جامعة سيدي محمد بن عبد الله بفاس
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⴰⵎⴻⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ ⴱ ⴼⴰⵙ
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES

كلية العلوم والتقنيات فاس
ⵜⴰⵖⵡⴰⵍⵜ ⵏ ⵜⵙⴰⵡⵍⵜ ⴷ ⵜⵙⵏⵉⵢⵉⵏ ⴰ ⵜⵙⵏⵉⵢⵉⵏ - ⴼⴰⵙ
FACULTÉ DES SCIENCES ET TECHNIQUES DE FÈS

Projet de Fin d'Etudes

Licence Sciences et Techniques Génie Informatique

Département Informatique

Développement d'une API REST pour gérer les hébergements des hôtes touristiques dans une plateforme digitale

Lieu de stage : NAWRA TECHNOLOGY

Réalisé par :

- Fatima AMIRI

Encadré par :

- Pr. ABDELALI BOUSHABA
- Mr. Mohamed KAARAR

Soutenu le 05/07/2022 devant le jury composé de :

Pr. Ilham CHAKER

Pr. Khalid ZENKOUAR

Pr. ABDELALI BOUSHABA

Remerciements

Au début, je tiens à adresser mes remerciements les plus sincères à toutes les personnes, qui, de près ou de loin, se sont impliquées dans la réalisation de ce rapport, tant par leur soutien opérationnel, que professionnel.

Une pensée particulière est adressée tout spécialement à mes parents pour leur présence et leur soutien inconditionnel pendant toutes les étapes de ma vie.

Je tiens à adresser mes remerciements les plus sincères à tout le corps professionnel et administratif de la Faculté des Sciences et Techniques de Fès. Je remercie sincèrement et tout particulièrement mes professeurs de la licence pour m'avoir permet de progresser sans cesse durant la période de la formation.

Je souhaite expirer mes gratitude à mon encadrant de stage Monsieur ABDELALI BOUSHABA, enseignant à la FSTF, pour avoir m'encadré et me dirigé tout au long de ce stage.

Mes vif remerciements s'adressent à tous les membres du jury pour avoir agréé d'évaluer ce travail.

En fin, je tiens également à adresser mes remerciements à mon encadrant professionnel Monsieur KAARAR MOHAMED, le directeur générale de la société NAWRA TECHNOLOGY, pour ses directives et ses explications et aussi pour le temps qu'il ma a consacré tout au long de cette période.

Résumé

Ce document est le rapport d'un projet de fin d'études pour l'obtention du diplôme Licence Génie Informatique de la Faculté des sciences et techniques de Fès. Le projet est réalisé dans le cadre d'un stage de deux mois au sein de la société NAWRA TECHNOLOGY.

Le sujet de stage consiste à développer une API REST pour une plateforme digitale de secteur touristique. L'API « *accommodationsApi* » permet la gestion des hébergements des biens touristiques dans cette plateforme. Elle va permettre à ses acteurs de gérer plusieurs ressources à savoir : les hôtes, les publications, les réservations et les opinions.

Abstract

This document is the report of an end-of-study project to obtain the degree in computer engineering from the Faculty of Sciences and Techniques of Fez. The project is carried out as part of a two-month internship within the company NAWRA TECHNOLOGY.

The internship subject is to develop a REST API for a digital platform for the tourism sector. The "accommodationsApi" API allows the management of tourist property accommodations in this platform. It will allow its actors to manage several resources, namely hosts, publications, reservations and opinions.

Sommaire

Liste des figures

Liste des tables

Liste des acronymes

Abréviation	Désignation
UML	UnifiedModelingLanguage
API	Application Programming Interface
REST	Representational state transfer
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JPA	Java Persistence API
HATEOAS	Hypermedia As The Engine of Application State
CRUD	Create, Read, Update, Delete
MLD	Modèle Logique des Données
JAR	Java ARchive
SGBD	Système de gestion de base de données
IDE	Intrgrated Development Environment

Introduction générale

Dans le cadre de notre formation LST génie informatique à la FST Fès, nous sommes amenés à réaliser un projet de fin d'études en vue d'obtenir le diplôme de licence. Dans ce but j'ai passé un stage de deux mois au sein de la société NAWRA TECHNOLOGY, où j'ai contribué à la conception et le développement de la partie BackEnd d'une plateforme digitale pour le secteur touristique. En particulier le développement d'une API REST qui va permettre la gestion des hébergements des hôtes touristiques dans cette plateforme.

Pour mettre en place cette API, plusieurs outils ont été utilisées : Spring Boot, Spring web, Spring Data et le SGBD PostgreSQL pour la gestion de la base de données.

Ce rapport est le compte rendu du ce stage, il sera devisé en trois chapitres, le premier contient la présentation de l'organisme d'accueil et de projet *Atlas Project* dont j'ai fait partie, il contient aussi le cahier des charges du système. Le deuxième sera consacré à l'analyse et la conception du système, et dans le dernier chapitre on présente les outils utilisés pour la réalisation ainsi que des démonstrations du fonctionnement du système.

Le rapport se termine par une conclusion dans laquelle on présente le travail réalisé ainsi que quelques perspectives futures.

Chapitre 1 :Contexte générale

1.1 Présentation de lieu de stage

NAWRA TECHNOLOGY est une société marocaine basé à Fès, crée en 2022 par des jeunes ingénieurs marocains intéressé par le développent des nouvelles solutions informatiques.

NAWRA TECHNOLOGY est un jeune éditeur logiciel qui se lance dans l'industrie de solutions digitales modernes et innovantes.

L'activité de NAWRA consiste à concevoir, réaliser et commercialiser des produits et des solutions pour les différents secteurs, en particulier le secteur de tourisme, d'où le projet *Atlas Project.*

1.2 Le projet *Atlas Project*

Atlas Project est le premier projet de la société NAWRA TECHNOLOGY, il vise la création d'une plateforme digitale pour le secteur touristique. Cette plateforme va offrir plusieurs services à ses clients, on cite :

- La réservation des biens entre particuliers.
- La réservation des biens proposés par des établissements touristiques.
- Des voyages organisés par des établissements spécialisés dans le domaine.

Et éventuellement des autres services qui seront ajoutés par la suite.

La plateforme va être utilisée par les acteurs suivants:

Acteurs	Description	Rôle
Les partenaires	Représentants des établissements touristiques (des hôtels, des raids…)	• Ajouter des hôtes. • Publier des annonces pour permettre aux clients (visiteurs de la plateforme) de réserver ces hôtes. • Gérer les réservations. • Gérer les opinions sur une annonce.
Les particuliers	personnes qui veulent proposer des chambres ou maisons à louer	
Les clients	visiteurs de la plateforme	• Consulter les publications proposées par la plateforme. • Passer des réservations. • Publier des opinions sur les annonces.

Tableau 1 : les cas d'utilisation de la plateforme

Remarque :

La plateforme propose d'autres services qui ne sont pas mentionner ici, car ils sont en relation avec des autres APIs développés par des autres stagiaires. Cette partie concerne seulement la description de fonctionnalités proposées par l'API « *AccommodationsApi* » sujet de stage.

Notre mission durant ce stage était d'intégrer une équipe de développement BackEnd, et contribuer au développement de la partie BackEnd de la plateforme.

Pour développer la partie BackEnd la société a choisi une architecture microservices.

Le schéma suivant représente l'architecture de BackEnd de la plateforme, il est constitué de plusieurs APIs.

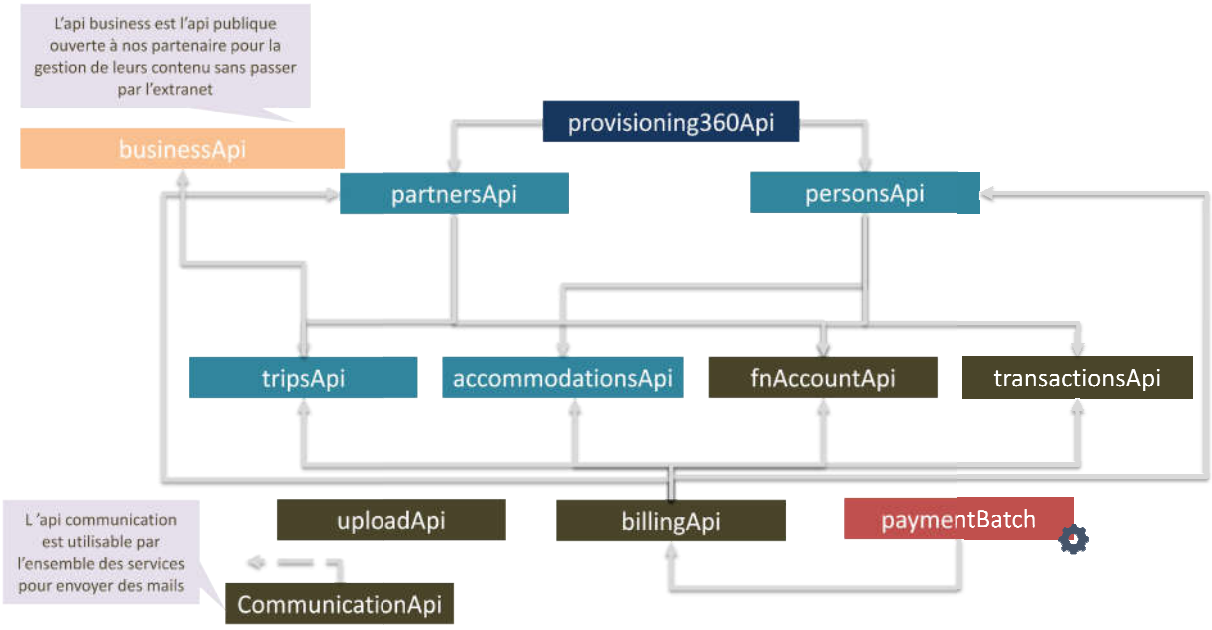

Figure 1 : l'architecture microservice de la plateforme

Dans ce qui suit on présente les responsabilités de quelques APIs de cette architecture.

L'API « *provisioning360Api* » est responsable de la création des comptes utilisateurs.

L'API « *partnersApi* » est responsable de la gestion des partenaires.

L'API « *personsApi* » est responsable de la gestion despersonnes.

L'API « *accommodationsApi* » est responsable de la gestion des hôtes : location, création et réservations.

L'API « *communicationApi* » est responsable de la gestion de communication : Mail, SMS et notifications.

Le sujet de notre stage consiste à concevoir et développer l'API REST « *accommodationsApi* ».

1.3 Cahier des charges de l'API « *accommodationsApi* »

L'API REST « *accommodationsApi* » est l'API responsable de la gestion des hébergements des hôtes touristiques dans la plateforme, elle doit accomplir les tâches suivantes :

Gestion des hôtes *:*

Un hôte est le produit touristique proposé par la plateforme, il peut s'agir d'un :maison, Riad, hôtel appartement, chambres (etc.)

L'Api doit fournir les services suivants à ses acteurs :

- Ajouter un hôte
- Modifier un hôte
- Supprimer un hôte
- Lister les hôtes
- Chercher un hôte bien spécifié

Gestion des annonces :

Pour un bien (hôte) les propriétaires peuvent publier des annonces pour présenter leurs biens et permettre aux visiteurs de la plateforme de faire des réservations.

L'API doit permettre à ses acteurs de publier des annonces sur les hôtes et les biens touristiques et de gérer ces annonces.

- Ajouter une annonce
- Modifier une annonce
- Supprimer une annonce
- Lister les annonces
- Chercher une annonce bien spécifiée

Gestion des réservations :

A partir des annonces publiées l'API doit permettre la réservation des biens proposés et de gérer ces réservations.

