VALIDATION DES DONNEES AVEC JOI DANS UNE APPLICATION EXPRESS

Introduction

Une des règles de base de la programmation web c'est de ne jamais faire confiance aux donnés envoyés vers votre serveur. Il y aura toujours le risque d'écrire un numéro de téléphone là où on lui indique de mettre une adresse émail, etc. Cependant écrire ses propres fonctions de validations peut devenir vite très fastidieux. Ainsi on peut utiliser une librairie qui s'appelle Joi.

Joi est un langage de description de schémas et de validateur pour les objets JavaScript. Elle est facile à utiliser et permet de faire des validations vraiment complexes en peu de ligne de code. La validation peut se faire de manière synchrone ou asynchrone.

Joi permet de définir des règles de validation précises et de générer des messages d'erreur clairs. Le code utilisera des fonctions asynchrones pour gérer les interactions avec la base de données et les requêtes HTTP.

Le package

D'abord installer le package

```
npm install joi
```

Joi Validation Schema

Créer le fichier `validation/article.js` ayant ce code :


```
const Joi = require('joi');

const articleSchema = Joi.object({
  reference: Joi.string().required().trim().min(3).max(20).label('Reference
must be between 3 and 20 characters'),
  designation:
Joi.string().required().trim().min(3).max(50).label('Designation must be
between 3 and 50 characters'),
  prix: Joi.number().optional().min(0).label('Price must be non-negative'),
  marque: Joi.string().required().trim().min(3).max(30).label('Marque must be
between 3 and 30 characters'),
  qtestock: Joi.number().optional().min(0).label('Stock quantity must be non-
negative'),
  imageart: Joi.string().optional().trim(),

});
```

Explication :

const articleSchema = Joi.object({ ... });

Cette ligne crée un schéma d'objet Joi pour définir des règles de validation pour un objet article.

Règles de validation des propriétés :

reference:

Joi.string().required().trim().min(3).max(20).label(...)

Garantit qu'il s'agit d'une chaîne, obligatoire, dépourvue d'espaces, entre 3 et 20 caractères, et inclut un message d'erreur personnalisé.

designation:

Règles similaires à référence, avec des limites de longueur de 3 à 50 caractères.

prix:

Joi.number().optional().min(0).label(...)

Nombre facultatif, doit être égal ou supérieur à 0, avec un message d'erreur personnalisé.

marque:

Règles similaires à référence, avec des limites de longueur de 3 à 30 caractères.

qtestock:

Règles similaires à celles du prix, garantissant une quantité non négative.

imageart:

Joi.string().optional().trim()

Chaîne facultative, débarrassée des espaces.

Appel du Schema Joi

Dans notre route dans la méthode post on va faire appel au schéma Joi créé.

