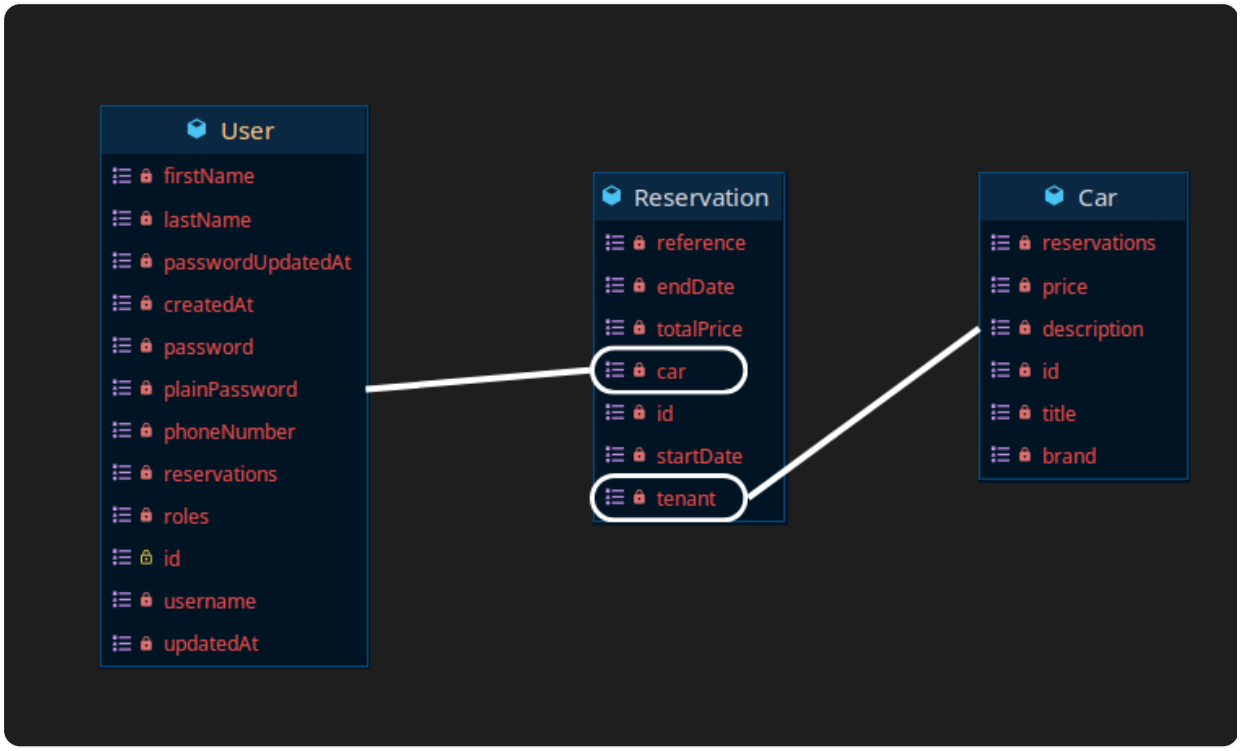
# Agence de location des voitures

> La document technique du project, ce project est basé sur l'achitecture MVC dans Symfony à l'aide de l'utilisation d'API Platform pour facilté/accéleré la création des Endpoints, et assuré la qualité de l"application avec les tests unitaire.

## 1 - Le stack technique:

- **Symfony 6.4:** La version LTS (long  term support) du framework.
- **PHP 8.3:** La version LTS du PHP .
- **API Platform 4.0:** La version d'API Platform responsable facilite la création des Endpoints.

## 2 - La conception:




Le schéma des entities

- **User**: représente les utilisateurs de l'application.
- Car : représente les voitures.
- Reservation : représente les réservations des utilisateurs avec une voiture choisie dans une periode de temps.

# 3 - Les Endpoints

```php
#[Get(
    uriTemplate: '/users/me',
    normalizationContext: ['groups' => [OperationEnum::USER_AUTH]],
    provider: AuthenticatedUserProvider::class,
)]
#[GetCollection(
    normalizationContext: ['groups' => [OperationEnum::UserListing->name]]
)]
#[ApiFilter(filterClass: OrderFilter::class)]
#[ORM\Table(name: '`user`')]
#[ORM\UniqueConstraint(name: 'user_unique', columns: ['username'])]
#[UniqueEntity(fields: 'username', message: Validator::UNIQUE_ENTITY)]
#[ORM\Entity]
class User implements UserInterface, PasswordAuthenticatedUserInterface
```

