


Projet 6

Construire une API sécurisée pour une application d'avis gastronomique

OPENCLASSROOMS Formations ▼ Alternance Financements Pour les entreprises

[← Mon parcours](#)


 SOUTENANCE À PLANIFIER

Construisez un site e-commerce en JavaScript


95%

[Mission](#) [Cours](#) [Ressources](#) [Évaluation](#)

Ces cours peuvent vous aider dans la réussite de votre projet. Ils ne sont pas obligatoires.



DÉVELOPPEMENT
Apprenez à programmer avec JavaScript
Easy 15 heures
Établissez une base solide en développement web en apprenant et en pratiquant JavaScript, l'un des principaux langages de programmation sur le web, et créez une application simple !



DÉVELOPPEMENT
Déboguez l'interface de votre site internet
Medium 10 heures

MISE EN PLACE

DU

BACKEND



Mise en place Backend



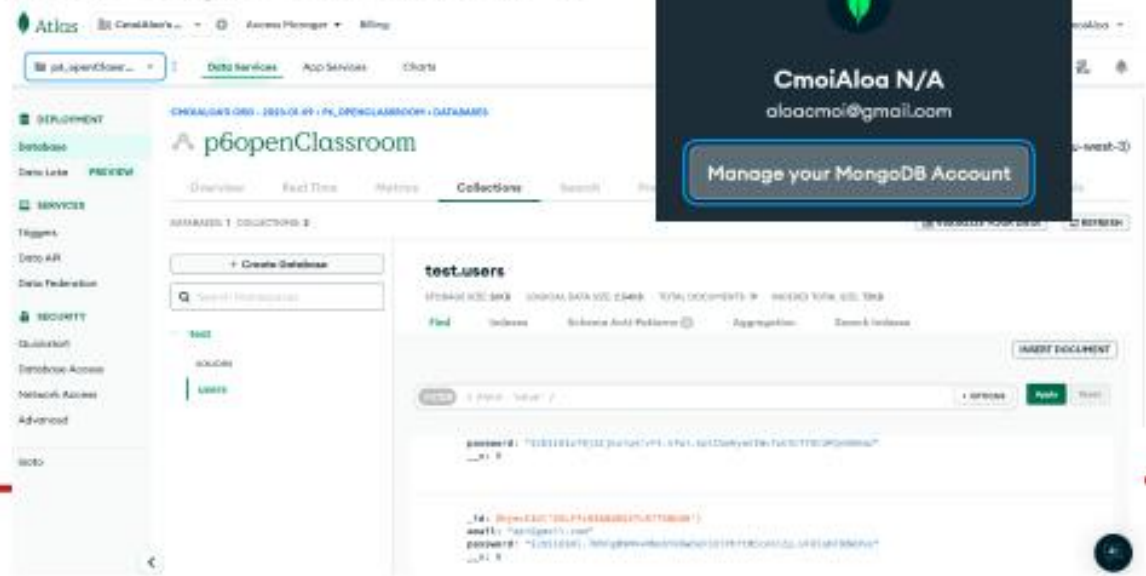
Mise en place package.json & node_modules

⇒ npm init



Création compte MongoDB

⇒ <https://www.mongodb.com/fr-fr>



Mise en place Backend



Installation des dépendances liée à la DB

- ⇒ express
- ⇒ mongoose
- ⇒ cors



Installation des dépendances autres

- ⇒ nodemon
- ⇒ morgan
- ⇒ dotenv



PIQUANTE

```
package.json X
package.json > ...
You: 3 hours ago | 1 author (You)
1  {
2    "name": "piquante",
3    "version": "1.0.0",
4    "description": "p6 open classroom",
5    "main": "server.js",
6    "scripts": {
7      "start": "nodemon server.js"
8    },
9    "author": "aloacmoi",
10   "license": "ISC",
11   "devDependencies": {
12     "morgan": "^1.10.0",
13     "nodemon": "^2.0.20"
14   },
15   "dependencies": {
16     "bcrypt": "^5.1.0",
17     "cors": "^2.8.5",
18     "dotenv": "^16.0.3",
19     "express": "^4.18.2",
20     "jsonwebtoken": "^9.0.0",
21     "mongoose": "^6.8.3",
22     "mongoose-unique-validator": "^3.1.0",
23     "multer": "^1.4.5-lts.1",
24     "validator": "^13.7.0"
25   }
26 }
27
```

