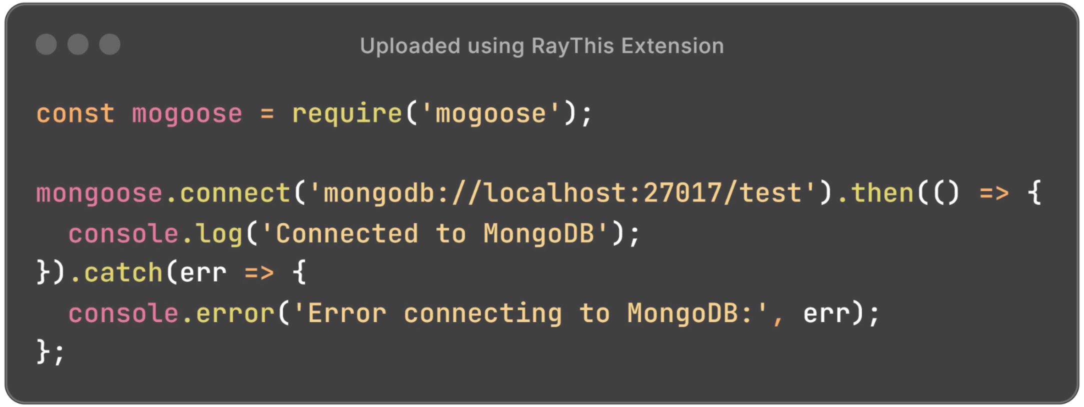
Chapitre 4
Gestion des données
Partie 2

CRUD en No SQL

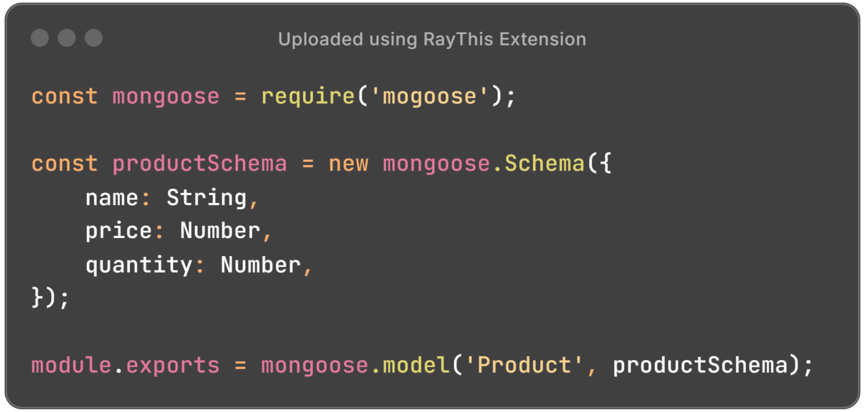
Connexion avec Mongoose



```javascript
const mogoose = require('mogoose');

mongoose.connect('mongodb://localhost:27017/test').then(() => {
  console.log('Connected to MongoDB');
}).catch(err => {
  console.error('Error connecting to MongoDB:', err);
};
```