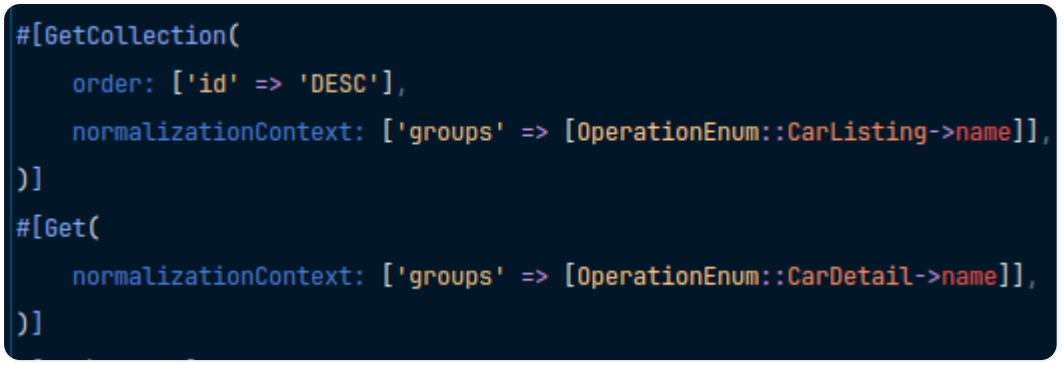
Les endpoints créer par API Platform dans l'entité User

Cet photo représente le code utilisé pour créer ces endpoints dont ils ont une relation avec l'utilisatuer.

**/api/users/me :** Obtenir les informations de l'utilisateur connecté en passant le token on header (bearer).

/**api/users :** Obtenir la listes des utilisateurs.

Appliqué l'unicité du nom d'utilisateur.



```
#[GetCollection(
    order: ['id' => 'DESC'],
    normalizationContext: ['groups' => [OperationEnum::CarListing->name]],
)]
#[Get(
    normalizationContext: ['groups' => [OperationEnum::CarDetail->name]],
)]
```

Les endpoints créer par API Platform dans l'entité Car (véhicule)

Cet photo représente le code utilisé pour créer ces endpoints dont ils ont une relation avec l'utilisatuer.

/**api/cars :** Obtenir la listes des voiture.

**/api/cars/[id] :** Obtenir le détails d'une voiture en replaçant [id] par l'id du voiture.

```php
#[GetCollection(
    order: ['startDate' => 'DESC', 'endDate' => 'ASC', 'id' => 'DESC'],
    normalizationContext: ['groups' => [OperationEnum::ReservationListing->name]],
)]
#[Get(
    normalizationContext: ['groups' => [OperationEnum::ReservationDetail->name]],
)]
#[GetCollection(
    uriTemplate: '/users/{tenantId}/reservations',
    uriVariables: [
        'tenantId' => new Link(
            fromClass: User::class,
            fromProperty: 'reservations',
        )
    ],
    controller: GetUserReservations::class,
    normalizationContext: ['groups' => [OperationEnum::UserReservationListing->name]],
    read: false,
    provider: null,
    name: 'user_reservations',
)]
#[Post(
    denormalizationContext: ['groups' => [OperationEnum::ReservationCreate->name]],
    normalizationContext: ['groups' => [OperationEnum::ReservationDetail->name]],
)]
#[Put(
    denormalizationContext: ['groups' => [OperationEnum::ReservationUpdate->name]],
    normalizationContext: ['groups' => [OperationEnum::ReservationDetail->name]],
)]
#[Delete]
#[AvailableCar]
#[ORM\Entity]
class Reservation
```

Les endpoints créer par API Platform dans l'entité Reservation (réservation)

**/api/users/[id_user]/reservations :** Obtenir toute les reservations d'un utilisateur passé en parametre et doit aussi être connecté, sinon il vas générer une page non trouvé.
**/api/users avec la method DELETE :** Annuler une réservation.

# 4 - L'architecture du dossier src:

```
-Platform:~/Desktop/PROJECTS/car-rental$ ls -a src
Controller  DataFixtures  Entity  EventListener  Factory  Kernel.php  Listener  Repository  Service  State  Story  Validator
```