- Ajouter une réservation
- Modifier une réservation
- Supprimer une réservation
- Lister les réservations
- Chercher une réservation bien spécifiée

Gestion des opinions :

L'API doit offrir à ses acteurs les moyennes de publier des opinions sur les annonces et de gérer ces opinions :

- Ajouter une opinion
- Modifier une opinion
- Supprimer une opinion
- Lister les opinions
- Chercher une opinion bien spécifiée

Gestion des erreurs :

L'API doit prend en charge la gestion des erreurs :

- côté client : des mauvaises requêtes ou des données invalide.
- côté serveur : erreurs techniques.

1.4 Diagramme de Gantt

Pour éclaircir les étapes de réalisation du ce projet on a effectué le diagramme de Gantt suivant :

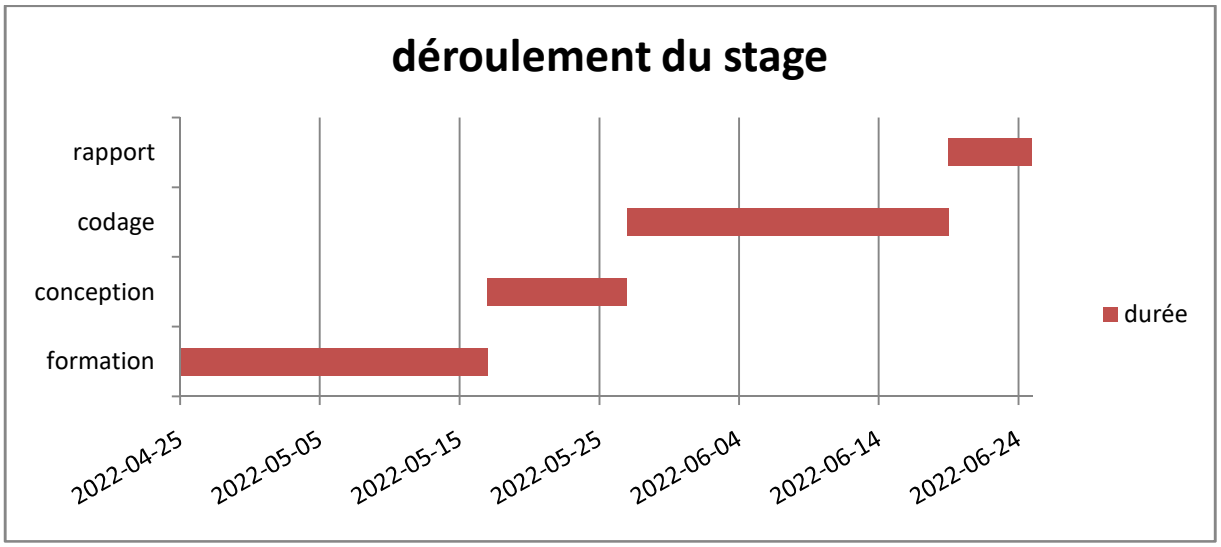

Figure 2 : le diagramme de Gantt

Chapitre 2 : Analyse et conception

2.1 Analyse des besoins fonctionnels

2.2.1 Les acteurs du système

Un acteur est une entité externe du système qui interagisse directement avec le système, il peut être une personne physique ou un autre système qui offert ou consomme les services du système.

Pour une API REST les acteurs sont des autres systèmes ; des clients applicatifs ou applications FrontEnd (applications web, mobiles…), ces clients vont envoyer des requêtes HTTP aux EndPoints (URI sur une API auxquels une application peut accéder) d'API pour gérer les différentes ressources.

La figure suivante montre le processus d'utilisation de l'API, les acteurs de la plateforme vont interagissent avec le front et ce dernier va envoyer des requêtes HTTP à l'API pour répondre aux actions des acteurs de la plateforme.

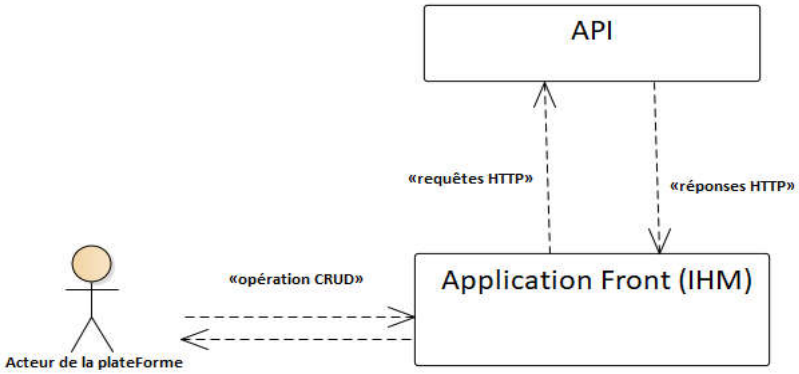

Figure 3 : les acteurs de l'API

Le tableau suivant représente les acteurs de l'API et leur cas d'utilisation :

Acteur	Cas d'utilisation
Client applicatif	▪ Gestion des hôtes ▪ Gestion des annonces ▪ Gestion des réservations ▪ Gestion des opinions

Tableau 2 :les acteurs et leurs rôles

2.2.2 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation permet de représenter les principales fonctionnalités du système, pour notre API on propose le diagramme suivant :

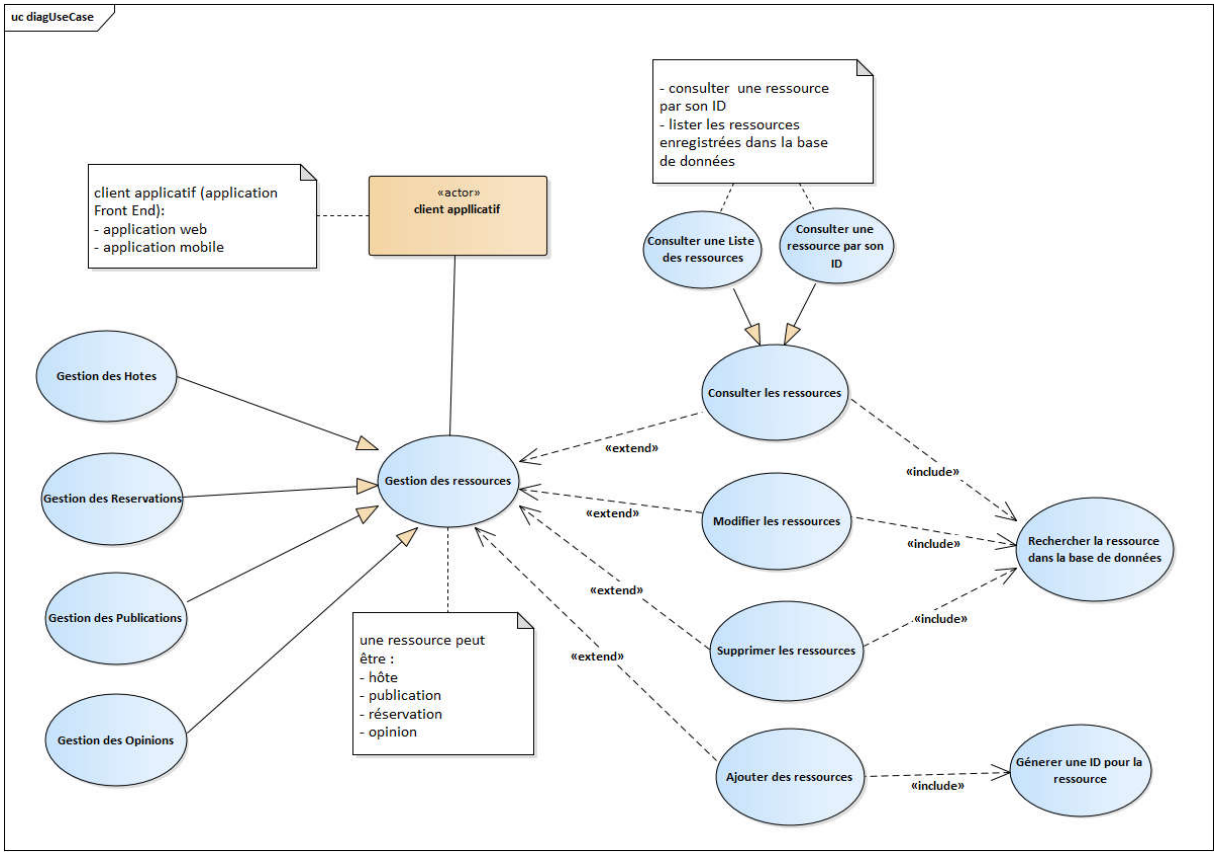

Figure 4 : le diagramme de cas d'utilisation

2.2.3 Description détaillée descas d'utilisation

Les cas d'utilisation de cette API sont des opérations CRUD qui s'effectue sur les différentes ressources gérées par l'API.

Dans ce que suit on va présenter les cas d'utilisation de l'API avec une description détaillée des scénarios normaux et alternatifs.

a) Cas d'utilisation : Ajouter des ressources

Nom de cas d'utilisation	Ajouter une ressource
Acteur	Client applicatif
Description	Persister une ressource (hôte, annonce, réservation, opinion) dans la base de données
Pré-condition	Envoyer une requête HTTP à l'URI convenable
Post-condition	Enregistrer les données dans la base

Tableau 3 : le cas d'utilisation ajouter une ressource

- **Scénario normale :**
  - Le client applicatif envoi une requête HTTP à l'URI avec des paramètres et des données valides.

- L'API persiste les données dans la base et renvoi un statut 201 *CREATED*, et un corps de réponse JSON (*Response Body*) qui inclut l'objet persisté avec son ID auto-généré et des liens HATEOAS.

➢ **Scénario alternatif 1:**
- Le client applicatif fait un appel avec des paramètres incorrects ou avec des données invalides.
- L'API renvoi un statut 404 *BAD REQUEST* avec un objet JSON qui représente des informations sur la source d'erreur.

➢ **Scénario alternatif 2 :**
- Le client applicatif fait un appel correct comme renseigné dans la documentation de l'API avec des données valides.
- Le client n'est pas autorisé d'ajouter la ressource, l'API renvoi le statut 403 *FORBIDDEN*, avec un message explicatif de la source d'erreur.

➢ **Scénario alternatif 3 :**
- Le client applicatif fait un appel correct comme renseigné dans la documentation de l'API avec des données valides et il est autorisé d'effectuer l'opération.
- Une erreur interne se produit (erreur technique), l'API renvoi le statut 500 *INTERNAL SERVER ERROR* et un message d'erreur.

b) Cas d'utilisation : Modifier une ressource

Nom de cas d'utilisation	Modifier une ressource
Acteur	Client applicatif
Description	Mettre à jour une ressource déjà persisté dans la base de données.
Pré-condition	Envoyer une requête HTTP à l'URI convenable
Post-condition	Modifier la ressource dans la base des données

Tableau 4 : le cas d'utilisation modifier une ressource

➢ **Scénario normale :**
- Le client applicatif envoi une requête HTTP à l'URI avec des paramètres et des données valides.
- L'API cherche la ressource dans la base.
- La ressource existe, et le client a le droit d'effectuer la modification, l'API renvoi un statut 201 *CREATED* avec un objet JSON qui représente la ressource après les modifications en plus des liens HATEOAS.

➢ **Scénario alternatif 1:**
- Le client fait un appel correct comme renseigné dans la documentation de l'API.
- L'API cherche la ressource dans la base.
- La ressource existe, le client n'est pas autorisé de modifier la ressource, l'API renvoi un statut 403 *FORBIDDEN*.

➢ **Scénario alternatif 2:**
- Le client fait un appel avec des paramètres incorrects ou avec des données invalides.
- L'API renvoi un statut 400 *BAD REQUEST* avec un message explicatif de la source d'erreur.

