

Piiquante



Contexte

- Mon client souhaite développer une application web de critique des sauces piquantes.
- Mission : Concevoir une API pour cette Application Web

Mission

À partir du front-end mis à ma disposition :

- Réaliser une API respectant les spécifications fournies
- Système d'inscription et de connexion
- Les utilisateurs doivent pouvoir publier des sauces, ajouter des images stockées sur leur ordinateur, éditer leurs posts, ainsi que les supprimer
- Fonction permettant le liker / disliker les sauces
- Exigence en matière de sécurité à respecter

Présentation

- 1 - Démonstration des fonctionnalités
- 2 - Présentation détaillée du code
- 3 - Bilan, Axes d'amélioration en termes de sécurité

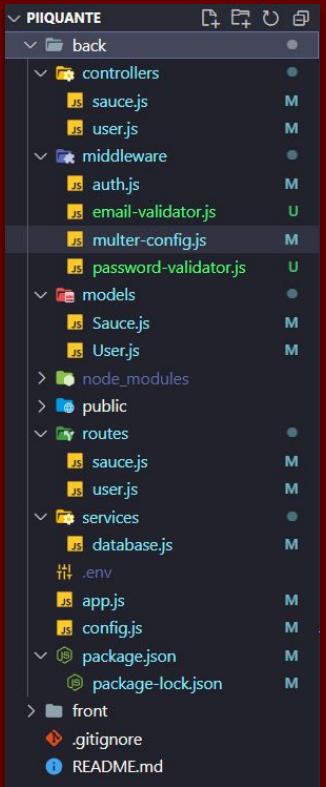


1 - Test

2- *Code*

Mise en place du projet

Structure :
Séparation des préoccupations



```
"dependencies": {
    "bcrypt": "^5.0.1",          Hashage password
    "crypto-js": "^4.1.1",       Cryptage Email
    "dotenv": "^16.0.1",         Données sensibles
    "express": "^4.18.1",
    "helmet": "^5.1.1",          Pack Sécurité (XSS...)
    "jsonwebtoken": "^8.5.1",      Authentification
    "mongoose": "^6.5.2",        Librairie MongoDB
    "mongoose-unique-validator": "^3.1.0",
    "multer": "^1.4.5-lts.1",     Upload images
    "password-validator": "^5.3.0", Contrôle
    "validator": "^13.7.0"
}
```

package.json :
Librairies utilisées

Application Express

Application | Imports, Express, Port

JS app.js

```
/* Imports - Librairie :  
require("dotenv").config(); // Dotenv (Sécurité)  
const express = require('express'); // Express  
const helmet = require('helmet'); // Helmet (Sécurité)  
const path = require('path') // Accès path serveur (route /images pour Multer)  
  
/* Imports - Code :  
require('./services/database'); // Database (connexion)  
const userRouter = require('./routes/user') // Routes User  
const sauceRouter = require('./routes/sauce') // Routes Sauce  
const { port, errorHandler } = require('./config') // Config : Port, Erreurs  
  
/* Express : Déclaration Express  
const app = express(); // Application Express  
  
// Config : Port, Erreurs  
app.on('error', errorHandler);  
app.on('listening', () => {  
    const address = app.address();  
    const bind = typeof address === 'string' ? 'pipe ' + address : 'port ' + port;  
    console.log('Listening on ' + bind);  
});  
  
Imports Librairie  
Imports Code  
Déclaration Express  
+ Gestion des erreurs  
+ Normalisation du port pour stabilité
```

JS config.js

```
/* Gestion du Port :  
// normalizePort renvoie un port valide, qu'il soit fourni sous la forme d'un numéro ou d'une chaîne  
normalizePort = val => {  
    const port = parseInt(val, 10);  
    if (isNaN(port)) { return val; }  
    if (port >= 0) { return port; }  
    return false;  
};  
exports.port = normalizePort(process.env.PORT || '3000');  
  
/* Gestion des Erreurs :  
// errorHandler recherche les différentes erreurs et les gère de manière appropriée  
exports.errorHandler = error => {  
    if (error.syscall !== 'listen') {  
        throw error;  
    }  
    const address = server.address();  
    const bind = typeof address === 'string' ? 'pipe ' + address : 'port: ' + port;  
    switch (error.code) {  
        case 'EACCES':  
            console.error(bind + ' requires elevated privileges.');//  
            process.exit(1);  
            break;  
        case 'EADDRINUSE':  
            console.error(bind + ' is already in use.');//  
            process.exit(1);  
            break;  
        default:  
            throw error;  
    };  
};  
  
Renvoie un port valide  
Gestion des erreurs  
Gain de stabilité = Plus facile à débuguer
```