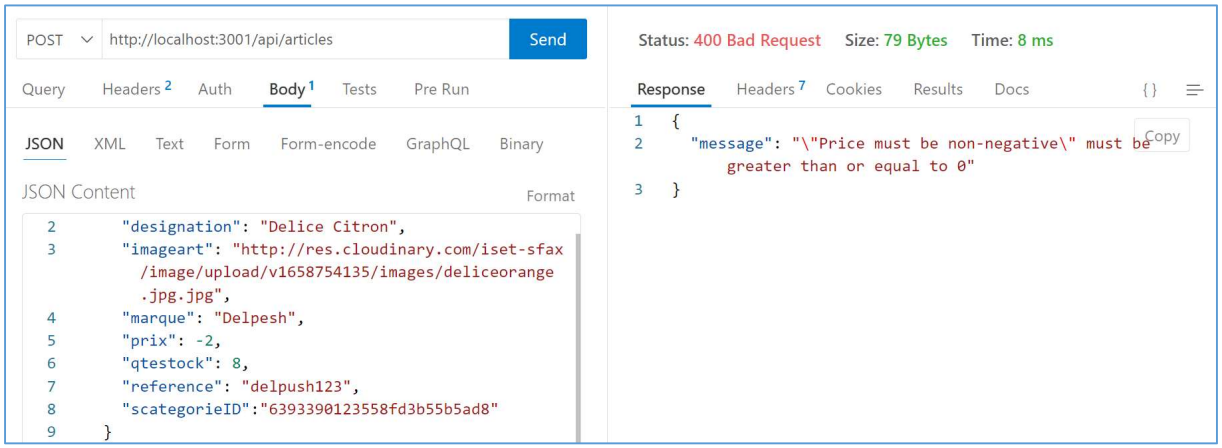
routes > JS article.route.js >

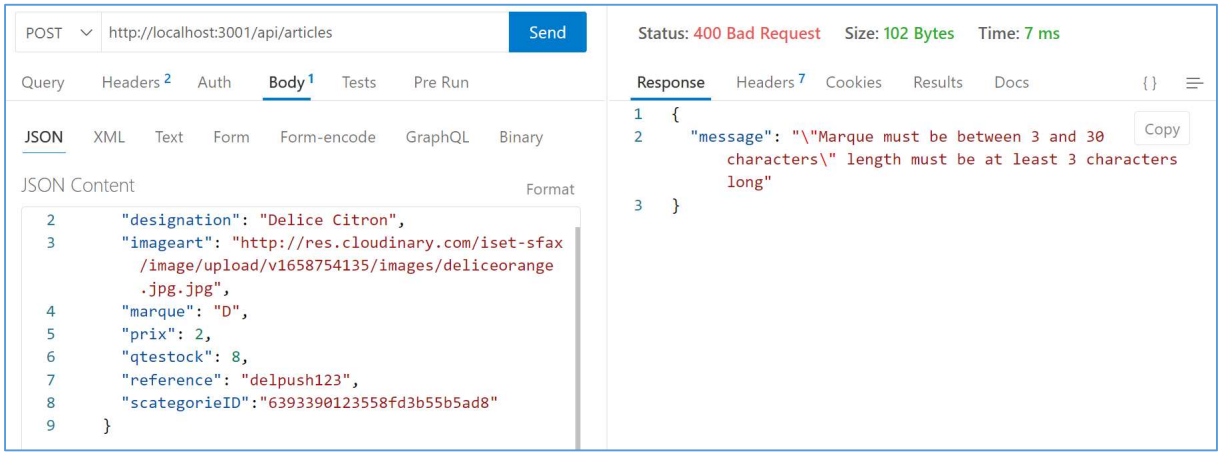
```
const articleSchema = require("../validation/article")
```

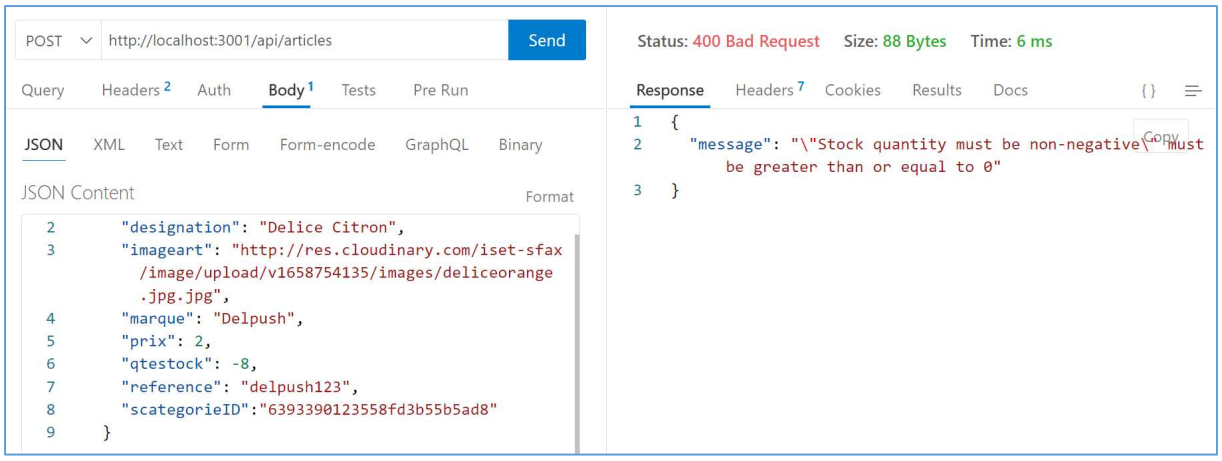
```
// créer un nouvel article
router.post('/', async (req, res) =>  {
```

```
    // Validation de la requête
  const { error } = articleSchema.validate(req.body);

  // Traitement en cas d'erreur de validation
  if (error) {
    return res.status(400).send({ message: error.details[0].message });
  }
  else {
    const nouvarticle = new Article(req.body)


    try {
        await nouvarticle.save();

        res.status(200).json(nouvarticle );
    } catch (error) {
        res.status(404).json({ message: error.message });
    }
}
});
```