- **Scénario alternatif 3:**
  - Le client fait un appel correct comme renseigné dans la documentation de l'API.
  - L'API cherche la ressource dans la base.
  - La ressource n'existe pas, l'API envoi un statut 404 *NOT FOUND* et un message d'erreur.
- **Scénario alternatif 4:**
  - Le client fait un appel correct comme renseigné dans la documentation de l'API avec des données valides.
  - L'API cherche la ressource dans la base.
  - La ressource existe, et le client a le droit d'effectuer la modification.
  - Une erreur interne se produit (erreur technique) l'API renvoi le statut 500*INTERNAL SERVER ERROR* et un message d'erreur.

c) Cas d'utilisation : consulter une ressource

Nom de cas d'utilisation	Consulter une ressource
Acteur	Client applicatif
Description	Chercher une ressource persistée dans la base des données
Pré-condition	Envoyer une requête HTTP à l'URI convenable
Post-condition	Envoyer les ressources recherchées

Tableau 5 : le cas d'utilisation consulter une ressource par son ID

- **Scénario normale :**
  - Le client applicatif envoi une requête HTTP à l'URI avec les bons paramètres.
  - L'API cherche la ressource dans la base.
  - La ressource existe, et le client a le droit de consulter cette ressource.
  - L'API renvoi une représentation de la ressource avec le statut 200 *OK*.
- **Scénario alternatif 1:**
  - Le client applicatif fait un appel avec des paramètres incorrects.
  - L'API renvoi un statut 400 *BAD REQUEST* avec un message explicatif de la source d'erreur.
- **Scénario alternatif 2 :**
  - Le client applicatif fait un appel correct comme renseigné dans la documentation de l'API.
  - L'API cherche la ressource dans la base.
  - La ressource n'existe pas, l'API renvoi un statut 404 *NOT FOUND* avec des informations sur la source d'erreur.
- **Scénario alternatif 3 :**
  - Le client applicatif fait un appel correct comme renseigné dans la documentation de l'API.
  - L'API cherche la ressource dans la base.
  - La ressource existe et le client n'est pas autorisé de consulter la ressource, l'API envoi un statut 403 *FORBIDDEN* avec des informations sur la source d'erreur.
- **Scénario alternatif 4 :**
  - Le client applicatif fait un appel correct comme renseigné dans la documentation de l'API.
  - L'API cherche la ressource dans la base.
  - La ressource existe et le client est autorisé de consulter la ressource.

- Une erreur interne se produit, l'API renvoi le code 500 *INTERNAL SERVER ERROR* avec des informations sur la source d'erreur.

d) Cas d'utilisation : lister les ressources

Nom de cas d'utilisation	Lister les ressources
Acteur	Client applicatif
Description	Chercher une liste des ressources persistée dans la base des données
Pré-condition	Envoyer une requête HTTP à l'URI correspondant
Post-condition	Envoyer les ressources recherchées

Tableau 6 : le cas d'utilisation lister les ressources

- **Scénario normale :**
  - Le client applicatif envoi une requête HTTP à l'URI avec les bons paramètres.
  - L'API cherche les données des ressources dans la base et renvoi ses données avec le statut 200 *OK* en plus des liens HATEOAS.
- **Scénario alternatif 1:**
  - Le client applicatif fait un appel avec des paramètres incorrects.
  - L'API renvoi un statut 400 *BAD REQUEST* avec un message explicatif de la source d'erreur.
- **Scénario alternatif 2 :**
  - Le client applicatif fait un appel correct comme renseigné dans la documentation de l'API.
  - Le client applicatif n'est pas autorisée de consulter les ressources, l'API envoi un statut 403 *FORBIDDEN* avec un message explicatif de la source d'erreur.
- **Scénario alternatif 3 :**
  - Le client applicatif fait un appel correct comme renseigné dans la documentation de l'API.
  - Une erreur interne se produit, l'API renvoi le code 500*INTERNAL SERVER ERROR* avec un message explicatif de la source d'erreur.

e) Cas d'utilisation : supprimer une ressource

Nom de cas d'utilisation	Supprimer une ressource
Acteur	Client applicatif
Description	Supprimer une ressource persistée dans la base de données
Pré-condition	Envoyer une requête HTTP à l'URI convenable
Post-condition	Supprimer la ressource

Tableau 7 : le cas d'utilisation supprimer une ressource

- **Scénario normale :**
  - Le client applicatif envoi une requête HTTP à l'URI avec les bons paramètres.
  - L'API cherche la ressource dans la base de données.
  - La ressource existe et le client est autorisé de supprimer la ressource, l'API supprime les données de la base et renvoi un statut 204 *NO CONTENT*.
- **Scénario alternatif 1 :**
  - Le client fait un appel correct comme renseigné dans la documentation de l'API.

- L'API cherche la ressource dans la base de données.
- La ressource n'existe pas. L'API envoi un statut 404 *NOT FOUND* et un message d'erreur.

➢ **Scénario alternatif 2:**
- Le client applicatif fait un appel avec des paramètres incorrects.
- L'API renvoi un statut 400 *BAD REQUEST* avec un message explicatif de la source d'erreur.

➢ **Scénario alternatif 3 :**
- Le client applicatif fait un appel correct comme renseigné dans la documentation de l'API.
- L'API cherche la ressource dans la base de données.
- La ressource existe et le client n'est pas autorisé de supprimer la ressource, l'API renvoi le statut 403 *FORBIDDEN* avec un message qui contient plus d'information sur la source d'erreur.

➢ **Scénario alternatif 4 :**
- Le client applicatif fait un appel correct comme renseigné dans la documentation de l'API.
- L'API cherche la ressource dans la base de données.
- La ressource existe et le client est autorisé de supprimer la ressource.
- Une erreur interne se produit, l'API renvoi le code 500 *INTERNAL SERVER ERROR* un message qui contient plus d'information sur la source d'erreur.

2.2.4 Diagrammes de séquence

Les diagrammes de séquence permettent de décrire comment les éléments du système interagissent entre eux et avec les acteurs.

Les objets au cœur du système interagissent entre eux par l'échangedes messages.

Les acteurs interagissent avec le système au moyen des requêtes HTTP.

Ci-dessous on présente les diagrammes de séquences : ajouter une ressource, consulter une ressource, supprimer une ressource.

a) Diagramme de séquence : Ajouter une ressource

La figure suivante montre le diagramme de séquence correspond au cas d'utilisation ajouter une ressource :

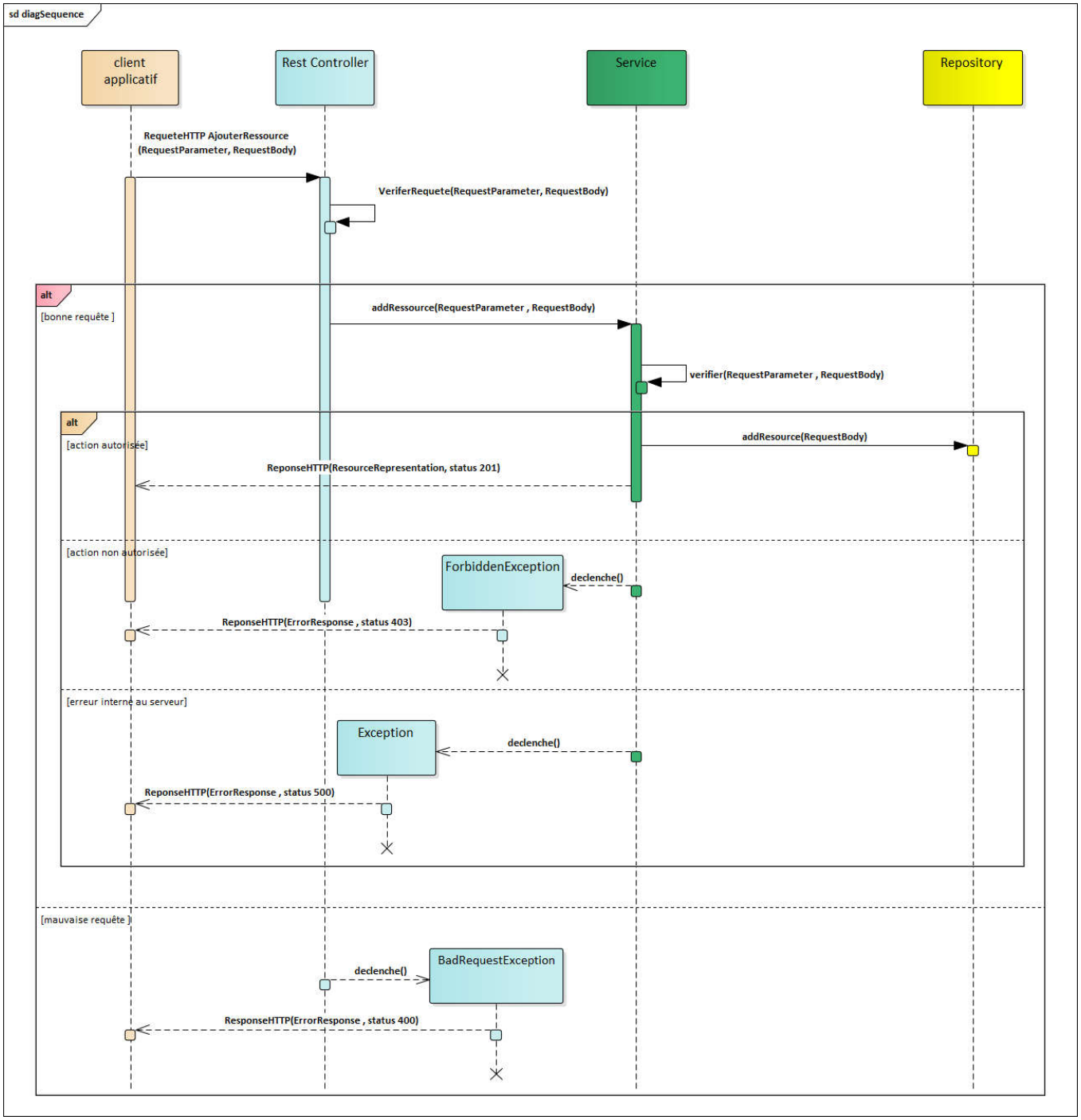

Figure 4 : diagramme de séquence de l'ajoute d'une ressource

Le client applicatif envoi une requête HTTP pour la création d'une ressource.

Si les paramètres et les données (*RequestBody*) sont valides, le contrôleur fait appel à la méthode «*addResource*» du Service, par la suite cette méthode fait appelle à une autre méthode qui s'appelle « *save*» de *Repository*, cette dernière se charge de la persistance de la ressource dans la base de données.

Ensuite la méthode de service envoi au contrôleur une représentation de la ressource qui contient, en outre des données de la ressource, des liens HATEOAS.

Le contrôleur envoi cette représentation avec un statut 201 au client applicatif.

Si il y a un paramètre manquant ou les données ne sont pas valides une exception se déclenche et l'API renvoi une réponse avec un code 400 *BAD REQUEST* et un objet Erreur qui contient plus d'informations sur la source d'erreur.

Si le client n'est pas autorisé d'ajouter une ressource à la base le service déclenche une exception et l'API renvoi une réponse HTTP avec un code 403 *FORBIDDEN* et un objet Erreur qui contient plus d'informations sur la source d'erreur.

Si l'exécution du code déclenche une erreur (problème technique notamment un *NULLPOINTER EXCEPTION* ou l'indisponibilité d'un composant intermédiaire), l'API renvoi un code 500 *INTERNAL SERVER ERROR* et un objet Erreur qui contient plus d'informations sur la source d'erreur.

b) Diagramme de séquence : consulter une ressource

La figure suivante montre le diagramme de séquence correspond au cas d'utilisation consulter une ressource.