**Controller**: les chemins personalisé.

**DataFixtures**: les classes responsables des chargements des données dans la base de données.

**Entity**: Les entités de l'application.

**Factory**: Comment chaque occurence d'une entité vas être générer les données chargé.

**Respository**: La couche responsable de la communications avec la base de données.

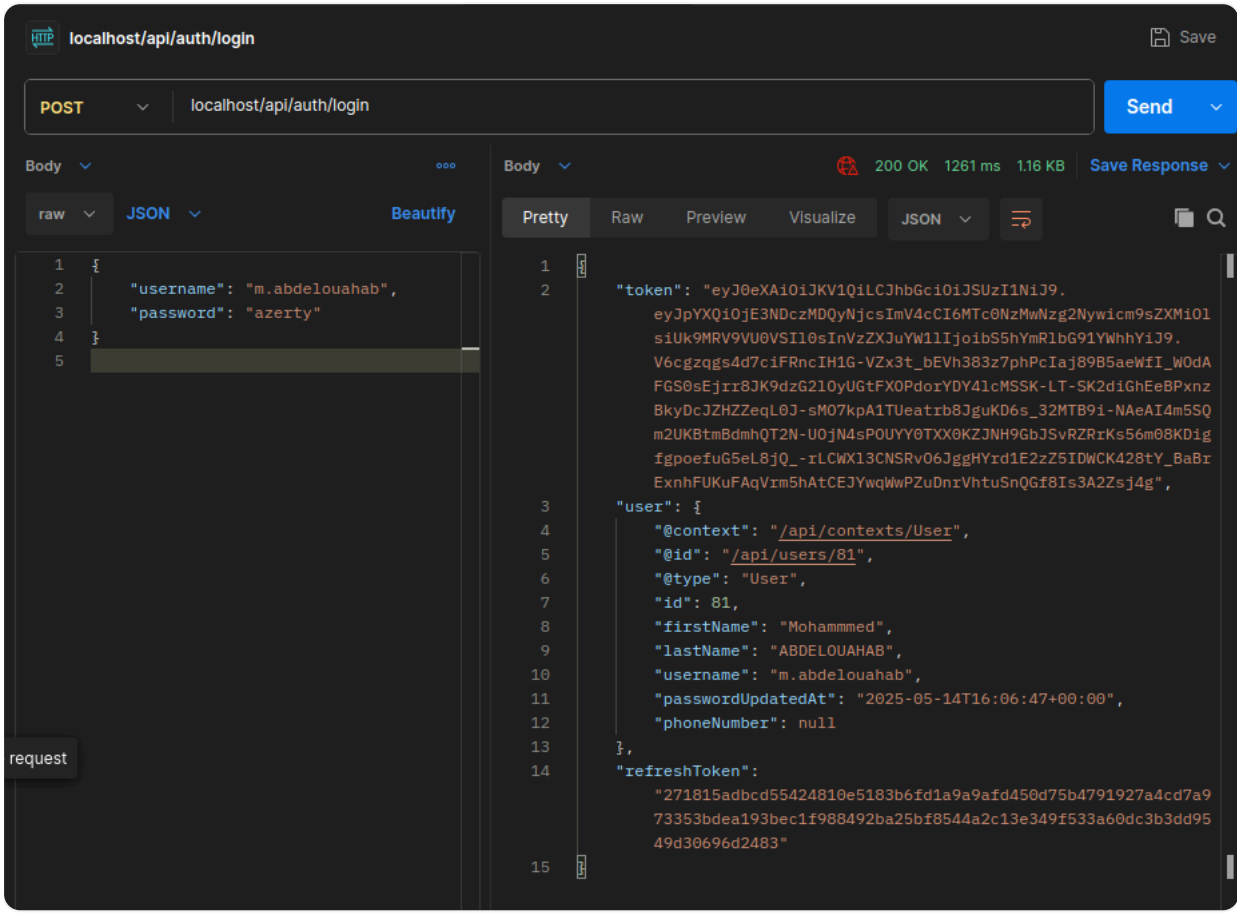
**Service**: Le code métier de l'application.

**State**: Dossier dédié pour créer des classes qui aident les Endpoint d'API Platform de créer des couches après/avant par example pour faire un traitement X.

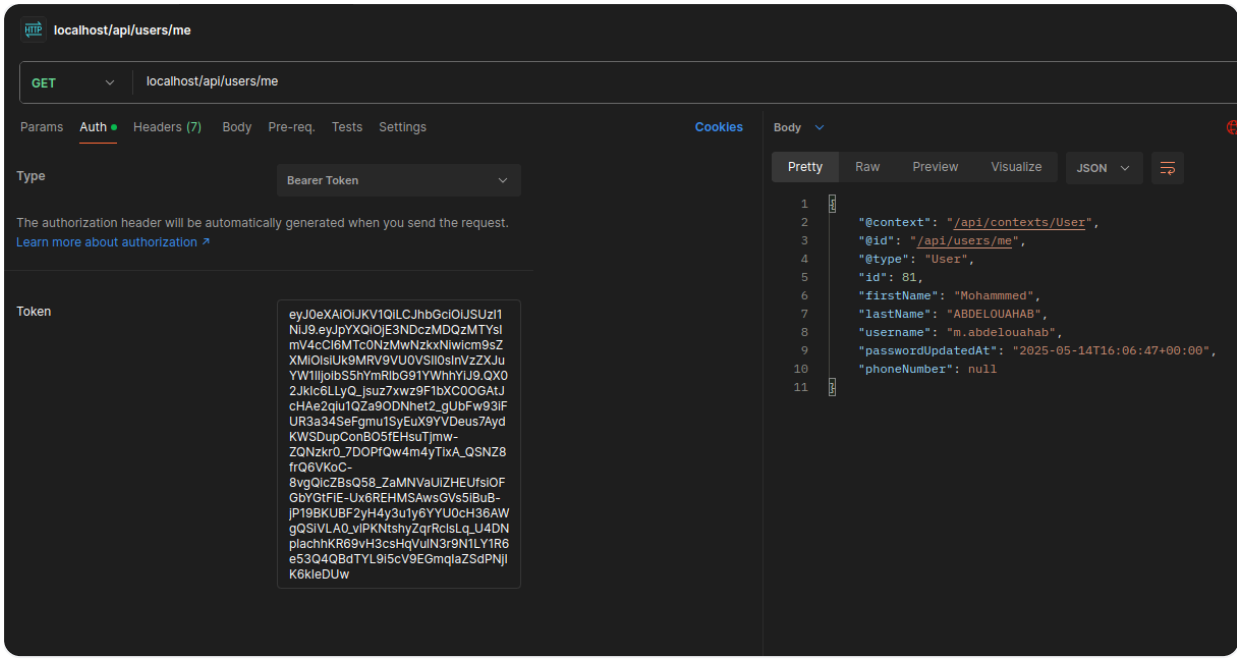
**Story**: La séparation du chargement des données au niveau Dev/Prod/Test.