App.js

⇒ centralisation

- des dépendances utilitaires
 - express
 - mongoose
 - cors
 - dotenv
- des routes

Routes



JS app.js > ...

```
1  You, 2 days ago | 1 author (You)
2  const express = require("express");
3  const app = express();
4  const path = require("path");
5
6  /* DOTENV
7  require("dotenv").config();
8
9  /* CORS BETWEEN API AND APPLICATION
10 const cors = require("cors");
11 app.use(cors());
12
13 /* ACCES AU CORPS DE LA REQUETE
14 app.use(express.json());
15
16 /* IMPORT MORGAN
17 const morgan = require("morgan");
18 app.use(morgan('dev'));
19
20 /* ACCES AUX ROUTES
21 const routesUser = require("./routes/routesUser");
22 app.use("/api/auth", routesUser);
23 const routesSauce = require("./routes/routesSauce");
24 app.use("/api/sauces", routesSauce);
25 app.use("/images", express.static(path.join(__dirname, "images")));
26
27 /* MONGOOSE => BDD
28 const mongoose = require("mongoose");
29 mongoose.set("strictQuery", true);
30
31 mongoose
32   .connect(process.env.MONGO_URL, {
33     useNewUrlParser: true,
34     useUnifiedTopology: true,
35   })
36   .then(() => console.log("Connexion à MongoDB réussie !"))
37   .catch(() => console.log("Connexion à MongoDB échouée !"));
38
39 module.exports = app;
```



PIQUANTE

PARTIE UTILISATEURS



Parcours utilisateur



Création du schéma utilisateur pour la DB



Création des routes, via le router d'Express

⇒ `router.post("/signup", ctrlUser.signup)`

⇒ `router.post("/login", ctrlUser.login)`



Création des contrôleurs

⇒ ensemble des fonctions utiles pour les différentes routes





Mise en place de sécurisation sur le modèle



- ⇒ contrôle que l'adresse email n'est pas déjà dans la data
- ⇒ mongoose-unique-validator
- ⇒ validation que la donnée saisie est bien un email
- ⇒ validator



Cryptage du mot de passe

- ⇒ utilisation de bcrypt dans la fonction signup
- ⇒ bcrypt



Mise en place d'un token d'identification

- ⇒ création d'un middleware avec la mise en place de jsonwebtoken
- ⇒ jsonwebtoken



Création du schéma utilisateur pour la DB



⇒ Installation des dépendances

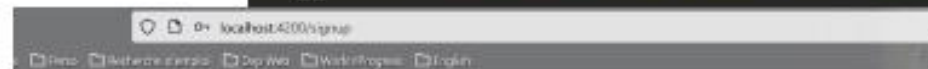
- mongoose-unique-validator
- validator

⇒ Utilisation de mongoose.Schema

⇒ Exportation du modèle

➡ pour une utilisation dans le controller

```
You, 3 hours ago | 1 author (You)
1  const mongoose = require("mongoose");
2  const uniqueValidator = require("mongoose-unique-validator");
3  const validator = require("validator");
4
5  const userSchema = mongoose.Schema({
6    email: {
7      type: String,
8      required: true,
9      unique: true,
10     validate: {
11       validator: function (v) {
12         return validator.isEmail(v)
13       },
14     },
15   },
16   password: { type: String, required: true },
17 });
18
19 userSchema.plugin(uniqueValidator);
20 module.exports = mongoose.model("User", userSchema);
21
22
```



HOT TAKES
THE WEB'S BEST HOT SAUCE REVIEWS

[SIGN UP](#)

[LOGIN](#)