Essais des contrôles







Valeurs référencées

Installer le package joi-objectid pour pouvoir tester si le type est objectID valable pour mongoDB :

```
npm i joi-objectid
```

validation > article.js >

```
const Joi = require('joi');
const JoiObjectId = require('joi-objectid')(Joi);
const Scategorie = require("../models/scategorie");

const articleSchema = Joi.object({
  reference: Joi.string().required().trim().min(3).max(20).label('Reference must be between 3 and 20 characters'),
  designation: Joi.string().required().trim().min(3).max(50).label('Designation must be between 3 and 50 characters'),
  prix: Joi.number().optional().min(0).label('Price must be non-negative'),
  marque: Joi.string().required().trim().min(3).max(30).label('Marque must be between 3 and 30 characters'),
  qtestock: Joi.number().optional().min(0).label('Stock quantity must be non-negative'),
  imageart: Joi.string().optional().trim(),
  scategorieID: JoiObjectId().custom(async (value, helpers) => {
    try {
      const scategory = await Scategorie.findById(value);
     if(scategory===null) {
         return helpers.error('Invalid Scategorie reference');
     }
     return value;
     } catch (error) {
      console.error('Error validating Scategorie reference:', error);
```

```
      return helpers.error('An error occurred while verifying the Scategorie
reference');
    }
  })
});

module.exports = articleSchema;
```

Explication :

custom(...) une fonction de validation personnalisée pour vérifier une référence Scategorie valide.

custom((value, helpers) => { ... });

Utilise une fonction de validation personnalisée : récupère le document Scategorie avec l'ID donné à l'aide de Scategorie.findById(value).

Si le document n'existe pas, renvoie une erreur de validation avec le message 'Référence Scategorie invalide'.

Si le document existe, renvoie la valeur.

