**Documentation du Projet**

**Technologies Utilisées**

1. **Frontend**:
   * **React**: Une bibliothèque JavaScript pour construire des interfaces utilisateur interactives.
   * **Vite**: Un outil de build rapide pour les applications web basées sur JavaScript et TypeScript.
   * **Figma**: Une plateforme de design collaborative pour créer des prototypes d'interface utilisateur et des maquettes graphiques.
2. **Backend**:
   * **Spring Boot**: Un framework Java pour développer des applications web et des services RESTful rapidement et facilement.
   * **MongoDB**: Une base de données NoSQL orientée documents pour stocker les données de manière flexible.
3. **Base de Données**:
   * **MongoDB**: Une base de données NoSQL très évolutive et flexible, utilisée pour stocker les données de l'application.

**Modules Principaux**

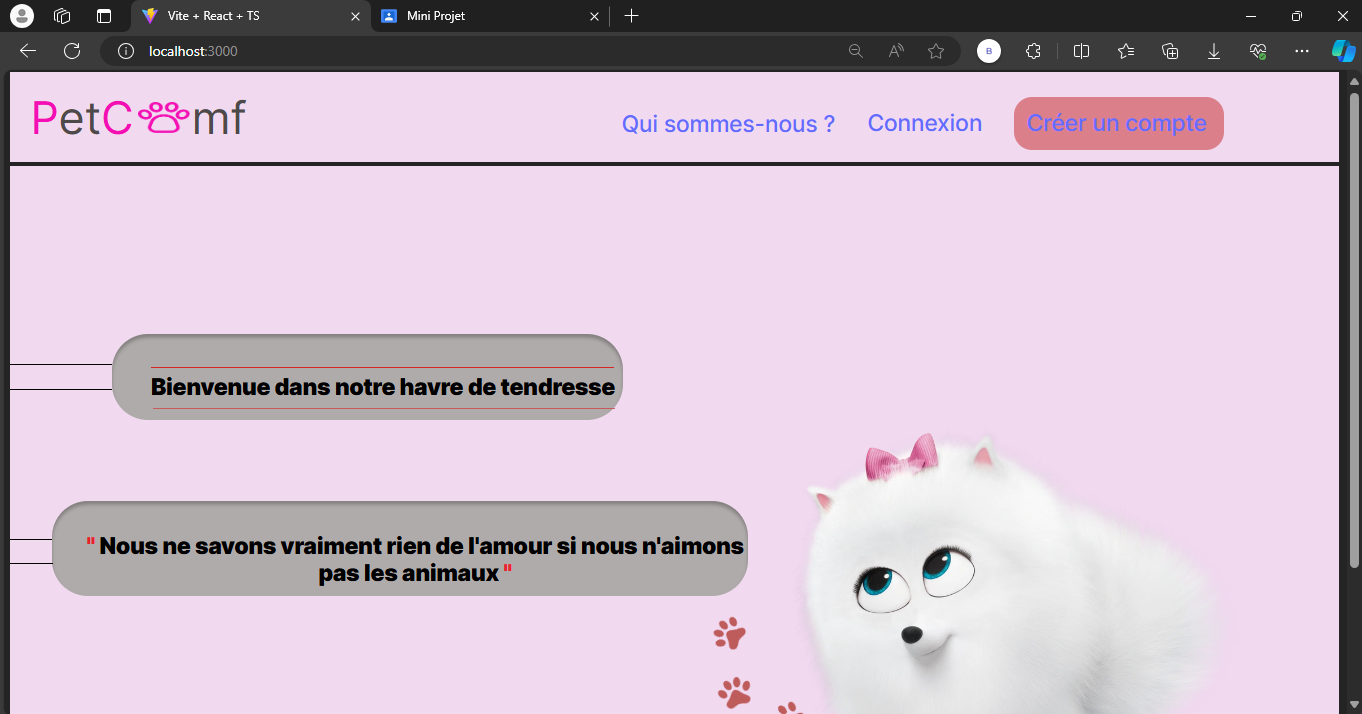
1. **Frontend**:
   * **React Components**: Comprend tous les composants réutilisables pour construire l'interface utilisateur de l'application, tels que les boutons, les formulaires, les cartes, etc.
   * **API Services**: Module pour gérer les appels API vers le backend à l'aide de fetch ou d'autres bibliothèques comme Axios.
   * **Pages**: Contient les pages principales de l'application, organisées en composants React pour la navigation et l'affichage des données.
   * **Design Assets**: Tous les fichiers de conception créés dans Figma, y compris les maquettes graphiques, les prototypes et les ressources visuelles.
2. **Backend**:
   * **Controllers**: Gère les requêtes HTTP entrantes, traite les données et retourne les réponses appropriées.
   * **Services**: Contient la logique métier de l'application, telle que la manipulation des données, les opérations CRUD, etc.
   * **Repositories**: Interfaces pour interagir avec la base de données, y compris les opérations de lecture, d'écriture, de mise à jour et de suppression.
   * **Models**: Définit la structure des données utilisées par l'application, y compris les entités et les schémas de base de données.
3. **Base de Données**:
   * **Collections MongoDB**: Stocke les données de l'application, y compris les animaux, les employés, les adoptions, etc.

**Flux de Développement**

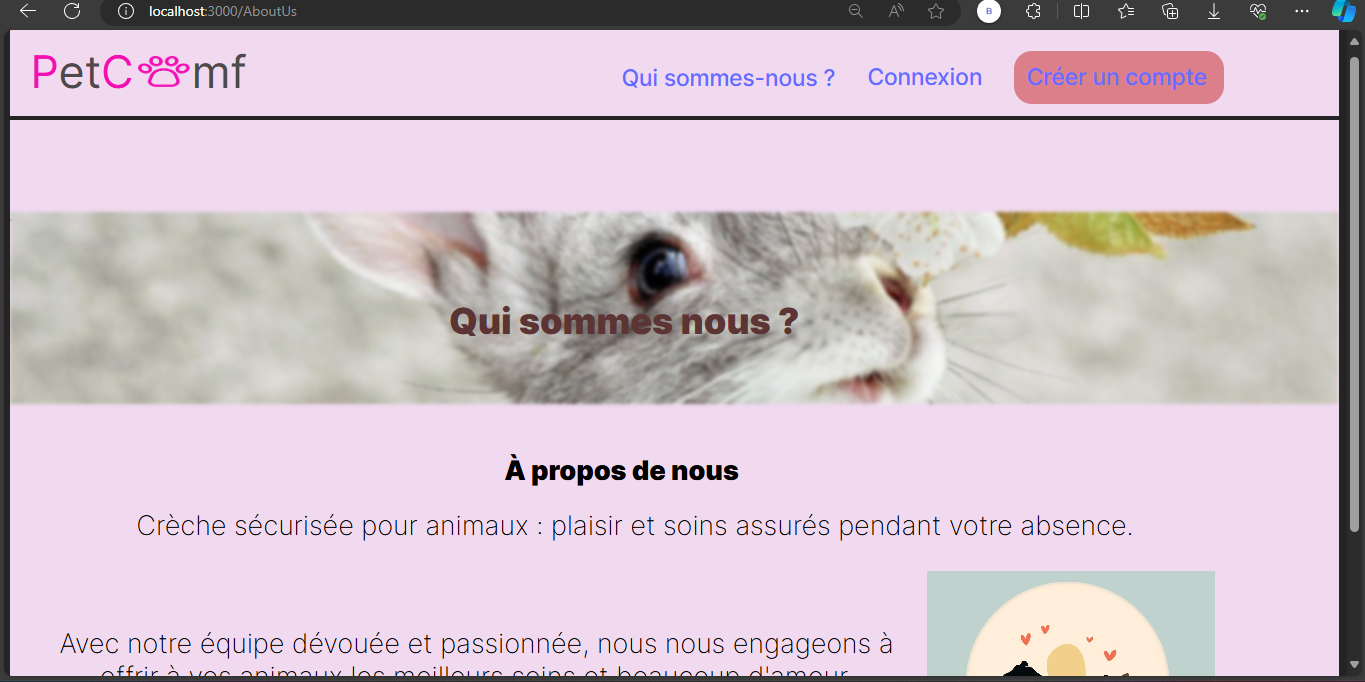
1. **Conception et Prototypage**:
   * Utilisez Figma pour créer des maquettes graphiques et des prototypes interactifs de l'interface utilisateur.
2. **Développement Frontend**:
   * Créez des composants React réutilisables pour construire l'interface utilisateur.
   * Utilisez Vite pour compiler et exécuter l'application frontend localement.
   * Intégrez les API services pour communiquer avec le backend et récupérer les données nécessaires.
3. **Développement Backend**:
   * Utilisez Spring Boot pour créer des endpoints RESTful pour les différentes fonctionnalités de l'application.
   * Implémentez la logique métier dans les services et les contrôleurs, en interagissant avec les repositories pour accéder aux données de la base de données.
4. **Tests et Déploiement**:
   * Effectuez des tests unitaires et d'intégration pour garantir le bon fonctionnement de l'application.
   * Déployez l'application frontend sur un service d'hébergement, comme Vercel ou Netlify.
   * Déployez l'application backend sur une plateforme cloud, comme Heroku ou AWS, en utilisant MongoDB Atlas pour héberger la base de données

Interfaces :

Page d’accueil :

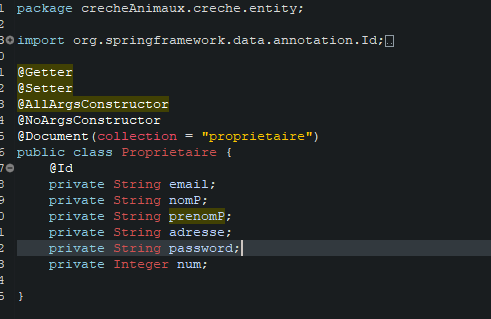


Page de présentation :

