```
1   const express = require('express');
2   const mongoose = require('mongoose')
3   const path = require('path');
4
5   const app = express();
6
7   require('dotenv').config();
8
9   const saucesRoutes = require('./routes/sauces');
10  const userRoutes = require('./routes/user');
11
12  mongoose.connect(
13      'mongodb+srv://' + process.env.DB_USER + ':' + process.env.DB_PASSWORD +
    '@cluster0.ovbyqz9.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0',
14      {
15          useNewUrlParser: true,
16          useUnifiedTopology: true
17      })
18      .then(() => console.log('Connexion à MongoDB réussie !'))
19      .catch(() => console.log('Connexion à MongoDB échouée !'));
20
21  app.use(express.json());
22
23  app.use((req, res, next) => {
24      res.setHeader('Access-Control-Allow-Origin', '*');
25      res.setHeader('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content,
    Accept, Content-Type, Authorization');
26      res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, PATCH,
    OPTIONS');
27      next();
28  });
29
30  app.use('/images', express.static(path.join(__dirname, 'images')));
31  app.use('/api/sauces', saucesRoutes);
32  app.use('/api/auth', userRoutes);
33
34
35  module.exports = app;
```

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.4.1",
    "mongoose-unique-validator": "^5.0.0",
    "multer": "^1.4.5-lts.1"
  }
}
```

```
1   const http = require('http');
2   const app = require('./app');
3
4   const normalizePort = val => {
5     const port = parseInt(val, 10);
6
7     if (isNaN(port)) {
8       return val;
9     }
10    if (port >= 0) {
11      return port;
12    }
13    return false;
14  };
15  const port = normalizePort(process.env.PORT || '3000');
16  app.set('port', port);
17
18  const errorHandler = error => {
19    if (error.syscall !== 'listen') {
20      throw error;
21    }
22    const address = server.address();
23    const bind = typeof address === 'string' ? 'pipe ' + address : 'port: ' + port;
24    switch (error.code) {
25      case 'EACCES':
26        console.error(bind + ' requires elevated privileges.');
27        process.exit(1);
28        break;
29      case 'EADDRINUSE':
30        console.error(bind + ' is already in use.');
31        process.exit(1);
32        break;
33      default:
34        throw error;
35    }
36  };
37
38  const server = http.createServer(app);
39
40  server.on('error', errorHandler);
41  server.on('listening', () => {
42    const address = server.address();
43    const bind = typeof address === 'string' ? 'pipe ' + address : 'port ' + port;
44    console.log('Listening on ' + bind);
45  });
46
47  server.listen(port);
```

```
1   const express = require('express');
2   const router = express.Router();
3
4   const userCtrl = require('../controllers/user');
5
6   router.post('/signup', userCtrl.signup);
7   router.post('/login', userCtrl.login);
8
9   module.exports = router;
```

```
1   const express = require('express');
2   const router = express.Router();
3
4   const saucesCtrl = require('../controllers/sauces'); // Importation des controlleurs
    et middlewares contenant la logique métier => logique de routing claire et explicite
5   const auth = require('../middleware/auth');
6   const multer = require('../middleware/multer-config');
7
8   router.post('/', auth, multer, saucesCtrl.createSauce); // Requêtes authentifiées
    même pour l'envoi d'images (également au niveau de la modification d'une sauce)
9   router.post('/:id/like', auth, saucesCtrl.likeOrDislike);
10  router.put('/:id', auth, multer, saucesCtrl.modifySauce);
11  router.delete('/:id', auth, saucesCtrl.deleteSauce);
12  router.get('/:id', auth, saucesCtrl.getOneSauce);
13  router.get('/', auth, saucesCtrl.getAllSauces);
14
15  module.exports = router;
```

```
1   const mongoose = require('mongoose');
2   const uniqueValidator = require('mongoose-unique-validator');
3
4   const userSchema = mongoose.Schema({
5     email: { type: String, required: true, unique: true },
6     password: { type: String, required: true }
7   });
8
9   userSchema.plugin(uniqueValidator);
10
11  module.exports = mongoose.model('User', userSchema);
```

```
1   const mongoose = require('mongoose');
2
3   const sauceSchema = mongoose.Schema({
4
5       name: { type: String, required: true },
6       manufacturer: { type: String, required: true },
7       description: { type: String, required: true },
8       heat: { type: Number, required: true },
9       likes: { type: Number, required: true },
10      dislikes: { type: Number, required: true },
11      imageUrl: { type: String, required: true },
12      mainPepper: { type: String, required: true },
13      usersLiked: { type: Array, required: true },
14      usersDisliked: { type: Array, required: true },
15      userId: { type: String, required: true },
16
17  });
18
19  module.exports = mongoose.model('Sauce', sauceSchema);
```

```
1   const jwt = require('jsonwebtoken');
2
3   module.exports = (req, res, next) => {
4      try {
5          const token = req.headers.authorization.split(' ')[1];
6          const decodedToken = jwt.verify(token, process.env.TOKEN);
7          const userId = decodedToken.userId;
8          req.auth = {
9              userId: userId
10         };
11          next();
12     } catch(error) {
13         res.status(401).json({ error });
14     }
15  };
```

