self-service-machine-api - STEP 10

Dans cette étape, nous allons mettre en place la validation des valeurs envoyées par le consommateur de notre API REST.

Pour cela, nous allons maintenant modifier notre modèle `ProductModel` .

Les 1ères règles de validation

Nous voulons nous assurer que :

- le nom ne soit composé que de lettres et du caractère espace.
- le prix soit un float.
- le nom et le prix soient renseignés (non null et non vide)

Voila les changements dans `models/products.mjs`

```
// https://sequelize.org/docs/v7/models/data-types/

const ProductModel = (sequelize, DataTypes) => {
  return sequelize.define(
    "Product",
    {
      id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true,
      },
      name: {
        type: DataTypes.STRING,
        allowNull: false,
        validate: {
          is: {
            args: /^[A-Za-z\s]*$/,
            msg: "Seules les lettres et les espaces sont autorisées.",
          },
          notEmpty: {
            msg: "Le nom ne peut pas être vide.",
          },
          notNull: {
            msg: "Le nom est une propriété obligatoire.",
          },
        },
      },
      price: {
        type: DataTypes.FLOAT,
        allowNull: false,
        validate: {
          isFloat: {
            msg: "Utilisez uniquement des nombres pour le prix.",
          },
          notEmpty: {
            msg: "Le prix ne peut pas être vide.",
          },
          notNull: {
            msg: "Le prix est une propriété obligatoire.",
          },
        },
      },
    },
    {
      timestamps: true,
      createdAt: "created",
      updatedAt: false,
```

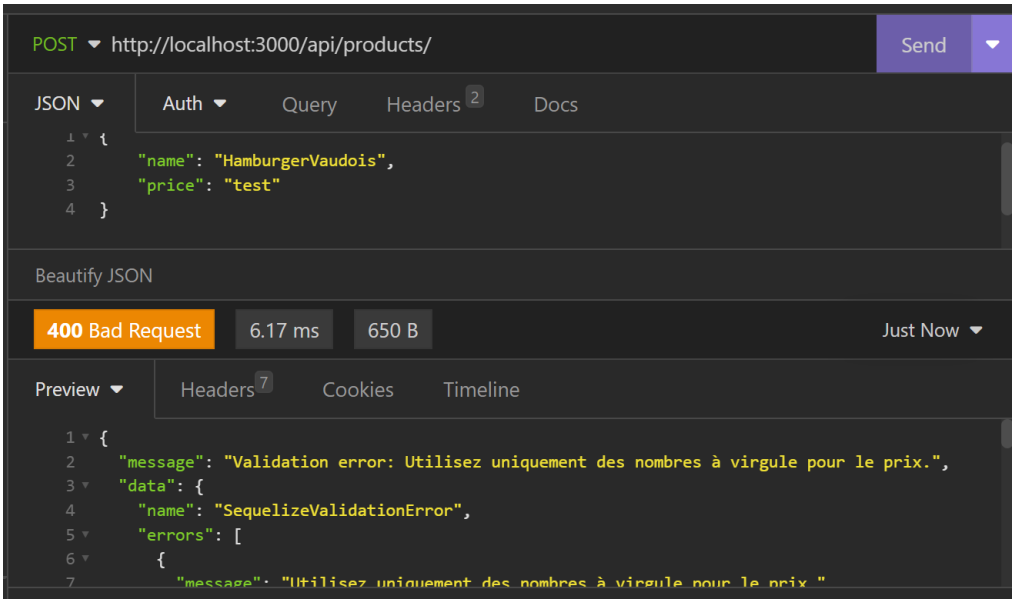
```
    }
  );
};

export { ProductModel };
```

Pour la route POST /api/products/ on a regarde si l'erreur est une instance de `ValidationError`. Si c'est le cas, on récupère et on affiche l'erreur personnalisée.

```
productsRouter.post("/", (req, res) => {
  Product.create(req.body)
    .then((createdProduct) => {
      // Définir un message pour le consommateur de l'API REST
      const message = `Le produit ${createdProduct.name} a bien été créé !`;

      // Retourner la réponse HTTP en json avec le msg et le produit créé
      res.json(success(message, createdProduct));
    })
    .catch((error) => {
      if (error instanceof ValidationError) {
        return res.status(400).json({ message: error.message, data: error });
      }
      const message =
        "Le produit n'a pas pu être ajouté. Merci de réessayer dans quelques instants.";
      res.status(500).json({ message, data: error });
    });
});
```

Maintenant si on fait une requête HTTP POST /api/products avec le json suivant

```
{
  "name": "HamburgerVaudois",
  "price": "test"
}
```