routes > article.route.js >

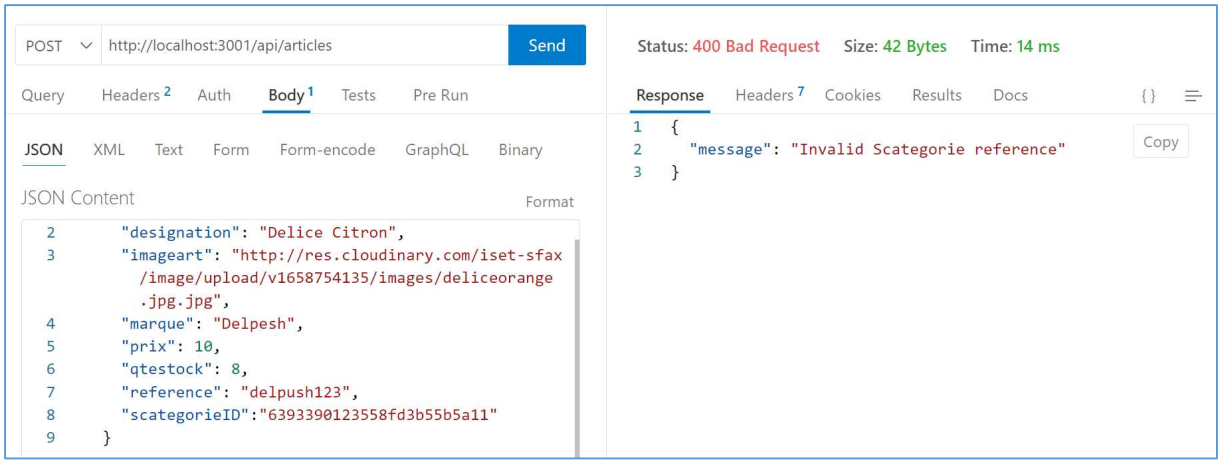
Custom retourne sa réponse dans value. ScategorieID obtenue dans le value est une promesse ayant le code qui contient le message d'erreur

```
router.post('/', async (req, res) =>  {

    // Validation de la requête
  const { error,value } = articleSchema.validate(req.body);
 const valueProm = await value.scategorieID;

  // Traitement en cas d'erreur de validation
  if (error) {
    return res.status(400).send({ message: error.details[0].message });
  }
  else {
    if(valueProm.code)
    { // cas custom scategorieID
        return res.status(400).send({ message: valueProm.code });
    }
    else{
    const nouvarticle = new Article(req.body)
      try {
        await nouvarticle.save();

        res.status(200).json(nouvarticle );

    } catch (error) {
```

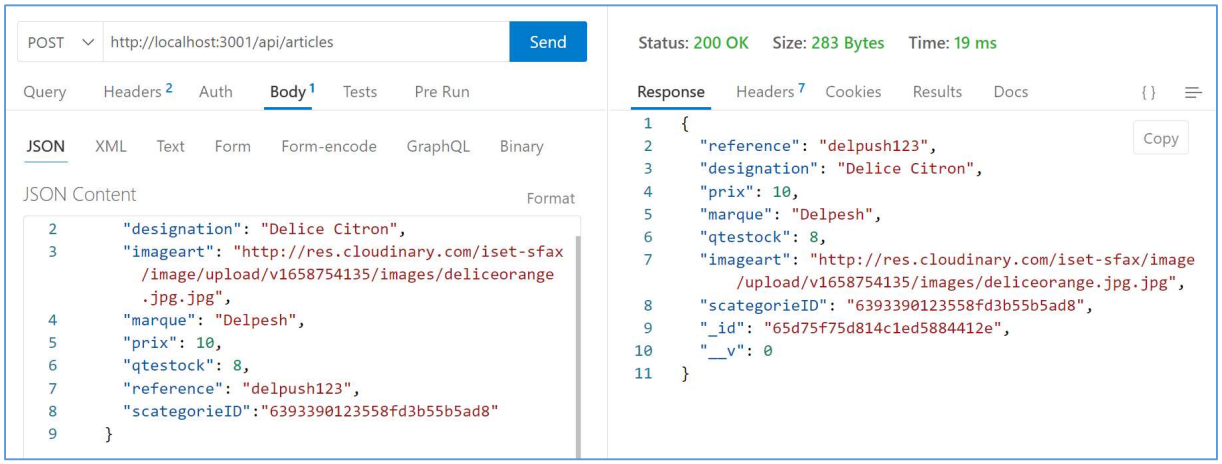
```
        res.status(404).json({ message: error.message });
    }
   }
  }
});
```

Essai du contrôle

On va essayer de donner une valeur inexistante pour scategorieID



Si existe :



Middleware

Créez ensuite un middleware qui intercepte les charges utiles des requêtes et les vérifie par rapport à un schéma fourni.

