système essaie de sauvegarder l'objet *message* dans la base de données, mais il ne trouve pas la table. La figure ci-dessus présente la suppression de la table de la base de données « communicationApi ».

```
communicationapi=# \d
             Liste des relations
 SchÚma |      Nom       | Type  | PropriÚtaire
--------+----------------+-------+--------------
 public | communications | table | postgres
 public | contacts       | table | postgres
 public | messages       | table | postgres
(3 lignes)


communicationapi=# drop table messages;
DROP TABLE
communicationapi=# \d
             Liste des relations
 SchÚma |      Nom       | Type  | PropriÚtaire
--------+----------------+-------+--------------
 public | communications | table | postgres
 public | contacts       | table | postgres
(2 lignes)
```

Figure 62: Suppression de la table messages de la base de données communicationapi

Ensuite, j'essaie d'envoyer une requête http valide. Puisque la table où le système doit sauvegarder l'objet message n'existe plus, il déclenche une exception 500.

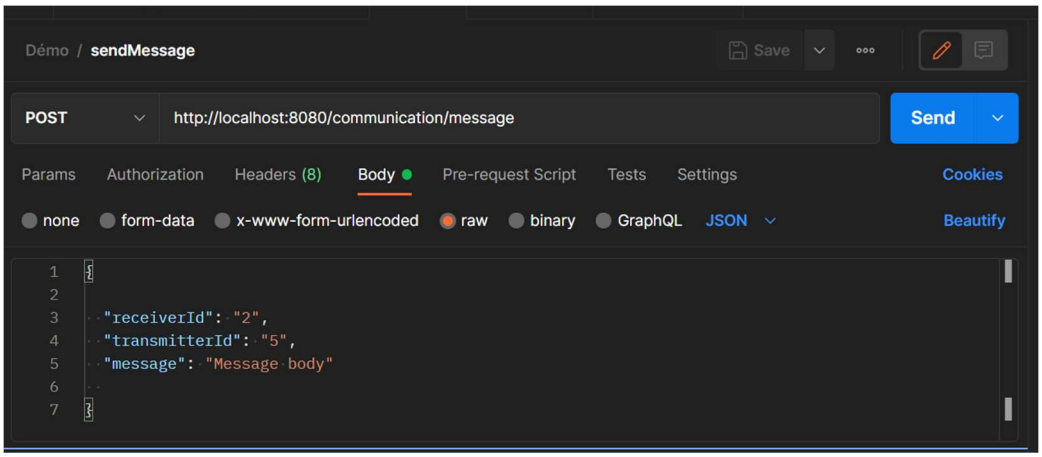


Figure 63: POST request - sendMessage

La réponse de l'API est la suivante :

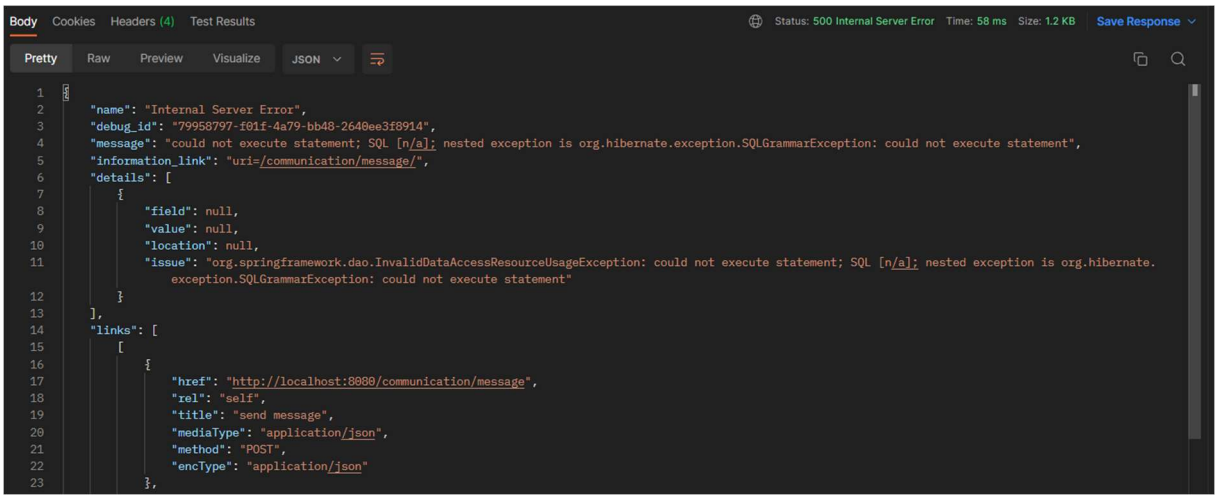


Figure 64: Réponse de l'api avec le code erreur 500 à cause d'une erreur dans le SGBD