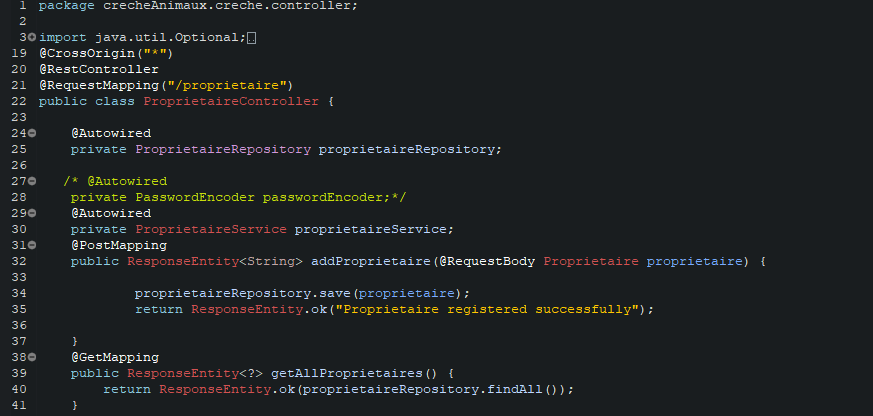


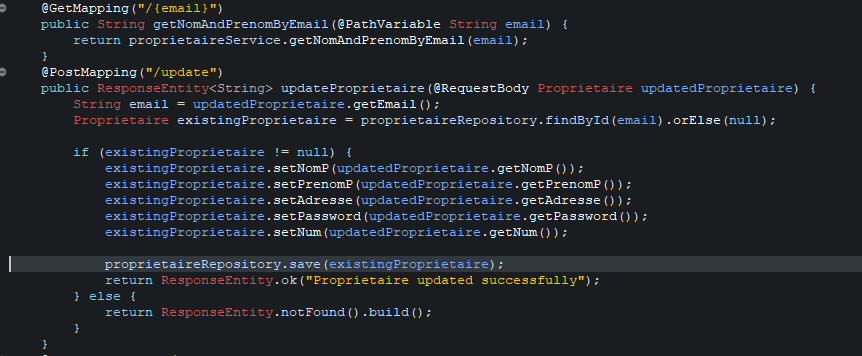
Propriétaire :

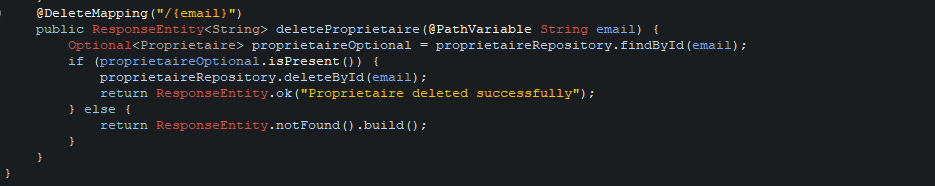
Nous avons la classe Propriétaire qui contient les champs suivants :



Il a comme fonctionnalités :



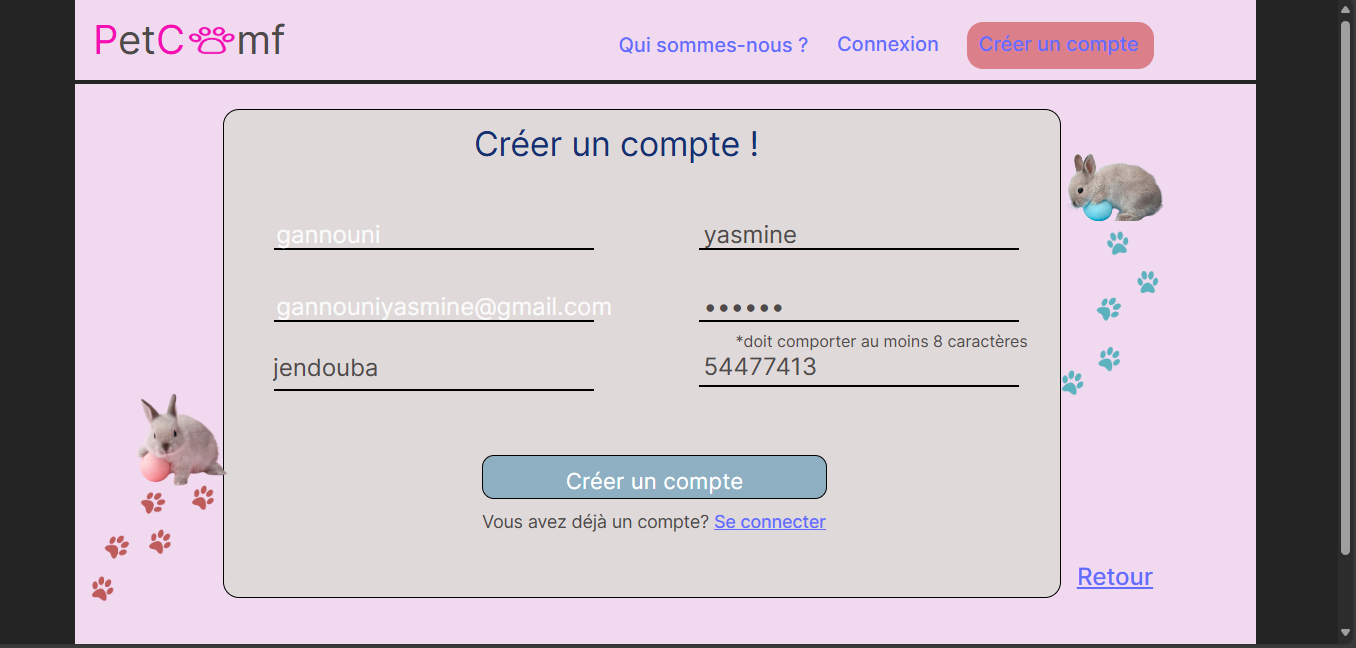


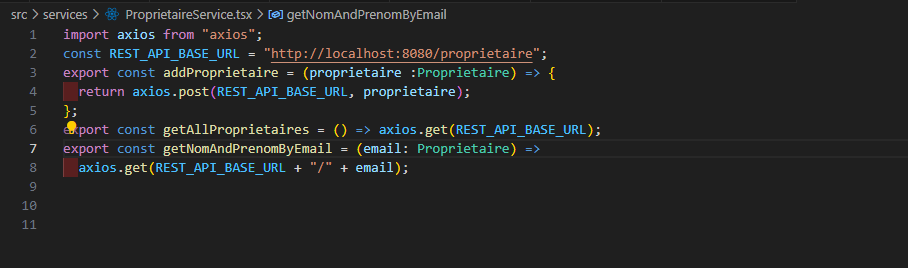


1. Ajout d'un propriétaire (@PostMapping) :

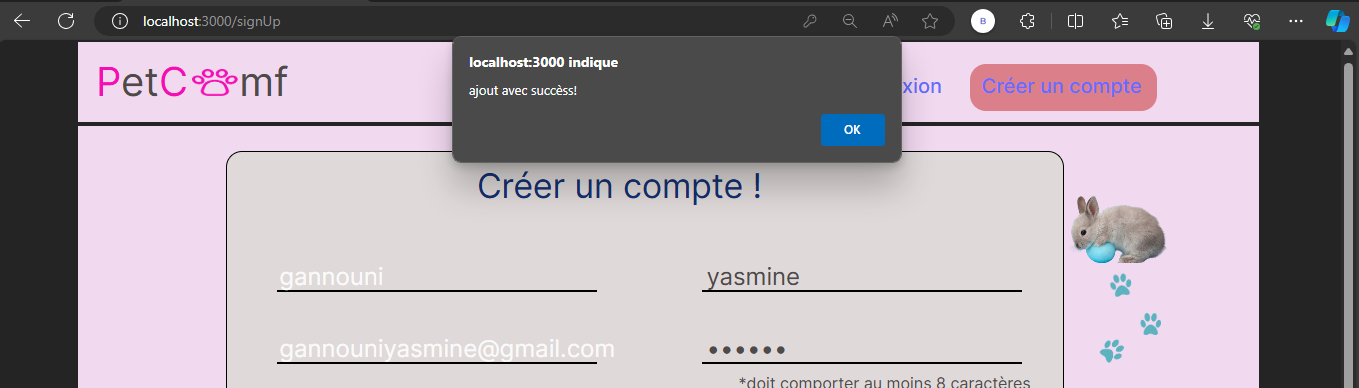
Lorsqu'une requête POST est envoyée à /proprietaire, un nouveau propriétaire est ajouté à la base de données en utilisant les données JSON fournies dans le corps de la requête. Les données du propriétaire sont extraites à partir du corps de la requête à l'aide de l'annotation @RequestBody.

Nous allons tester l'ajout d'un propriétaire avec React en saisissant nos données :

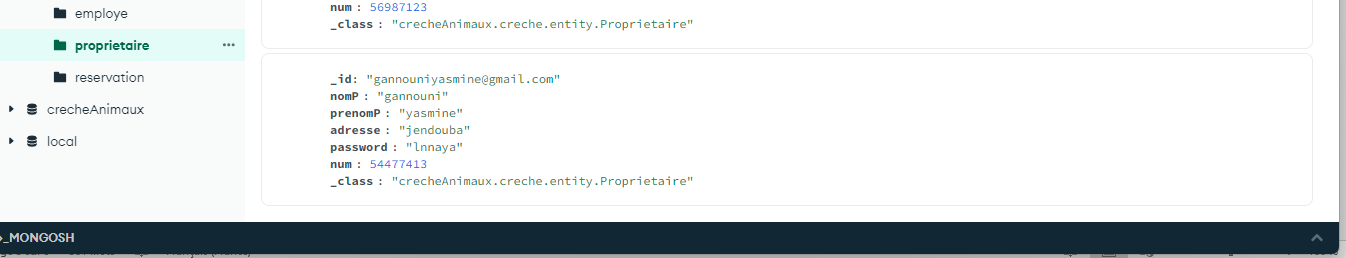
Mais avant tous, on doit lier entre le frontend et le backend de l'application et donc on utilise la bibliothèque Axios qui facilite l'envoi de requêtes http. Nous pouvons facilement faire appel aux fonctions définies dans notre backend Spring Boot.