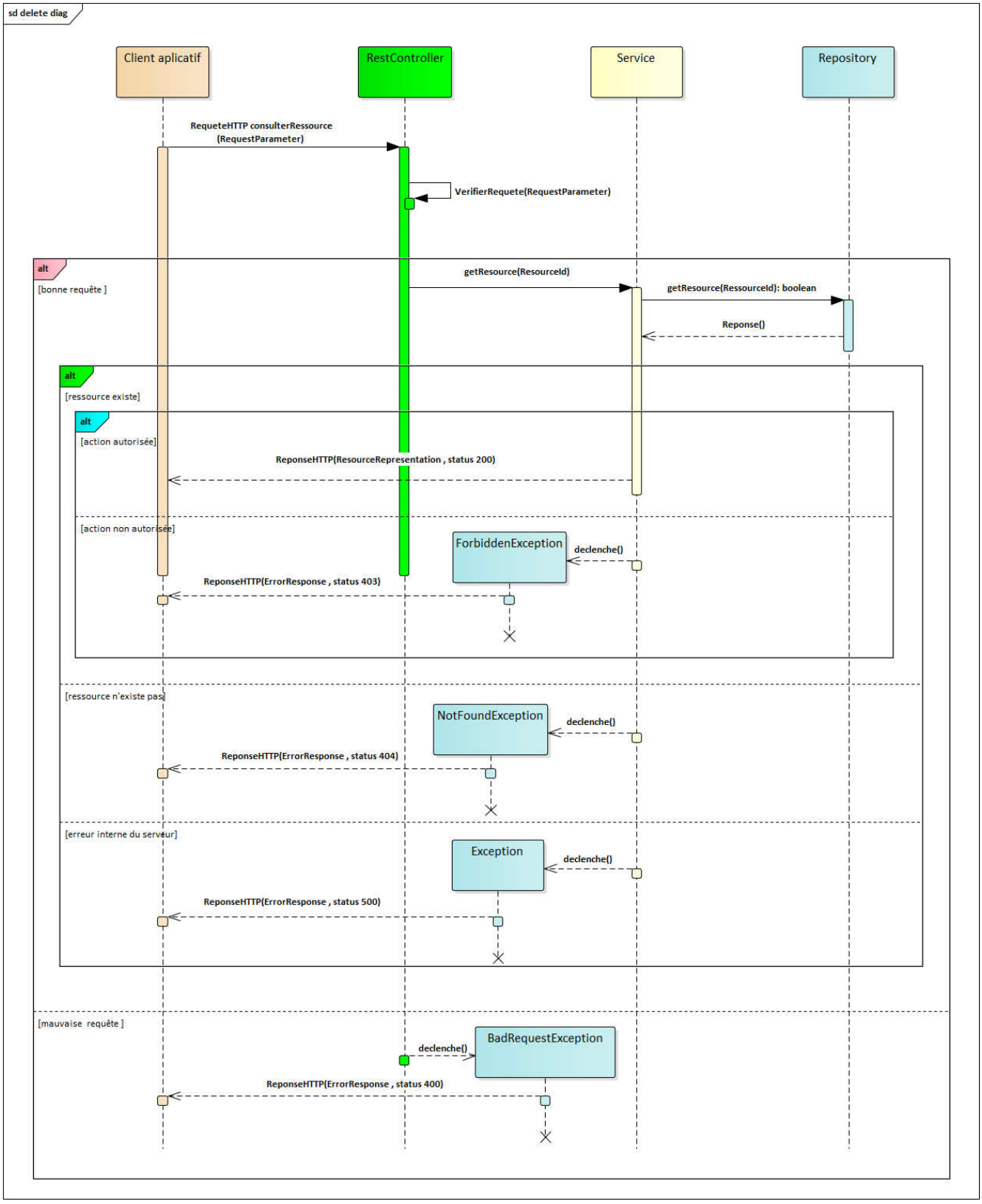

Figure 5 : diagramme de séquence de consultation d'une ressource par sonID

Le client applicatif envoi une requête HTTP pour consulter une ressource.

Le contrôleur vérifie la requête (les paramètres de la requête).

Si les paramètres sont valides, le contrôleur fait appel à la méthode «*getResource*» du Service, qui permet au premier temps de chercher la ressource dans la base.

La ressource existe dans la base, la méthode de service envoi au contrôleur une représentation de la ressource qui contient, en outre des données de la ressource, des liens HATEOAS.

Le contrôleur renvoi cette représentation avec un statut 200 *OK* au client HTTP.

Si la ressource n'existe pas le service déclenche une exception *NOT FOUND*, l'API envoi une réponse avec un statut 404 *NOT FOUND* et un message d'erreur qui contient plus d'informations sur la source d'erreur.

Si le client n'est pas autorisé de consulter la ressource, le service déclenche une exception et renvoi une réponse HTTP avec un code 403 *FORBIDDEN* et un objet Erreur qui contient plus d'informations sur la source d'erreur.

S'il y a un paramètre manquant le contrôleur déclenche une exception et renvoi une réponse avec un code 400 *BAD REQUEST* et un objet Erreur qui contient plus d'informations sur la source d'erreur.

Si l'exécution du code déclenche une erreur (problème technique notamment un *NULLPOINTER EXCEPTION* ou l'indisponibilité d'un composant intermédiaire), l'API renvoi un code 500 *INTERNAL SERVER ERROR* et un objet Erreur qui contient plus d'informations sur la source d'erreur.

a) Diagramme de séquence : Supprimer une ressource

La figure suivante montre le diagramme de séquence correspond au cas d'utilisation supprimer une ressource.

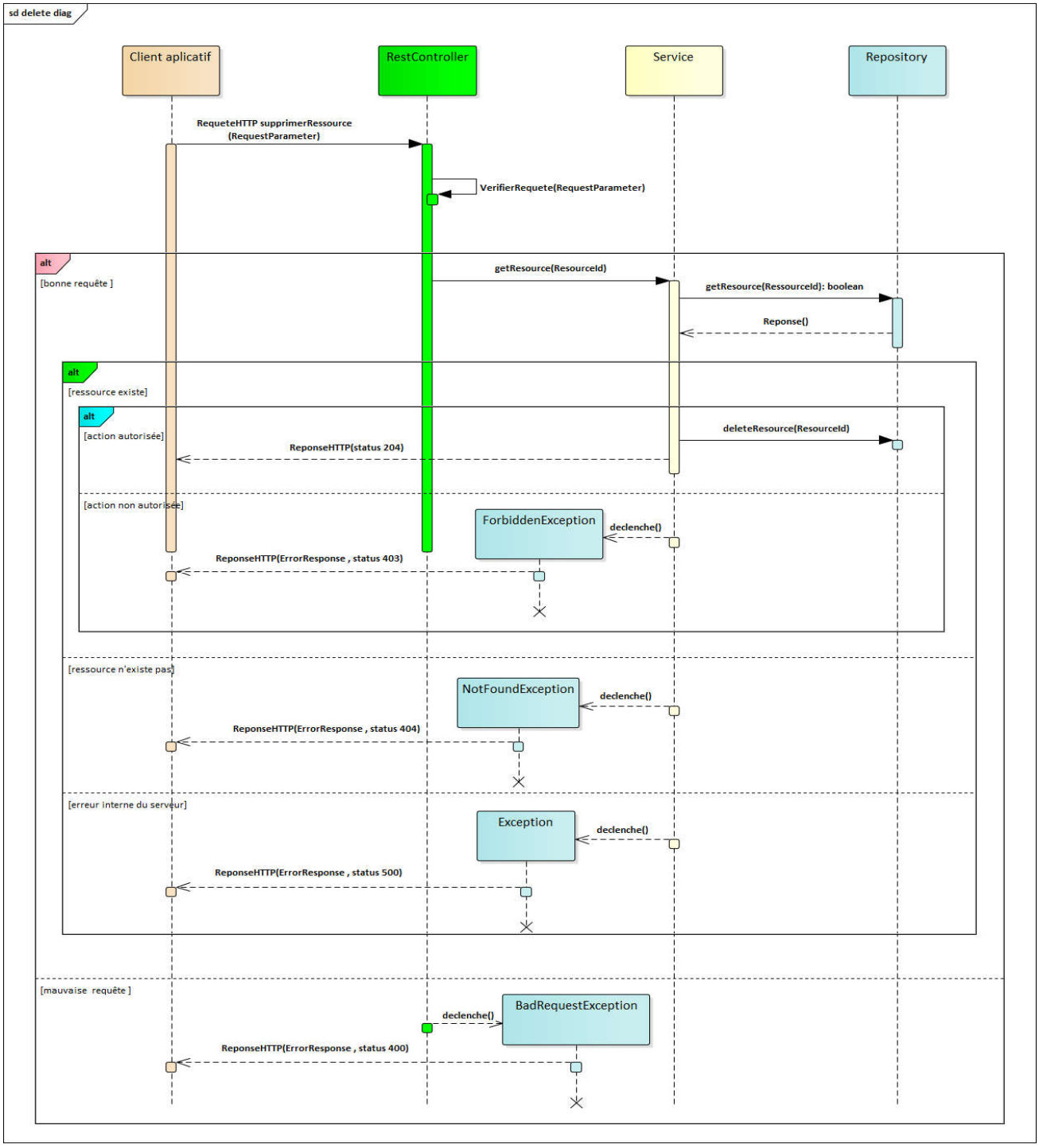

Figure 6 : diagramme de séquence de suppression d'une ressource

Pour supprimer une ressource l'API réalise les même étapes et tests que le cas de consultation, elle vérifie la requête HTTP, l'existence de la ressource, les droits du client puis elle supprime la ressource de la base et renvoi une réponse HTTP avec un statut 204 *NO CONTENT*, la réponse est éventuellement sans corps.

2.3 Model statique

2.3.1 Diagramme de classes

Le diagramme de classes permet de représenter les classes du système ainsi que leurs relations.

La figure suivante représente le diagramme de classe correspond à notre API :

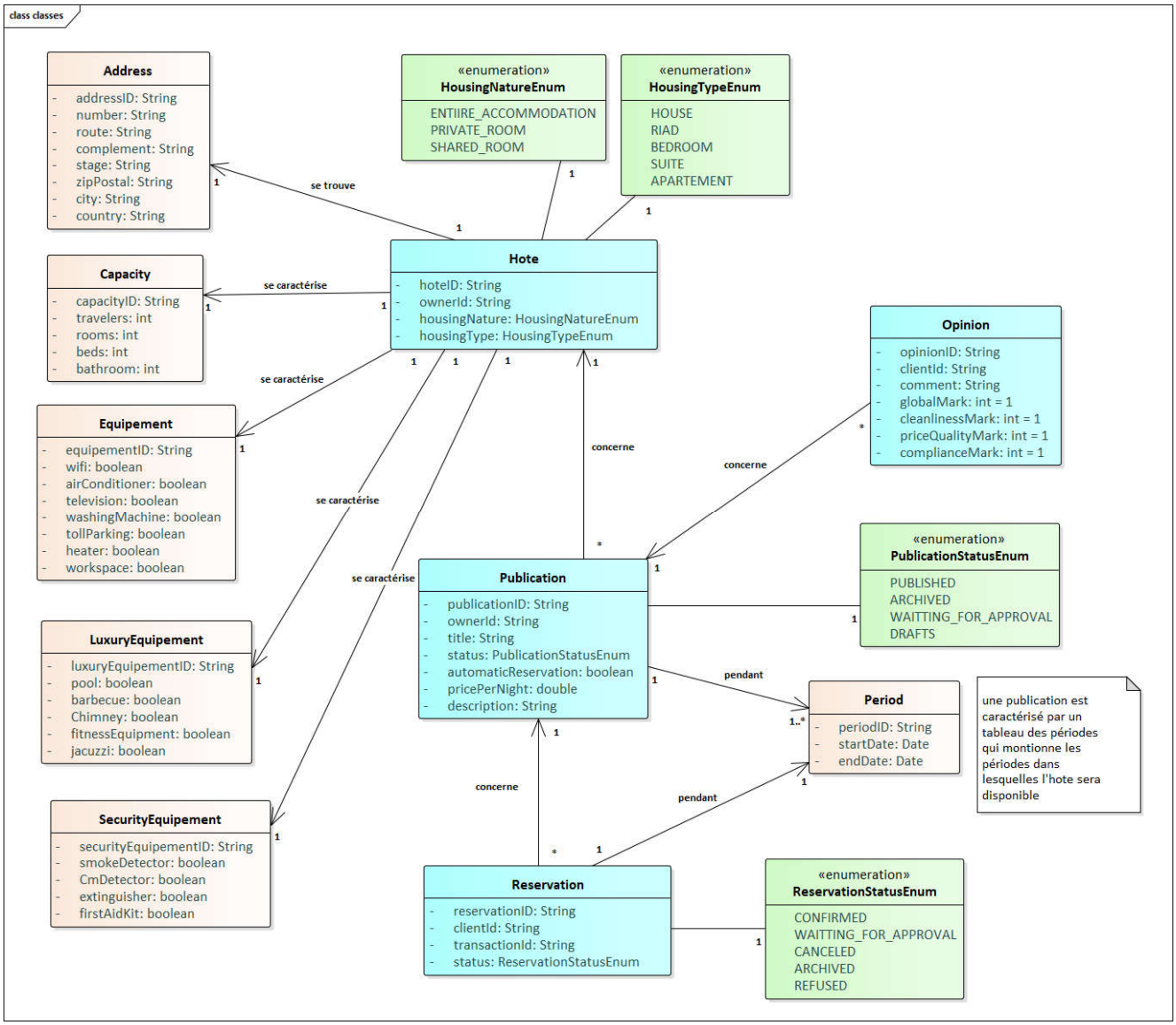

Figure 7 : diagramme de classes

L'API «*accommodationsApi*» gère quatre classes principales : Hote, Réservation, Publication, Opinion.

