Piquante A ‘so pekocko’ API

Présentation MVP

Piquante A ‘so pekocko’ API

- Présentation API
- Structure du code
- Sécurité de l’API
- Mais encore...?
- Questions / Réponses

Présentation API /piquante

- Démonstration des fonctionnalités

Structure du code /piquante

- Organisation logique et découpage
- Explication des “modules”
- Explications des différentes routes

Structure du code/organisation /piquante

- Controllers (fonctions “métiers” pour chaque routes)
  - → un fichier par route (sauces et utilisateurs)
- Middleware (configuration de certains modules)
  - → un fichier par module (auth, limit, multer)
- Models (schéma de données MongoDb)
  - → un fichier par collection (sauces et utilisateurs)
- Routes (fonctions “logiques” de chaque routes)
  - → un fichier par route (sauces et utilisateurs)
- app.js (coeur du serveur qui rassemble les infos nécessaires)
- dossier Images (dossier qui stocke les images envoyé par les utilisateurs)

Controllers /sauce

```
// Récupérer la liste de toutes les sauces
exports.getAllSauces = (req, res, next) => {
    Sauce.find()
        .then(sauces => res.status(200).json(sauces))
        .catch(error => res.status(400).json({ error: error }))
}
```

Controllers /sauce

```
// Récupérer une seule sauce
exports.getOneSauce = (req, res, next) => {
    Sauce.findOne({  id: req.params.id })
        .then(sauce => res.status(200).json(sauce))
        .catch(error => res.status(404).json({ error: error }))
}
```

Controllers /sauce

```
// Créer une sauce
exports.createSauce = (req, res, next) => {
    const sauceObject = JSON.parse(req.body.sauce)
    delete sauceObject. id
    const sauce = new Sauce({
        ...sauceObject,
        imageUrl: `${req.protocol}://${req.get('host')}/images/${req.file.filename}`
    });
    sauce.save()
        .then(() => res.status(201).json({ message: 'Sauce enregistré !' }))
        .catch(error => res.status(400).json({ error }))
}
```

Controllers /sauce

```
// Modifier une sauce
exports.modifySauce = (req, res, next) => {
    const sauceObject = req.file ?
        {
            ...JSON.parse(req.body.sauce),
            imageUrl: `${req.protocol}://${req.get('host')}/images/${req.file.filename}`
        } : { ...req.body }
    Sauce.updateOne({  id: req.params.id }, { ...sauceObject,  id: req.params.id })
        .then(() => res.status(200).json({ message: 'Sauce modifiée !' }))
        .catch(error => res.status(400).json({ error }))
}
```

Controllers /sauce

```
// Supprimer une sauce
exports.deleteSauce = (req, res, next) => {
    Sauce.findOne({  id: req.params.id })
        .then(sauce => {
            const filename = sauce.imageUrl.split('/images/')[1]
            fs.unlink(`images/${filename}`, () => {
                Sauce.deleteOne({  id: req.params.id })
                    .then(() => res.status(200).json({ message: 'Sauce supprimée !' }))
                    .catch(error => res.status(400).json({ error: error }))
            })
        })
        .catch(error => res.status(500).json({ error }))
}
```

→ Controllers /sauce

```
// Aimer ou pas une sauce
exports.likeOrNot = (req, res, next) => {
    if (req.body.like === 1) {
        Sauce.updateOne({  id: req.params.id }, { $inc: { likes: req.body.like++ }, $push: { usersLiked: req.body.userId } })
            .then((sauce) => res.status(200).json({ message: 'Like ajouté !' }))
            .catch(error => res.status(400).json({ error }))
    } else if (req.body.like === -1) {
        Sauce.updateOne({ _id: req.params.id }, { $inc: { dislikes: (req.body.like++) * -1 }, $push: { usersDisliked:
req.body.userId } })
            .then((sauce) => res.status(200).json({ message: 'Dislike ajouté !' }))
            .catch(error => res.status(400).json({ error }))
    } else {
        Sauce.findOne({  id: req.params.id })
            .then(sauce => {
                if (sauce.usersLiked.includes(req.body.userId)) {
                    Sauce.updateOne({  id: req.params.id }, { $pull: { usersLiked: req.body.userId }, $inc: { likes: -1 } })
                        then((sauce) => { res.status(200).json({ message: 'Like supprimé !' }) })
                        catch(error => res.status(400).json({ error }))
                } else if (sauce.usersDisliked.includes(req.body.userId)) {
                    Sauce.updateOne({  id: req.params.id }, { $pull: { usersDisliked: req.body.userId }, $inc: { dislikes: -1 } })
                        then((sauce) => { res.status(200).json({ message: 'Dislike supprimé !' }) })
                        catch(error => res.status(400).json({ error }))
                }
            })
            .catch(error => res.status(400).json({ error }))
    }
}
```