En cliquant sur le bouton « créer compte » il affiche une alerte que ce soit un ajout avec succès ou échec :

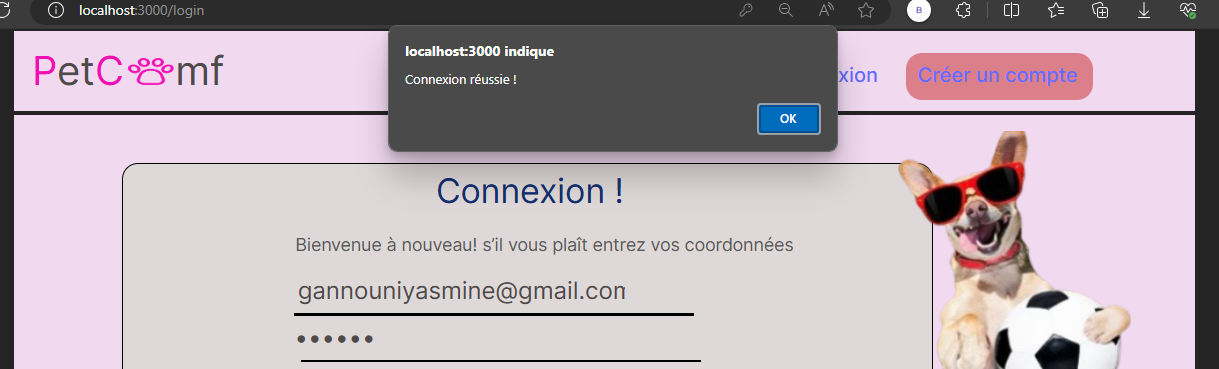


On vérifie l’ajout dans notre base de données :

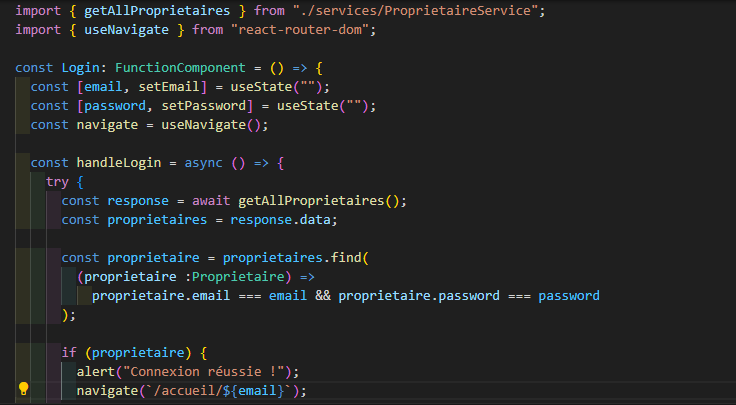


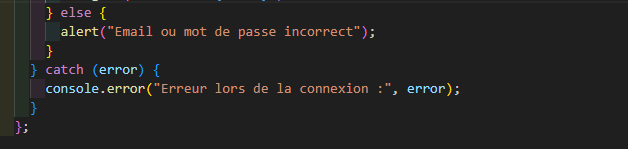
Une fois que nous avons créé un compte, nous serons automatiquement redirigés vers la page d'authentification.

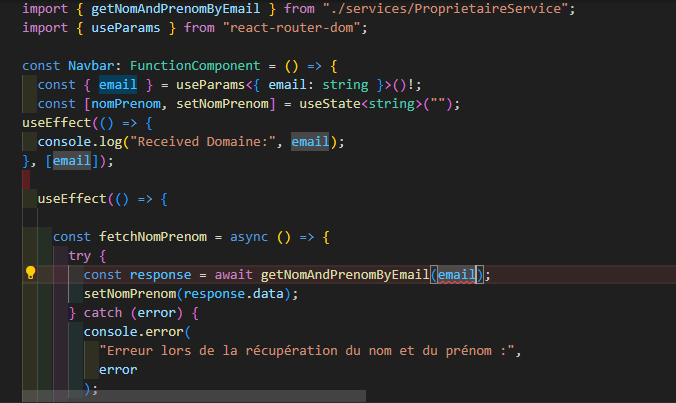




Lorsque la connexion réussit et qu'un utilisateur se trouve dans notre base de données avec ses données, je vais extraire l'e-mail à partir de l'URL pour récupérer le nom et le prénom de l'utilisateur.

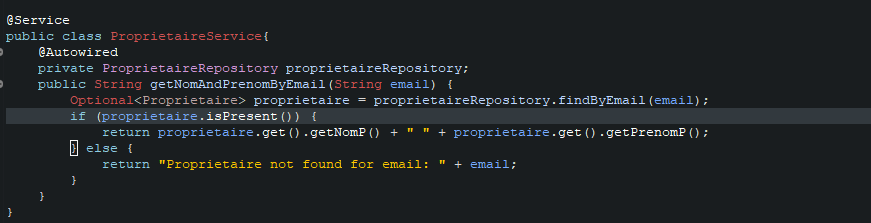




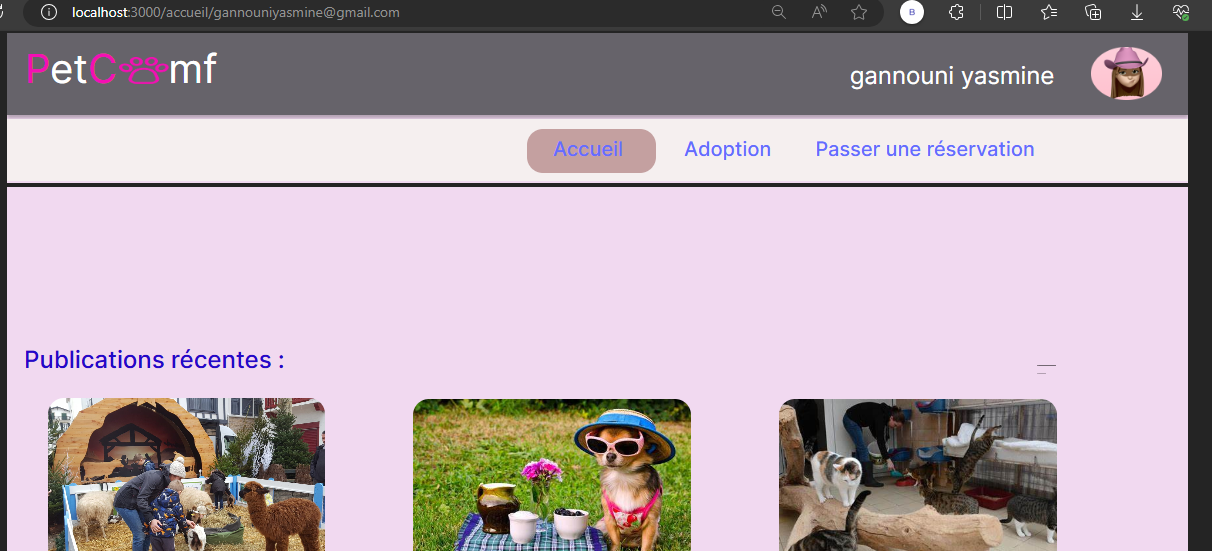


Ce code React extrait l'e-mail de l'URL avec useParams, puis utilise useEffect pour appeler une fonction asynchrone fetchNomPrenom. Cette fonction envoie une requête pour récupérer le nom et le prénom associés à cet e-mail à l'aide de getNomAndPrenomByEmail. Tout erreur est capturée et affichée dans la console. Une fois les données récupérées, elles sont stockées dans l'état local avec setNomPrenom.

La méthode de récupération du nom et prénom est écrite dans le service du propriétaire et elle est appelé ensuite dans le controlleur.



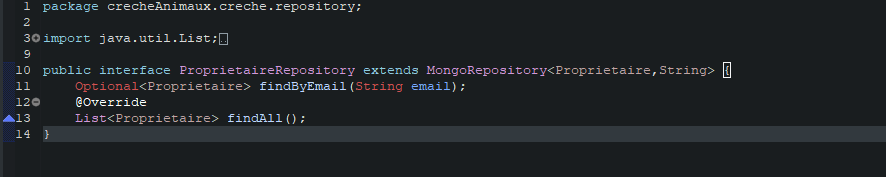
Voilà le résultat :



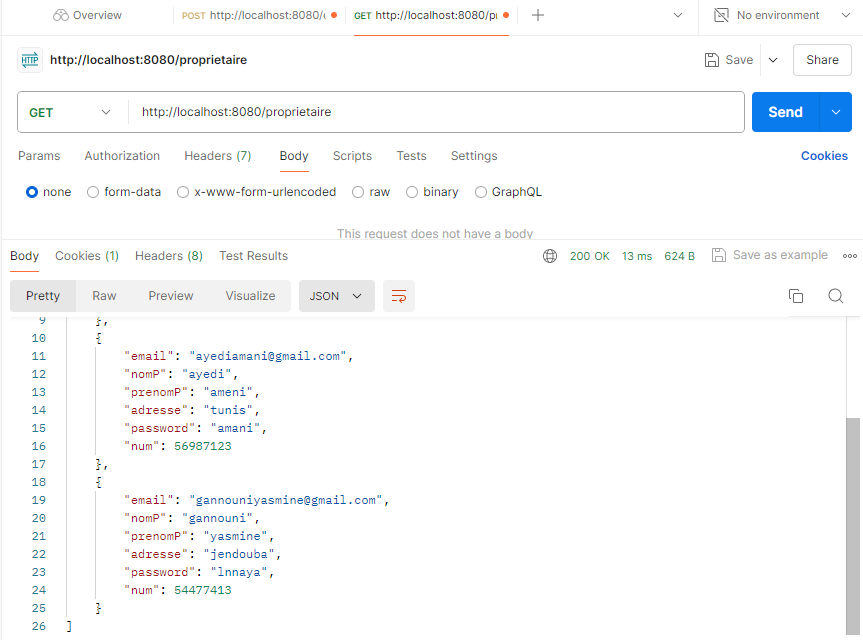
1. **Récupération de tous les propriétaires** (**@GetMapping**) :

Lorsqu'une requête GET est envoyée à /proprietaire, tous les propriétaires présents dans la base de données sont renvoyés sous forme de réponse JSON.