La classe « Hote » représente les biens touristiques qui vont être stocké dans la base de données, elle est constituée de plusieurs attributs qui représentent les caractéristiques de bien.

La classe « Publication » représente les annonces publiées dans la plateforme, elle est en relation avec la classe « Hote » car une publication concerne un bien déjà persisté dans la base de données.

La classe « Reservation » représente les réservations qui vont être effectué par les visiteurs de la plateforme à partir des annonces déjà publié.

La classe « Opinion » va présenter les avis publiés par les visiteurs de la plateforme concernant les annonces.

2.3.2 Schéma logique de la base de données

Le MLD (modèle logique de données) est la représentation des données du système, réalisé en vue d'une mise en œuvre au sein d'un système de gestion de base de donnes SGBD.

La figure suivante représente le MLD de l'API :

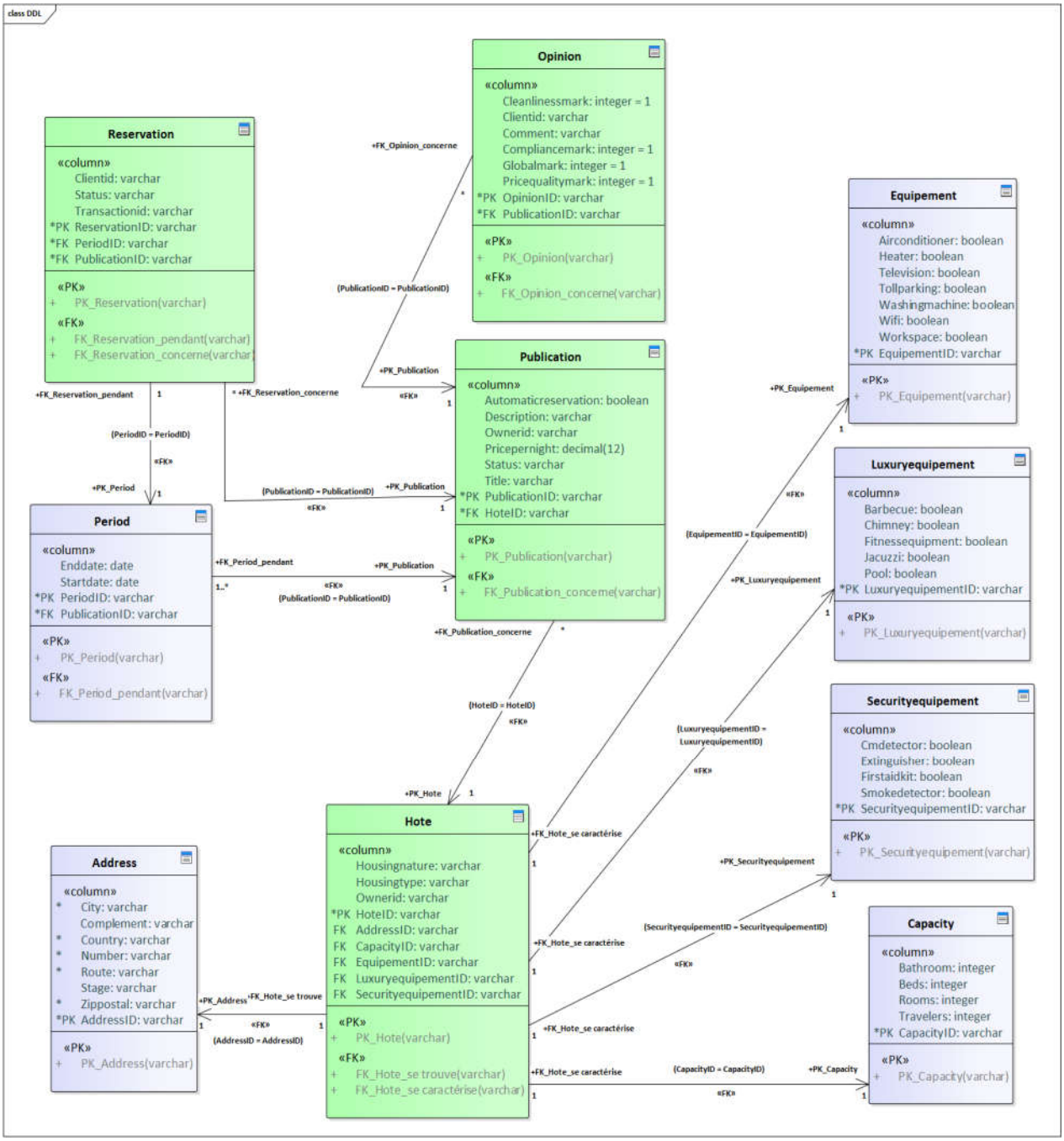

Figure 8 : modèle logique de données

Chapitre 3 : Réalisation

3.1 Architectures logicielles et le protocole HTTP

Pour mettre en place la plateforme digitale la société d'accueil a opté pour un ensemble des architectures jugées les plus convenables pour ce type des projets.

3.1.1 Architecture microservice

Les microservices[1] sont un style de développement logiciel qui vise la conception d'une application à partir d'un ensemble des modules (microservices). Chaque module est responsable d'accomplir un objectif métier spécifique en communiquant avec les autres modules pour assurer le fonctionnement cherché par l'application.

D'ici on peut définir un microservice comme une application qui ne fait qu'une seuletâche de manière optimale, car on se focalise plus sur le service rendu par le microservice sans se déranger du reste d'application.

Les principaux avantages de l'architecture microservices sont:

a) Réduction du temps de développement

Venant du faite que les microservices sont indépendants, les développeurs peuvent travailler et programmer plusieurs microservices simultanément.

b) Évolutivité accrue

Avec les microservices et leur indépendance en termes de développement et de déploiement, les développeurs peuvent mettre à jour un composant sans que cela n'ait d'incidence sur les autres composants de l'application.

Cela rendre l'évolution d'application rapide parce qu'on n'est pas amené à reconstruire l'application de zéro à chaque mise a jour.

c) Réduction des pannes

Les microservices étant indépendants, lorsqu'un élément tombe en panne ou rencontre un problème, l'ensemble de l'application ne cesse pas de fonctionner contrairement aux applications monolithiques. Il est également plus facile d'identifier et de résoudre une panne dans ce type d'écosystème.

d) L'adaptabilité de chaque microservice aux outils de travail

L'indépendance des composants offre la possibilité de développer les microservices avec des langages de programmationdifférents, tous en travaillant ensemble pour faire fonctionner la même application.

e) La flexibilité par rapport au cloud

Une entreprise qui utilise l'architecture en microservices peut réduire ou augmenter son usage du *cloud* en fonction de la charge de l'application, en prévoyant plus de stockage *cloud* pour les composants les plus demandés.

3.1.2 API REST

Une API [2](*application programming interface*) est un ensemble normalisé de classes, de méthodes, de fonctions qui sert de façade par laquelle un logiciel offre ses services à des autres logiciels. Une API est accompagnée d'une description qui spécifie comment des programmes consommateurs peuvent se servir des fonctionnalités du programme fournisseur.

REST[3](*REpresentational State Transfer*) est un type d'architecture d'API. Pour qu'une API soit qualifiable comme système *RESTful*, elle doit respecter les six règles suivantes :

a) La séparation entre client et serveur

Chaque côté peut être implémenté, modifiés indépendamment de l'autre. En continuant toujours à communiquer dans le même format. Dans une architecture REST, différents clients applicatifs envoient les mêmes requêtes, effectuent les mêmes actions et obtiennent les mêmes réponses.

b) L'absence d'état de sessions (stateless)

La communication entre client et serveur ne conserve pas l'état des sessions d'une requête à l'autre, ce que signifie quechaque requête envoyée par le client au serveur de ressources doit contenir les tous données nécessaire pour exécuter la commande, cette règle garantit une grande tolérance à l'échec. De plus, cela permet aux APIs REST de répondre aux requêtes de plusieurs clients différents sans saturer les ports du serveur. L'exception à cette règle est l'authentification, pour que le client n'ait pas à préciser ses informations d'authentification à chaque requête.

c) L'URI comme identifiant des ressources

REST se base sur les URI [4]**(Uniform Resource Identifier)** afin d'identifier une ressource. Ainsi une API REST doit définir ses URI d'une manière précise et uniforme.

Pour bien avoir la ressource demandée le client doit bien formuler la requête HTTP selon ce qui est déterminé dans le contrat d'interface [5] de l'API.

d) les verbes HTTP comme identifiant des opérations

L'architecture REST utilise les verbes HTTP existants plutôt que d'inclure l'opération dans l'URI de la ressource. Ainsi, généralement pour une ressource, il y aquatre opérations possibles (CRUD), HTTP propose les verbes correspondant :

- Créer (*Create*) ➔ **POST**
- Afficher (*Read*) ➔ **GET**
- Mettre à jour (*Update*) ➔ **PUT**
- Supprimer (*Delete*) ➔ **DELETE**

e) La mise en cache

La mise en cache des données fréquemment demandées pour réduire la consommation de la bande passante et accélérer les traitements et les chargements des réponses.

f) HATEOAS [6] (Hypermédia comme moteur d'état de l'application)

La réponse du serveur comprend les URI des méthodes supplémentaires auxquelles le client peut accéder à l'aide des données de réponse.

3.1.3 Architecture MVC

Modèle Vue Contrôleur (MVC) [7] est un patron de conception, qui découpe l'application en trois couches principales : les modèles, les vues et les contrôleurs.

- Un modèle (*Model*) représente les données qui vont être utilisées dans l'application.
- Une vue (*View*) représente la partie exposée au client.
- Un contrôleur (*Controller*) contient la logique métier de l'application.C'est aussi l'intermédiaire principal entre la vue et le modèle.