Application | Database

JS app.js

```
/* Imports - Librairie :          Dotenv
require("dotenv").config(); // Dotenv (Sécurité)

/* Imports - Code :              Database
require('./services/database'); // Database (connexion)
```

Fichier .env

```
# Database MongoDB
DB_USERNAME=Frank
DB_PASSWORD=H0FzKZId0hfA1WIH
DB_CLUSTER=Lab
DATABASE=Piquante
```

```
# Json-Web-Token (Authentification)
JWT_KEY=LJqjK$DfaSpNsi8k5rk3?P3ERcakf?3FTEN7QFfs

# Crypto-JS (Cryptage email)
CRYPTOJS_KEY=xJJT$M$p6K?8tC$BbtMs$oNjtR4FF!LjmQTqqkXB
```

Variables d'environnement pour connexion Database

JS database.js

```
/* Mongoose import
const mongoose = ...require("mongoose"); Mongoose

/* (Sécurité) Variables d'environnement ( => .env )
const username = process.env.DB_USERNAME;
const password = process.env.DB_PASSWORD;
const cluster = process.env.DB_CLUSTER      Variables
const database = process.env.DATABASE;      environnement

/* MongoDB adresse
const uri = `mongodb+srv://${username}:${password}@${cluster}.zdqnwhr.mongodb.net/${database}?retryWrites=true&w=majority`; Inclusion variables

/* MongoDB connexion Connexion
mongoose.connect(uri)
  .then(() => console.log('Connexion à MongoDB réussie !'))
  .catch((err) => console.error('Connexion à MongoDB échouée !', err));

/* Exports
module.exports = { mongoose };
```

Application | Middleware et lancement

JS

app.js

```
/* Middleware :  
// Helmet : Protège l'application de certaines vulnérabilités en configurant de manière appropriée des headers HTTP  
app.use(helmet({ crossOriginResourcePolicy: { policy: "same-site" } }));  
                                              Helmet : Package sécurité  
// Paramétrage des headers HTTP :      Configuration headers  
app.use((req, res, next) => {  
    // Accéder à notre API depuis n'importe quelle origine   Origines autorisées  
    res.setHeader('Access-Control-Allow-Origin', '*');  
    // Ajouter les headers mentionnés aux requêtes envoyées vers notre API   Ajout headers aux requêtes  
    res.setHeader('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content, Accept, Content-Type,  
    Authorization');  
    // Envoyer des requêtes avec les méthodes mentionnées   Méthodes autorisées  
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, PATCH, OPTIONS');  
    next();  
});  
  
// Parser : Analyse le corps d'une requête HTTP, assemble les données, crée un objet body exploitable  
app.use(express.json())  Parser  
  
// Servir des fichiers statiques (images...)  
app.use('/public/images', express.static(path.join(__dirname, '/public/images')))  
  
app.use(userRouter); // Routeur User      Routes  
  
app.use(sauceRouter); // Routes Sauce  
  
/* Lancement :  
app.listen(port, () => {  
    console.log(`App listening on port ${port}`);  
});  
                                              Inclue createServer  
                                              Lie application au port  
                                              Écoute les connexions  
/* Exports :  
module.exports = app;
```

User

User | Model + Router

JS models/user.js

```
/* Mongoose import
const mongoose = require('mongoose');

/* Plugin "mongoose-unique-validator" :
/* Facilite la gestion des erreurs de la propriété "unique" ;
Prévalide les informations avant de les enregistrer */
const uniqueValidator = require('mongoose-unique-validator');

/* Schéma utilisateur
const userSchema = mongoose.Schema({
  email: { type: String, required: true, unique: true}, // Email unique
  password: { type: String, required: true}
});

/* Appliquer "mongoose-unique-validator" à "userSchema"
userSchema.plugin(uniqueValidator);

/* Exports
module.exports = mongoose.model('User', userSchema);
```

