Hot Takes

Application pour créer, liker ou disliker des sauces

Proposée par "piiquante.com" créateur de sauces épicées

Projet 6 Formation OCR Développeur Web

Magali Bernard

Sommaire

Structure du code de l'API

A/Création du serveur et de l'application

B/Sécurité de l'API : Modules User d'authentification, Variables d'environnement

C/Traitement des objets Sauce: CRUD : Create, Read, Update et Delete

Fonction Like/Dislike

Création de l'appli en modules...

- routeurs

```
//route pour créer une sauce:
router.post("/", auth, multer, stuffCtrl.createSauce);
```

- modèles

"constructeurs" d'objet user et sauce

- contrôleurs

fonctions exécutées lors des différentes requêtes

- autres middleware utiles
  - authentification
  - configuration de multer

... et en deux parties :

- **parcours user "/api/auth"**

système d'inscription et connexion pour la sécurité de l'API

- **parcours sauce "/api/sauces"**
  - **CRUD**
  - **Like/Dislike**

A/ Serveur et application

server.js dans node

- package http de node + méthode createServer (application express)
- écoute le port 3000 par défaut ou un autre si spécifié par l'utilisateur

application express app.js

- permet de se connecter à notre base de données MongoDb
- configure les réponses aux requêtes pour la com entre l'API et le front (Cors)
- indique les routes à utiliser

B/Sécuriser l'API

- Rendre la connexion obligatoire pour toute requête. Utiliser le cryptage de mot de passe pour enregistrer les users.

- Créer et envoyer des tokens au front-end pour authentifier toutes les requêtes grâce à un middleware d'authentification pour sécuriser toutes les routes de l' API. Seules les requêtes authentifiées seront gérées.

- Utiliser des variables d'environnement et de Helmet, un "casque de sécurité" pour les requêtes

Sécuriser l'API: modèle user

Création d'un schema mongoose User afin de stocker les informations utilisateur dans votre base de données. Empêcher pour une adresse mail la création de plusieurs comptes grâce au plugin mongoose unique validator.

```
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);
//Dans notre schéma, la valeur unique =true,
//avec l'élément mongoose-unique-validator passé comme plug-in,
//s'assurera que deux utilisateurs ne puissent partager la même adresse e-mail

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

Sécuriser l'API: routes user

Les routes prévues sont:

api/auth/signup

api/auth/login

Sécuriser l'API: signup

Enregistrer les données de connexion de manière sécurisée grâce à bcrypt permettant le cryptage des mot de passe.
Pour chaque couple email/motDePasse est créé un userID

```
const User = require("../models/users");
const bcrypt = require("bcrypt");//package de cryptage
//nous appelons la fonction de hachage de bcrypt dans notre mot de passe et lui demandons de « saler » le
mot de passe 10 fois.
//Plus la valeur est élevée, plus l'exécution de la fonction sera longue, et plus le hachage sera sécurisé.
//il s'agit d'une fonction asynchrone qui renvoie une Promise dans laquelle nous recevons le hash généré ;
//dans notre bloc then , nous créons un utilisateur et l'enregistrons dans la base de données
exports.signup = (req, res, next) => {
  //on brouille le mot de passe entré par le user 10 fois
  bcrypt
    .hash(req.body.password, 10)
    //puis on créée un nouvel objet user avec le mail du user et le mot de passe brouillé
    .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 }));
};
```

Sécuriser l'API: login

Comparer le hash (version cryptée) stockée dans l'API et le hash obtenu à partir du mdp entré par l'utilisateur.

Si la comparaison est valide, créer un token d'authentification valable pour une durée limitée et contenant le userId chiffré. Ce token sera inclu dans les entêtes des requêtes du parcours sauce.