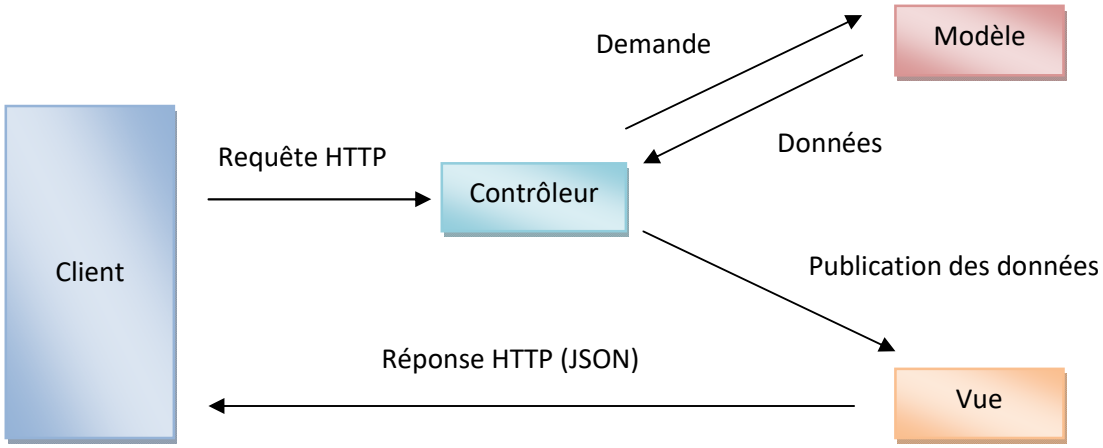

Figure 9 : le modèle MVC

L'API « *accommodationsApi* » est développée en respectant cette architecture, car l'utilisation du MVC donne une meilleure organisation du code et diminue la complexité lors de la conception grâce à la séparation des données de la vue et de contrôleur.

3.1.4 Objet de transfert de données : DTO

Un objet de transfert de données [8] (*data transfer object*) est un patron de conception utilisé dans les architectures logicielles objets.Son but est de simplifier les transferts de données entre les sous-systèmes d'une application logicielle.

Le DTO ne doit pas contenir de logique métier, mais il doituniquementpermettre de modifier ou d'accéder aux données (avec des mutateurs et accesseurs).

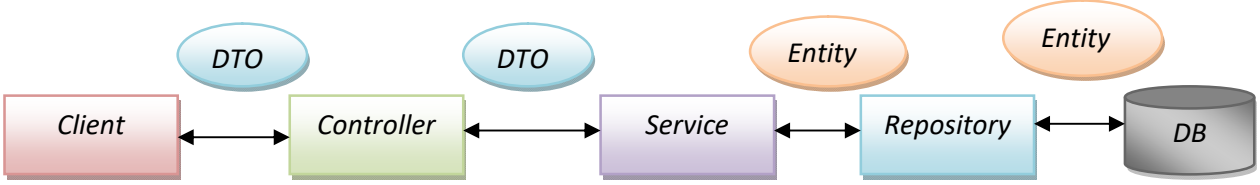

Figure 10 : patron de conception DTO

3.1.5 Protocole HTTP

HTTP [9] signifie « **Hypertext Transfer Protocol** ». Ce protocole a été développé par Tim Berners-Lee au CERN (Suisse) pour établir une connexion entre le client qui envoi des requêtes et le serveur qui réceptionne ces requêtes, les traites puis renvoi une réponse au client.

- **Requête http**

Les requêtes HTTP se composent des éléments suivants :

- Une méthode HTTP : ou un verbe HTTP tel que GET, POST qui définit l'opération que le client souhaite effectuer.
- Le chemin de la ressource à extraire.
- La version du protocole HTTP.
- Les en-têtes optionnels qui transmettent des informations supplémentaires.
- Un corps, pour certaines méthodes comme POST, qui contiennent la ressource envoyée.

- **Réponse http**

Les réponses HTTP se composent des éléments suivants :

- La version du protocole HTTP qu'elle suit.
- Un code de statut, qui indique si la requête a réussi ou non.
- Un message de statut qui est une description rapide, informelle, du code de statut.
- Les en-têtes HTTP, comme pour les requêtes.
- Éventuellement un corps contenant la ressource récupérée.

- **Les codes de réponse**

Ce sont les codes que vous voyez lorsque le navigateur n'arrive pas à vous fournir la page demandée. Le code de réponse est constitué de trois chiffres : le premier indique la classe de statut et les suivants la nature exacte de l'erreur.

Code	Message
1xx	Message d'information
2xx	Réussite
3xx	Redirection
4xx	Erreur due au client
5xx	Erreur due au serveur

Tableau 8 : les codes de réponses HTTP

3.2 Environnement de développement

3.2.1 Environnement matériel

Ce projet est réalisé sous un ordinateur portable Delldes caractéristiques suivantes :

- Système d'exploitation : Microsoft Windows 10 Professionnel
- Processeur : Intel(R) Core(TM) i5-4310U CPU @ 2.00GHz 2.60 GHz
- RAM : 4 GO

3.2.2 Les outils de développement

Pour mettre en ouvre cette API on a utiliséJAVA comme langage de programmation, en outre d'un ensemble des outils qui facilitent la conception et le développement d'un API REST. Parmi ces outils on cite :

Figure 11 : logo Enterprise Architect

Enterprise Architect[10] est un logiciel de modélisation et de conception UML, édité par la société australienne Sparc Systems. Couvrant, par ses fonctionnalités, l'ensemble des étapes du cycle de conception d'application, il est l'un des logiciels de conception et de modélisation les plus reconnus.

Figure 12 : logo STS

STS[11](*spring tool suite*) est un IDE JAVA conçu pour développer des applications d'entreprise basées sur Spring, c'est plus facile, plus rapide et plus pratique. Et surtout, il est basé sur ECLIPCE IDE. STS est gratuit, open source et optimisé par VMware.

Figure 13 :logo spring boot

Spring boot [12]C'est un composant très particulier de Spring Framework. Il est au service des autres composants**,**car il nous permet de mettre en œuvre tous les autres. Ce composant a été créé pour la**s**implification du développement de nos projets avec Spring**.** Les principaux avantages de spring boot :

- La gestion des dépendances est simplifiée grâce aux **starters**qui regroupent plusieurs dépendances et homogénéisent les versions**.**
- L'auto configuration permet de se concentrer sur le code métier, etsimplifie énormément la mise en œuvre des composants Spring qui sont utilisés.
- Le déploiement de l'application est facilité par la génération d'un JAR, et pour les projets web, un tomcat est embarqué**.**

Figure 14 : logo spring

Spring Web[15] est le module de spring consacré au développement d'application web et d'API. C'est lui qui va permettre d'exposer l'API comme API REST.

Figure 15 : logo spring data

Spring Data [13] : fourni un modèle de programmation familier et cohérent basé sur Spring pour l'accès aux données. Il facilite l'utilisation des technologies d'accès aux données, des bases de données relationnelles et non relationnelles.

Dans ce projet on a utilisé **Spring Data JPA**[14] qui fait partie de la grande famille Spring Data.

La mise en œuvre d'une couche d'accès aux données d'une application a été fastidieuse pendant un certain temps. Trop de code passe partout doit être écrit pour exécuter des requêtes simples ainsi que pour effectuer la pagination et l'audit. Pour gérer les données il suffit d'écrire interfaces de référentiel, y compris les méthodes de recherche personnalisées, et Spring fournira automatiquement l'implémentation.

Figure 16 : logo maven

Apache Maven [16]: est un outil de gestion et d'automatisation de production des projets logiciels Java en général et Java EE en particulier. Il est utilisé pour automatiser l'intégration continue lors d'un développement de logiciel. Maven est géré par l'organisation Apache Software Foundation.

Figure 17 : logo postgreSQL

PostgreSQL[17]est un système de gestion de base de données relationnelle et objet. C'est un outil libre disponible selon les termes d'une licence de type BSD. Il se dispose d'interface graphique, des bibliothèques facilitant l'accèsa partir des programmes écrit en (JAVA, C, C++, etc.) et d'une API ODBC permettant a n'import quelle application supportant ce type d'interface d'accéder a des bases de données de type PostgreSQL.

Figure 18 : logo git

Git[18]est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. Le principal contributeur actuel de git et depuis plus de 16 ans est Junio C Hamano.

Figure 19 : logo postman

Postman[19] est une application permettant de tester des API, créée en 2012 par AbhinavAsthana, AnkitSobti et Abhijit Kane à Bangalore pour répondre à une problématique de test d'API partageable. Il permet de créer des requêtes HTTP simples ou complexe, ainsi que les enregistrer et d'analyser les réponses envoyées par l'API.

Figure 20 : logo Trello

Trello[20]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, très utilisé dans la gestion de projets agiles. Il se présente de la manière suivante :des tableaux, des listes et des cartes.

Figure 21 : logo JSON

JavaScript Object Notation[21] (**JSON**) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée, ce format se génère et s'analyse facilement. Pour les humains, il est pratique à écrire et à lire grâce à une syntaxe simple et à une structure en arborescence, il est principalement utilisé pour transférer des données entre un serveur et un client. JSON accepte les valeurs suivantes : objet, Array, Nombre, chaîne de caractères, true, false, null.

3.3 Présentation de l'API « *accommodationsApi* »

L'API « *accommodationsApi* » est conçu en respectant le contrat d'interface fournis par la société, cette contrat décrire quelle est la responsabilité d'une API et comment la consommée.

3.3.1 L'architecture de l'API« *accommodationsApi* »

Le schéma suivant représente la structure de l'api « *accommodationsApi* »:

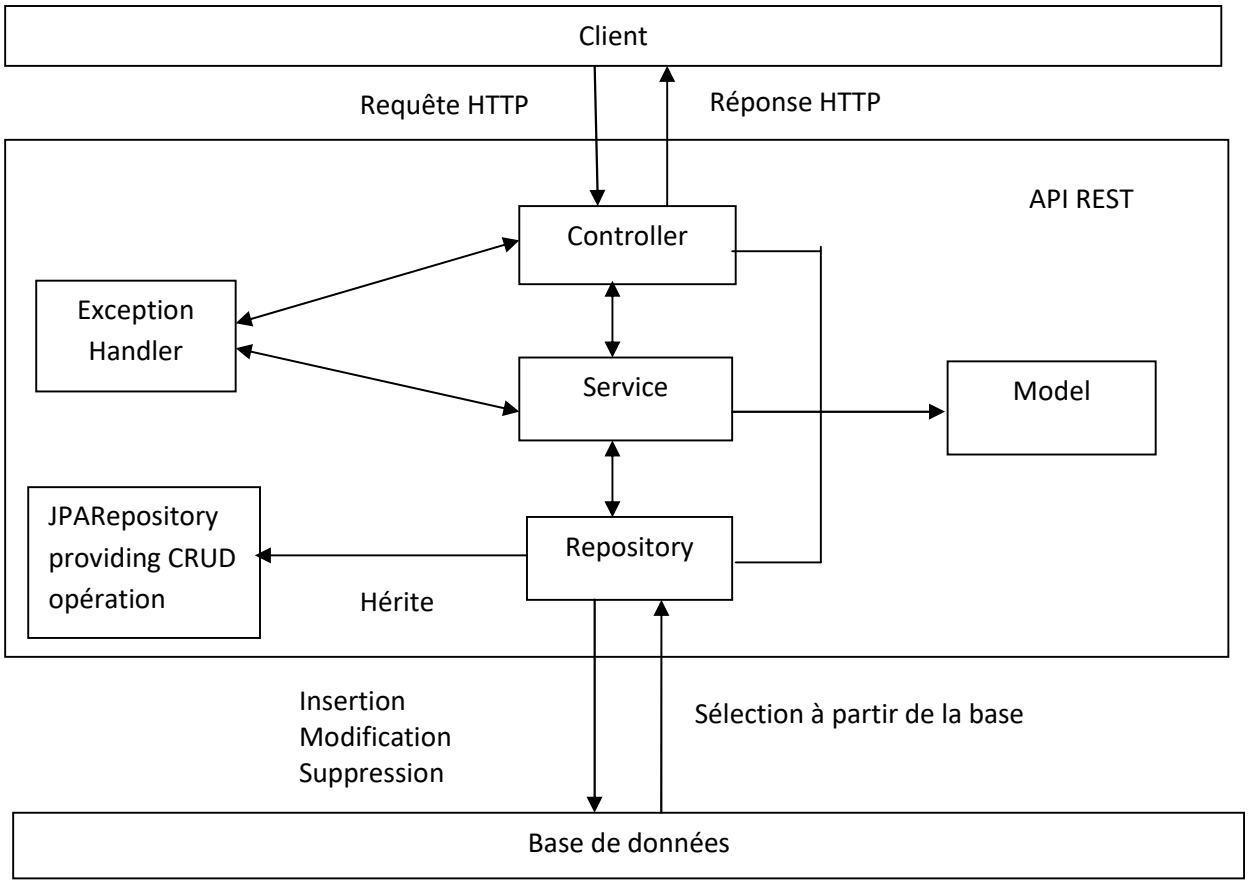

Figure 22 : l'architecture de l'API

L'API est constituée des éléments suivants :

- *Controller* : réceptionner la requête et fournir la réponse.
- *Service* : exécuter les services métiers.
- *Repository* : communiquer avec la base de données.
- *Model* : contenir les objets métiers.
- *Exception Handler* :gérer les erreurs client et serveur.
- *JPARepository* [22]: interface qui facilite la gestion des données.