En utilisant proprietaireRepository.findAll() : Cette méthode récupère tous les propriétaires de la base de données à l'aide du repository et les renvoie sous forme de réponse HTTP.



Vérification avec Postman :

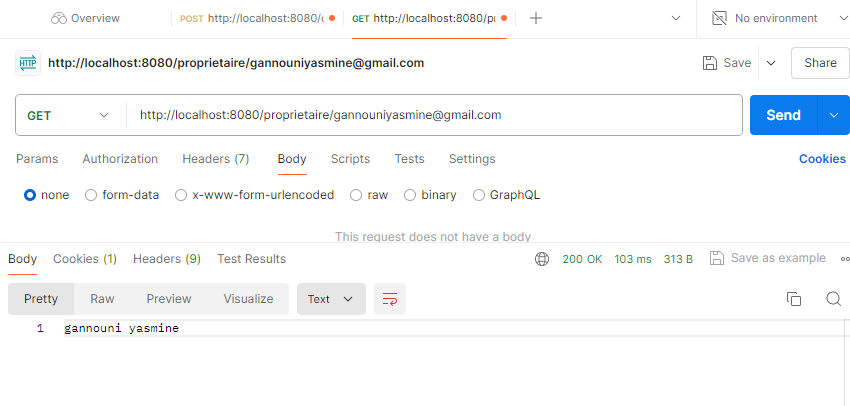


1. **Récupération d'un propriétaire par e-mail** (**@GetMapping("/{email}")**) :

Requête GET est envoyée à /proprietaire/{email}, qui récupère le nom et le prénom du propriétaire correspondant à l'adresse e-mail.

En utilisant proprietaireRepository.findById(email) : Cette méthode récupère le propriétaire correspondant à l'e-mail spécifié dans la base de données à l'aide de la méthode écrite dans repository (dans l’image en haut) et le renvoie sous forme de réponse HTTP.

Vérification :

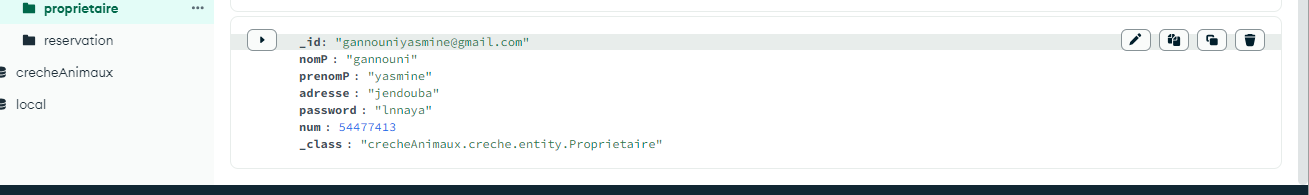


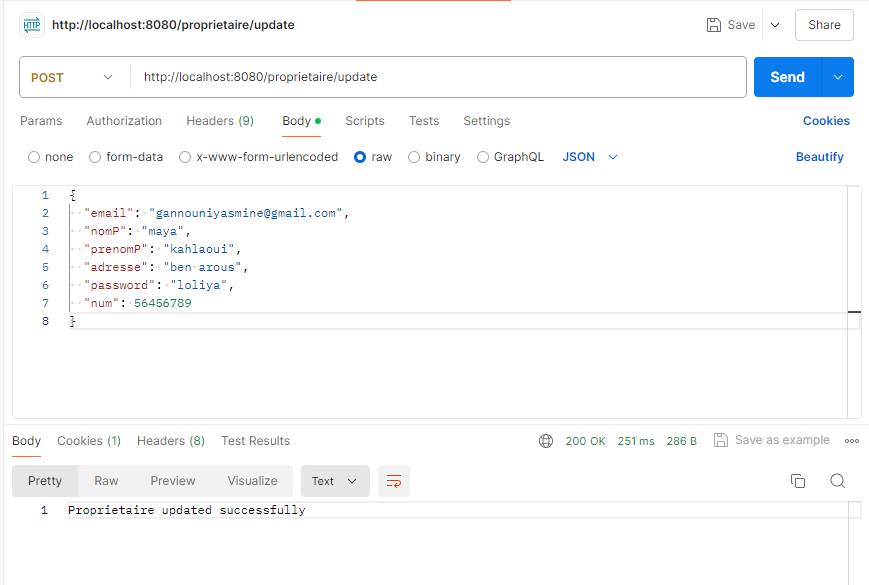
**Mise à jour d'un propriétaire** (**@PostMapping("/update")**) :

La méthode commence par extraire l'e-mail du propriétaire à partir de l'objet updatedProprietaire fourni dans le corps de la requête. L'e-mail sert d'identifiant unique pour trouver le propriétaire dans la base de données. On utilise le proprietaireRepository pour chercher dans la base de données un propriétaire correspondant à cet e-mail. La méthode findById renvoie un Optional<Proprietaire>, ce qui permet de gérer le cas où aucun propriétaire n'est trouvé. Si aucun propriétaire n'est trouvé renverra null.On vérifie si un propriétaire correspondant à l'e-mail existe. Si existingProprietaire n'est pas null, cela signifie que le propriétaire a été trouvé dans la base de données et que la mise à jour peut être effectuée. Pour chaque champ du propriétaire, on met à jour les valeurs avec celles fournies dans l'objet updatedProprietaire. Cela assure que les informations du propriétaire existant sont mises à jour avec les nouvelles valeurs. On enregistre le propriétaire mis à jour dans la base de données en utilisant le proprietaireRepository.

Vérification :

Propriétaire existant :





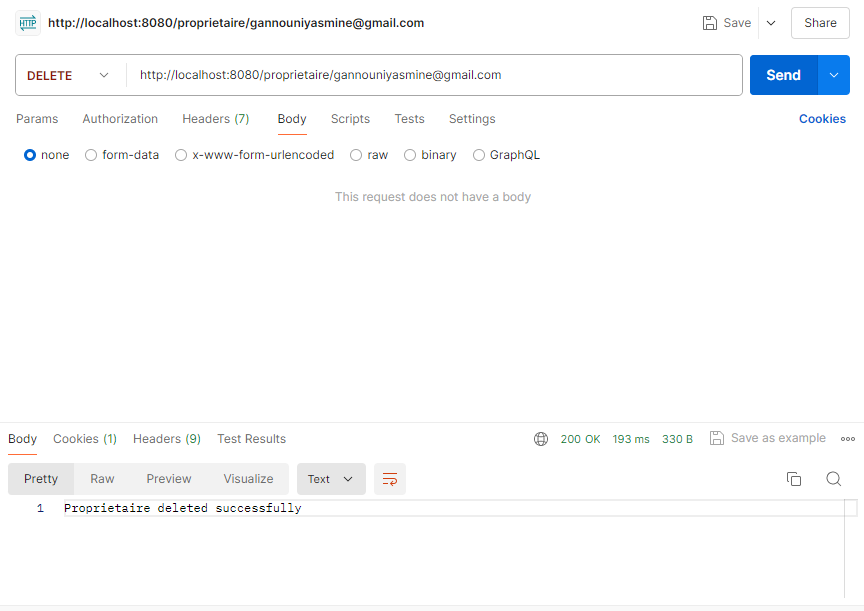
Après modification :



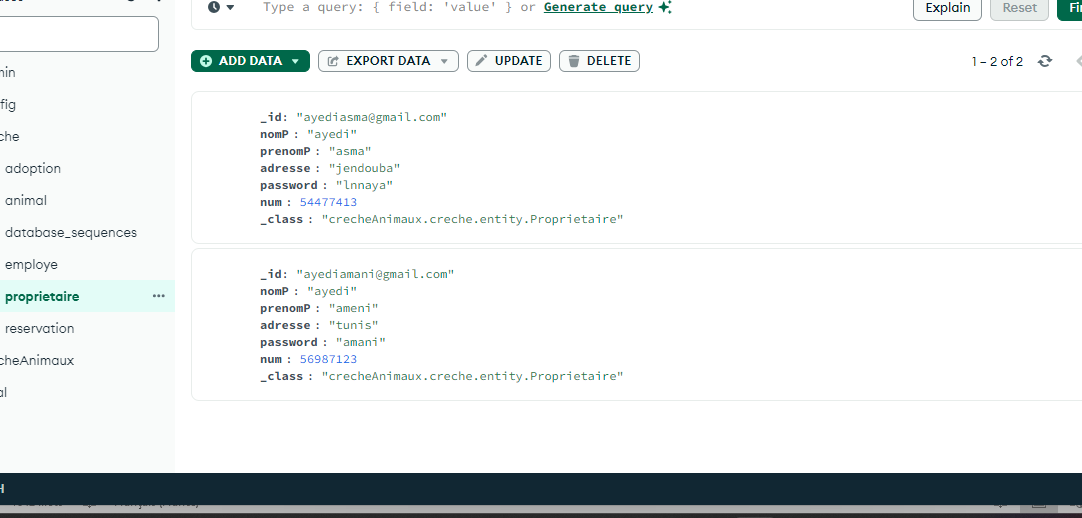
**Suppression d'un propriétaire par e-mail** :

On utilise proprietaireRepository.deleteById(email) : Cette méthode supprime le propriétaire correspondant à l'e-mail spécifié de la base de données à l'aide du repository.