```
1   const multer = require('multer');
2
3   const MIME_TYPES = { // Dictionnaire mime types pour ceux disponibles depuis le
    frontend
4       'image/jpg': 'jpg',
5       'image/jpeg': 'jpg',
6       'image/png': 'png'
7   };
8
9   const storage = multer.diskStorage({ // Objet de configuration pour multer. Nécessite
    2 élements: destination et filename
10      destination: (req, file, callback) => {
11          callback(null, 'images')
12      },
13      filename: (req, file, callback) => { // Générer le nouveau nom
14          const name = file.originalname.split(' ').join('_'); // Partie avant
    l'extension
15          const extension = MIME_TYPES[file.mimetype]; // Extension
16          callback(null, name + Date.now() + '.' + extension); // Nom global: partie
    avant l'extension + Timestamp créant un fichier unqiue + extension
17      }
18  });
19
20  module.exports = multer({ storage: storage }).single('image'); // Export du
    middleware configuré avec méthode multer précisant l'unicité du fichier et le type
    image
```

```
const bcrypt = require('bcryptjs');
const User = require('../models/User');
const jwt = require('jsonwebtoken');


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 }));
  };

  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.TOKEN,
                            { expiresIn: '24h' }
                        )
                    });
                })
                .catch(error => res.status(500).json({ error }));
        })
        .catch(error => res.status(500).json({ error }));
 };
```

```
1  const Sauce = require('../models/Sauce');
2  const fs = require('fs'); // Importation du package fs (file system) de Node. Permet
   d'accéder aux fonctions de modification du système de fichiers
3
4  exports.createSauce = (req, res, next) => {
5    const sauceObject = JSON.parse(req.body.sauce);
6    delete sauceObject._id;
7    const sauce = new Sauce({
8      ...sauceObject,
9      imageUrl: `${req.protocol}://${req.get('host')}/images/${req.file.filename}`, //
   Résolution complète de l'URL de l'image
10     // req.protocol => premier segment, req.get('host') => résolution de l'hôte,
   dossier images, req.file.filename => nom du fichier
11     likes: 0,
12     dislikes: 0,
13     usersLiked: [],
14     usersDisliked: []
15   });
16   sauce.save()
17     .then(() => res.status(201).json({ message: 'Sauce enregistrée' }))
18     .catch(error => res.status(400).json({ error }));
19 };
20
21 exports.modifySauce = (req, res, next) => {
22   const sauceObject = req.file ? // L'opérateur ternaire permet de traiter 2 cas.
   L'image a ou n'a pas été modifiée.
23     {
24       ...JSON.parse(req.body.sauce),
25       imageUrl: `${req.protocol}://${req.get('host')}/images/${req.file.filename}`
26     } : { ...req.body };
27   Sauce.updateOne({ _id: req.params.id }, { ...sauceObject, _id: req.params.id })
28     .then(() => res.status(200).json({ message: 'Sauce modifiée' }))
29     .catch(error => res.status(400).json({ error }));
30 };
31
32 exports.deleteSauce = (req, res, next) => {
33   Sauce.findOne({ _id: req.params.id })
34     .then(sauce => {
35       if (!sauce) {
36         return res.status(404).json({ error: new Error('Objet non trouvé!') });
37       }
38       // On vérifie que la sauce a bien été créée par l'utilisateur faisant la
   requête
39       if (sauce.userId !== req.auth.userId) {
40         return res.status(401).json({ error: new Error('Requête non autorisée!') });
41       }
42       const filename = sauce.imageUrl.split('/images/')[1]; // Récupération du 2ième
   élément du tableau créé par split
43       fs.unlink(`images/${filename}`, () => {
44         Sauce.deleteOne({ _id: req.params.id })
45           .then(() => res.status(200).json({ message: 'Sauce supprimée' }))
46           .catch(error => res.status(400).json({ error }));
47       });
48     })
49     .catch(error => res.status(500).json({ error }));
50 };
51
52 exports.getOneSauce = (req, res, next) => {
53   Sauce.findOne({ _id: req.params.id })
54     .then(
55       (sauce) => { res.status(200).json(sauce); })
56     .catch((error) => { res.status(404).json({ error: error }); });
57 };
58
59 exports.getAllSauces = (req, res, next) => {
60   Sauce.find()
```