Pour erreurs
mail unique

Schema User

Ajout plugin

JS routes/user.js

```
/* Imports :                                     Imports : middleware, controller
const express = require('express');
const emailValidator = require('../middleware/email-validator')
const passwordValidator = require('../middleware/password-validator')
const userController = require('../controllers/user')

/* Routeur :
// Déclarer routeur User
const userRouter = express.Router();

// POST => Permet à l'utilisateur de créer un compte
userRouter.post('/api/auth/signup', emailValidator,
passwordValidator, userController.signup)
// POST => Permet à l'utilisateur de s'identifier
userRouter.post('/api/auth/login', userController.login)

/* Exports :
module.exports = userRouter;
```

Imports : middleware, controller

Router

Routes

User | Middleware pour Inscription

package.json

```
latest  
"password-validator": "^5.3.0",  
latest  
"validator": "^13.7.0"      Rappel
```

JS middleware/email-validator.js

```
/* Import  
const emailValidator = require('validator');  
  
/* Fonction  
module.exports = (req, res, next) => {  
    const { email } = req.body;  
    if (emailValidator.isEmail(email)) {  
        next();  
    } else {  
        return res  
            .status(403)  
            .json({ error: `${email} n'est pas un email valide` })  
    }  
}
```

Contrôle des entrées
pour "email"

JS middleware/password-validator.js

```
/* Import  
const passwordValidator = require('password-validator');  
  
/* Déclaration  
const passwordSchema = new passwordValidator();  
  
/* Critères à respecter, à accorder avec front-end  
passwordSchema  
    .is().min(8)                                // Minimum length 8  
    .is().max(100)                               // Maximum length 100  
    .has().uppercase()                           // Must have uppercase letters  
    .has().lowercase()                           // Must have lowercase letters  
    .has().digits(1)                             // Must have at least 1 digits  
    .has().not().spaces()                        // Should not have spaces  
    .is().not().oneOf(['Passw0rd', 'Password123']); // Blacklist these values  
  
/* Critères de validation  
    à définir  
  
/* Exports  
module.exports = (req, res, next) => {  
    if (passwordSchema.validate(req.body.password)) {  
        next();  
    } else {  
        return res.status(403).json({ error: `Le mot de passe n'est pas assez fort ${passwordSchema.validate('req.body.password', { list: true })}` })  
    }  
};
```

Contrôle des entrées
pour "password"

User controller | Imports + Inscription

JS controllers/user.js

```
/* Imports :  
// Modèle "User"  
const User = require('../models/User'); Model  
// Crypto-js (chiffrage email)  
const cryptojs = require('crypto-js'); Email  
// Bcrypt (hashage password)  
const bcrypt = require('bcrypt'); Password  
// Json-Web-Token (authentification)  
const jwt = require('jsonwebtoken'); Token  
// Clé secrète JWT (.env)  
const jwtKey = process.env.JWT_KEY;  
// Clé secrète Crypto-js (.env) Clés secrètes  
const cryptojsKey = process.env.CRYPTOJS_KEY;
```

Database : objet User

_id: ObjectId("63013b1c3c14bfea6e80537c")
email: "9bf376ffaf44d9d940d7869b3539dfb729bec51bc0bc8c0f984a52683d420b18"
password: "\$2b\$10\$p4YlnSBBHNVozC7fdA2vuuJEuGcYi/Sg/SPJftSiXEJKbw323ejPW"

Informations protégées

Fonction 'signup'

```
// Inscription => Créer un nouveau utilisateur  
exports.signup = (req, res, next) => {  
    // Chiffrer email  
    const emailCryptojs = [cryptojs.HmacSHA256(req.body.email, cryptojsKey)]  
    // Hasher password (10 fois)  
    bcrypt.hash(req.body.password, 10) Hashage password  
    // Transmettre mail/pass à un objet User  
    .then(hashedPassword => {  
        // Crée un new utilisateur  
        const user = new User({  
            email: emailCryptojs, Objet User  
            password: hashedPassword  
        });  
        // Méthode "save" : Engistrer dans la base de données  
        user.save() Enregistrement  
        // 201 = Création de ressources  
        .then(() => res.status(201).json({ message: 'Utilisateur créé !' }))  
        // 500 = Erreur Serveur  
        .catch(error => res.status(500).json({ error }));  
    })  
    .catch(error => res.status(501).json({ error }));  
};
```