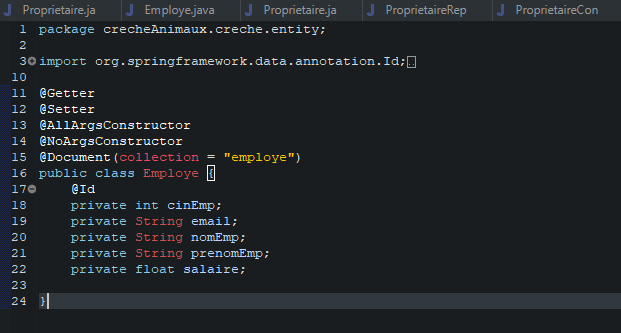
Vérification :



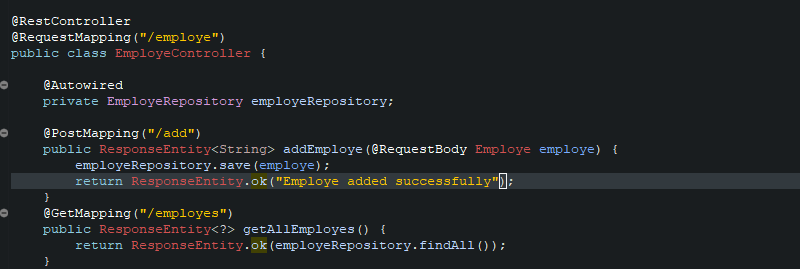
Vérification dans la base :

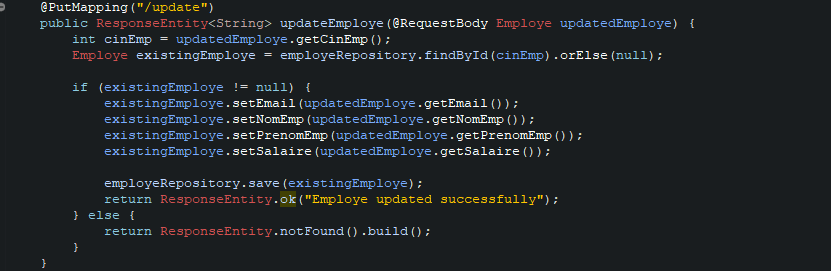


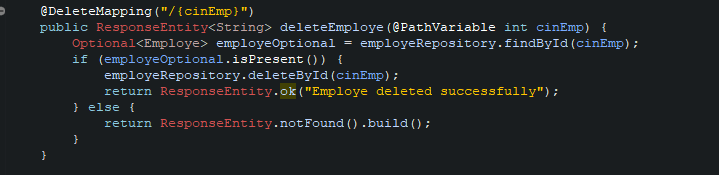
Classe Employee qui va s’occuper de l’animal :



Il a les fonctionnalités suivantes :



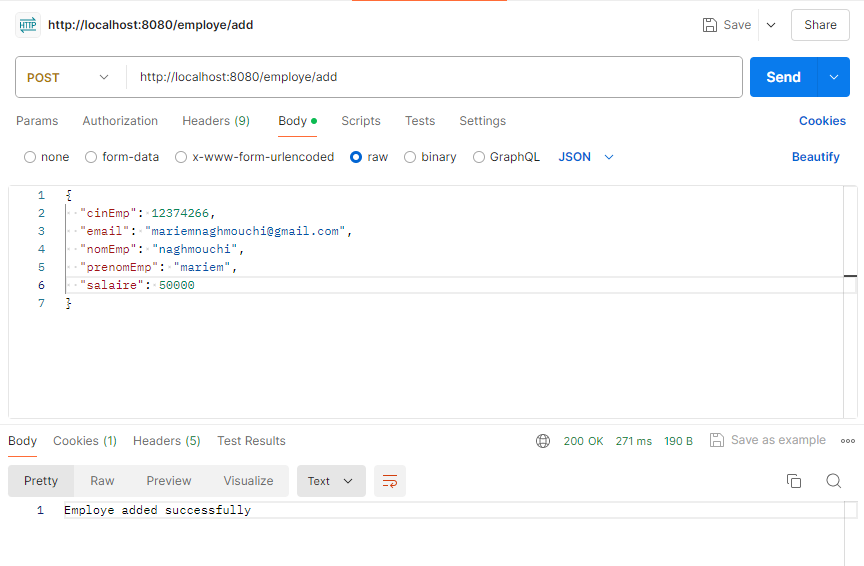




Ajout d’un Employee :

Cette méthode prend un objet Employe depuis le corps de la requête, le sauvegarde dans la base de données via le employeRepository.

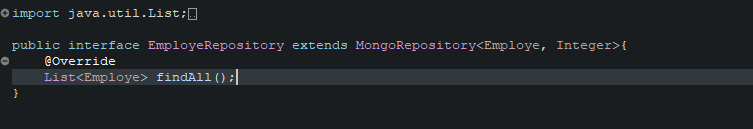
Vérification :



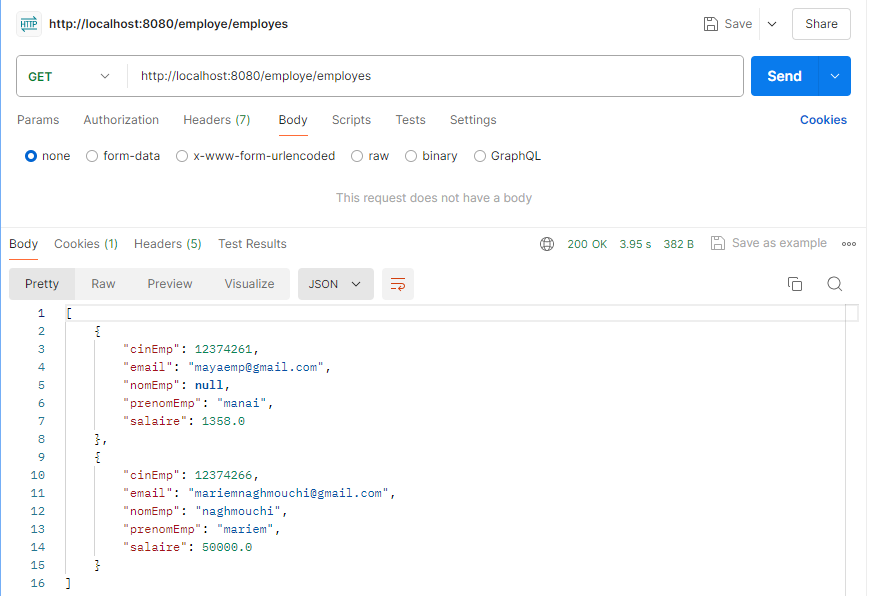


**Récupérer tous les employés** (**getAllEmployes**):

Cette méthode renvoie tous les employés présents dans la base de données en appelant findAll() sur le employeRepository.



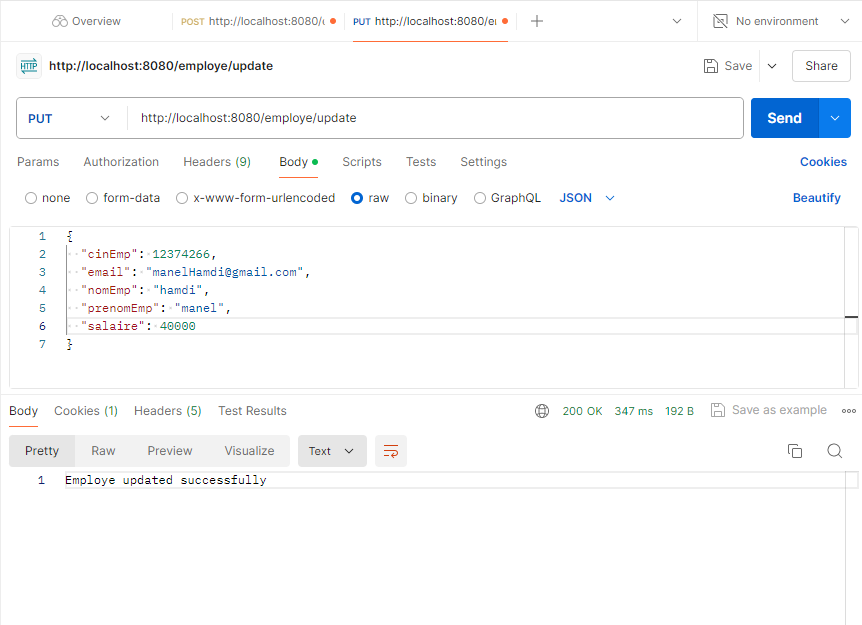
Vérification :



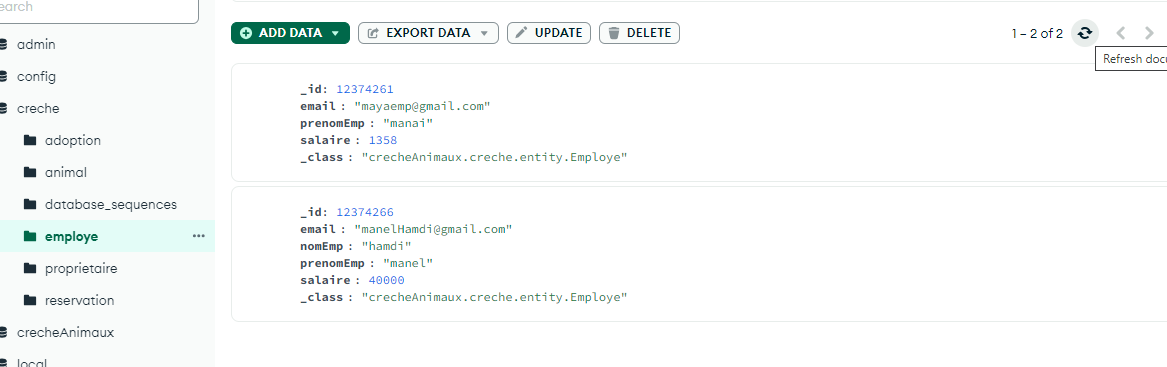
**Mettre à jour un employé** (**updateEmploye**):

Cette méthode prend un objet Employe avec les informations mises à jour depuis le corps de la requête. Elle cherche l'employé existant dans la base de données par cinEmp. Si l'employé existe, ses informations sont mises à jour et sauvegardées .