```
61        .then(sauces => res.status(200).json(sauces))
62        .catch(error => res.status(400).json({ error }));
63    };
64
65    exports.likeOrDislike = (req, res, next) => {
66
67      let like = req.body.like
68      let userId = req.body.userId
69      let sauceId = req.params.id
70
71      if (like === 1) {
72        Sauce.updateOne({ _id: sauceId }, { $push: { usersLiked: userId }, $inc: { likes:
      +1 } })
73          .then(() => res.status(200).json({ message: 'LIKE!' }))
74          .catch(error => res.status(400).json({ error }))
75      }
76      if (like === -1) {
77        Sauce.updateOne({ _id: sauceId }, { $push: { usersDisliked: userId }, $inc:
      { dislikes: +1 } })
78          .then(() => res.status(200).json({ message: 'DISLIKE!' }))
79          .catch(error => res.status(400).json({ error }))
80      }
81      if (like === 0) {
82        Sauce.findOne({ _id: sauceId })
83          .then((sauce) => {
84            if (sauce.usersLiked.includes(userId)) {
85              Sauce.updateOne({ _id: sauceId }, { $pull: { usersLiked: userId }, $inc:
      { likes: -1 } })
86                .then(() => res.status(200).json({ message: 'LIKE supprimé!' }))
87                .catch(error => res.status(400).json({ error }))
88            }
89            if (sauce.usersDisliked.includes(userId)) {
90              Sauce.updateOne({ _id: sauceId }, { $pull: { usersDisliked: userId }, $inc:
      { dislikes: -1 } })
91                .then(() => res.status(200).json({ message: 'DISLIKE supprimé!' }))
92                .catch(error => res.status(400).json({ error }))
93            }
94          })
95          .catch(error => res.status(404).json({ error }))
96      }
97    }
```

```
1   # PIIQUANTE V2 (2024)
2
3   ## Construisez une API sécurisée pour une application d'avis gastronomiques
4
5   *Projet 6 de la formation Développeur web d'OpenClassRooms.*
6
7   Le projet consiste à développer le backend d'une application permettant d'ajouter des
    sauces afin de les partager avec d'autres utilisateurs. Il est également possible de
    liker ou disliker les sauces.
8
9   Le frontend est fourni et a été développé/compilé à l'aide d'Angular. Il nous est
    demandé de créer une API en utilisant Node, le framework Express et une base de
    données afin de stocker les utilisateurs et sauces créés.
10
11  Cette version de 2024 a été faite car la version 2021, lorsqu'on l'installe à l'heure
    actuelle, dispose de certaines failles de sécurité sur certains packages.
12
13  ## Compétences évaluées
14
15  #### La création de cette API permet de:
16
17  - Stocker des données de manière sécurisée
18  - Implémenter un modèle logique de données conformément à la réglementation
19  - Mettre en œuvre des opérations CRUD de manière sécurisée
20
21  ## Exigences de sécurité
22
23  *Il est primordial de porter une attention particulière sur la sécurité de
    l'application.*
24
25  - Le mot de passe de l'utilisateur doit être haché.
26  - L'authentification doit être renforcée sur toutes les routes sauce requises.
27  - Les adresses électroniques dans la base de données sont uniques et un
28  plugin Mongoose approprié est utilisé pour garantir leur unicité et signaler
29  les erreurs.
30  - La sécurité de la base de données MongoDB (à partir d'un service tel que
31  MongoDB Atlas) ne doit pas empêcher l'application de se lancer sur la
32  machine d'un utilisateur.
33  - Un plugin Mongoose doit assurer la remontée des erreurs issues de la base
34  de données.
35  - Les versions les plus récentes des logiciels sont utilisées avec des correctifs
36  de sécurité actualisés.
37  - Le contenu du dossier images ne doit pas être téléchargé sur GitHub.
38
39  ## Comment utiliser l'application?
40
41  #### Les installations et commandes suivantes sont nécessaires:
42
43  - Installer Node, Sass, Npm sur votre poste de travail
44  - Télécharger le projet compressé au format ZIP et l'extraire
45  - Extraire les deux dossiers compressés frontend et backend
46  - Aller dans le frontend (via un terminal) et faire `npm install`
47  - Faire `npm start` , le serveur se lance sur http://localhost:4200
48  - Aller dans le backend (via un terminal) et  faire `npm install`
49  - Toujours dans le dossier backend,  créer un fichier .env avec les données fournies
    par le développeur (DB_USER, DB_PASSWORD, DB_NAME, TOKEN) (ignoré avec gitignore pour
    des raisons de sécurité)
50  - Créer également un dossier /images à la racine (ignoré avec gitignore pour
    optimiser la taille du projet)
51  - Faire `node server`(ou nodemon server si celui-ci est installé). La connexion se
    fait sur le port 3000.
52
53  #### Sécurisation de l'application et des données?
54
55  - bcryptjs (hash et salage du mot de passe utilisateur)
56  - jsonwebtoken (sécurisation de l'authentification pour les requêtes )
```

- mongoose (schémas stricts)
- mongoose-unique-validator (création d'un compte par adresse mail)
- dotenv (sécurisation des informations de connexion et d'accès à la base de données)
- dossier images et dotenv non téléchargés sur github (via gitignore)
- helmet et cookie-session auraient été judicieux et sont préconisés dans les recommandations de sécurité d'Express mais ils nécessitaient quelques ajustements côté frontend

Environnement de développement

####

- Visual Studio Code
- Node / Sass / Npm
- Nodemon
- Express
- Mongo DB / Mongo Atlas / Mongoose

Pour en savoir plus sur la sécurité informatique (développement/production/utilisation):

- [Framework Express: meilleures pratiques en production](*https://expressjs.com/fr/advanced/best-practice-security.html*)
- [GitHub Advisory Database](*https://github.com/advisories*)
- [OWASP Top 10 Web Application Security Risks](*https://owasp.org/www-project-top-ten/*)