Projet 6

Piiquante

Présentation des livrables

❖ Création du repo Github contenant:

- un dossier Front-End (fourni),
- un dossier Back-End.

❖ Afin de faire fonctionner l'API :

- Dans le dossier Back-End, exécuter via le terminal => nodemon server
- Dans le dossier Front-End, exécuter via le terminal => npm run start

Routes : User et Sauce

❖Mise en place de la logique pour la création et connexion utilisateur.

```
router.post("/signup", userCtrl.signup);
router.post("/login", userCtrl.login);

module.exports = router;
```

❖Puis la logique de création d'objet, avec protection d'accès à la route avec « auth » et « multer » pour la modification des images.

```
router.post("/", auth, multer, sauceControllers.createSauce);
router.post("/:id/like", auth, sauceControllers.likeDislikeSauce);

// modifier un objet existant dans la base de donnée
router.put("/:id", auth, multer, sauceControllers.modifySauce);

// supprimer un objet existant dans la base de donnée
router.delete("/:id", auth, sauceControllers.deleteSauce);

// :id <= parti de la route dynamique pour une recherche à l'unité
router.get("/:id", auth, sauceControllers.getOneSauce);
// pour trouver tous les objets
router.get("/", auth, sauceControllers.getAllSauces);

module.exports = router;
```

Models : User

- Création d’un schéma pour le modèle utilisateur intégrant l’obligation d’un mail unique sous forme de chaîne de caractère et un mot de passe.

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

Models : Sauce

- Création d’un schéma pour le modèle sauce suivant les recommandations du data ModelsSauce.

```
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, required: false, default: 0 },
  dislikes: { type: Number, required: false, default: 0 },
  usersLiked: { type: [String], required: false, defaultValue: [] },
  usersDisliked: { type: [String], required: false, defaultValue: [] },
});
```

Middleware : auth

- Le middleware auth permet avec l'aide de Jsonwebtoken de vérifier les tokens.

- Nous récupérons le token dans le header Authorization que l'ont décode avec la clé secrète ensuite ont compare le userId récupérer dans le token avec celui de la requête.

```
module.exports = (req, res, next) => {
  try {
    // récupérer le token dans le header authorization
    // [1] => ont récupère le 2éme élement du tableau en cas d'erreur ça r
    const token = req.headers.authorization.split(" ")[1];
    // ont décode le token avec la clé secréte en cas d'erreur ça renvoi v
    const decodedToken = jwt.verify(token, "RANDOM_TOKEN_SECRET");
    // ont récupére le userId récupérer dans le token et ont le compare av
    const userId = decodedToken.userId;
    if (req.body.userId && req.body.userId !== userId) {
      // si le userID est faux ont envoi une erreur
      throw "User ID non valable !";
    } else {      You, a month ago • Mise en place de la partie authentifi
      // si c'est bon on passe la requéte au prochain middleware
      next();
    }
  } catch (error) {
    res.status(403).json({ error: error | "unauthorized request" });
  }
```

Middleware : multer

❖Mise en place de multer avec restrictions de format d'image, choix de l'emplacement de stockage et modification du nom du fichier afin d'intégrer la date.

```
const MIME_TYPES = {
  "image/jpg": "jpg",
  "image/jpeg": "jpg",
  "image/png": "png",
};
```

```
const storage = multer.diskStorage({
  // function pour l'emplacement de sauvegarde des fichiers
  destination: (req, file, callback) => {
    callback(null, "images");
  },
  // explique à multer quel nom de fichier utilisé
  filename: (req, file, callback) => {
    // supprime les espaces avec "split" et les remplace av
    const name = file.originalname.split(" ").join("_");
    // création de l'extension du fichier
    const extension = MIME_TYPES[file.mimetype];
    callback(null, name + Date.now() + "." + extension);
  },
});
```

Controllers : user (1/2)

- Mise en place de la création d'utilisateur avec dissimulation de l'adresse mail et cryptage du mot de passe, puis enregistrement dans la base de donnée si new utilisateur.

```
exports.signup = (req, res, next) => {
  // hash (fonction asynchrome qui prend du temps) pour crypter
  // salt = 10, tour pour l'algorithme de hashage, suffisant pour un mot de p
  bcrypt
    .hash(req.body.password, 10)
    .then((hash) => {
      // création d'un nouvel utilisateur avec le mot de passe crypté et emai
      const user = new User({
        email: MaskData.maskEmail2(req.body.email, emailMask2Options),
        password: hash,
      });
      //enregistrement de l'utilisateur dans la base de donnée
      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 (2/2)

- Nous récupérons l'utilisateur qui correspond à l'adresse mail entrée par l'utilisateur si il n'y a pas de correspondance ont renvoi une erreur, si le mot de passe ne respecte pas le bon schéma ont renvoi une erreur.

- Si l'identification est correct, ont renvoi le user_id attendu par le Front-End accompagné d'un token avec une validité de 24h.

```
exports.login = (req, res, next) => {
  // ont récupère l'utilisateur dans la base de donnée qui correspond à l'adresse email
  entrée par l'utilisateur
  User.findOne({
    email: MaskData.maskEmail2(req.body.email, emailMask2Options),
  })
    .then((user) => {
      // si ont ne trouve pas de correspondance ont renvoi une erreur
      if (!user) {
        return res.status(401).json({ error: "Utilisateur non trouvé !" });
      }
      if (!schema.validate(req.body.password)) {
        //Renvoie une erreur si le schema de mot de passe n'est pas respecté
        return res.status(400).json({
          message:
            "Le mot de passe doit contenir au moins 10 caractères, une majuscule, une
            minuscule, 2 chiffres, un symbole ainsi qu'aucun espace.",
        });
      }      You, a month ago • Mise en place de password validator, modification…
      // ont compare le mot de passe entré avec le hash enregistré dans la base de donn
      bcrypt
        .compare(req.body.password, user.password)
        .then((valid) => {
          // si la comparaison n'est pas bonne on renvoi une erreur
          if (!valid) {
            return res.status(401).json({ error: "Mot de passe incorrect !" });
          }
          // si l'identification est bonne on renvoi le user._id attendu par le
          un token
          res.status(200).json({
            userId: user._id,
            token: jwt.sign(
              // donnée que l'ont veux encodé à l'intérieur du token
              { userId: user._id },
              // clé secrete pour l'encodage
              "RANDOM_TOKEN_SECRET",
              // argument de configuration (expiration 24H)
              { expiresIn: "24h" }
            ),
          });
        })
        .catch((error) => res.status(500).json({ error }));
    })
    .catch((error) => res.status(500).json({ error }));
};
```

Controllers : sauce (1/4)

- Pour la création des sauces nous récupérons tous les champs du corps de la requête, ont génère une url pour l'image et ont sauvegarde dans la base de donnée MongoDB.

```
exports.createSauce = (req, res, next) => {
  // chaine de caractére sous forme javascript req.body.sauce
  const sauceObject = JSON.parse(req.body.sauce);
  // ont enléve l'id car mongoDB en génére un de lui même
  delete sauceObject._id;
  const sauce = new Sauce({
    // permet de récupérer tous les champs du corp de la requête
    ...sauceObject,
    // ont génére une URL de l'image
    imageUrl: `${req.protocol}://${req.get("host")}/images/${
      req.file.filename
    }`,
  });
  // save dans la base de donnée MongoDB
  sauce
    .save()
    .then(() => res.status(201).json({ message: "Objet enregistré !" }))
    .catch((error) => res.status(400).json({ error }));
};
```

Controllers : sauce (2/4)

- Pour la suppression d'une sauce ont récupère l'objet dans la base de donnée ainsi que l'image associée et nous supprimons l'objet.

```
exports.deleteSauce = (req, res, next) => {
  //ont trouve l'objet dans la base de donnée
  Sauce.findOne({ _id: req.params.id })
    .then((sauce) => {
      // ont récupère le nom du fichier à supprimer
      const filename = sauce.imageUrl.split("/images/")[1];
      // ont supprime l'objet
      fs.unlink(`images/${filename}`, () => {
        // ont renvoi une réponse si fonctionne ou non
        Sauce.deleteOne({ _id: req.params.id })
          .then(() => res.status(200).json({ message: "Objet supprimé !" }))
          .catch((error) => res.status(400).json({ error }));
      });
    })
    .catch((error) => res.status(500).json({ error }));
};
```

Controllers : sauce (3/4)

- Pour modifier une sauce :
  - Ont vérifie que l'image soit modifié ou non.
  - Puis ont vérifie tous les champs du corps de la requête afin de trouver d'autres modifications si il y en a.
  - Pour finir nous mettons à jour l'objet dans la base de donnée.

```
exports.modifySauce = (req, res, next) => {
  // permet de savoir si image existante ou si nouvelle
  const sauceObject = req.file
    ? {
        ...JSON.parse(req.body.sauce),
        // ont génére une URL de l'image
        imageUrl: `${req.protocol}://${req.get("host")}/images/${
          req.file.filename
        }`,
      }
    : { ...req.body }; // si il n'existe pas on fait une copie de req.
  Sauce.updateOne(
    { _id: req.params.id },
    { ...sauceObject, _id: req.params.id }
  )
    .then(() => res.status(200).json({ message: "Objet modifié !" }))
    .catch((error) => res.status(400).json({ error }));
};
```

Controllers : sauce (4/4)

- Pour l’incrémentation des likes et dislikes => ici

- Utilisation de switch afin de n’avoir qu’un like ou dislike actif basé sur l’user_id. Le case 1 permet de like une sauce, le case -1 de dislike une sauce et le case 0 permet quand à lui de décrémenté aussi bien un like qu’un dislike selon se que l’utilisateur avait précédent choisi.

Sécurité OWASP (1/2)

- Mise en place de :
  - Mongo sanitize => permet de nettoyer les données utilisateur pour éviter les injections.
  - Xss clean => permet de nettoyer les entrées utilisateur provenant du corps POST, des requêtes GET et des paramètres d'URL, Protection contre les attaques XSS.
  - Helmet => permet de sécurisé les en-têtes HTTP.
  - Maskdata => permet de masqué différent type de donnée e-mail, mot de passe, etc...
  - Password-validator => permet d'imposer une complexité pour valider un mot de passe en définissant des règles.

Sécurité OWASP (2/2)

- Mise en place de :
  - Bcrypt => permet que le mot de passe de l'utilisateur soit hashé.
  - L'authentification est demandé sur toutes les routes sauce.
  - mongoose-unique-validator => permet l'ajout d'une validation de pré-sauvegarde pour les champs uniques dans un schéma pour que les adresses électroniques soit uniques.
  - Jsonwebtoken => permet de créer et contrôler les tokens.
  - Les versions les plus récentes des logiciels sont utilisées avec des correctifs de sécurité actualisés.

  - Le contenu du dossier images n'est pas téléchargé sur Github.