Vérification :



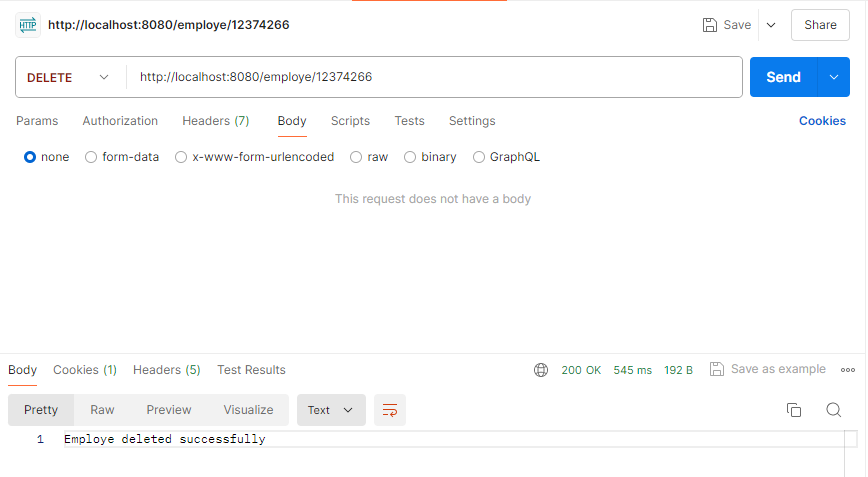
Vérification dans la base :



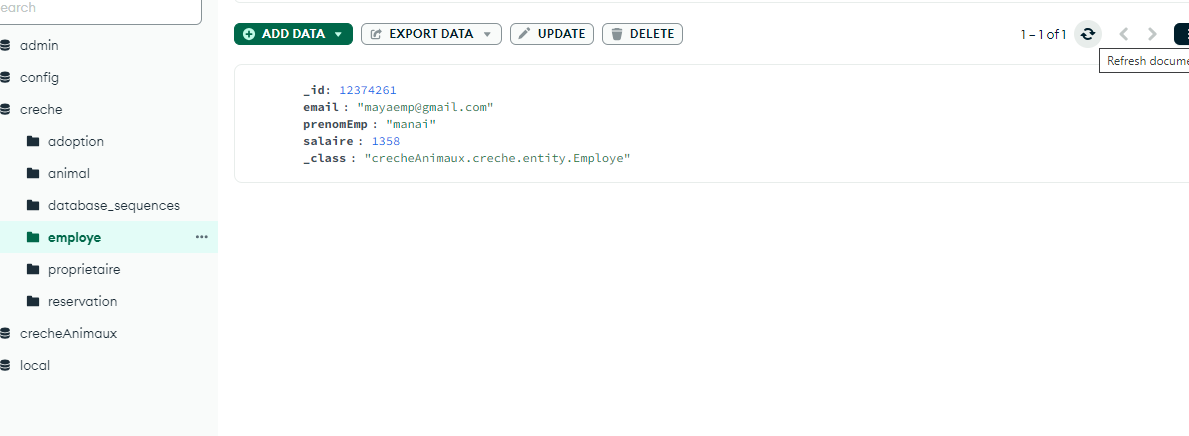
**Supprimer un employé** (**deleteEmploye**):

Cette méthode prend l'identifiant **cinEmp** de l'employé depuis l'URL, cherche l'employé correspondant dans la base de données. Si l'employé est trouvé, il sera supprimé.

Vérification :

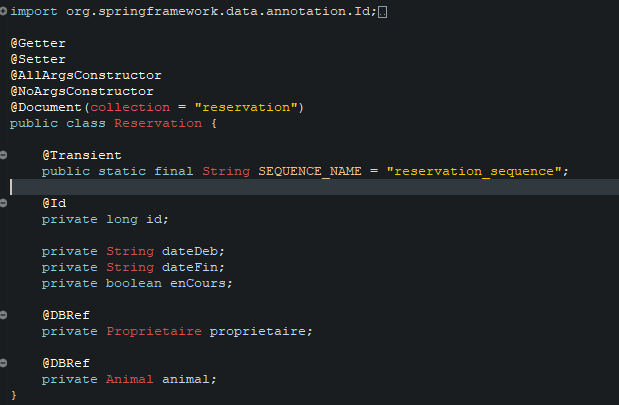


Vérification dans la base :

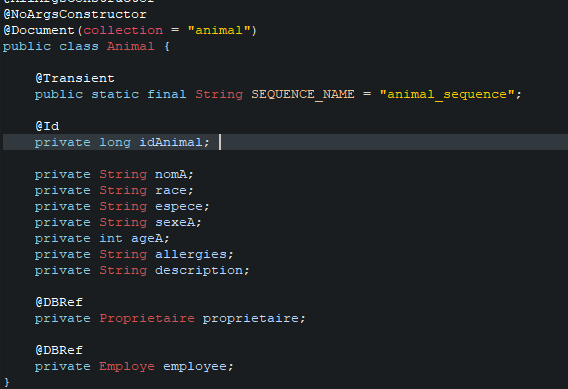


Classe Réservation :

Cette classe est utilisée pour représenter les réservations dans l'application. Chaque réservation a une date de début et de fin, un état indiquant si elle est en cours, et des références à un propriétaire et à un animal.



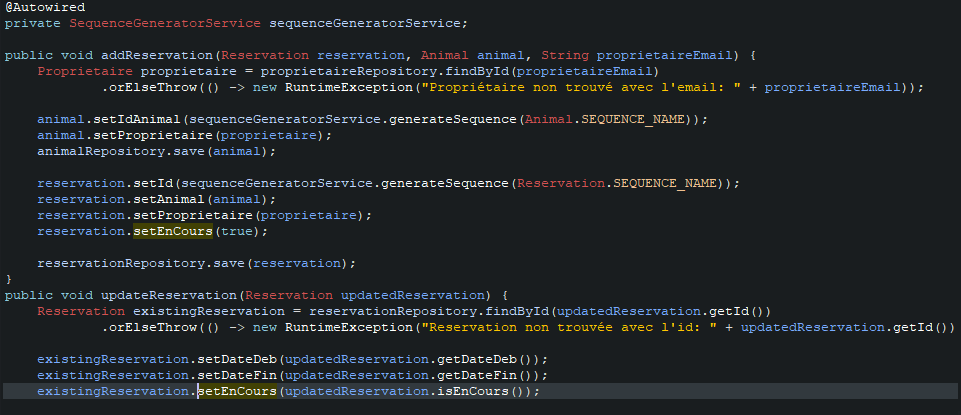
Si on ajoute une réservation , l’animal sera ajouté automatiquement dans la base .Un animal a les champs suivants :

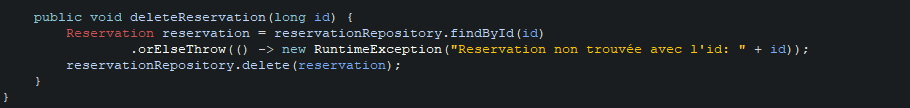


* **@Document(collection = "reservation")** : Indique que cette classe est un document MongoDB .
* **@Transient** : Indique que le champ annoté n'est pas persisté dans la base de données.
* **SEQUENCE\_NAME** : Une constante utilisée pour la génération de séquences d'identifiants pour les réservations.
* @DBRef : Indique une référence à un autre document dans MongoDB, proprietaire et animal sont des références à d'autres documents (Proprietaire et Animal respectivement).

Ce service permet de générer des identifiants uniques pour des documents MongoDB en maintenant une séquence incrémentale dans une collection dédiée. Lorsqu'un nouveau document est ajouté à une collection qui nécessite un identifiant unique, la méthode **generateSequence** est appelée pour obtenir la prochaine valeur de la séquence, garantissant ainsi l'unicité des identifiants dans la base de données.