Côté front



Email
piquante@gmail.com

Password

[SIGN UP](#)

Création des routes



⇒ Utilisation du router d'Express

⇒ Importation du controller user

⇒ Création des routes

- signup
- login

⇒ Exportation du router

routes > JS routesUser.js > ...

```
You, 7 days ago | 1 author (You)
1  /* IMPORT CONTROLLERS      You, 7 days ago • version
2  const express = require("express");
3  const router = express.Router();
4
5  const ctrlUser = require("../controllers/ctrlUser");
6
7
8  /* ROUTES
9  //POST => CREATE
10 router.post("/signup", ctrlUser.signup);
11 router.post("/login", ctrlUser.login);
12
13 module.exports = router;
14
```



Création des contrôleurs

JS

controllers > JS ctrlUser.js > login > login

```
You, 3 hours ago | 1 author (You)
1  /* IMPORT SCHEMA USER
2  const User = require("../models/User");
3
4  /*PACKAGES
5  /* BCrypt => CRYPTAGE DES MDP
6  const bcrypt = require("bcrypt");
7  /* TOKEN
8  const jwt = require("jsonwebtoken");
9  /* DOTENV
10 require("dotenv").config();
11
12 /* FONCTIONS
13 exports.signup = (req, res, next) => {
14   // UTILISATION BCrypt
15   bcrypt
16     .hash(req.body.password, 10)
17     .then((hash) => {
18       const user = new User({
19         email: req.body.email,
20         password: hash,
21       });
22       user
23         .save()
24         .then(() => res.status(201).json({ message: "Utilisateur créé !" }))
25         .catch((error) => res.status(400).json({ error }));
26     })
27     .catch((error) => res.status(500).json({ error }));
28 };
29
```

⇒ Importation du schéma utilisateur

⇒ Installation des dépendances

- bcrypt
- jsonwebtoken

⇒ Exportation des fonctions

➡ pour une utilisation sur la route correspondante



PIQUANTE

Côté console



201	POST	localhost:3000	signup	polyfills.js:5978...	json	306 B	34 B
204	OPTIONS	localhost:3000	login	xhr	plain	343 B	0 B
200	POST	localhost:3000	login	polyfills.js:5978...	json	492 B	224 B
204	OPTIONS	localhost:3000	saucis	xhr	plain	330 B	0 B

18 requests | 3.36 MB / 1.81 kB transferred | Finish: 444 ms

Headers | Cookies | Request | **Response** | Timings | Stack Trace

Filter properties

JSON Raw ☐

userId: "63d7c930f2b64def55c76d2c"

token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2M2Q3YzkyMGYyYjY0ZGVmNTVjNzZkMmMiLCJpYXQiOiJl2NzUwODYxMjksImV4cCI6MTY3NTEyMjE5OX0.1X8lotPqPuu9s-2gCVrArACxhmWk5D55XAo8ZE4SfuA"

⇒ POST Signup ⇒ 201 = created 🎉

⇒ POST Login ⇒ 200 = OK 🎉

- utilisation de l'userId & création du token



PIQUANTE

Création des contrôleurs

JS

```
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.ACCESS_TOKEN_SECRET,  
              { expiresIn: "10h" }  
            ),  
          });  
        })  
      .catch((error) => res.status(500).json({ error }));  
    })  
    .catch((error) => res.status(500).json({ error }));  
};
```