**Validator**: Des validateurs personalité qui ne sont pas pré-définis dans Symfony/Validator.

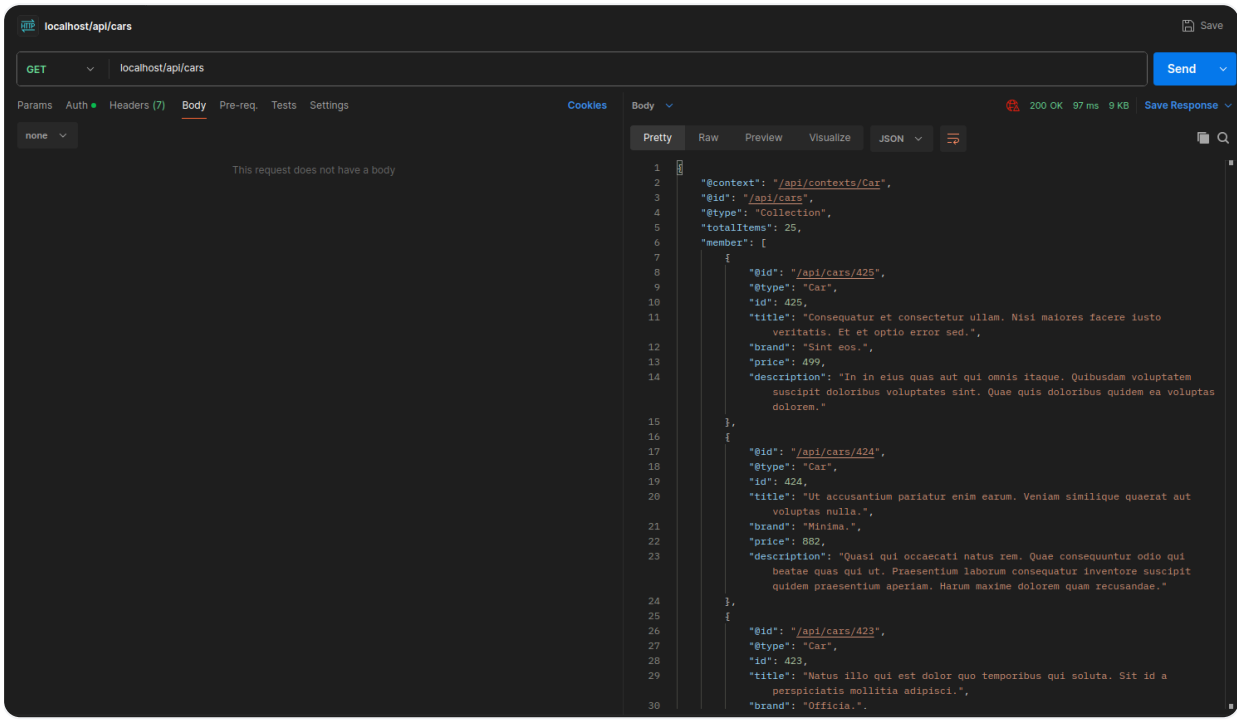
# 5 - Les tests dans postman:



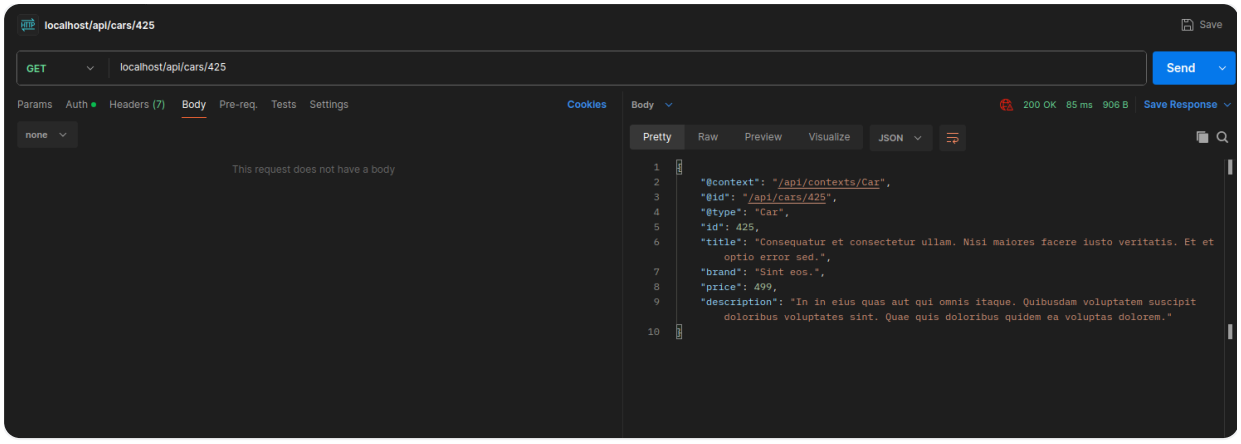

L'authentification avec succès



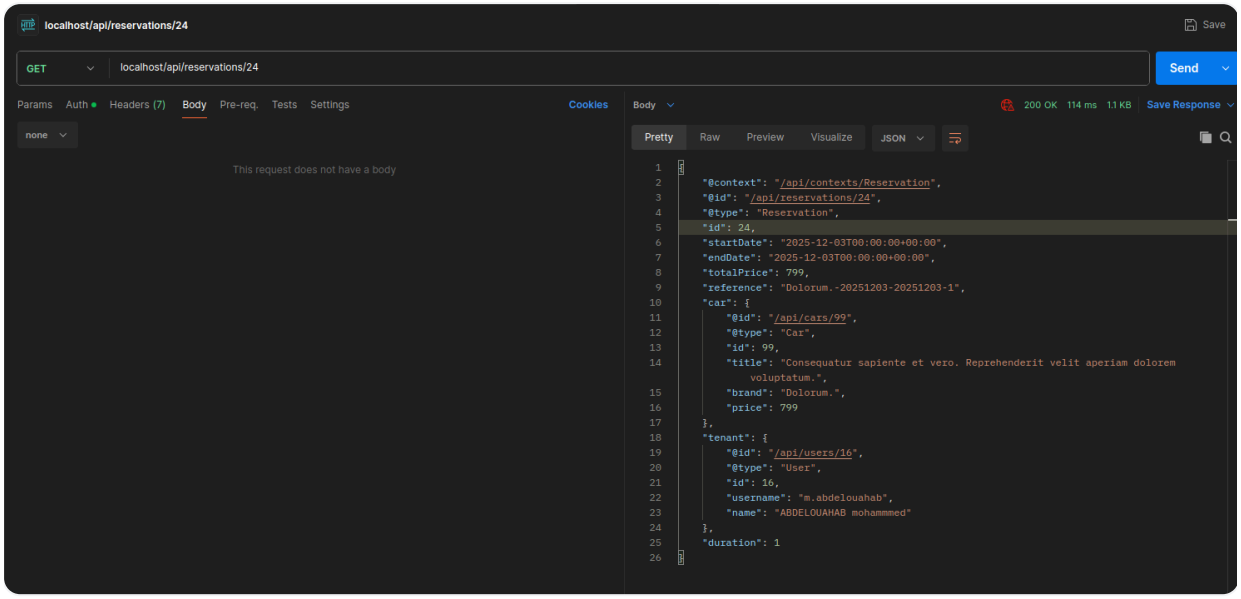

Les informations d'utilisateur connecté



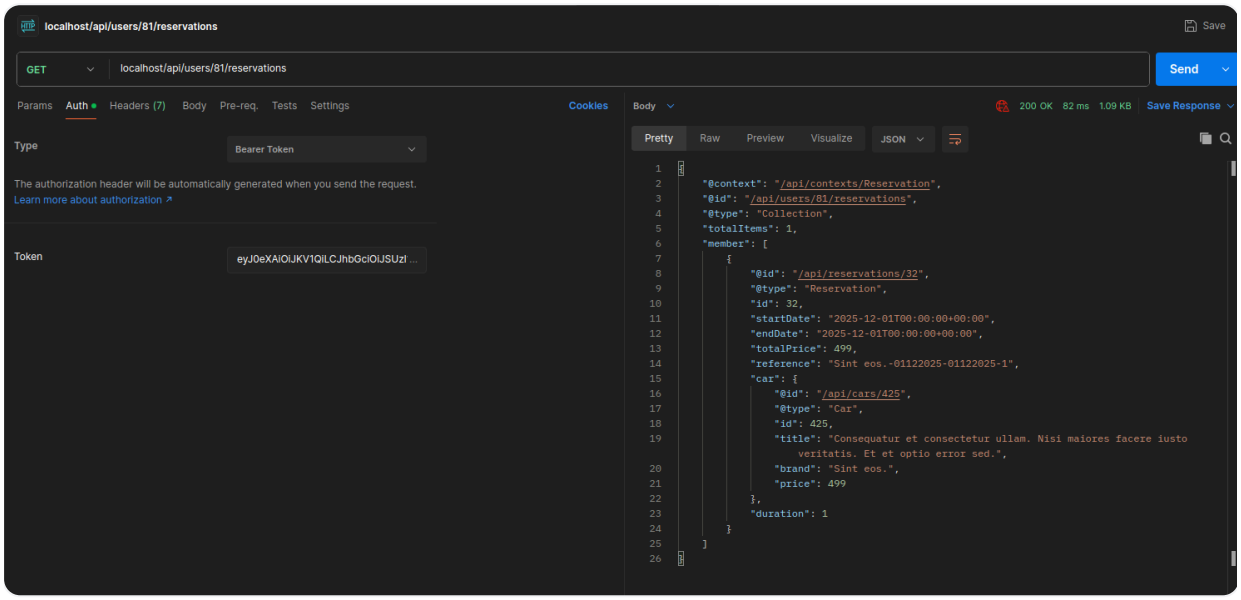