Créer middleware/SchemaValidator.js


```
const validationMiddleware = (articleSchema) => {
```

```
    return async (req, res, next) => {
        // Validation de la requête
  const { error,value } = articleSchema.validate(req.body);
  const valueProm = await value.scategorieID;

   // Traitement en cas d'erreur de validation
   if (error) {
     return res.status(400).send({ message: error.details[0].message });
   }
   else {
     if(valueProm.code)
     { // cas custom scategorieID
         return res.status(400).send({ message: valueProm.code });
     }
     else{
        // Data is valid, proceed to the next middleware
      next();
     }
    }
  }
}
  module.exports = validationMiddleware;
```

Explication :

Lorsqu'une requête est effectuée, le middleware appelle la méthode validate du schéma pour valider le corps de la requête. Si des erreurs de validation se produisent, le middleware envoie une réponse 400 Bad Request avec les messages d'erreur extraits des détails de l'erreur de validation.

En revanche, si la validation se déroule sans erreur, le middleware appelle la fonction next().

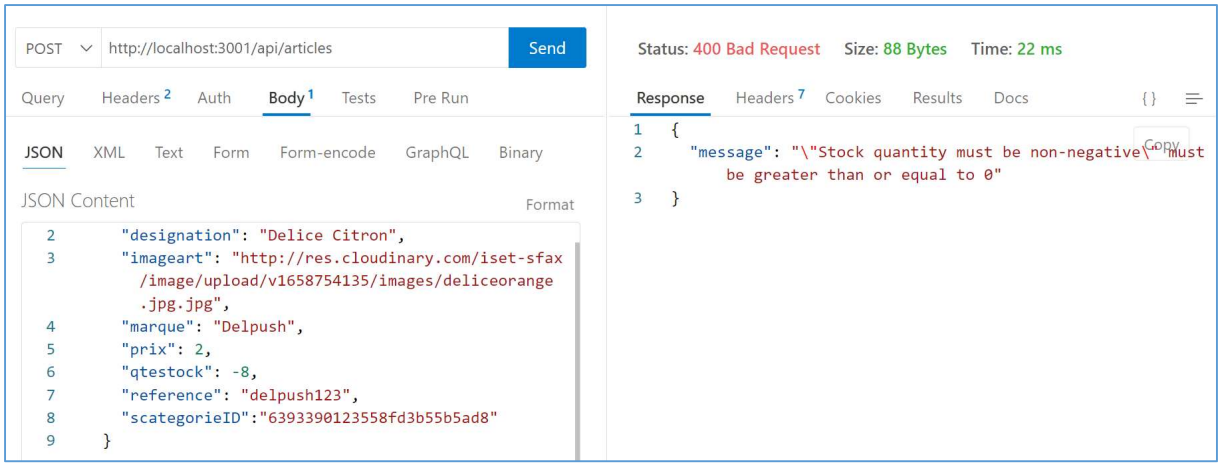
Enfin, exportez le validationMiddleware.