Controllers /user

```
// Créer un compte utilisateur
exports.signup = (req, res, next) => {
    bcrypt.hash(req.body.password, 10)
        .then(hash => {
            const user = new User({
                email: req.body.email,
                password: hash
            })
            user.save()
                .then(() => res.status(201).json({ message: 'Utilisateur créé !' }))
                .catch(error => res.status(400).json({ error }))
        })
        .catch(error => res.status(500).json({ error }))
}
```

→ Controllers /user

```
// Se connecter à un compte utilisateur
exports.login = (req, res, next) => {
    User.findOne({ email: req.body.email })
        .then(user => {
            if (!user) {
                return res.status(401).json({ error: 'Utilisateur non trouvé !' })
            }
            bcrypt.compare(req.body.password, user.password)
                .then(valid => {
                    if (!valid) {
                        return res.status(401).json({ error: 'Mot de passe incorrect !' })
                    }
                    res.status(200).json({
                        userId: user._id,
                        token: jwt.sign(
                            { userId: user._id }, `${process.env.RND_TKN}`, { expiresIn: '24h' }
                        )
                    })
                })
                .catch(error => res.status(500).json({ error }))
        })
        .catch(error => res.status(500).json({ error }))
}
```

Middleware /auth

```
module.exports = (req, res, next) => {
    try {
        const token = req.headers.authorization.split(' ')[1]
        const decodedToken = jwt.verify(token, `${process.env.RND TKN}`)
        const userId = decodedToken.userId
        if (req.body.userId && req.body.userId !== userId) {
            throw 'User ID non valable !'
        } else {
            next()
        }
    } catch (error) {
        res.status(401).json({ error: error | 'Requête non-authentifiée !' })
    }
}
```

→ Middleware /limit

```
const rateLimit = require("express-rate-limit")

const limiter = rateLimit({
    windowMs: 5 * 60 * 1000,
    max: 3,
    message: "Trop de tentatives de connexion. Compte bloqué pour 5 minutes"
})

module.exports = { limiter }
```

→ Middleware /multer

```
const multer = require('multer')

const MIME TYPES = {
    'image/jpg': 'jpg',
    'image/jpeg': 'jpg',
    'image/png': 'png'
}

const storage = multer.diskStorage({
    destination: (req, file, callback) => {
        callback(null, 'images')
    },
    filename: (req, file, callback) => {
        name = file.originalname.split('.')[0]
        name = name.split(' ').join(' ')//underscore sur le join
        const extension = MIME TYPES[file.mimetype]
        callback(null, name + '_' + Date.now() + '.' + extension)
    }
})

module.exports = multer({ storage }).single('image')
```

Models /sauce

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

const sauceSchema = mongoose.Schema({
    userId: { type: String, required: true },
    name: { type: String, required: true },
    manufacturer: { type: String, required: true },
    description: { type: String, required: true },
    mainPepper: { type: String, required: true },
    imageUrl: { type: String, required: true },
    heat: { type: Number, required: true },
    likes: { type: Number, default: 0 },
    dislikes: { type: Number, default: 0 },
    usersLiked: { type: [String] },
    usersDisliked: { type: [String] },
})

module.exports = mongoose.model('Sauce', sauceSchema)
```

Models /user

```
const mongoose = require('mongoose')
const uniqueValidator = require('mongoose-unique-validator')