Ici je demande de retrouver l'utilisateur
via l'email de connexion



Si pas retrouvé Erreur 401 🦴



Si retrouvé, alors il doit vérifier le mot
de passe crypté via Bcrypt



Si pas bon Erreur 401 🦴



Si OK

- il utilise mon userId
- il m'attribue un token d'une validité
de 10 heures ici



PIQUANTE



Création du schéma sauce pour la DB



Création de middlewares

⇒ d'authentification

⇒ gestion des images



Création des 6 routes, via le router d'Express

⇒ POST, GET, PUT, DELETE



Création des contrôleurs

⇒ CRUD

⇒ ensemble des fonctions utiles pour les différentes routes



PARTIE

SAUCES



Création du schéma sauce pour la DB



models > JS Sauce.js > [x] sauceSchema

You, last week | 1 author (You)

```
1 const mongoose = require("mongoose");
2
3 const sauceSchema = mongoose.Schema({
4   userId: { type: String, required: true },
5   name: { type: String, required: true },
6   manufacturer: { type: String, required: true },
7   description: { type: String, required: true },
8   mainPepper: { type: String, required: true },
9   imageUrl: { type: String, required: true },
10  heat: { type: Number, required: true },
11  likes: { type: Number, default: 0 },
12  dislikes: { type: Number, default: 0 },
13  usersLiked: { type: [String] },
14  usersDisliked: { type: [String] },
15 });
16
17 module.exports = mongoose.model("Sauce", sauceSchema);
18
```

Côté base de données

- Certains champs sont générés directement par la BDD, comme l'userId ou les tableaux usersLiked et Disliked.
- D'autres sont complétées par l'utilisateur dans le cadre d'un POST



PIQUANTE

Côté front

ALL SAUCES

ADD SAUCE



HOT TAKES

THE WEB'S BEST HOT SAUCE
REVIEWS

LOGOUT

Name

Ketchup à l'ail

Manufacturer

Ketchup à l'ail

Description

Sauce crèmeuse à l'ail de Provence

ADD IMAGE



Main Pepper Ingredient

Ail

Heat



3

SUBMIT

Création de middlewares



⇒ d'authentification

```
middleware > JS auth.js > <unknown> > exports > decodedToken
You, 7 days ago | 1 author (You)
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.ACCESS_TOKEN_SECRET);
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 };
16
```

La fonction ici récupère le Token



J'enlève le mot clé "Bearer" situé devant le token avec la méthode split (" ")



Nous vérifions le Token avec la fonction verify()



Nous décodons l'userId du token



On ajoute cette valeur (l'userId) à l'objet request qui est transmise aux routes

⇒ utilisation du paquet jsonwebtoken

Mis en place sur toutes les routes puisque l'on a besoin d'être identifié systématiquement



PIQUANTE

Création de middlewares



⇒ pour la gestion des images

⇒ utilisation du paquet multer,
pour les images

Mis en place sur les routes qui ont
besoin de gérer des images

- **POST** api/sauces
- **PUT** api/sauces/:id

middleware > **JS** multer-config.js > ...

```
1  const multer = require("multer");
2
3  const MIME_TYPES = {
4    "image/jpg": "jpg",
5    "image/jpeg": "jpg",
6    "image/png": "png",
7  };
8
9  const storage = multer.diskStorage({
10    destination: (req, file, callback) => {
11      callback(null, "images");
12    },
13    filename: (req, file, callback) => {
14      const name = file.originalname.split(" ").join("_");
15      const extension = MIME_TYPES[file.mimetype];
16      callback(null, name + Date.now() + "." + extension);
17    },
18  });
19
20 module.exports = multer({ storage: storage }).single("image");
21
```



PIQUANTE

Création des routes ⇒ CRUD



⇒ POST

- CREATE : création de données comme les sauces

⇒ GET

- READ : lecture de la data
Ici, l'ensemble des sauces ou une sauce grâce à la seconde route sur l'ID