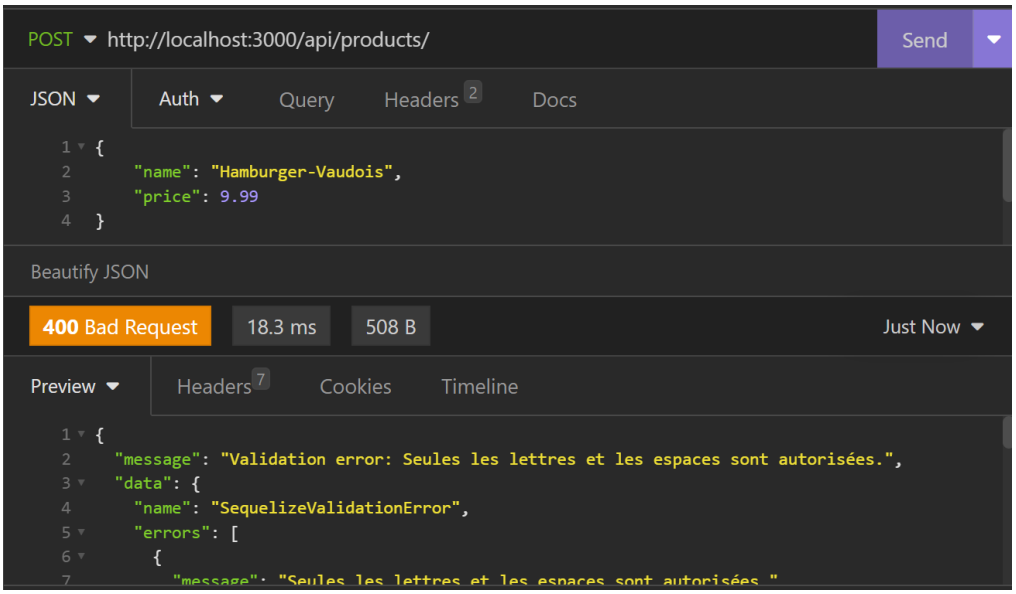


De même, si on fait une requête HTTP POST /api/products avec le json suivant :

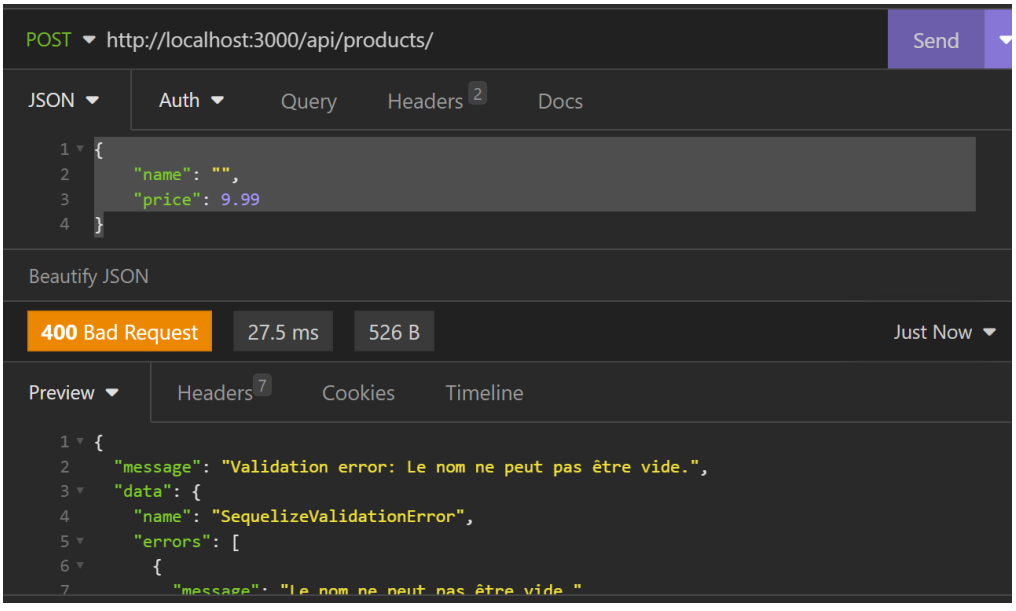
```
{
  "name": "Hamburger-Vaudois",
```

```
  "price": 9.99
}
```