La requête est réceptionnée, la couche *Controller*délègue les traitements métieràexécuter, la couche *Service* pourra ensuite faire appel à la couche *Repository*.

3.3.2 Tester le fonctionnement de l'API

La partie FrontEnd de la plateforme n'est pas encore développée, alors on ne se dispose pas des interfaces homes machines pour visualiser le fonctionnement de l'API, pour cela on utilise l'outil POSTMAN pour tester le fonctionnement de l'API.

L'API expose 20 Endpoints, elle gère 4 ressources, pour chaque ressource elle expose 5 Endpoints.

Le tableau suivant représente les différentes routes de l'API « *accommodationsApi* » :

Méthode HTTP	Route	Description	Type d'opération
POST	**/accommodations/hotes**	**Ajouter un hôte à la base de données**	**Ecriture**
PUT	**/accommodations/hotes/ {hote_id}**	**Modifier les informations d'hôte d'ID=hote_id**	**Ecriture**
GET	**/accommodations/hotes/{hote_id}**	**Récupérer les informations d'hôte d'ID=hote_id**	**Lecture seule**
GET	**/accommodations/hotes**	**Récupérer la liste des hôtes**	**Lecture seule**
DELETE	**/accommodations/hotes/{hote_id}**	**Supprimer l'hôte d'ID=hote_id**	**Ecriture**
POST	**/accommodations/publications**	**Ajouter une publication d'un hôte à la base de données**	**Ecriture**
PUT	**/accommodations/publications/ {publication_id}**	**Modifier les informations de la publication d'ID = publication_id**	**Ecriture**
GET	**/accommodations/publications/ {publication_id}**	**Récupérer les informations de la publication d'ID = publication_id**	**Lecture seule**
GET	**/accommodations/publications**	**Récupérer la liste des publications**	**Lecture seule**
DELETE	**/accommodations/publications/ {publication_id}**	**Supprimer la publication d'ID = publication_id**	**Ecriture**
POST	**/accommodations/publications/ {publication_id}/reservations**	**Ajouter une réservation a partir d'une publicationà la base de données**	**Ecriture**
PUT	**/accommodations/publications/ {publication_id}/reservations/ {reservation_id}**	**Modifier les informations de la réservation d'ID = reservation_id**	**Ecriture**
GET	**/accommodations/publications/ {publication_id}/reservations/ {reservation_id}**	**Récupérer les informations de la réservation d'ID = reservation_id**	**Lecture seule**
GET	**/accommodations/publications/ {publication_id}/reservations**	**Récupérer la liste des réservations**	**Lecture seule**
DELETE	**/accommodations/publications/ {publication_id}/reservations/ {reservation_id}**	**Supprimer la réservation d'ID = reservation_id**	**Ecriture**
POST	**/accommodations/opinions**	**Ajouter une opinion sur une publicationà la base de données**	**Ecriture**
PUT	**/accommodations/opinions/ {opinion_id}**	**Modifier les informations de l'opinion d'ID = opinion_id**	**Ecriture**
GET	**/accommodations/opinions/ {opinion_id}**	**Récupérer les informations de l'opinion d'ID = opinion_id**	**Lecture seule**

GET	/accommodations/opinions	Récupérer la liste des opinions	Lecture seule
DELETE	/accommodations/opinions/ {opinion_id}	Supprimer l'opinion d'ID = opinion_id	Ecriture

Tableau 9 : les Endpoints de l'API « accommocationsApi »

Dans ce qui suit, on va présenter quelque Endpoints avec les différentes réponses possibles.

- **L'Endpoint addHote**

Un client veut ajouter un hôte à la base de données pour cela il envoi une requête HTTP pour accomplir cette tache.

La requête envoyée avec un verbe HTTP POST en vue de persister les informations dans la base de données.On envoi la requête suivant à l'Endpoint *addHote* :

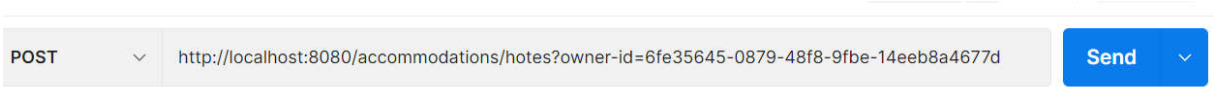

Figure 23 : la requête HTTP pour ajouter un bien a la base de données

Les données à persister sont envoyées comme un objet JSON qui va être convertit à un objet JAVA puis persisté dans la base de données grâce à Spring Data. Dans le corps de la requête on envoie l'objet JSON suivant :

```
{
    "ownerId":"6fe35645-0879-48f8-9fbe-14eeb8a4677d",
    "housingType": "RIAD",
    "housingNature": "PRIVATE_ROOM",
    "address": {
        "number": "4",
        "route": "Derb Matrouh",
        "complement": "Foundouk Lihoudi",
        "stage": "Fes El Bali",
        "zipPostal": "30110",
        "city": "Fès",
        "country": "MA"
    },
    "capacity": {
        "travelers": 50,
        "beds": 10,
        "rooms":10,
        "bathroom": 10
    },
    "equipements": {
        "wifi": true,
        "television": true,
        "kitchen": true,
        "washingMachine": false,
        "freeParking": true,
        "tollParking": true,
        "airConditioner": true,
        "heater": true,
        "workspace": false
    },
    "luxuryEquipments": {
        "pool": true,
        "jacuzzi": false,
```

```
        "barbecue": false,
        "Chimney": true,
        "fitnessEquipment": false
    },
        "securityEquipments": {
            "smokeDetector": true,
            "CmDetector": true,
            "extinguisher": false,
            "firstAidKit": true
    }
}
```


Figure 24 : l'objet JSON qui représente les données à persister dans la base

L'API traite la requête ; le client a bien renseigné tous les paramètres et respecte les conditions de gestion, alors l'API renvoi un code 201 *CREATED* pour mentionner que l'objet est bien créé et persisté dans la base de données.

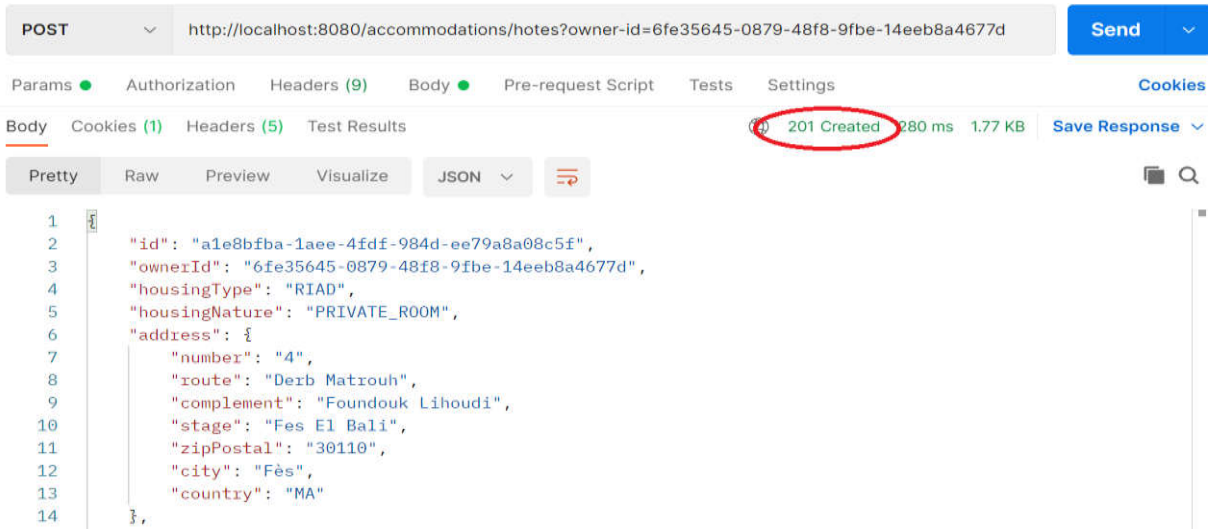

Figure 25 : réponse HTTP à la requête d'ajout

La figure suivante représente le corps de réponse détaillé :


```
{
    "id": "8ce8cc9c-5dc9-4a84-9ee2-4f8ca0251006",
    "ownerId": "6fe35645-0879-48f8-9fbe-14eeb8a4677d",
    "housingType": "RIAD",
    "housingNature": "PRIVATE_ROOM",
    "address": {
        "number": "4",
        "route": "Derb Matrouh",
        "complement": "Foundouk Lihoudi",
        "stage": "Fes El Bali",
        "zipPostal": "30110",
        "city": "Fès",
        "country": "MA"
    },
    "capacity": {
        "travelers": 50,
        "beds": 10,
        "rooms": 10,
        "bathroom": 10
    },
    "equipements": {
        "wifi": true,
```

```
        "television": true,
        "kitchen": true,
        "washingMachine": false,
        "freeParking": true,
        "tollParking": true,
        "airConditioner": true,
        "heater": true,
        "workspace": false
    },
    "luxuryEquipments": {
        "pool": true,
        "jacuzzi": false,
        "barbecue": false,
        "Chimney": true,
        "fitnessEquipment": false
    },
    "securityEquipments": {
        "smokeDetector": true,
        "CmDetector": true,
        "extinguisher": false,
        "firstAidKit": true
    },
    "links": [
        {
            "href": "/accommodations/hotes",
            "rel": "self",
            "title": "link to self",
            "mediaType": "application/json",
            "method": "POST",
            "encType": "application/json"
        },
        {
            "href": "/accommodations/hotes",
            "rel": "hotes",
            "title": "link to list of hotes",
            "mediaType": "application/json",
            "method": "GET",
            "encType": "application/json"
        },
        {
            "href": "/accommodations/hotes/8ce8cc9c-5dc9-4a84-9ee2-4f8ca0251006",
            "rel": "update hote",
            "title": "link to update the hote",
            "mediaType": "application/json",
            "method": "PUT",
            "encType": "application/json"
        },
        {
            "href": "/accommodations/hotes/8ce8cc9c-5dc9-4a84-9ee2-4f8ca0251006",
            "rel": "delete hote",
            "title": "link to delete the hote",
            "mediaType": "application/json",
            "method": "DELETE",
            "encType": "application/json"
        },
        {
            "href": "/accommodations/hotes/8ce8cc9c-5dc9-4a84-9ee2-4f8ca0251006",
            "rel": "hote",
            "title": "link to the hote",
            "mediaType": "application/json",
            "method": "GET",
            "encType": "application/json"
        }
    ]
}
```


Figure 26 : réponse HTTP pour la requête d'ajoute d'un hôte

En outre de la ressource persistée, l'API renvoi un objet Links qui contient des liens vers des prochaines étapes potentielles, ces liens facilitent la navigation du client dans l'application en renvoyant des liens supplémentaire avec la réponse.Par exemple dans la requête précédente l'API renvoi des liens pour modifier, supprimer l'objet récemment persisté et un lien vers une collection des hôtes de même propriétaire.