Elle a les fonctionnalités suivantes :

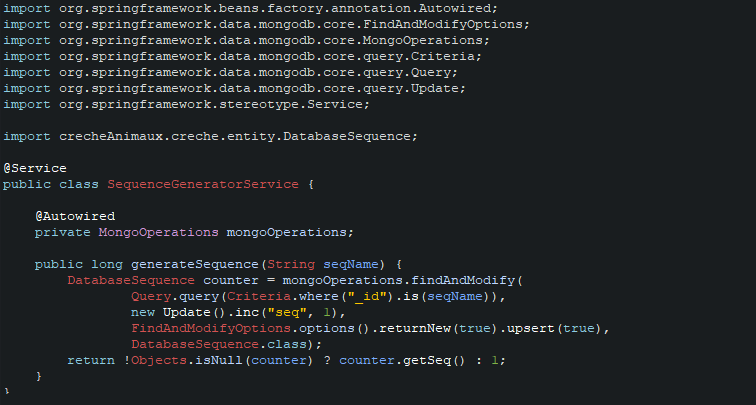


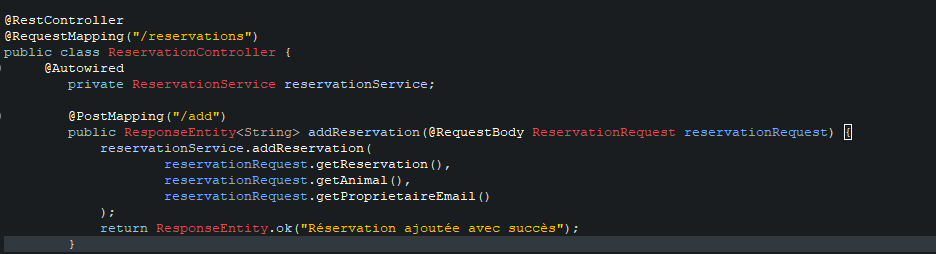


Méthode d’ajout d’une réservation :

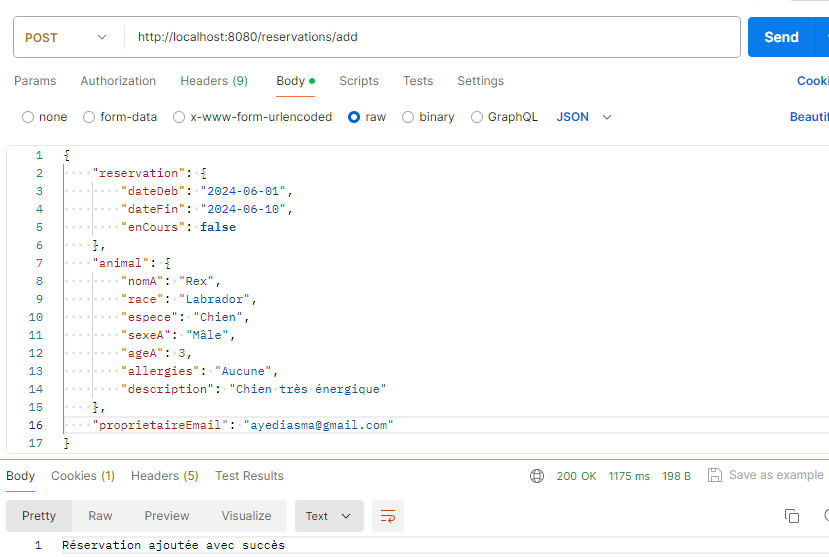
Elle Vérifie et récupère le propriétaire à partir de la base de données en utilisant l'e-mail fourni, crée un nouvel animal avec un ID unique et l'associe au propriétaire et crée une nouvelle réservation avec un ID unique, associe l'animal et le propriétaire à cette réservation, et marque la réservation comme en cours et sauvegarde à la fois l'animal et la réservation dans la base de données.

La méthode generateSequence dans la classe SequenceGeneratorService sert à générer des séquences uniques pour les entités de la base de données.

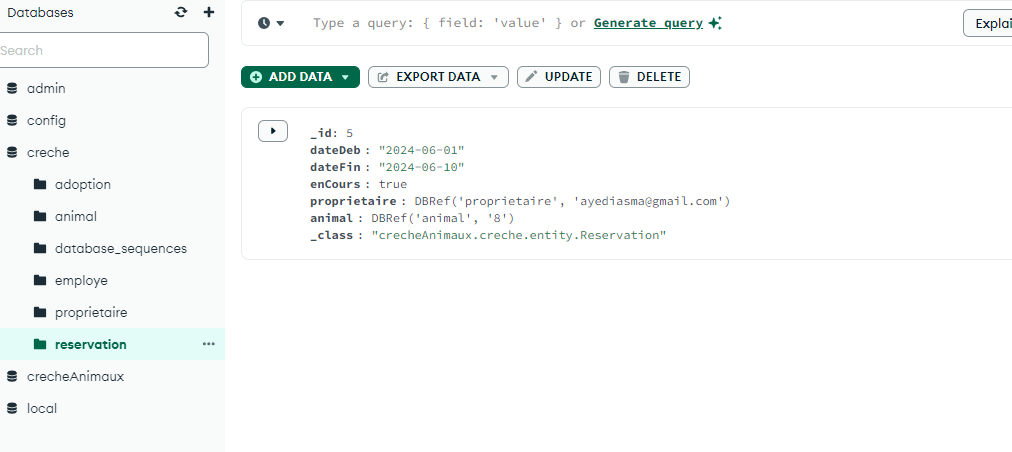




Vérification :



Vérification dans la base :

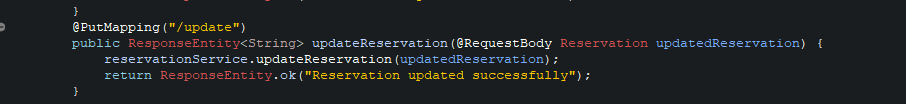


On vérifie aussi l’ajout de l’animal dans le document « Animal »:

****

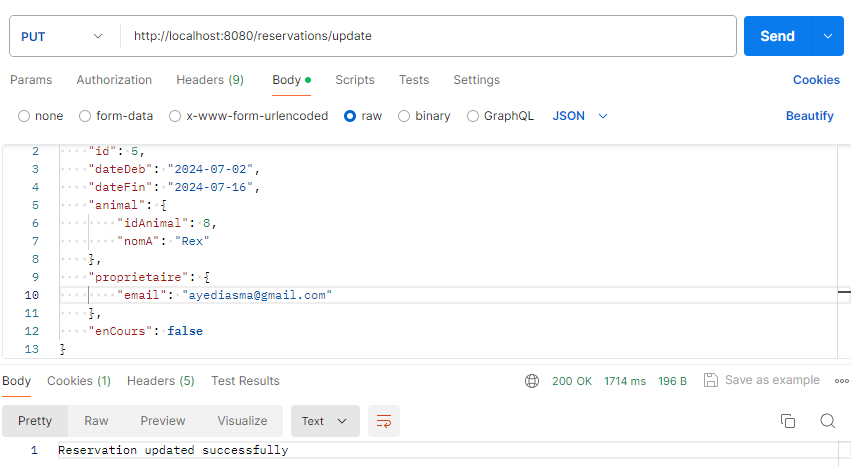
**Mise à Jour de Réservation :**

La méthode updateReservation prend une requête avec les détails de la réservation mise à jour et utilise le service pour mettre à jour cette réservation.

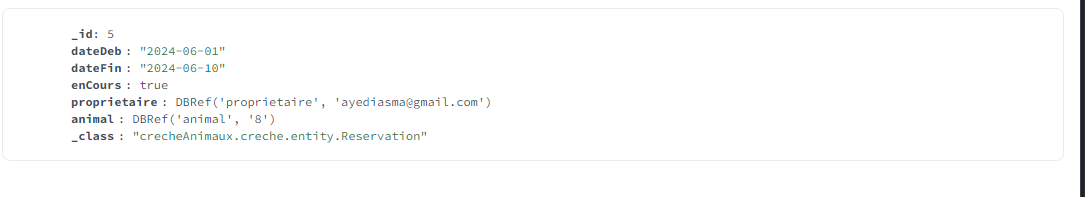


Cette méthode prend en paramètre un objet Reservation mis à jour, elle recherche la réservation existante dans la base de données par son ID. Si la réservation n'est pas trouvée, une exception est levée sinon, elle met à jour les champs dateDeb, dateFin et enCours de la réservation existante avec les nouvelles valeurs fournies. Ensuite, elle sauvegarde la réservation mise à jour dans la base de données.

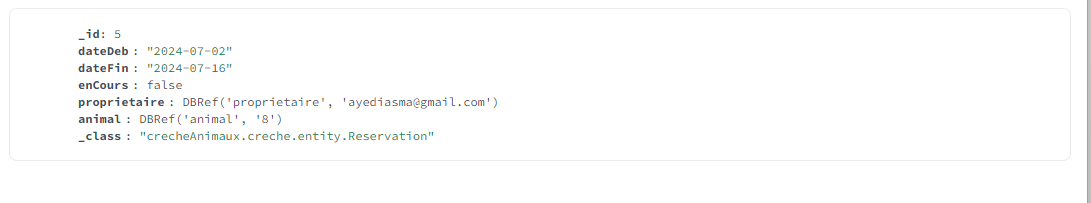
Vérification :



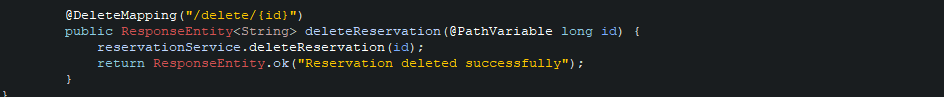
Réservation avant la mise à jour :



Réservation après la mise à jour :

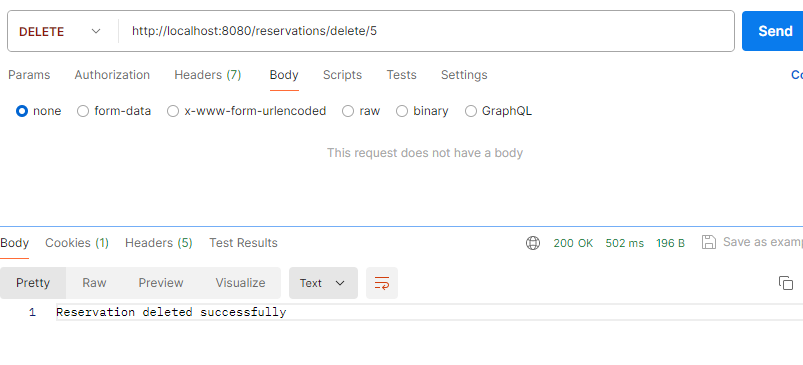


Suppression d’une réservation :