⇒ PUT

- UPDATE : mise à jour de la data

⇒ DELETE

- DELETE : effacer la data



PIQUANTE

```
routes > JS routesSauce.js > ...
```

```
You, 6 hours ago | 1 author (You)
1  /* IMPORT CONTROLLERS
2  const express = require("express");
3  const router = express.Router();
4
5  /* ⚠ METTRE EN PLACE L'AUTHT LORS DE L'ECRITURE DES ROUTES ⚠
6  const auth = require("../middleware/auth");
7  const multer = require("../middleware/multer-config");
8
9  const ctrlSauce = require("../controllers/ctrlSauce");
10 const ctrlLike = require("../controllers/ctrlLike");
11
12 /* ROUTES
13 //POST => CREATE
14 router.post("/", auth, multer, ctrlSauce.sauceCreated);
15 router.post("/:id/like", auth, ctrlLike.sauceLike);
16
17 //GET => READ
18 router.get("/", auth, ctrlSauce.sauceReadAll);
19 router.get("/:id", auth, ctrlSauce.sauceReadOne);
20
21 //PUT => UPDATE
22 router.put("/:id", auth, multer, ctrlSauce.sauceUpdate);
23
24 //DELETE => DELETE
25 router.delete("/:id", auth, ctrlSauce.sauceDelete);
26
27 module.exports = router;
28
```

Mise en place des middlewares d'authentification et image

Création des contrôleurs



⇒ ensemble des fonctions utiles pour les différentes routes

controllers > JS ctrlSauce.js > ...

```
1    You, 1 second ago | 1 author (You)
2    /* IMPORT SCHEMA SAUCE
3    const Sauce = require("../models/Sauce");
4
5    /* INTERACTION AVEC LE SYSTEME DE FICHIERS DU SERVEUR
6    const fs = require("fs");
7
8    /* POST
9    // router.post("/", auth, ctrlSauce.sauceCreated);
10   > exports.sauceCreated = (req, res, next) => { ...
25   };
26
27   /* GET
28   // router.get("/", (req, res, next) => {});
29   > exports.sauceReadAll = (req, res, next) => { ...
33   };
34
35   // router.get("/:id", (req, res, next) => {});
36   > exports.sauceReadOne = (req, res, next) => { ...
40   };
41
42   /* PUT
43   // router.put("/:id", (req, res, next) => {});
44   // ⚠ Format de la requête change si l'utilisateur nous envoie un fichier ou non => const sauceSend
45   > exports.sauceUpdate = (req, res, next) => { ...
71   };
72
73   /* DELETE
74   // router.delete("/:id", (req, res, next) => {});
75   > exports.sauceDelete = (req, res, next) => { ...
94   };
95
```

POST

- sauceCreated

GET

- sauceReadAll
- sauceReadOne

PUT

- sauceUpdate

DELETE

- sauceDelete



PIQUANTE



Création de la route

⇒ **POST** api/sauces/:id/like



Création du controller des likes

⇒ Like = +1

⇒ Dislike = -1

⇒ J'annule mon vote = 0



Like les sauces



⇒ création du controller like

```
1 // You'll need to import the
2 // import the SAUCE
3 const Sauce = require("../models/Sauce");
4
5 // POST
6 // router.post('/:id/like', (req, res, next) => {});
7 exports.sauceLike = (req, res, next) => {
8   // GET THE SAUCE => FINDONE ON ID
9   Sauce.findOne({ _id: req.params.id })
10    .then((sauce) => {
11      //
12    })
13    .catch((error) => res.status(404).json({ error }));
14 }
15
16
```

⇒ gestion des like

Mise en place d'un switch qui gère les 3 cas ...

- Like +1
- Dislike -1
- J'annule mon vote 0

➡ En fonction du vote, l'userId est placé dans le tableau

- userLike
- userDislike

créé dans le schéma des sauces

```
sauce.findOne({ _id: req.params.id })
  .then((sauce) => {
    // MISE EN PLACE D'UN SWITCH
    switch (req.body.like) {
      case 1:
        // USERID N'EST PAS DANS LE TABLEAU DES VOTES && USERID LIKE LA SAUCE => FICHE SAUCE MAJ
        if (
          !sauce.usersLiked.includes(req.body.userId) &&
          req.body.like === 1
        ) {
          // MAJ BDD
          Sauce.updateOne(
            { _id: req.params.id },
            {
              $inc: { likes: 1 },
              $push: { usersLiked: req.body.userId },
            }
          )
            .then(() => res.status(201).json({ message: "Like +1" }));
            .catch((error) => res.status(400).json({ error }));
        }
        break;
    }
  })
```



PIQUANTE