const userSchema = mongoose.Schema({
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true }
})

userSchema.plugin(uniqueValidator)

module.exports = mongoose.model('User', userSchema)
```

Routes /sauce

```
const express = require('express')
const router = express.Router()

const auth = require('../middleware/auth')
const multer = require('../middleware/multer-config')

const sauceCtrl = require('../controllers/sauce')

router.get('/', auth, sauceCtrl.getAllSauces)
router.get('/:id', auth, sauceCtrl.getOneSauce)
router.post('/', auth, multer, sauceCtrl.createSauce)
router.put('/:id', auth, multer, sauceCtrl.modifySauce)
router.delete('/:id', auth, sauceCtrl.deleteSauce)
router.post('/:id/like', auth, sauceCtrl.likeOrNot)

module.exports = router
```

Routes /user

```
const express = require('express')
const router = express.Router()
const max = require("../middleware/limit")

const userCtrl = require('../controllers/user')

router.post('/signup', userCtrl.signup)
router.post('/login', max.limiter, userCtrl.login)

module.exports = router
```

→ app.js

```
const express = require ('express')
const helmet = require('helmet')
const bodyParser = require('body-parser')
const mongoose = require('mongoose')
const path = require('path')

require('dotenv').config({ path: process.cwd() + '/.env' });

const sauceRoutes = require('./routes/sauce')
const userRoutes = require('./routes/user')
```

→ app.js

```
const app = express()
app.use(helmet())

mongoose.connect(`mongodb+srv://${process.env.DB USER}:${process.env.DB PASS}@${process.env.DB HOST}/${process.env.DB_NAM
E}?retryWrites=true&w=majority`, { useNewUrlParser: true, useUnifiedTopology: true, useCreateIndex: true})
    .then(() => console.log('Connexion à MongoDB réussie !'))
    .catch(() => console.log('Connexion à MongoDB échouée !'))

app.use((req, res, next) => {
    res.setHeader('Access-Control-Allow-Origin', '*')
    res.setHeader('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content, Accept, Content-Type,
Authorization')
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, PATCH, OPTIONS')
    next()
})

app.use(bodyParser.json())

app.use('/images', express.static(path.join(__dirname, 'images')))
app.use('/api/sauces', sauceRoutes)
app.use('/api/auth', userRoutes)

module.exports = app
```

Sécurité de l'API /piquante

- Top 10 OWASP
- Mesures mises en place

1. Les injections

- validations des entrées
- module mongoose

2. Piratage de sessions

- utilisation d'un token avec jsonwebtoken
- limitation de tentative de connexion avec express-rate-limit
- cryptage du mot de passe avec bcrypt

3. Exposition des données sensibles

- utilisations des routes CRUD seulement si nécessaire
- sécurisation des chaînes de caractères avec mongoose, bcrypt

4. Entités Externes XML (XXE)

→ utilisation du format json

5. Contournement contrôle d'accès

→ utilisation du module helmet

6. Mauvaise configuration

→ utilisation du module helmet

7. Cross-Site Scripting (XSS)

→ validation des données
→ token unique et ré-authentification pour chaque actions

8. Désérialisation non-sécurisée

→ utilisation du module helmet

9. Utilisations de composants non-sécurisés

→ recherche et documentation sur les modules utilisés

10. Manque de surveillance et monitoring

→ -

Mais encore /piquante

- Mesures de sécurité à implémenter

Mais encore/Reste à mettre en place /piquante

2. Piratage de session

→ mise en place d'un mot de passe fort, durée de vie du mot de passe

3. Exposition des données sensibles

→ utilisation du protocole HTTPS

6. Mauvaise configurations

→ personnalisé les codes erreurs et les messages associés
→ faire attention aux modèles de pages et au références direct

10. Manque de surveillance et de monitoring

→ mettre en place un monitoring et surveiller les logs du serveur

Questions / Réponses /piquante

> “Parfois, le plus gros problème dans une réponse, c’est la question.”
>
> — Jean Dion / *Le Devoir - 4 mai 2000*

Piquante

A ‘so pekocko’ API