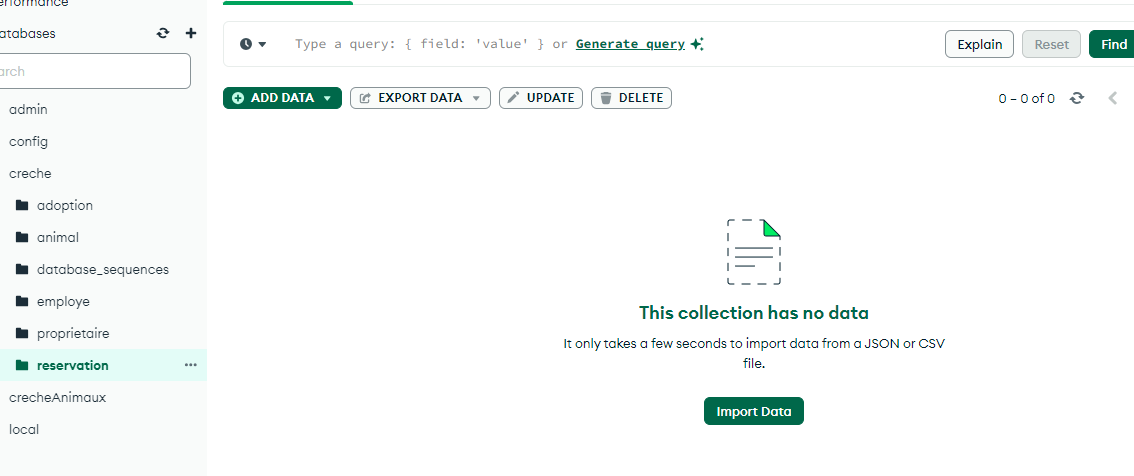


Cette méthode prend en paramètre l'ID de la réservation à supprimer. Elle recherche la réservation correspondante dans la base de données par son ID. Si la réservation n'est pas trouvée, une exception est levée sinon, elle supprime la réservation trouvée de la base de données.

Vérification :

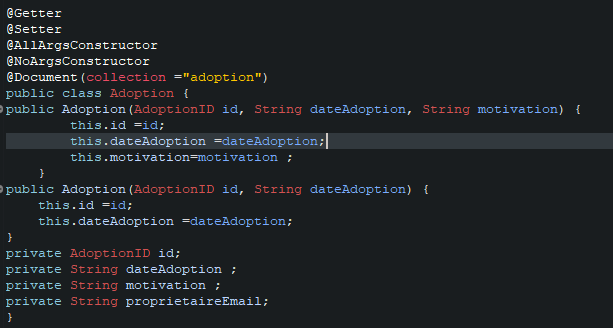


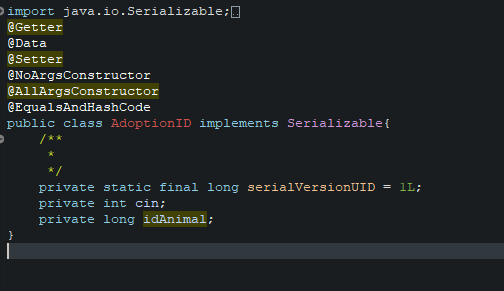
Vérifiaction dans la base :



Classe d’adoption :

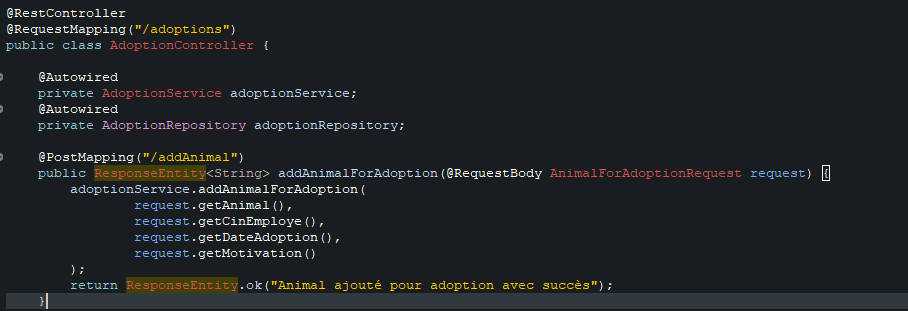
Elle a une clé primaire composée constituée du CIN de l'employé et de l'ID de l'animal.



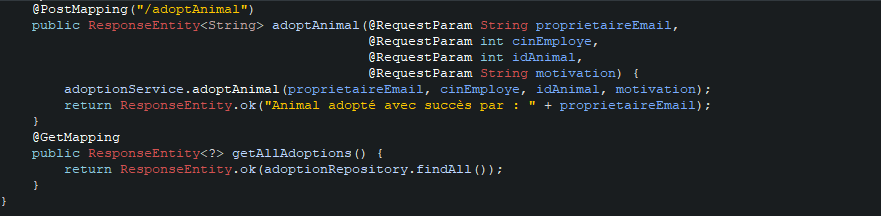


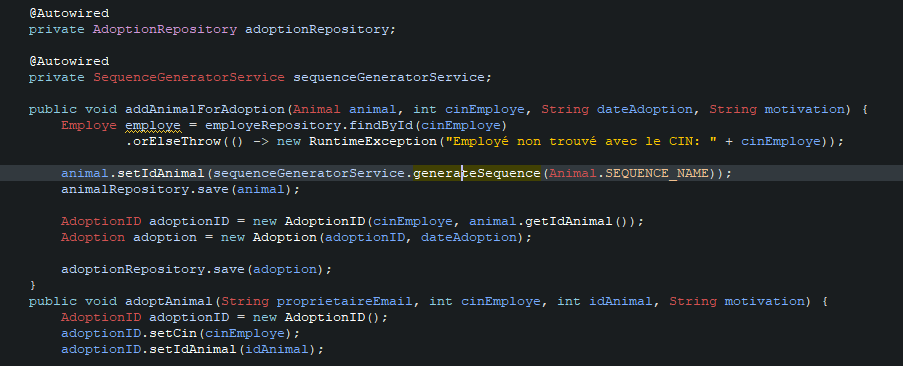
Elle a les méthodes suivantes :

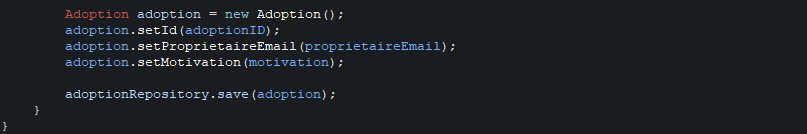
C’est pour l’ajout un animal pour l’adoption par un employée



Cette méthode pour l’adoption de l’animal par un propriétaire



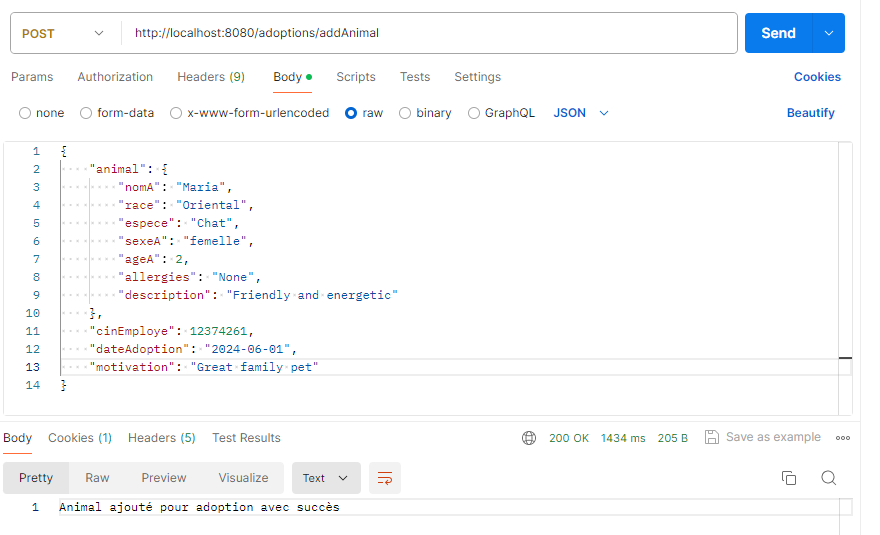




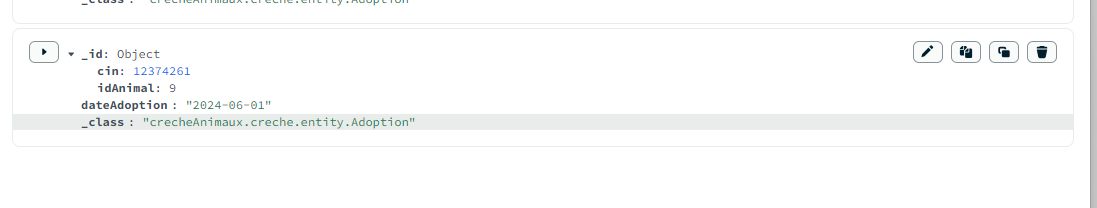
Méthodes d’ajout un animal pour adoption :

Cette méthode prend un objet Animal, le CIN de l'employé, une date d'adoption et une motivation. Trouve l'employé par son CIN. Si l'employé n'est pas trouvé, lance une exception. Elle génère un nouvel ID pour l'animal et sauvegarde l'animal dans le référentiel animalRepository , crée une instance d'AdoptionID avec le CIN de l'employé et l'ID de l'animal ,crée une nouvelle instance d'Adoption avec l'ID d'adoption et la date d'adoption et sauvegarde l'adoption dans le référentiel adoptionRepository.

Vérification :



Vérification dans la base :



On vérifie aussi l’ajout de l’animal dans le document ‘animal’



Listage des animaux pour adoption :