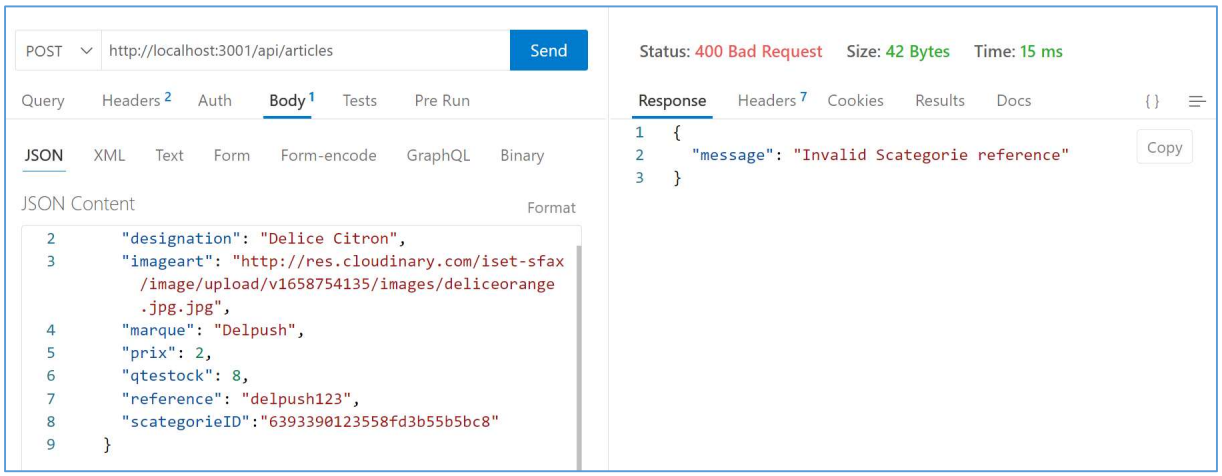
routes > article.route.js >

Importez validationMiddleware dans votre fichier article.route.js et configurez le middleware comme ceci :

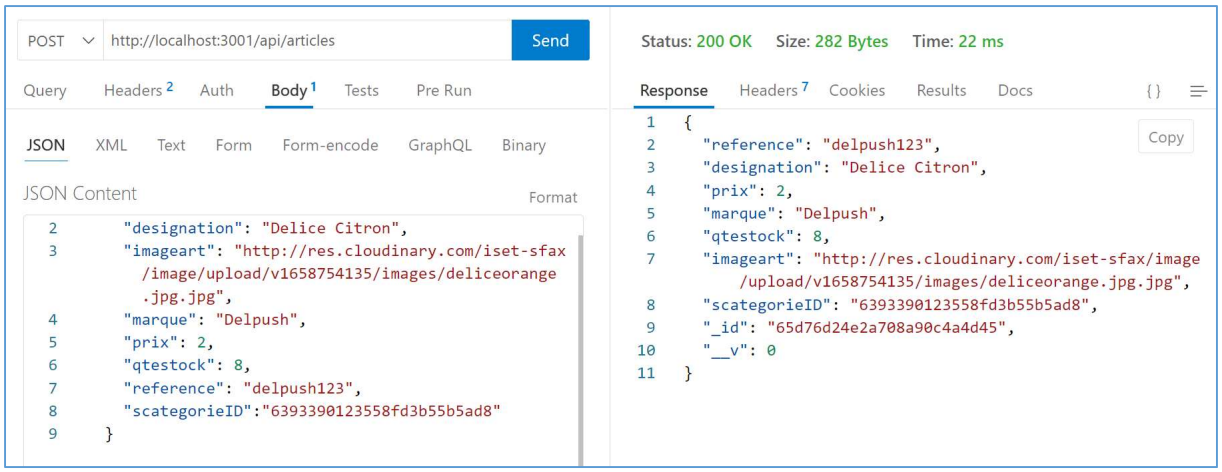
```
const articleSchema = require("../validation/article")

const validationMiddleware = require("../middleware/SchemaValidator.js")


router.post('/', validationMiddleware(articleSchema), async (req, res) =>  {

    const nouvarticle = new Article(req.body)
     try {
       await nouvarticle.save();
```

```
        res.status(200).json(nouvarticle );

    } catch (error) {
        res.status(404).json({ message: error.message });
    }

});
```





Lorsque tout est correct


POST
http://localhost:3001/api/articles
Send
Status: 200 OK
Size: 282 Bytes
Time: 22 ms
Query
Headers 2
Auth
Body 1
Tests
Pre Run
Response
Headers 7
Cookies
Results
Docs
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
Copy
"designation": "Delice Citron",
"imageart": "http://res.cloudinary.com/iset-sfax/image/upload/v1658754135/images/deliceorange.jpg.jpg",
"marque": "Delpush",
"prix": 2,
"qtestock": 8,
"reference": "delpush123",
"scategorieID":"6393390123558fd3b55b5ad8"
{
"reference": "delpush123",
"designation": "Delice Citron",
"prix": 2,
"marque": "Delpush",
"qtestock": 8,
"imageart": "http://res.cloudinary.com/iset-sfax/image/upload/v1658754135/images/deliceorange.jpg.jpg",
"scategorieID": "6393390123558fd3b55b5ad8",
"_id": "65d76d24e2a708a90c4a4d45",
"__v": 0
}