Le liste des voitures



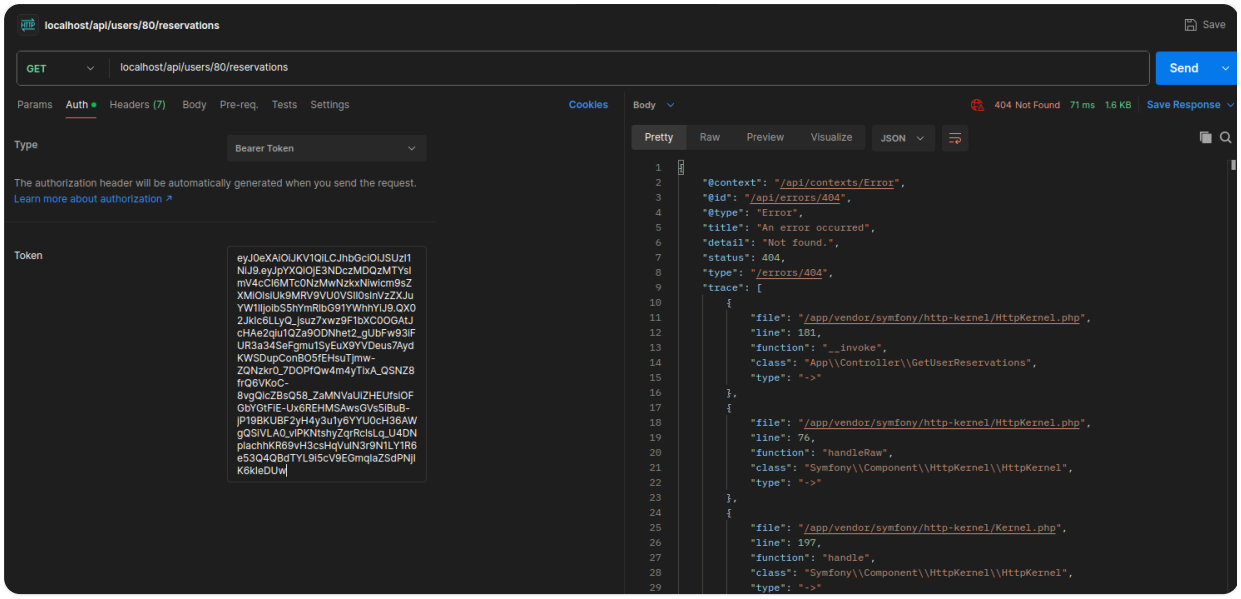

Le détails d'une voiture



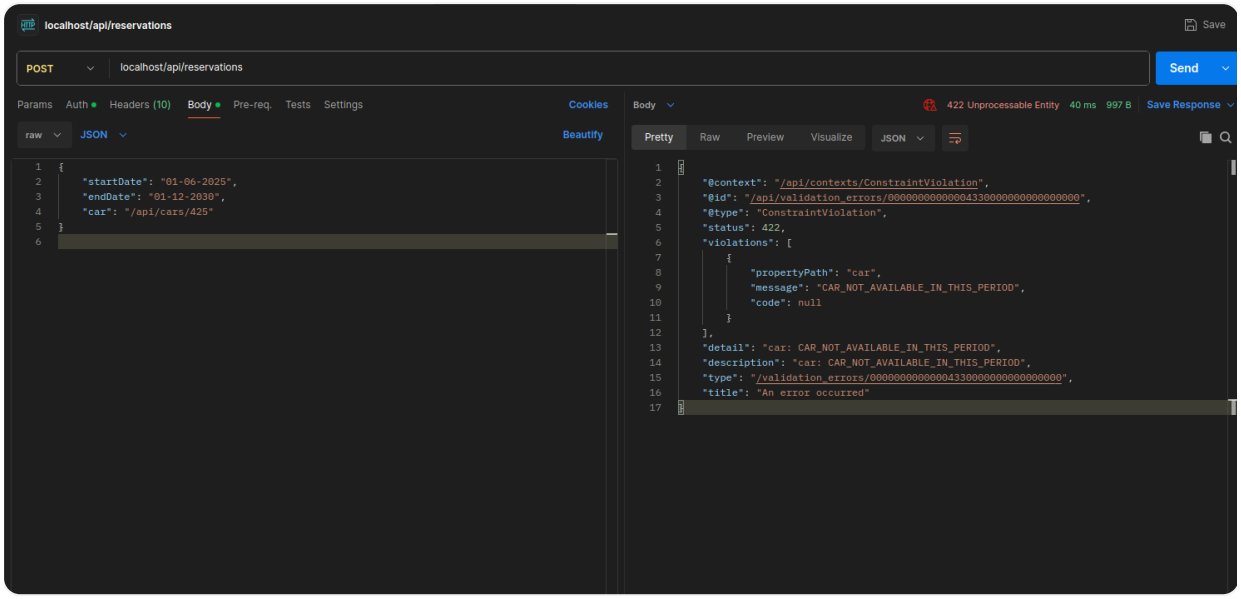

Les détails d'une réservation



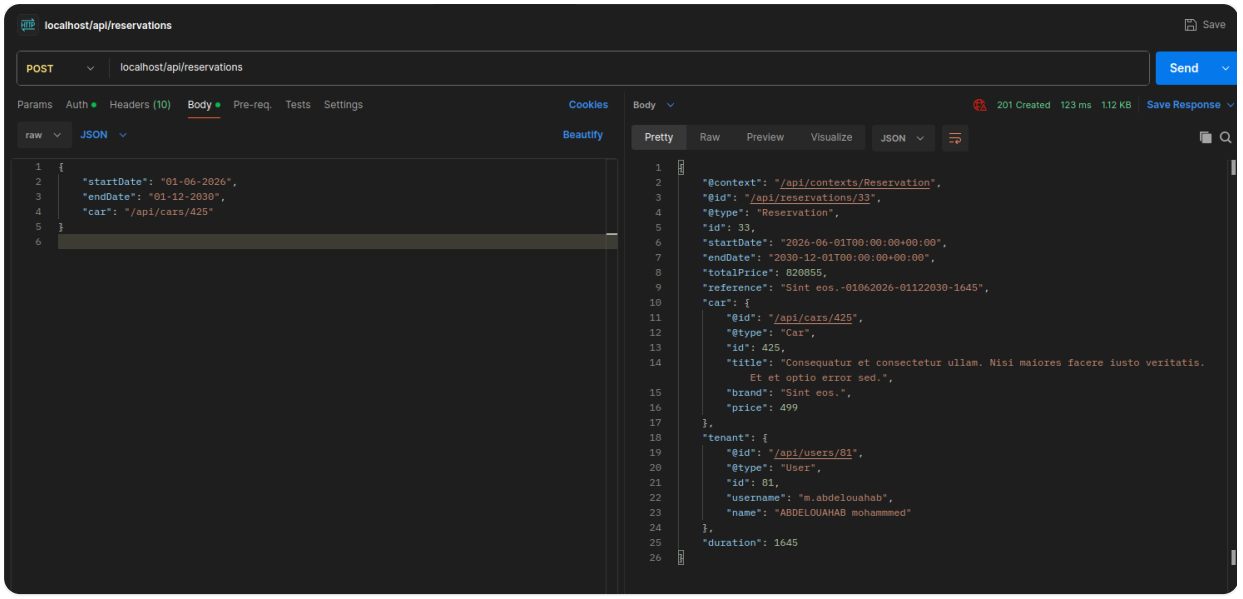