```
      //   Nous utilisons la fonction compare de bcrypt pour comparer le mot de passe entré par
l'utilisateur
      //  avec le hash enregistré dans la base de données.bcrypt peut savoir si 2 hash proviennent du même
mdp.
      bcrypt
        .compare(req.body.password, user.password)
        .then((valid) => {
          if (!valid) {
            return res.status(401).json({ error: "Mot de passe incorrect, vous n'êtes pas autorisé à vous
connecter." });
          }
        //   S'ils correspondent, les informations d'identification de notre utilisateur sont valides.
        //    Dans ce cas, nous renvoyons une réponse 200 contenant l'ID utilisateur et un token:
          res.status(200).json({
            userId: user._id,
            //Nous utilisons la fonction sign de jsonwebtoken (jwt) pour chiffrer un nouveau token:
            token: jwt.sign({ userId: user._id },
                //Nous utilisons une chaîne secrète de développement temporaire RANDOM_SECRET_KEY pour
crypter notre token:
                process.env.CHIFFREMENT_TOKEN, {
              expiresIn: "72h",
            }),
          });
        })
        .catch((error) => res.status(500).json({ error }));
    })
    .catch((error) => res.status(500).json({ error }));
};
```

Sécuriser l'API: auth.js

Récupèration du token dans l'entête de la requête.
Vérification du token par décodage.
Extraction du userId et ajout de ce userId à la requête.

Permet d'authentifier les requêtes. Requis pour toutes les rautes "sauce" de l'API.

```
    //Nous utilisons donc la fonction split pour tout récupérer après l'espace dans le header.
    //Les erreurs générées ici s'afficheront dans le bloc catch.
       const token = req.headers.authorization.split(' ')[1];

    //Nous utilisons ensuite la fonction verify pour décoder notre token. Si celui-ci n'est pas valide,
une erreur sera générée.
       const decodedToken = jwt.verify(token, process.env.CHIFFREMENT_TOKEN);
    //Nous extrayons l'ID utilisateur de notre token et le rajoutons à l'objet Request afin que nos
différentes routes puissent l'exploiter.
       const userId = decodedToken.userId;
       req.auth = {
           userId: userId
       };
       //tout fonctionne et notre utilisateur est authentifié. Nous passons à l'exécution à l'aide de la
fonction next().
    next();
   } catch(error) {
       res.status(401).json({ error });
   }
};
```

Sécuriser l'API: variables d'environnement

```
# port

PORT= number

# variables d'environnement de connexion à la base de données MongoDb

LIEN_MDB = string

# chaine de chiffrement du token

CHIFFREMENT_TOKEN = string
```

CRUD : les routes

```
const express = require("express");
const router = express.Router();

const auth = require("../middleware/auth");
//nous importons le middleware auth et le passons comme argument aux routes à protéger.
const multer = require("../middleware/multer-config");//pourquoi?
const stuffCtrl = require("../controllers/sauce");

//route pour voir toutes les sauces:
router.get("/", auth, stuffCtrl.getAllSauces);
//route pour créer une sauce:
router.post("/", auth, multer, stuffCtrl.createSauce);
//route pour voir une sauce:
router.get("/:id", auth, stuffCtrl.getOneSauce);
//route pour modifier une sauce
router.put("/:id", auth, multer, stuffCtrl.modifySauce);
//pour supprimer une sauce:
router.delete("/:id", auth, stuffCtrl.deleteSauce);
//pour liker ou disliker une sauce
router.post("/:id/like", auth, stuffCtrl.likeSauce);
```

Middleware multer-config.js

multer.single permet de capturer les images et d'enregistrer les images dans un dossier images du backend (config diskstorage)

```
};
//Nous créons une constante storage , à passer à multer comme configuration,
//qui contient la logique nécessaire pour indiquer à multer où enregistrer les fichiers entrants:
// la méthode diskStorage()  configure le chemin et le nom de fichier pour les fichiers entrants.
const storage = multer.diskStorage({
  //la fonction destination indique à multer d'enregistrer les fichiers dans le dossier images:
  destination: (req, file, callback) => {
    callback(null, 'images');
  },
  //filename indique à multer d'utiliser le nom d'origine, de remplacer les espaces par des
  //underscores et d'ajouter un timestamp Date.now() comme nom de fichier
  filename: (req, file, callback) => {
    const name = file.originalname.split(' ').join('_');
    //utilise ensuite la constante dictionnaire de type MIME pour résoudre l'extension de fichier
appropriée:
    const extension = MIME_TYPES[file.mimetype];
    callback(null, name + Date.now() + '.' + extension);
  }});
//Nous exportons ensuite l'élément multer entièrement configuré, lui passons notre constante storage et lui
//indiquons que nous gérerons uniquement les téléchargements de fichiers image:
module.exports = multer({storage: storage}).single('image');
//La méthode single()  crée un middleware qui capture les fichiers d'un certain type (passé en argument),
//et les enregistre dans des fichiers du serveur à l'aide du storage configuré.
```

C/Modèle sauce

```
const mongoose = require("mongoose");
const modelSauce = 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", modelSauce);
```

C/Create (post)

A partir d'un objet sauce string et d'un fichier image, créer un nouvel objet sauce et l'enregistre dans la BDD.