2.6. L'endpoint getMessage

2.6.1. Status code : 200 OK

GetMessage permet de récupérer un objet de type message persisté dans la base de données par une requête http GET.

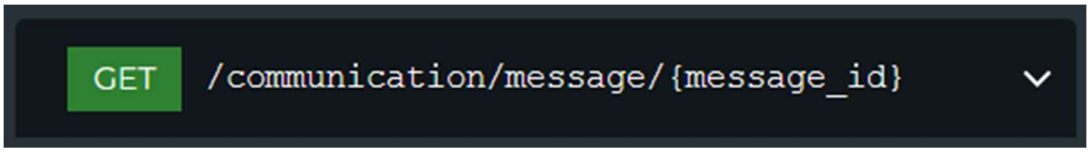


Figure 65: URI getMessage

Si l'identifiant du message fournie dans l'URI est associé à un objet dans la base de données, l'API retourne cet objet sous format de JSON, avec le code http 200 OK.

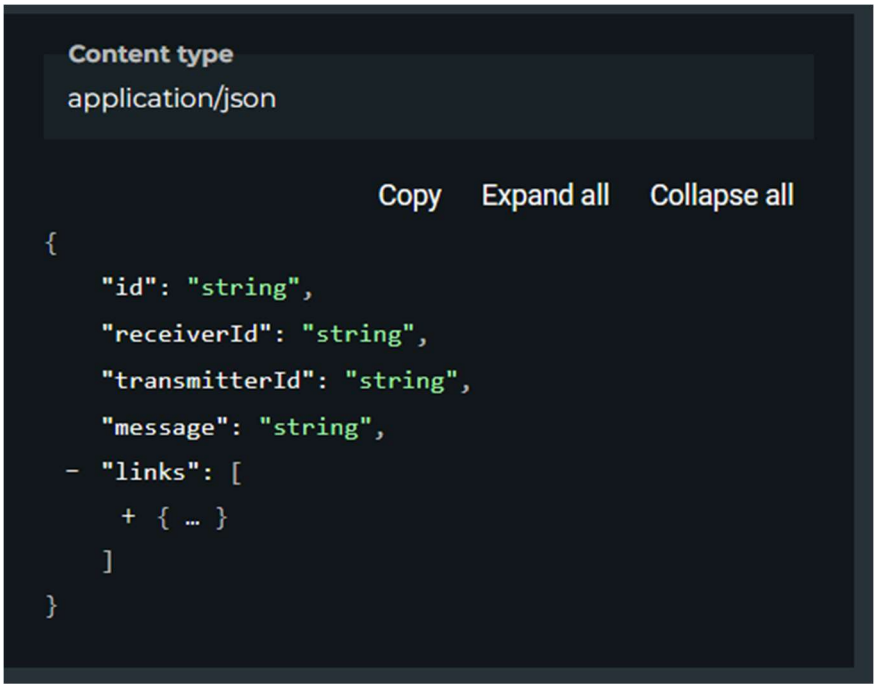


Figure 66: Réponse de l'API avec le code 200 Ok – getMessage

Voici une requête http pour consommer getMessage :

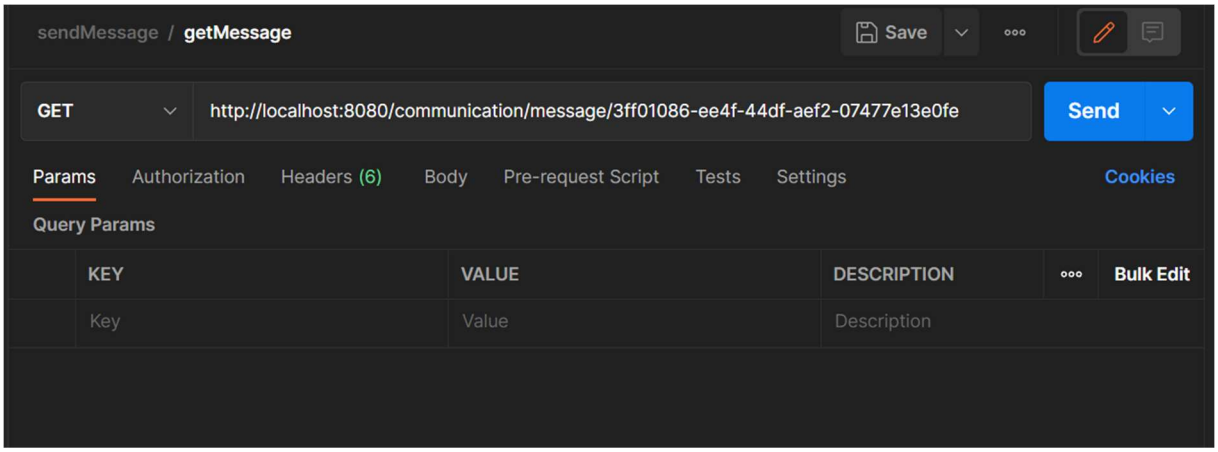


Figure 67: GET request – getMessage

La réponse de l'API est la suivante :

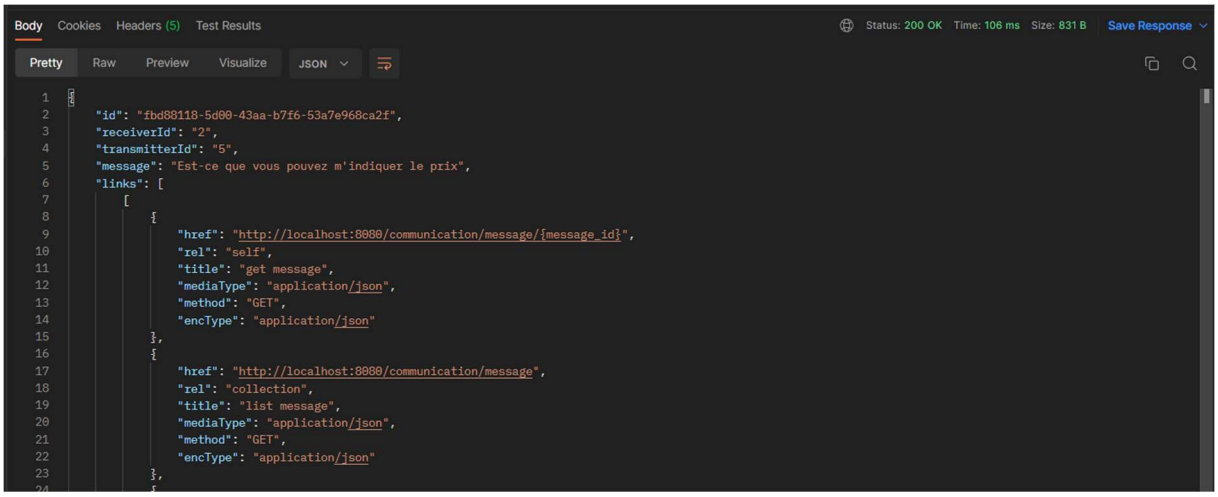


Figure 68: Réponse de l'api avec le code 200 OK – getMessage

2.6.2. Status code : 404 Not Found

On peut prévoir que le client fournie un identifiant qui n'existe pas dans la base de données. Dans ce cas l'API répond avec le code erreur qui indique que la ressource recherchée n'est pas trouvée, c'est le code erreur 404 Not Found.

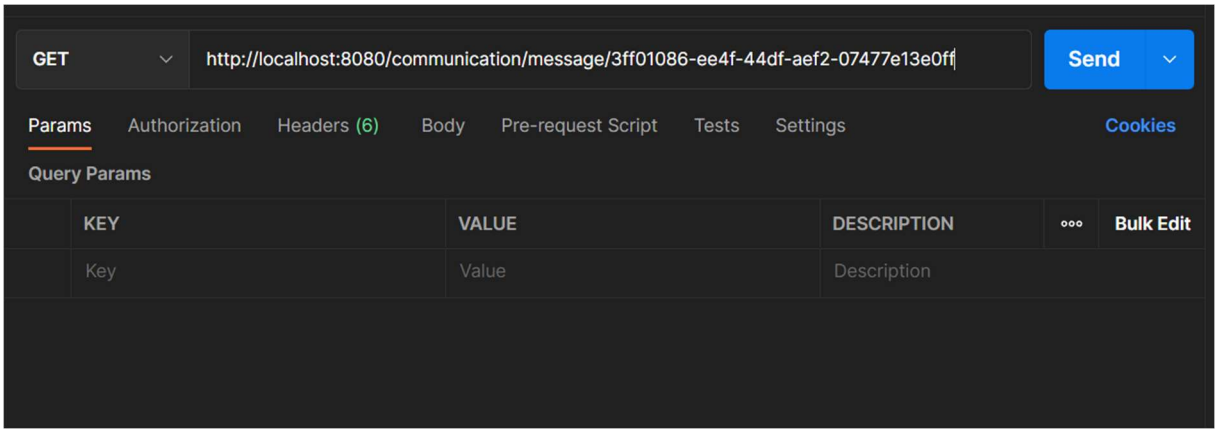


Figure 69: GET request - getMessage - cas d'un message qui n'existe pas

La réponse de l'API est la suivante :

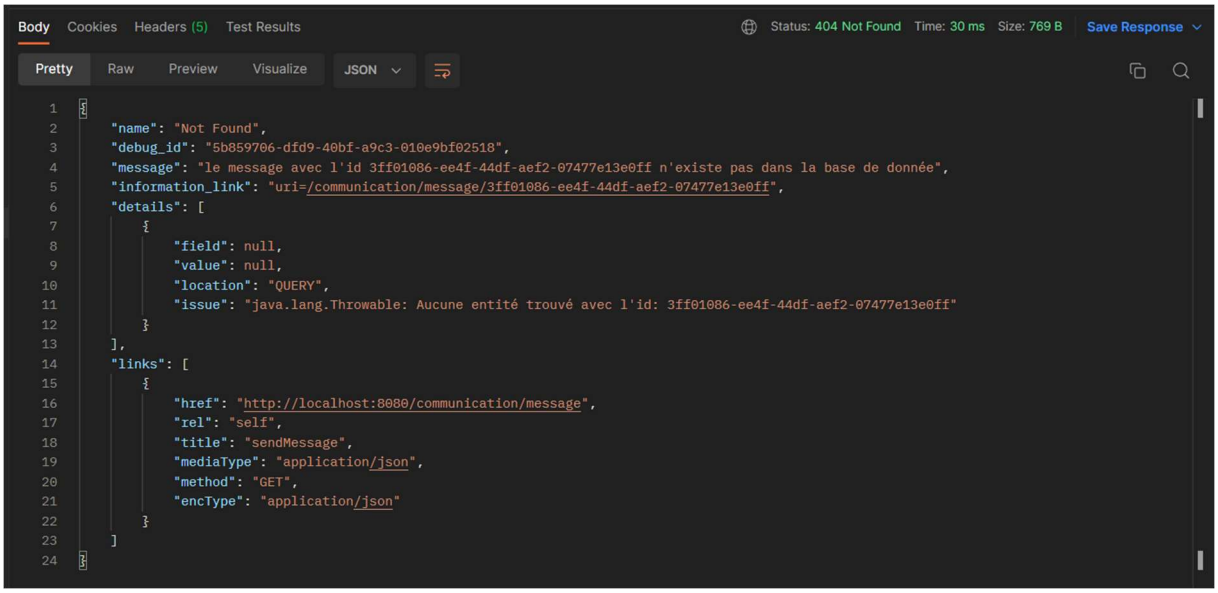


Figure 70: Réponse de l'API avec le code erreur 404 NOT FOUND – getMessage

2.6.3. Status code : 500 Internal Server Error

Dans le cas d'une erreur du serveur, l'API doit informer le client de cette erreur par une réponse de code 500.

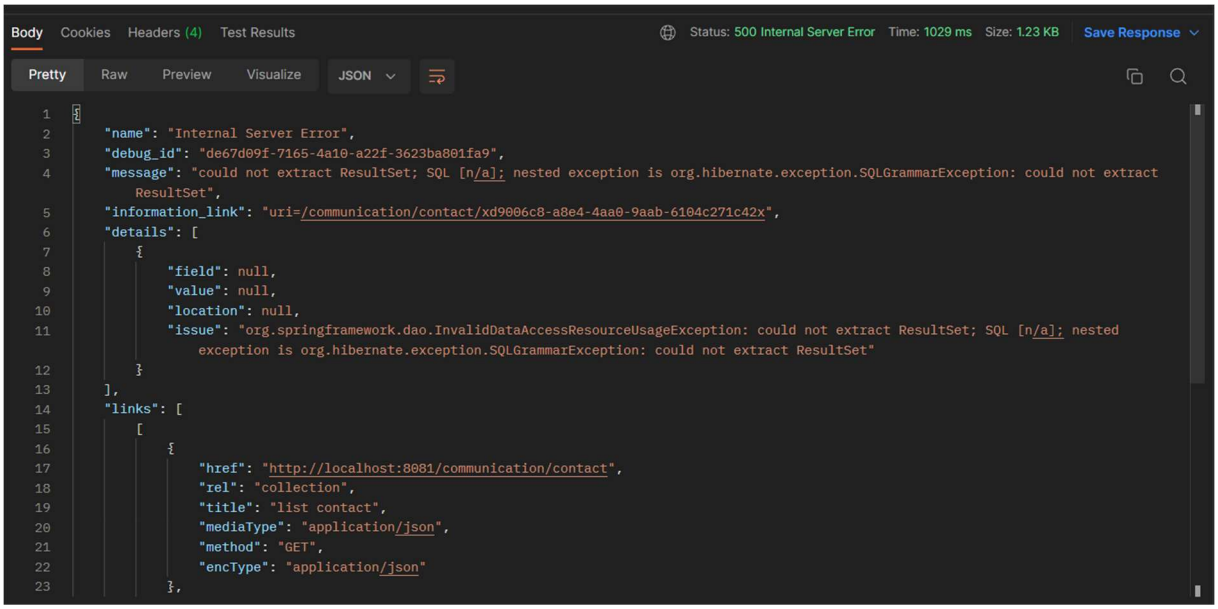


Figure 71: Réponse de l'API avec le code erreur 500 Internal Server Error – getMessage

2.7. L’endpoint listMessage

2.7.1. Status code : 200 OK

L’endpoint listMessage fournie deux informations :

- totalItems : représente le nombre d’objets de type message sauvegardés dans la base de données.
- totalPages : représente le nombre de pages nécessaires pour afficher les objets de type message.

Il est consommé par une requête http GET.

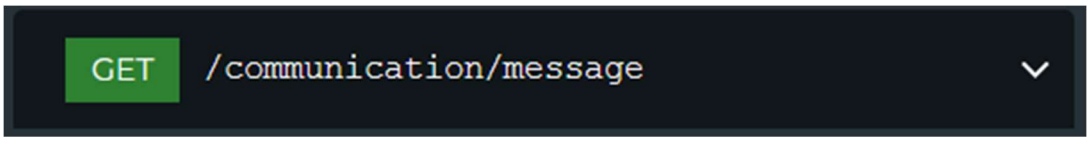


Figure 72: URI listMessage

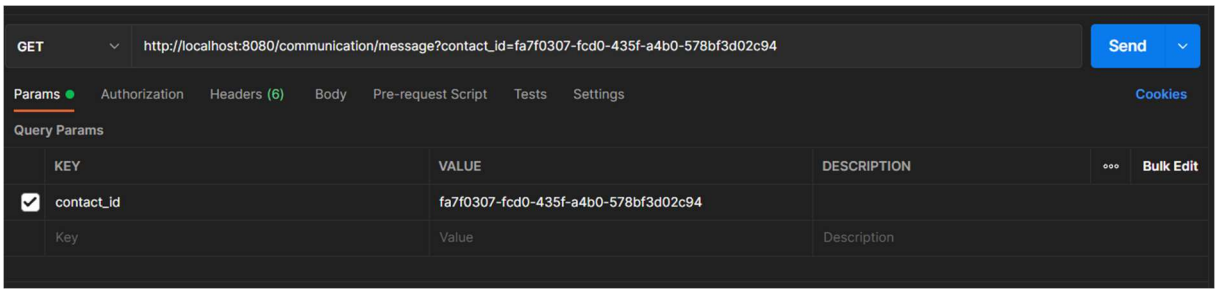


Figure 73: GET request - listMessage

La réponse de l’API est la suivante :

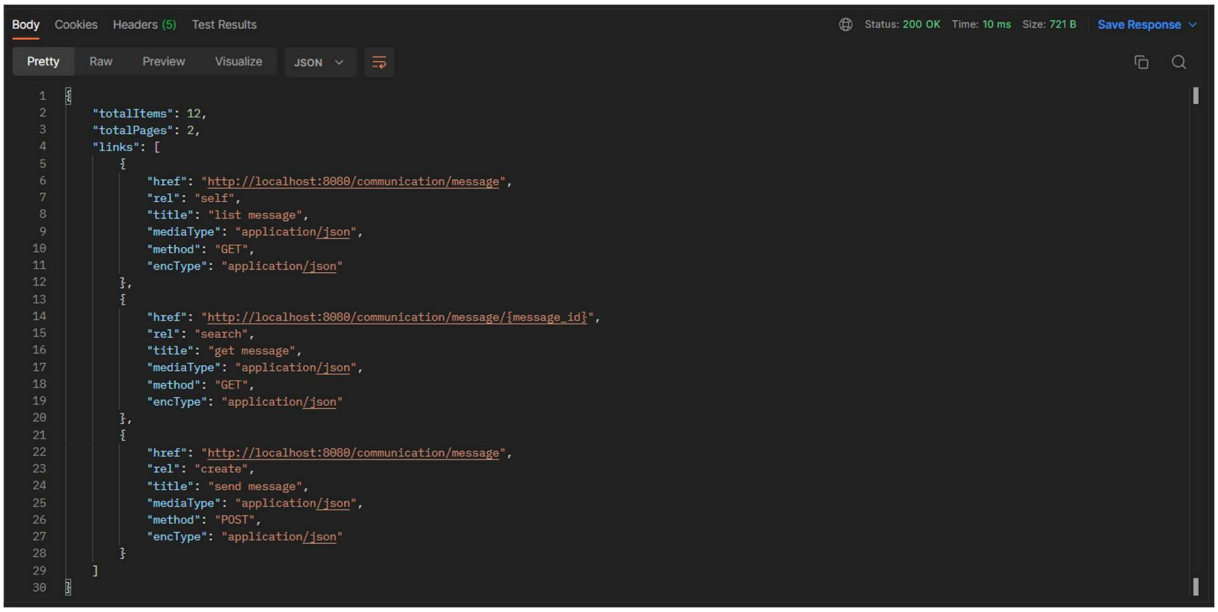


Figure 74: La réponse de l'API par le code 200 OK – listMessage

2.7.2. Status code : 500 Internal Server Error

L'une des raisons qui peut provoquer un erreur 500 est la recherche dans une table inexistante. La figure ci-dessous montre la suppression de la table « *messages* » qui persiste les objets de type *message*.

```
communicationapi=# \d
             Liste des relations
 SchÚma |      Nom       | Type  | PropriÉtaire
--------+----------------+-------+--------------
 public | communications | table | postgres
 public | contacts       | table | postgres
 public | messages       | table | postgres
(3 lignes)


communicationapi=# drop table messages;
DROP TABLE
communicationapi=# \d
             Liste des relations
 SchÚma |      Nom       | Type  | PropriÉtaire
--------+----------------+-------+--------------
 public | communications | table | postgres
 public | contacts       | table | postgres
(2 lignes)
```

Figure 75: Suppression de la table messages

Lorsque le système essaie de récupérer le nombre total d'objets persistés dans la base de données, il ne trouve pas la table dans laquelle il doit chercher, une exception se déclenche avec le code erreur 500. La figure ci-dessous représente une réponse 500 de l'api.