Les réservation d'une utilisateur connecté et passé en paramètre



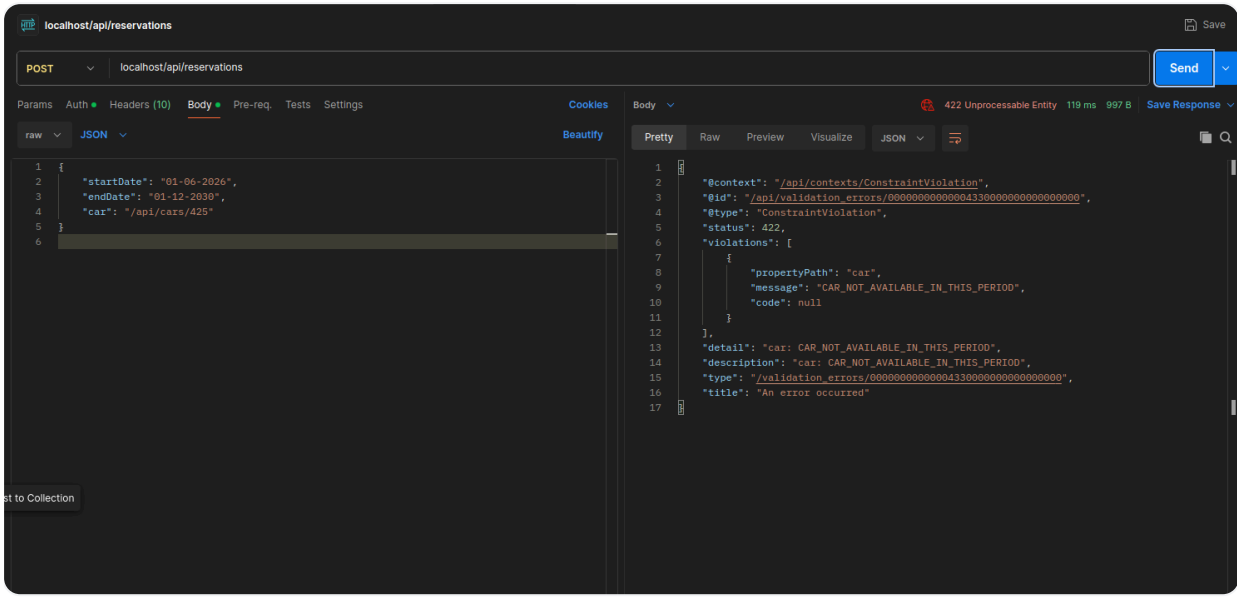

Les réservation d'une utilisateur connecté et passé en paramètre : lors de consultation des