ou encore avec un nom vide :




```
{
  "name": "",
  "price": 9.99
}
```



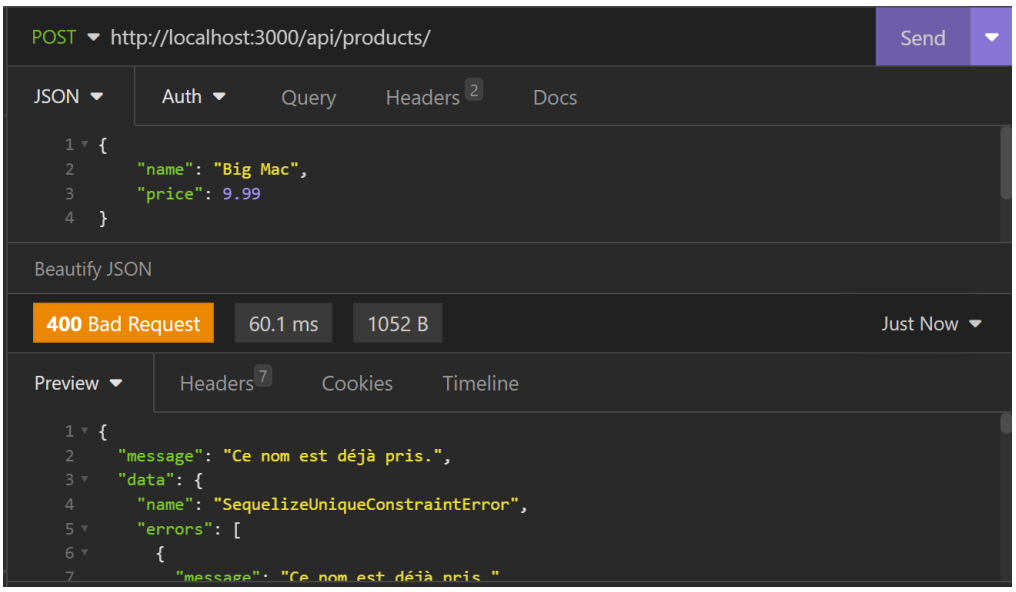
Les autres règles de validation

Nous voulons que :

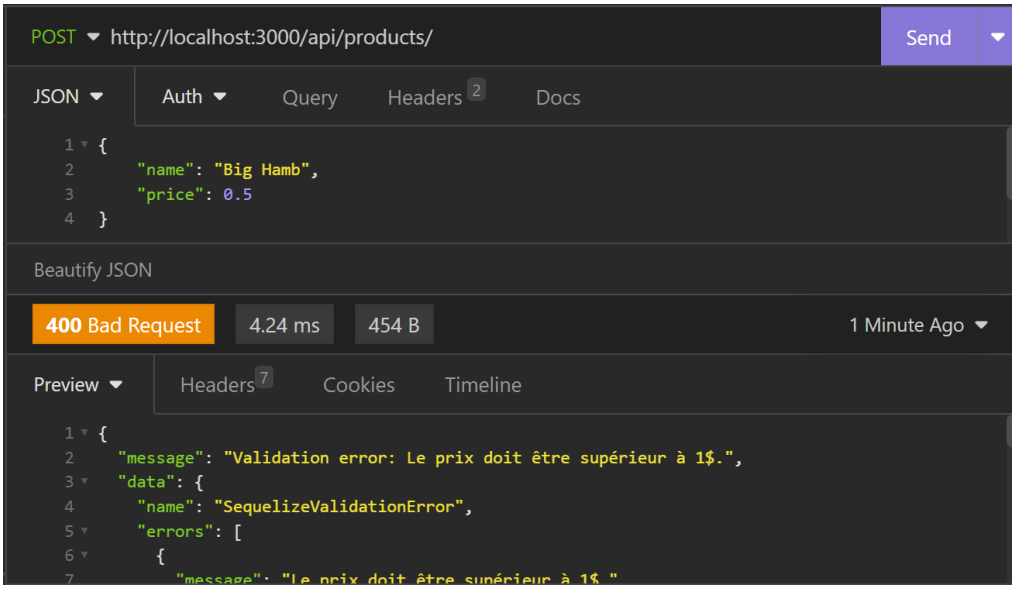
- le nom soit unique
- le prix soit supérieur à 1$ et inférieur à 1000$

```
// https://sequelize.org/docs/v7/models/data-types/

const ProductModel = (sequelize, DataTypes) => {
  return sequelize.define(
    "Product",
    {
      id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true,
      },
      name: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: {
          msg: "Ce nom est déjà pris.",
        },
        validate: {
          is: {
            args: /^[A-Za-z\s]*$/,
            msg: "Seules les lettres et les espaces sont autorisées.",
          },
          notEmpty: {
            msg: "Le nom ne peut pas être vide.",
          },
          notNull: {
            msg: "Le nom est une propriété obligatoire.",
          },
        },
      },
      price: {
        type: DataTypes.FLOAT,
        allowNull: false,
        validate: {
          isFloat: {
            msg: "Utilisez uniquement des nombres pour le prix.",
          },
          notEmpty: {
            msg: "Le prix ne peut pas être vide.",
          },
          notNull: {
            msg: "Le prix est une propriété obligatoire.",
          },
          min: {
            args: [1.0],
            msg: "Le prix doit être supérieur à 1$.",
          },
          max: {
            args: [1000.0],
            msg: "Le prix doit être inférieur à 1000$.",
          },
        },
      },
    },
    {
      timestamps: true,
      createdAt: "created",
      updatedAt: false,
    }
  );
};

export { ProductModel };
```

Il nous reste à tester que la validation pour le nom unique fonctionne :



et que la validation pour (par exemple) le prix inférieur à 1$ fonctionne également :



Passons à l'étape n°11