Chiffrage Email

User controller | Connexion

JS controllers/user.js

```
/* Imports :                                Rappel imports
// Modèle "User"
const User = ...require('../models/User');
// Crypto-js (chiffrage email)
const cryptojs = require('crypto-js');
// Bcrypt (hashage password)
const bcrypt = require('bcrypt');
// Json-Web-Token (authentification)
const jwt = require('jsonwebtoken');
// Clé secrète JWT (.env)
const jwtKey = process.env.JWT_KEY;
// Clé secrète Crypto-js (.env)
const cryptojsKey = process.env.CRYPTOJS_KEY;
```

```
// Connexion => Correspondance mail et password
exports.login = (req, res, next) => {
    // Recuperer mail crypté, le convertir en string
    const emailCryptojs = cryptojs.HmacSHA256(req.body.email, cryptojsKey).toString()
    // Chercher correspondance dans la database (Méthode "findOne")
    User.findOne({ email: emailCryptojs.toString() })
        .then(user => {
            if (user === null) {
                // Pas de correspondance, échec de la connexion
                res.status(401).json({ message: 'L\'adresse e-mail que vous avez saisie n\'est associée à aucun compte' });
            } else {
                // Si correspondance mail, comparer passwords requête/database (Fonction "compare" de Bcrypt)
                bcrypt.compare(req.body.password, user.password)
                    .then(valid => {
                        if (!valid) {
                            // Invalide, échec de la connexion
                            res.status(401).json({ message: 'Paire identifiant/mot de passe incorrecte' });
                        } else {
                            // Valide, envoi d'un token d'authentification (Méthode "sign")
                            res.status(200).json({
                                userId: user.id,
                                token: jwt.sign(
                                    // Arguments : userId, clé cryptage, durée de validité
                                    { userId: user._id },
                                    jwtKey,
                                    { expiresIn: '24h' }
                                )
                            });
                        }
                    })
                    .catch(error => res.status(500).json({ error }));
            }
        })
        .catch(error => res.status(501).json({ error }));
};
```

Fonction 'login'

Correspondance Email : requête / database

Correspondance Password : requête / database

Token web d'authentification

Sauce

Sauce | Model + Router

JS models/sauce.js

```
/* Mongoose import
const mongoose = require('mongoose');                                Mongoose

/* Schéma Sauce
const sauceSchema = mongoose.Schema({                               Schema "Sauce"
  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, required: true},
  dislikes: { type: Number, required: true},
  usersLiked: { type: [String], required: true},
  usersDisliked: { type: [String], required: true},
});

/* Exports
module.exports = mongoose.model('Sauce', sauceSchema);
```

Export Model

JS routes/sauce.js

```
/* Imports
const express = require('express'); // Express                                Imports : middleware, controller
const auth = require('../middleware/auth') // Authentification (sécurité)
const multer = require('../middleware/multer-config') // Multer (images)
const sauceController = require('../controllers/sauce') // Sauce contrôleur

/* Routeur
const sauceRouter = express.Router(); // Déclarer routeur Sauce               Router

// POST => Publier une sauce (/api/sauces)
sauceRouter.post('/api/sauces', auth, multer, sauceController.createSauce)

// GET => Afficher toutes les sauces (/api/sauces)
sauceRouter.get('/api/sauces', auth, sauceController.getAllSauces)

// GET => Afficher une sauce en particulier (/api/sauces/:id)            Routes
sauceRouter.get('/api/sauces/:id', auth, sauceController.getOneSauce)

// PUT => Modifier une sauce (/api/sauces/:id)
sauceRouter.put('/api/sauces/:id', auth, multer, sauceController.modifySauce)

// DELETE => Supprimer une sauce (/api/sauces/:id)
sauceRouter.delete('/api/sauces/:id', auth, sauceController.deleteSauce)

// POST => Liker/Disliker une sauce (/api/sauces/:id/like)
sauceRouter.post('/api/sauces/:id/like', auth, sauceController.likeDislikeSauce)

/* Exports
module.exports = sauceRouter;
```