Si le client envoi une mauvaise requête l'API renvoi un code 400 *BAD REQUEST* avec un objet Erreur qui contient plus d'informations sur la source d'erreur ; ici le client n'a pas mentionné le paramètre « *owner-id* ».

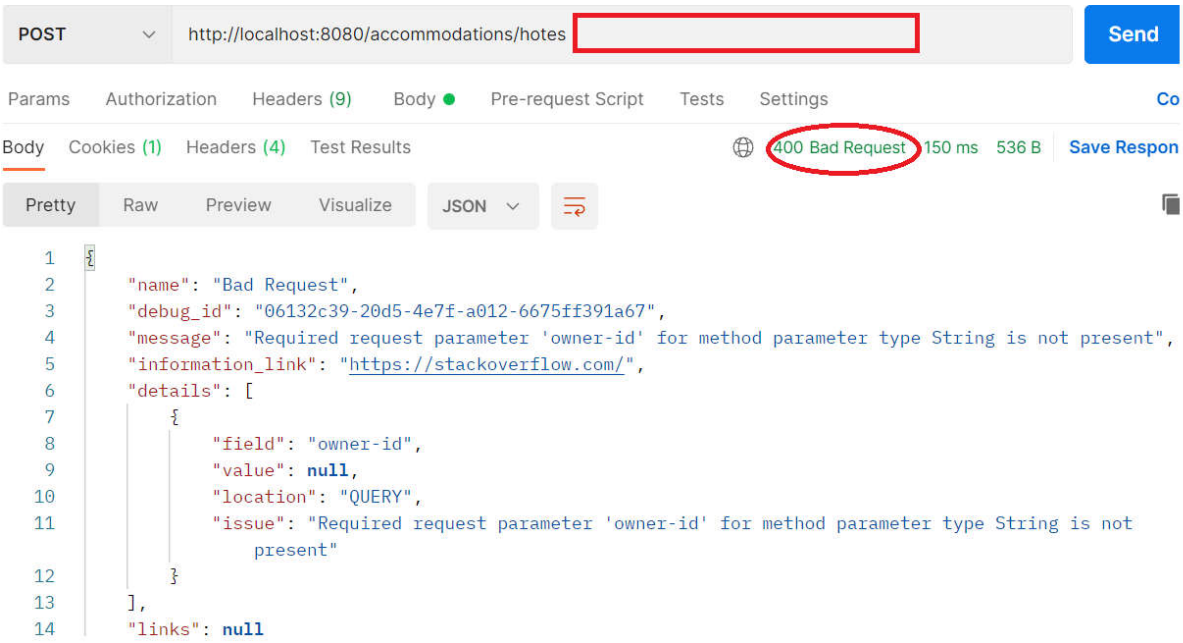

Figure 27 :réponse HTTP Bad Request

Si le client essai d'effectuer une opération interdite l'API renvoi un statut 403 avec un objet JSON qui contient plus de détails sur la source d'erreur.

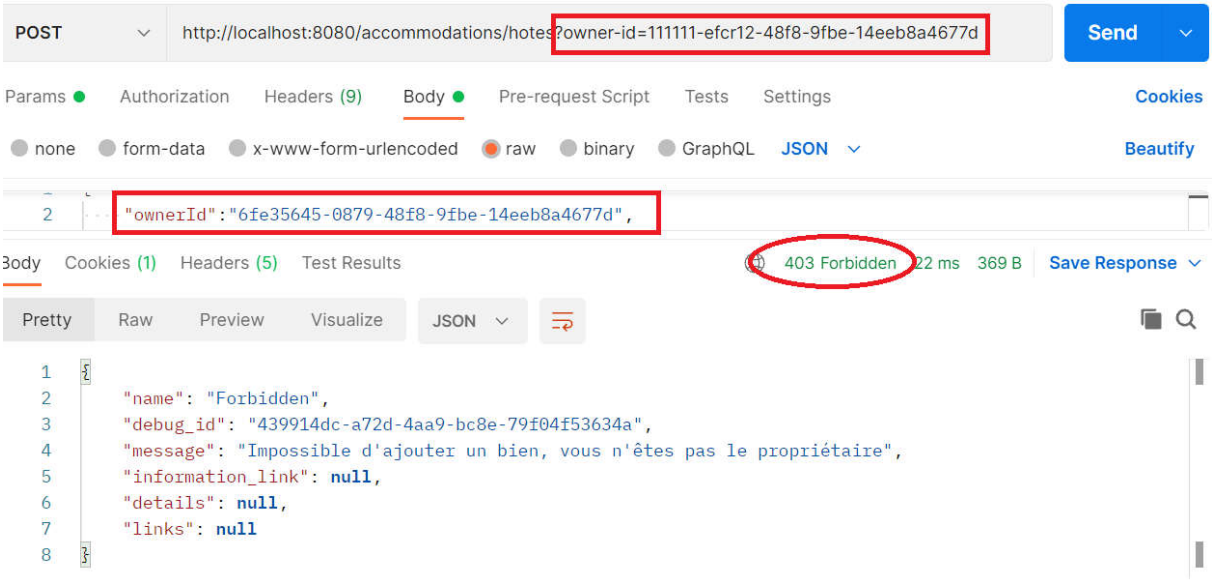

Figure 28 : réponse HTTP à une action interdite

➢ Endpoint listPublications

Cet Endpoint va permettre de retourner au client applicatif une liste des annonces qui vérifient les conditions envoyés par le client comme des paramètres de la requête.

Le client peut aussi spécifie le nombre des éléments à renvoyer en renseignant le paramètre « *size* » ainsi que l'offset de la page à renvoyer. C'est ce qu'on appel la pagination [23], elle est utilisée si une requête renvoi une grande quantité de données, elle permet l'organisation de ces données dans des pages qui s'identifient par un offset.

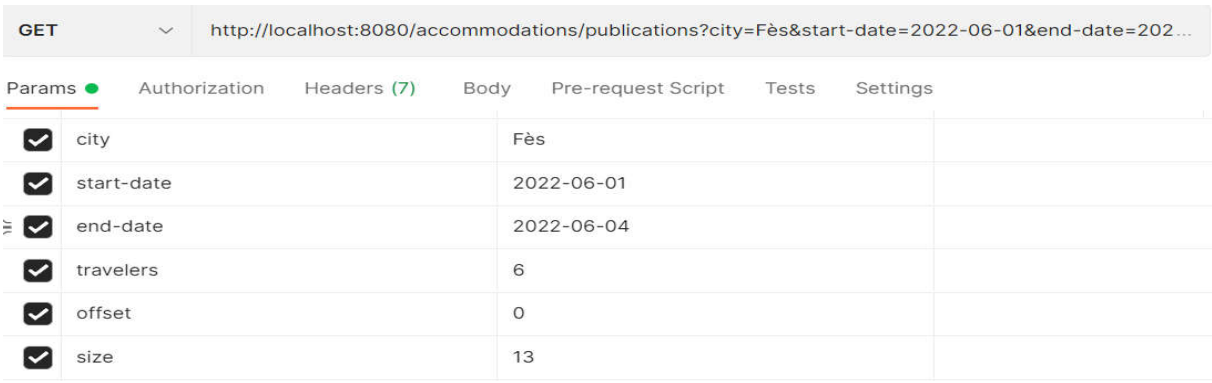

Figure 29 : requête pour lister les annonces

On envoi cette requête, on obtient la réponse suivante :

Figure 30 : réponse de la requête : listPublication

L'API renvoi une réponse HTTP avec le statut 200 OK, et un objet JSON qui est constitué de :

- nombre totale des éléments qui répond aux critères cités par le client.
- nombre total des pages de la liste.
- objet links contient une liste des liens HATEOAS vers les éléments de la liste qui vérifient les critères cherchés par le client.

Figure 31 : les liens HATEOAS vers les éléments de la liste des annonces

Conclusion et perspectives

Mon projet de fin d'étude a consisté en la mise en œuvre d'une API REST. Cette API fait partie d'un ensemble des APIs qui forment la partie BackEnd d'une plateforme digitale pour le secteur touristique.

L'API expose un certains nombre d'Endpoints pour permettre aux applications FrontEnd la mise en place des services de gestion des hébergements des hôtes touristiques.

Ce stage de fin d'étude, effectué au sein de la société NAWRA TECHNOLOGY, m'a permis de mettre en pratique les connaissances théoriques et techniques acquises au cours de ma formation. Ainsi que s'auto-former sur des nouvelles outils et technologies. Il m'a permis d'intégrer une équipe de développement et de se familiariser avec les outils de travail en groupe notamment le gestionnaire des versions Git et le gestionnaire des projets en ligne Trello.

Ce projet m'a permis également de découvrir l'écosystème Spring et ses différents modules notamment Spring Boot, Spring web, Spring Data et découvrir des nouvelles architectures logicielles tels que les microservices et le REST.

Pour conclure, ce projet a été très enrichissant. Néanmoins, ce projet pourrait être amélioré en ajoutant d'autres fonctionnalités. En particulier la partie des tests et l'amélioration des réponses d'erreur pour fournir plus d'informations sur la source d'erreur, ainsi que la mise en place des interfaces homes machine pour s'en profiter des services offerts par l'API.

Références

Webographie

[1] « Définition des microservices » https://www.talend.com/fr/resources/guide-microservices/ : 2022-06-20

[2] « Une API, qu'est-ce que c'est ? » https://www.redhat.com/fr/topics/api/what-are-application-programming-interfaces : 2022-06-20

[3] « Une API REST, qu'est-ce que c'est» https://www.redhat.com/fr/topics/api/what-is-a-rest-api : 2022-06-20

[4] « URI : ce qu'il faut savoir » https://www.ionos.fr/digitalguide/sites-internet/developpement-web/le-uniform-resource-identifier/: 2022-06-20

[5] « C'est quoi un contrat d'interface ?» https://definir-tech.com/info/2736/c-est-quoi-un-contrat-d-interface: 2022-06-20

[6] « Spring hateoas» https://spring.io/projects/spring-hateoas : 2022-06-24

[7] « MVC »https://practicalprogramming.fr/mvc/: 2022-06-25

[8] « DTO » https://stackabuse.com/data-transfer-object-pattern-in-java-implementation-and-mapping/: 2022-06-25

[9] « Le protocole HTTP » https://www.commentcamarche.net/contents/520-le-protocole-http : 2022-06-25

[10] « EnterpriseArchitect»https://fr.wikipedia.org/wiki/Enterprise_Architect 2022-06-17

[11] « STS »https://spring.io/tools : 2022-06-21

[12] « Spring Boot » https://spring.io/projects/spring-boot : 2022-06-24

[13] « Spring Data» https://spring.io/projects/spring-data : 2022-06-24

[14] « Spring Data jpa» https://spring.io/projects/spring-data-jpa: 2022-06-24

[15] « Spring Web » https://gayerie.dev/docs/spring/spring/spring_mvc_intro.html: 2022-06-24

[16] « Maven » https://www.jmdoudoux.fr/java/dej/chap-maven.htm : 2022-06-24

[17] « definition-postgresql » https://www.oracle.com/fr/database/definition-postgresql.html : 2022-06-24

[18] « git » https://git-scm.com/doc : 2022-06-24

[19] « utilisez postman pour formuler vos requetes » https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/7498761-utilisez-postman-pour-formuler-vos-requetes : 2022-06-24

[20] « trello » https://www.blogdumoderateur.com/tools/trello/ : 2022-06-24

[21] « définition de JSON » https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1445308-json-definition-et-presentation-de-ce-format-de-donnees/ :

2022-06-24

[22] « Spring Boot: jparepository » https://www.geeksforgeeks.org/spring-boot-jparepository-with-example/ : 2022-06-24

[23] « Spring data jpa pagination sorting » https://www.baeldung.com/spring-data-jpa-pagination-sorting : 2022-06-25