Définir un schéma

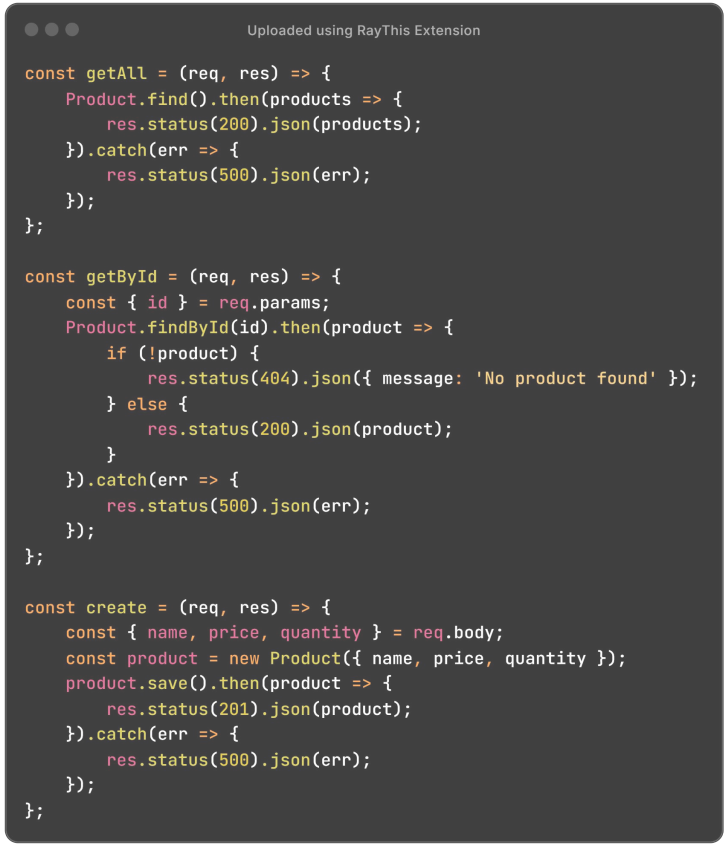


```
const mongoose = require('mogoose');

const productSchema = new mongoose.Schema({
    name: String,
    price: Number,
    quantity: Number,
});

module.exports = mongoose.model('Product', productSchema);
```

Faire des requêtes

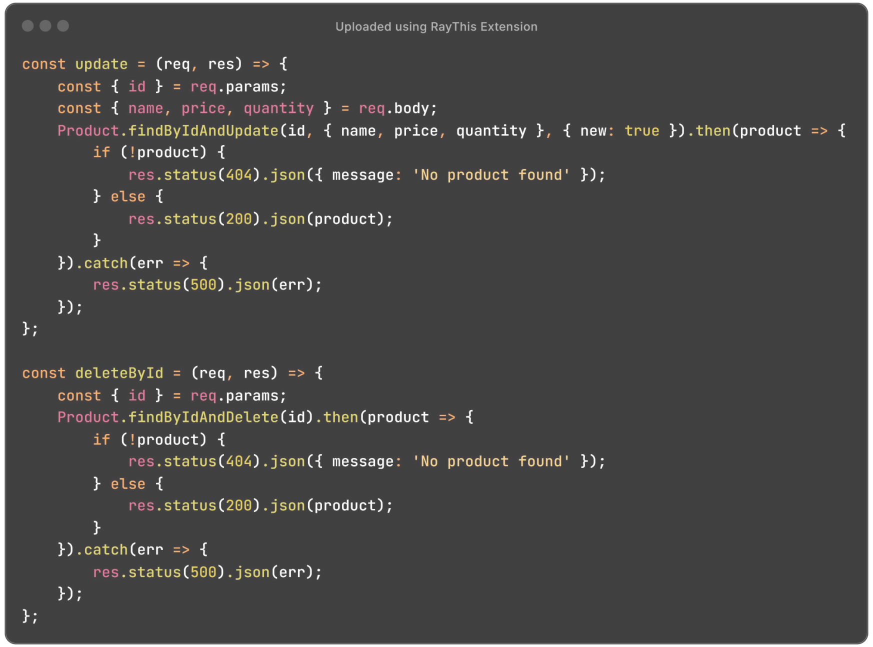


```javascript
const getAll = (req, res) => {
    Product.find().then(products => {
        res.status(200).json(products);
    }).catch(err => {
        res.status(500).json(err);
    });
};

const getById = (req, res) => {
    const { id } = req.params;
    Product.findById(id).then(product => {
        if (!product) {
            res.status(404).json({ message: 'No product found' });
        } else {
            res.status(200).json(product);
        }
    }).catch(err => {
        res.status(500).json(err);
    });
};

const create = (req, res) => {
    const { name, price, quantity } = req.body;
    const product = new Product({ name, price, quantity });
    product.save().then(product => {
        res.status(201).json(product);
    }).catch(err => {
        res.status(500).json(err);
    });
};
```