Export Router

Sauce Middleware 1/2 | Authentification

JS middleware/auth.js

```
/* Imports
const jwt = require('jsonwebtoken'); // JSON-Web-Token
const jwtKey = process.env.JWT_KEY; // Clé secrète JWT (.env)

/* Fonction :

module.exports = (req, res, next) => {
    try {
        // Récupérer header "Authorization" + Extraire le Token (split après "bearer")
        const token = req.headers.authorization.split(' ')[1];
        // Décoder le Token : méthode "verify" + arguments(token, clé secrète)
        const decodedToken = jwt.verify(token, jwtKey);
        // Extraire "userId" du token
        const userId = decodedToken.userId;
        // Renseigner "userId" afin que nos routes puissent l'exploiter
        req.auth = {
            userId: userId      Correspondance user Id
        };
        next();    passer au middleware suivant
    } catch (error) {
        res.status(403).json({ error })
    }
};

// Après import, ajouter "auth" aux routes Sauce, avant chaque controller.
```

Import
json-web-token
+ Clé

Récupérer Token (login) dans header
Décoder Token avec clé
Récupérer userId du Token

Dotenv

```
# Json-Web-Token (Authentification)
JWT_KEY=LJqqJK$DfaSpNsI8k5rk3?P3ERcakf?3FTEN7QFfs
```

JWT dans Headers

▼ Request Headers

[View source](#)

Accept: application/json, text/plain, /*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,fr-FR;q=0.8,fr;q=0.7
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJc2VyS
WQiOjI2MzAxM2IxYzNjMTRiZmVhNmU4MDUzM2MiLCJpYXQiOjE2NjExODE5NTAsImV
4cCI6MTY2MTI2ODM1MH0.PwNUEQjfJDdTayyrBv1ScJHasyfPCgrZ-Omk6UG4lns

JS routes/sauce.js -> Ajout de auth

```
// POST => Publier une sauce (/api/sauces)
sauceRouter.post('/api/sauces', auth, multer, sauceController.createSauce)
// GET => Afficher toutes les sauces (/api/sauces)
sauceRouter.get('/api/sauces', auth, sauceController.getAllSauces)
// GET => Afficher une sauce en particulier (/api/sauces/:id)
sauceRouter.get('/api/sauces/:id', auth, sauceController.getOneSauce)
// PUT => Modifier une sauce (/api/sauces/:id)
sauceRouter.put('/api/sauces/:id', auth, multer, sauceController.modifySauce)
// DELETE => Supprimer une sauce (/api/sauces/:id)
sauceRouter.delete('/api/sauces/:id', auth, sauceController.deleteSauce)
// POST => Liker/Disliker une sauce (/api/sauces/:id/like)
sauceRouter.post('/api/sauces/:id/like', auth, sauceController.likeDislikeSauce)
```

Sauce Middleware 2/2 | Multer configuration

JS middleware/multer-config.js

```
/* Import Multer : Package gestion de fichiers          Import Multer
const multer = require('multer');

/* Dictionnaire Mime-types (extensions)
const MIME_TYPES = {
  'image/jpg': 'jpg',    Pour ajouter extension pendant
  'image/jpeg': 'jpg',   écriture nom fichier
  'image/png': 'png'
};

/* Objet de configuration
// .diskStorage() configure : (1) Dossier de réception (2) Nom du fichier
const storage = multer.diskStorage({
  destination: (req, file, callback) => {
    // Callback (null = pas d'erreur) | Dossier de réception
    callback(null, 'public/images');
  },
  filename: (req, file, callback) => {
    // Configurer (nouveau) nom du fichier
    // Remplacer les " " (espaces) par des "_" (underscores) :
    const name = file.originalname.split(' ').join('_');
    // Ajouter une extension à partir du "mimetype"
    const extension = MIME_TYPES[file.mimetype];  Extension
    // Callback (null = pas d'erreur) | name + timestamp (milliseconde = unique) + '.' + extension
    callback(null, name + Date.now() + '.' + extension);
  }
});                                              Ajout milliseconde pour nom unique

/* Exports
// (Ajouter aux routes/sauce.js pour enregistrer images au système de fichier du serveur)
module.exports = multer({ storage }).single('image');          Export image
// = Appel multer({ notre objet storage }) | Méthode .single('image') = Fichier unique/image
```