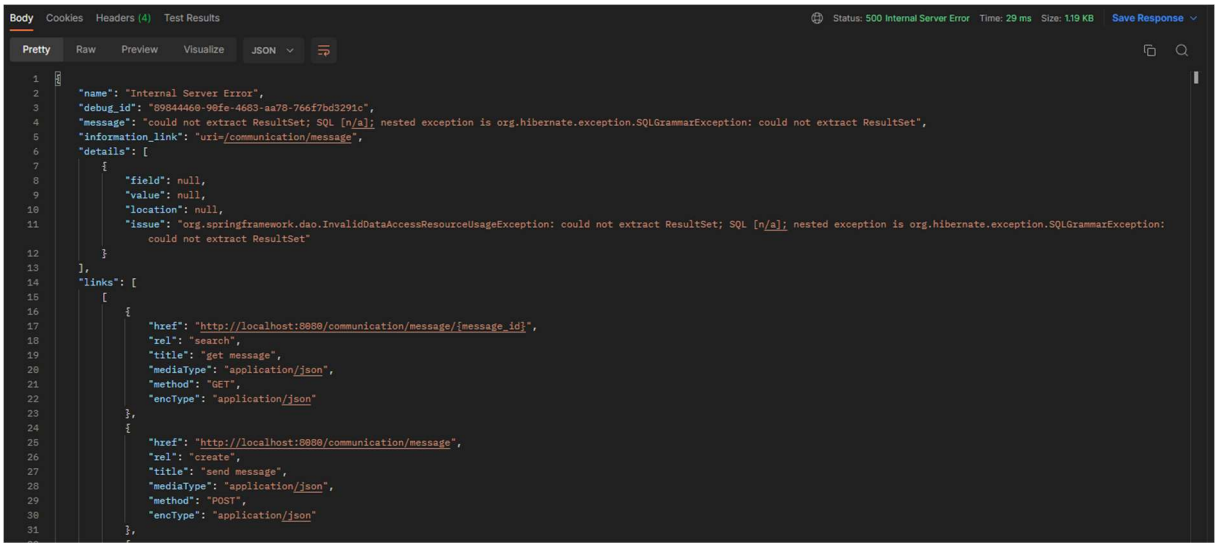


Figure 76: Réponse de l'api avec le code erreur 500 – listMessage

Conclusion

Mon projet de fin d'étude avait comme objectif le développement d'une API avec le Framework spring. Ce projet m'a fait énormément progresser techniquement.

L'API *communicationApi* est conçue pour gérer les communications entre le système et le client et entre les utilisateurs inscrits dans la plate-forme, à travers des e-mails et des messages.

On a déjà commencé par la persistance des données dans une base de données relationnelle, en plus de la mise en place de l'API en respectant le contrat d'interface. L'API est capable de persister les e-mails que les clients désirent envoyer, de chercher des messages dans la base de données et les renvoyer au client. En plus, de la gestion les exceptions de type BAD REQUEST, NOT FOUND et INTERNAL SERVER ERROR.

Perspective :

L'API reste toujours extensible par d'autres développeurs pour y ajouter d'autres fonctionnalités ou pour d'adapter aux besoins émergents. En fait, la fonctionnalité d'envoie des e-mails est encore en étude technique afin de faire le bon choix des outils pour mettre en place cette fonctionnalité.

Bibliographies

Booch, G., Rumbaugh, J., & Jacobson, I. (s.d.). *The Unified Modeling Language User Guide 2nd Edition* . Addison-Wesley Professional; 2nd edition (May 1, 2005).

Fowler, M. (s.d.). *UML Distilled: A Brief Guide to the Standard Object Modeling Language 3rd Edition* . Addison-Wesley Professional; 3rd edition (September 15, 2003).

Juba , S., Vannahme , A., & Volkov , A. (s.d.). *Learning PostgreSQL: Create, develop and manage relational databases in real world applications using PostgreSQL* . packt.

Miles, R., & Hamilton, K. (s.d.). *Learning UML 2.0: A Pragmatic Introduction to UML 1st Edition* . O'Reilly Media; 1st edition (May 16, 2006).

Narebski, J. (s.d.). *Mastering Git, Attain expert-level proficiency with Git for enhanced productivity and efficient collaboration by mastering advanced distributed version control features*. Packt.

Stonebraker, M., & Rowe, L. A. (s.d.). *THE DESIGN OF POSTGRES Michael Stonebraker and Lawrence A. Rowe Department of Electrical Engineering and Computer Sciences University of California Berkeley, CA 94720*.