```
exports.createSauce = (req, res, next) => {
  const sauceObject = JSON.parse(req.body.sauce);
  console.log(sauceObject);
  //(Le corps de la requête contient une chaîne sauce, qui est simplement un objet sauce converti en chaîne
string.
  //Nous devons donc le convertir en json à l'aide de JSON.parse() pour obtenir un objet utilisable.)
  delete sauceObject._id;
  //Nous supprimons le champ_userId de la requête envoyée par le client car nous ne devons pas lui faire
confiance
  delete sauceObject._userId;
  const sauce = new Sauce({
    ...sauceObject,
    //Nous remplaçons le userId supprimé par le _userId extrait du token par le middleware
d'authentification:
    userId: req.auth.userId,
    //Reconstitution de l'URL de l'image:
    //Nous utilisons req.protocol pour obtenir le premier segment (dans notre cas 'http').
    //Nous ajoutons '://', puis utilisons req.get('host') pour obtenir l'hôte du serveur (ici,
'localhost:3000').
    //Nous ajoutons finalement '/images/' et le nom de fichier pour compléter notre URL.
    imageUrl: `${req.protocol}://${req.get("host")}/images/${
      req.file.filename
    }`,
  });
  sauce
    .save()
    .then(() => {
      res.status(201).json({ message: "Sauce enregistrée !" });
    })
    .catch((error) => {
      res.status(400).json({ error });
    });
```

C/Read (get)

Trouve l'id passé en paramètre et retourne l'objet sauce correspondant de la BDD au format json grâce à findOne.

```
//FONCTION OBTENIR UNE 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({
        message: "Sauce non trouvée.",
      });
    });
};
```

C/Update

Passe en request body soit un objet sauce json, soit un objet sauce string et un fichier image.

En cas de présence d'une image, reconstitue son url et supprime l'ancienne image.

Dans tous les cas, remplace grâce à updateOne et l'id sauce en paramètre l'ancien objet sauce par le nouveau.

```
//FONCTION MODIFIER UNE SAUCE
exports.modifySauce = (req, res, next) => {
  //on crée un objet sauceObject et on demande si req.file existe ou non, ie s'il y a un fichier à
télécharger ou non.
  //en fonction de la réponse à cette question, on remplit différemment sauceObject
  const sauceObject = req.file
    ? {
        //S'il existe, on traite la nouvelle image :
        //on met dans sauceObject à l'aide de ... la partie Json du body, ie tout sauf l'image:
        ...JSON.parse(req.body.sauce),
        //Nous utilisons req.protocol pour obtenir le premier segment (dans notre cas 'http').
        //Nous ajoutons '://', puis utilisons req.get('host') pour obtenir l'hôte du serveur (ici,
'localhost:3000').
        //Nous ajoutons finalement '/images/' et le nom de fichier pour compléter notre URL.
        imageUrl: `${req.protocol}://${req.get("host")}/images/${
          req.file.filename
        }`,
      }
    : {
        // s'il n'existe pas, on traite simplement l'objet entrant: on met le contenu du body de la requête
dans sauceObject à l'aide de l'opérateur ...
        ...req.body,
      };
  // suppression du champ _userId envoyé par le client
  delete sauceObject.userId;
  //ON VERIFIE QUE LE MODIFICATEUR EST LE CREATEUR SANS QUOI IL N EST PAS AUTORISE A MODIFIE LA SAUCE:
  Sauce.findOne({ _id: req.params.id })
    .then((sauce) => {
      //Nous vérifions si l'utilisateur qui a fait la requête de suppression est bien celui qui a créé la
sauce.
      if (sauce.userId != req.auth.userId) {
        res.status(401).json({
          message: "Vous n'êtes pas autorisé à modifier cette sauce.",
        });
      } else {
        console.log("sauce à modifier:", sauce);
        //l'opérateur !!string vaut true si la chaîne n'est pas vide, ie ici si une image est présente
        if (!!sauce.imageUrl) {
          const filename = sauce.imageUrl.split("/images/")[1];
          fs.unlink(`images/${filename}`, () => {
            console.log("Ancienne image supprimée!", filename);
          });
        }
            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 }));
      }
    })
    .catch((error) => {
      res.status(400).json({ error });
    });
};
```

C/Delete

**Trouve et supprime l'objet sauce correspondant à l'id passé en paramètre grâce à deleteOne.
Supprime l'image correspondante dans le dossier images grâce à fs.unlink.**

```
exports.deleteSauce = (req, res, next) => {
  //Nous utilisons l'ID que nous recevons comme paramètre pour accéder a la sauce correspondante dans la base de données.
  Sauce.findOne({ _id: req.params.id })
    .then((sauce) => {
      //Nous vérifions si l'utilisateur qui a fait la requête de suppression est bien celui qui a créé la sauce.
      if (sauce.userId != req.auth.userId) {
        res
          .status(401)
          .json({
            message: "Vous n'êtes pas autorisé à supprimer cette sauce.",
          });
      } else {
        //Nous utilisons le fait de savoir que notre URL d'image contient un segment /images/ pour séparer le nom de fichier.
        const filename = sauce.imageUrl.split('/images/')[1];
        //Nous utilisons ensuite la fonction unlink du package fs pour supprimer ce fichier du système de fichiers du serveur
        //en lui passant le fichier à supprimer et le callback à exécuter une fois ce fichier supprimé.
        fs.unlink(`images/${filename}`, () => {
          //Dans le callback, nous implémentons le code pour supprimer la sauce de la base de données:
          Sauce.deleteOne({ _id: req.params.id })
            .then(() => {
              res.status(200).json({ message: "Sauce supprimée !" })})
            .catch(error => res.status(400).json({ error }));
        });
      }
    })
    .catch(error => {
      res.status(500).json({ error });
    });
};
```

Fonction liker/disliker (post)

Entrées: id sauce en paramètre , valeur de userId et de like en corps de requête

Agit sur les clés like et dislike et les tableaux de userId "usersLiked" et "usersDisliked" de la sauce.

Cas envisagés:

post like=1 : état initial 1, 0 ou -1
post like=-1: état initial -1, 0 ou 1
post like=0: état initial 0, 1 ou -1

Si état posté = état initial, rien ne change.
Si on passe de 0 à 1 ou -1, on incrémente like ou dislike et on enregistre le userId dans le tableau concerné.
Si on passe de 1 à -1 ou inversement, on efface en plus le userId du tableau qui n'est plus concerné.

Fonction liker/ disliker

```
//FONCTION DE GESTION DES "LIKE"
exports.likeSauce = (req, res, next) => {
  //on cherche la sauce que l'on veut liker dans l'api via son id passer dans l'url de la requête
  Sauce.findOne({ _id: req.params.id })
    .then((sauce) => {
      // console.log("je suis dans la fonction like");
      const likes = sauce.likes; //2
      const dislikes = sauce.dislikes; //1
      const usersLiked = sauce.usersLiked; //["id1","id2"]
      const usersDisliked = sauce.usersDisliked;
      //on recupère les infos de la requête dans des variables:
      let userIdLikeur = req.body.userId;
      let like = req.body.like;
      //****************si le user likeur n'est pas enregistré dans le tableau des users likeurs de la sauce
et qu'il a liker la sauce:
      if (!usersLiked.includes(userIdLikeur) && like === 1) {
        //s'il avait disliké au préalable, on supprime son dislike:décrément de dislikes et supression de
son Id dans usersDisliked
        if (usersDisliked.includes(userIdLikeur)) {
          Sauce.updateOne(
            { _id: req.params.id },
            {
              $inc: { dislikes: -1 },
              $pull: { usersDisliked: userIdLikeur },
            }
          )

            .then((sauce) => console.log(sauce))
            .catch((error) => console.log("mauvaise requête" + error));
        } else {
          //Dans tous les cas on enregistre son like:
          Sauce.updateOne(
            //le premier paramètre de cette méthode mongoDB updateOne(filter,update,options) est le filtre:
            { _id: req.params.id },
            //ensuite on écrit les modifications voulues: likes à 1 et mettre l'id du likeur dans le
tableaux des users likeurs:
            {
              //pour cela on utilise un opérateur de mise à jour mongoDB: $inc qui incrémente un champ avec
une valeur.
              //syntaxe: { $inc: { <field1>: <amount1>, <field2>: <amount2>, ... } }
              $inc: { likes: 1 },
              //$push permet d'ajouter un champ et sa valeur à un élément d'une collection ou de mettre à
jour la valeur du champ s'il existe:
              $push: { usersLiked: userIdLikeur },
            }
          )
            //le résultat de la méthode updateOne est une promesse donc on doit ajouter .then et .cath pour
traiter cette promesse:
            .then(() =>
              res
                .status(201)
                .json({
                  message: "Sauce mise à jour avec incrément d'un like!",
                })
            )
            .catch((error) => res.status(400).json({ error }));
        }
      }
```

Fonction liker/disliker

```
//*****************si le user likeur est enregistré dans le tableau des users likeurs de la sauce et qu'il ne
veut plus liker:
      else if (usersLiked.includes(userIdLikeur) && like === 0) {}
//*****************si le user likeur enregistré dans tableau des users DISlikeurs de la sauce et qu'il ne
veut plus DISliker:
      else if (usersDisliked.includes(userIdLikeur) && like === 0) {}
 //*****************si le user likeur n'est pas enregistré dans le tableau des users dislikeurs de la sauce
et qu'il a disliker la sauce:
      else if (!usersDisliked.includes(userIdLikeur) && like === -1) {}
//***************si l'utilisateur veut refaire la même action 2 fois, aucun changement ne se produit. */
      else if (
        (!usersLiked.includes(userIdLikeur) &&
          usersDisliked.includes(userIdLikeur) &&
          like == -1) ||
        (usersLiked.includes(userIdLikeur) &&
          !usersDisliked.includes(userIdLikeur) &&
          like == 1) ||
        (!usersLiked.includes(userIdLikeur) &&
          !usersDisliked.includes(userIdLikeur) &&
          like == 0)
      ) {
        res.json({ message: "Pas de changement." });
      }
      //*************************************************************************************************
      console.log("likes:", likes);
      console.log("dislikes:", dislikes);
    })
    .catch((error) => res.status(404).json({ message: "Sauce non trouvée." }));
};
```

Bilan Sécurité

Le Top Ten de l'OWASP fournit une base de référence avec une liste de contrôles à effectuer pour atténuer les risques les plus courants en matière de sécurité.

Open Web Application Security Project Top Ten 2021:

1 Pour protéger des contrôle d'accés cassés n°1 des attaques Top Ten 2021:
->Implémentez des mécanismes de contrôle d'accès une seule fois et réutilisez-les dans l'ensemble de l'application -> *création de token de session dans notre code.*
->Les jetons JWT doivent plutôt être de courte durée afin que la fenêtre d'opportunité pour un attaquant soit minimisée->réduire la durée valide à la durée de session ou x h.
2 échec cryptographique ->défaillances liées à la cryptographie (ou son absence)
-> Ne pas utiliser des mots de passe codés en dur mais des cryptés et des token chiffrés.
->chiffrer toutes les transitions de données à l'aide de directives telles que HTTP Strict Transport Security (HSTS)->possible en utilisant helmet et un certificat SSL (préparé dans app.js)
3 injection de code via le frontend (par exemple les failles XXS)
-> utiliser le package helmet définissant des entêtes de protection dans les requêtes dont l'entête x-xss-protection contre les injections XXS.

RGPD: RèglementGénéral pour la Protection des Données (europe) (prend OWASP en référence)
Grands principes, guides. ex: https://www.cnil.fr/fr/principes-cles/guide-de-la-securite-des-donnees-personnelles qui reprend authentification, habilitation, protocole de com TLS (nouveau SSL).

Et aussi:
Sécuriser les dépendances, par exemple avec npm audit on vérifie l'absence de vulnérabilité sur les packages installés.
Veille sécurité sur des bases de données de vulnérabilités comme Common Vulnerabilities and Exposures (en anglais), créée par le MITRE, https://snyk.io/vuln/ ou encore github https://github.com/advisories.

Amélioration continu de la sécurité!

Merci!