Fonction diskStorage

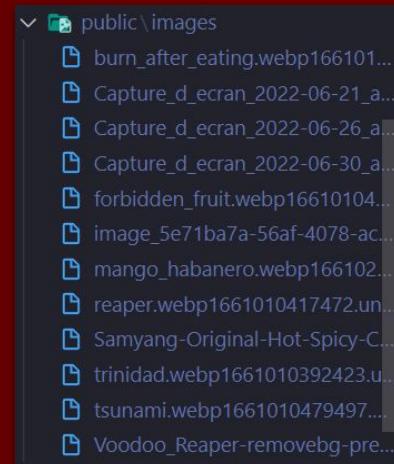
Argument 1 : Destination

Argument 2 : Filename

JS routes/sauce.js -> Ajout de multer

```
// POST => Publier une sauce (/api/sauces)
sauceRouter.post('/api/sauces', auth, multer, sauceController.createSauce)

// PUT => Modifier une sauce (/api/sauces/:id)
sauceRouter.put('/api/sauces/:id', auth, multer, sauceController.modifySauce)
```



Dossier de réception

Sauce Controller | Afficher

Toutes les sauces

JS controllers/sauce.js

```
/* Afficher toutes les sauces => GET
exports.getAllSauces = (req, res, next) => {
  // Appliquer méthode "find" à notre modèle Mongoose
  Model.Sauce.find()
    .then(sauces => res.status(200).json(sauces))
    .catch(error => res.status(400).json({ error }));
}
```

Une seule sauce

JS controllers/sauce.js

```
/* Afficher une sauce en particulier => GET
exports.getOneSauce = (req, res, next) => {
  // Appliquer méthode "findOne" à notre modèle + Paramètre id
  Model.Sauce.findOne({ _id: req.params.id })
    .then(thing => res.status(200).json(thing))
    .catch(error => res.status(404).json({ error }));
}
```

[ALL SAUCES](#) [ADD SAUCE](#)

HOT TAKES
THE WEB'S BEST HOT SAUCE REVIEWS

LOGOUT

THE SAUCES

VOODOO REAPER Heat: 9/10	TRINIDAD Heat: 10/10	FORBIDDEN FRUIT Heat: 10/10	SUPERHOT REAPER Heat: 10/10

VOODOO REAPER
by MIC'S CHILLI

Description

A real volcano in a bottle! It contains 50% Carolina Reaper, enough to wake the dead.
Origin: Ireland Scoville: 1,200,000 Capacity: 155ml Ingredients : Carolina Reaper 50%, xanthan gum, lemon juice, garlic, paprika, black pepper, salt, white vinegar

1 0

BACK MODIFY DELETE

Sauce Controller | Publier

JS

controllers/sauce.js

```
/* Publier une sauce => POST
exports.createSauce = (req, res, next) => {
  /* Parser objet requête :
  Si un fichier (image) est ajouté à la requête, le front-end revoie les données
  en chaîne.*/
  const sauceObject = JSON.parse(req.body.sauce);          Parser pour manipuler objet
  // Supprimer "userId" de la requête (Ne jamais faire confiance à l'utilisateur)
  delete sauceObject.userId;                                Supprimer userId requête
  // Créer un nouvel objet Sauce
  const sauce = new Sauce({    Nouvel objet à partir du Model
    ...sauceObject,
    // Récupérer "userId" depuis le Token d'authentification
    userId: req.auth.userId,      Récupérer userId depuis middleware
    // Générer URL de l'image
    imageUrl: `${req.protocol}://${req.get('host')}/public/images/${req.file.filename}`,   Générer url image
    likes: 0,
    dislikes: 0,
    usersLiked: [],
    usersDisliked: []
  });
  // Enregistrer dans database
  sauce.save()  Sauvegarde database
    .then(() => { res.status(201).json({ message: 'Sauce ajoutée !' }) })
    .catch(error => { res.status(400).json({ error }) })
};
```

Valeur 0,
Tableaux
vides

Name
VOODOO REAPER

Manufacturer
MIC'S CHILLI

Description
Capacity: 155ml

Ingredients : Carolina Reaper 50%, xanthan gum, lemon juice, garlic, paprika, black pepper, salt, white vinegar

ADD IMAGE



Main Pepper Ingredient
Carolina Reaper 50%, xanthan gum, lemon juice, garlic, paprika, black pepper, salt, white vinegar

Heat
10

SUBMIT

Database : objet Sauce

```
_id: ObjectId("62fbf5140e48d33e02b00817")
userId: "63013b1c3c14bfea6e80537c"
name: "VOODOO REAPER"
manufacturer: "MIC'S CHILLI"
description: "A real volcano in a bottle! It contains 50% Carolina Reaper, enough to..."
mainPepper: "Ingredients : Carolina Reaper 50%, xanthan gum, lemon juice, garlic, ..."
imageUrl: "http://localhost:3000/public/images/Voodoo_Reaper-removebg-preview_900.."
heat: 9
likes: 1
dislikes: 0
usersLiked: Array
usersDisliked: Array
__v: 0
```

Sauce Controller | Modifier + Supprimer

Supprimer

```
// Import File System (module Node)
/* Imports :
const fs = require('fs') // File system (Node.js)
// Pour utiliser la fonction "unlink" et supprimer une image stockée localement.

/* Supprimer une sauce => DELETE
exports.deleteSauce = (req, res, next) => {
  // Chercher correspondance objet requête/database
  Sauce.findOne({ _id: req.params.id }) Trouver id objet dans database
    .then(sauce => {
      // Vérifier correspondance user database/requête(token-auth)
      if (sauce.userId != req.auth.userId) { Vérifier utilisateur
        res.status(401).json({ message: 'Non-autorisé' })
      } else {
        // Récupérer nom fichier image avec méthode "split"
        const filename = sauce.imageUrl.split('public/images/')[1];
        // Supprimer fichier du serveur avec fonction "unlink" de FS
        fs.unlink(`public/images/${filename}`, () => {
          // Callback : Supprimer la Sauce de la database
          Sauce.deleteOne({ _id: req.params.id })
            .then(() => { res.status(200).json({ message: 'Objet supprimé !' }) })
            .catch(error => res.status(401).json({ error }));
        })
      }
    })
    .catch(err => {
      res.status(500).json({ error });
    })
}
```

Modifier

```
exports.modifySauce = (req, res, next) => {
  // Objet pour vérifier si il y a un fichier dans notre requête
  const sauceObject = req.file ? {
    // Si oui : Parser objet requête
    ...JSON.parse(req.body.sauce),
    // Générer URL de l'image
    imageUrl: `${req.protocol}://${req.get('host')}/public/images/${req.file.filename}`,
    // Si non : Traiter objet directement (string)
  } : { ...req.body };
  // Supprimer "_userId" de la requête (Ne jamais faire confiance à l'utilisateur)
  delete sauceObject.userId;
  // Chercher correspondance objet en database
  Sauce.findOne({ _id: req.params.id }) Trouver id objet dans database
    .then((sauce) => {
      // Vérifier id utilisateur database/requête(token-auth)
      if (sauce.userId != req.auth.userId) { Vérifier utilisateur
        // Mauvais utilisateur : Annuler requête
        res.status(401).json({ message: 'Not authorized' });
      } else {
        // Bon utilisateur : Si image dans requête, supprimer ancienne image
        if (req.file) {
          const filename = sauce.imageUrl.split('/images/')[1] Si nouvelle image ->
          fs.unlink(`public/images/${filename}`, () => { supprimer ancienne
          })
        }
        // Modifier objet avec contenu requête
        Sauce.updateOne({ _id: req.params.id }, { ...sauceObject, _id: req.params.id })
          .then(() => res.status(200).json({ message: 'Sauce modifiée !' }))
          .catch(error => res.status(401).json({ error }));
      }
    })
    .catch((error) => {
      res.status(400).json({ error });
    })
};

Si modification image, parser +
générer imageUrl
```

Sauce Controller

```
/* Liker / Disliker une sauce => POST
exports.likeDislikeSauce = (req, res, next) => {
  let like = req.body.like
  let userId = req.body.userId
  let sauceId = req.params.id
  // Utilisation de l'instruction "Switch"
  switch (like) {
    // Liker une sauce
    case 1:
      // Correspondance Sauce, Ajouter userId au tableau usersLiked, Incrémenter 1 dans Likes
      Sauce.updateOne({ _id: sauceId }, { $push: { usersLiked: userId }, $inc: { likes: +1 } })
        .then(() => res.status(200).json({ message: `Sauce likée` }))
        .catch((error) => res.status(400).json({ error }))
      break;
    // Annuler un Like ou un Dislike
    case 0:
      // Trouver correspondance
      Sauce.findOne({ _id: sauceId })
        .then((sauce) => {
          // 1 - Annulation d'un Like (si user présent dans tableau "usersLiked")
          if (sauce.usersLiked.includes(userId)) {
            // Retirer userId du tableau usersliked, Décrémenter 1 dans Likes
            Sauce.updateOne({ _id: sauceId }, { $pull: { usersLiked: userId }, $inc: { likes: -1 } })
              .then(() => res.status(200).json({ message: `Neutre` }))
              .catch((error) => res.status(400).json({ error }))
          }
          // 2 - Annulation d'un Dislike (si user présent dans tableau "usersDisliked")
          if (sauce.usersDisliked.includes(userId)) {
            // Retirer userId du tableau usersDisliked, Décrémenter 1 dans Dislikes
            Sauce.updateOne({ _id: sauceId }, { $pull: { usersDisliked: userId }, $inc: { dislikes: -1 } })
              .then(() => res.status(200).json({ message: `Neutre` }))
              .catch((error) => res.status(400).json({ error }))
          }
        })
        .catch((error) => res.status(404).json({ error }))
      break;
    // Disliker une sauce
    case -1:
      // Correspondance Sauce, Ajouter userId au tableau usersDisliked, Incrémenter 1 dans Dislikes
      Sauce.updateOne({ _id: sauceId }, { $push: { usersDisliked: userId }, $inc: { dislikes: +1 } })
        .then(() => { res.status(200).json({ message: `Sauce disliked` }) })
        .catch((error) => res.status(400).json({ error }))
      break;
    default:
      console.log(error);
  }
}
```

Déclarer variables utiles

Instruction switch -> Plusieurs cas de figure

Dans l'objet Sauce concerné :
"Pousser" userId dans tableau "users Liked"
Incrémenter valeur +1 dans Likes

Dans l'objet Sauce concerné :
"Pousser" userId dans tableau "users Disliked"
Incrémenter valeur +1 dans Dislikes

Like / Dislike

3-Bilan

Exigences respectées

Inscription : Email unique ; Hachage mot de passe

Connexion : Renvoie un Token web signé (id), durée limitée

Routes Sauce : Authentification renforcée sur toutes les routes Sauce par vérification du Token web signé

Database : Mongoose renvoie les erreurs ; Identifiants en variable d'environnement (.env n'est pas envoyé à GitHub)

Dépendances : Versions les plus récentes incluant les derniers correctifs de sécurité

Images : Contenu dossier image n'est pas envoyé à GitHub

Ajouts : Validateurs de conformité pour les champs Email/Password ; Cryptage Email en database (RGPD) ; Sécurité headers HTTP via Helmet

Axes d'amélioration

Prendre en considération les recommandations de l'OWASP

Injection : Protection des entrées, Fuzzing (test) -> Protection contre injection SQL, attaques XSS

Piratage de session : Limiter nombre de requête de connexion (Force brute)

Données en transit : HTTPS pour l'ensemble du site

Contrôles d'accès : Si l'application évolue, toujours s'assurer que toutes les pages sont verrouillées par contrôle d'accès

Tester la sécurité : Faire réaliser des tests de sécurité par un prestataire spécialisé en sécurité (tests d'intrusion, menaces potentielles...)

Veille sécurité : S'informer sur les nouvelles vulnérabilités