Faire des requêtes



```javascript
const update = (req, res) => {
    const { id } = req.params;
    const { name, price, quantity } = req.body;
    Product.findByIdAndUpdate(id, { name, price, quantity }, { new: true }).then(product => {
        if (!product) {
            res.status(404).json({ message: 'No product found' });
        } else {
            res.status(200).json(product);
        }
    }).catch(err => {
        res.status(500).json(err);
    });
};

const deleteById = (req, res) => {
    const { id } = req.params;
    Product.findByIdAndDelete(id).then(product => {
        if (!product) {
            res.status(404).json({ message: 'No product found' });
        } else {
            res.status(200).json(product);
        }
    }).catch(err => {
        res.status(500).json(err);
    });
};
```

Exercices

Chapitre 5
Middlewares

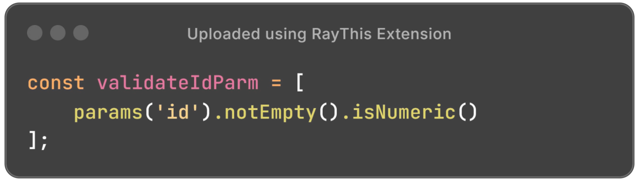
Express-Validator

Introduction à Express Validator

- Bibliothèque de validation pour Express.js.
- Utilisée pour valider les entrées des requêtes.

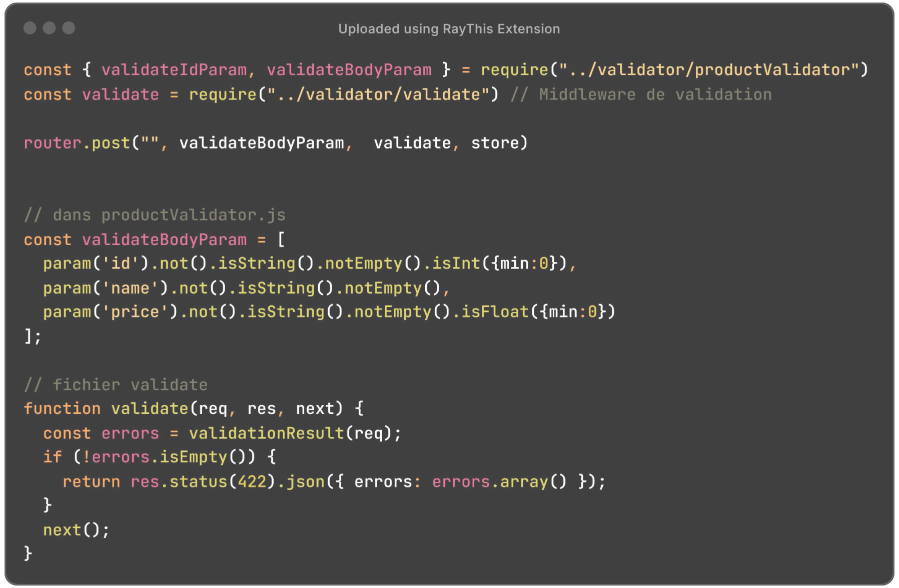
- /validator/productValidator.js
- /validator/validate.js

- Importation : const { body, param } = require('express-validator');
- Exemple de Validation de Paramètre d'URL



```javascript
const validateIdParm = [
    params('id').notEmpty().isNumeric()
];
```

Utilisation dans une route express


Uploaded using RayThis Extension

```js
const { validateIdParam, validateBodyParam } = require("../validator/productValidator")
const validate = require("../validator/validate") // Middleware de validation

router.post("", validateBodyParam,  validate, store)


// dans productValidator.js
const validateBodyParam = [
  param('id').not().isString().notEmpty().isInt({min:0}),
  param('name').not().isString().notEmpty(),
  param('price').not().isString().notEmpty().isFloat({min:0})
];

// fichier validate
function validate(req, res, next) {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(422).json({ errors: errors.array() });
  }
  next();
}
```

TP 3 : Validation des acquis