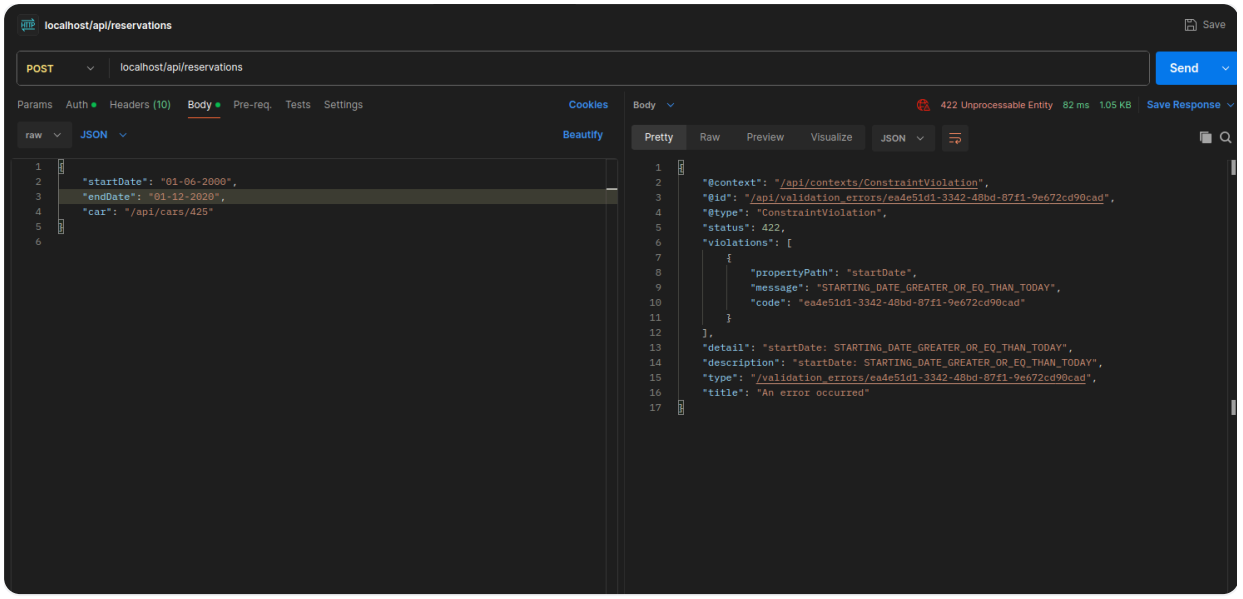

Cas d'erreur : Ajout d'une réservation




Ajout d'une réservation avec succès




Cas d'erreur : Ajout d'une réservation




Cas d'erreur : Ajout d'une réservation

# 6 - Les tests unitaire: