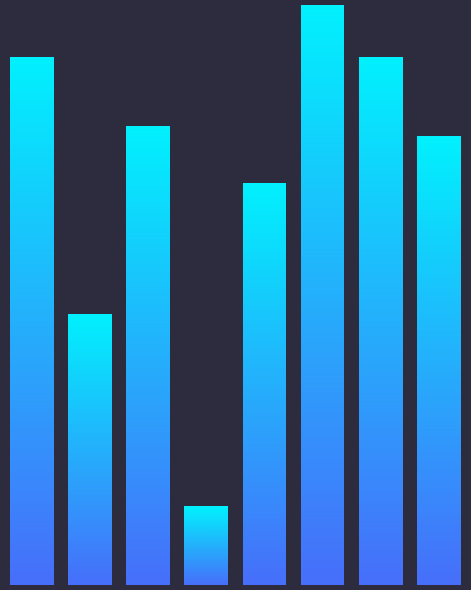
Réalisation API Piiquante



Lors de cette présentation, nous allons vous montrer comment nous avons conçu et développé une solution pour permettre aux utilisateurs d'ajouter, de consulter et de noter les sauces piquantes de leur choix. Au cours de cette présentation, nous allons couvrir les différentes étapes de la réalisation de cette API, des choix technologiques que nous avons faits jusqu'à la mise en œuvre de mesures de sécurité pour assurer la confidentialité des données utilisateur.

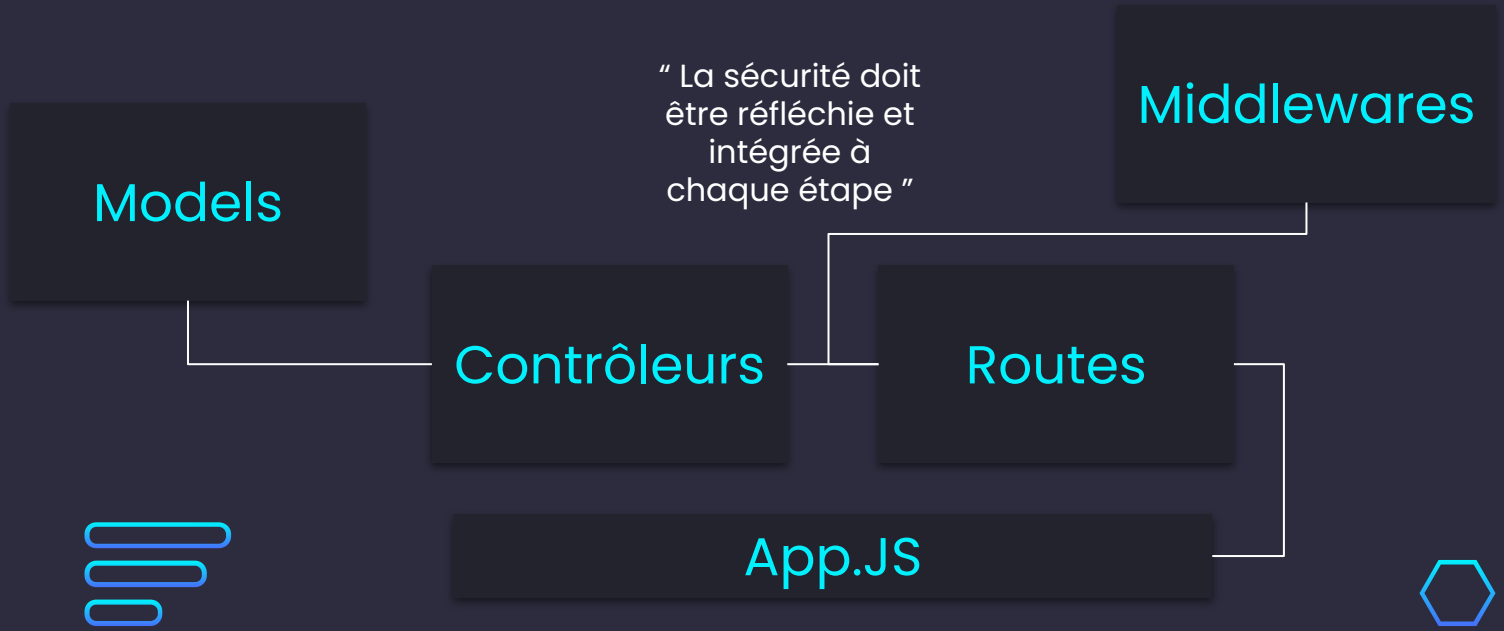
Technologies & outils 1/2

Node.js	Un environnement d'exécution pour JS sur le serveur, qui permet aux développeurs de créer des applications web côté serveur en utilisant JS
Express.js	Un framework Node.js pour construire des applications web et API grâce à sa flexibilité
MongoDB	Un système de gestion de données NoSQL performant et flexible qui permet de stocker et d'interagir avec des données de manière efficace

Technologies & outils 2/2

Bcrypt	Un module pour crypter et sécuriser les mots de passe
Dotenv	Un module pour stocker les informations sensibles de l'application
Express-rate-limite	Un module pour limiter le nombre de requêtes HTTP à une API
Helmet	Un module pour renforcer la sécurité des applications Express
Jsonwebtoken	Un module pour gérer les tokens JWT pour l'authentification
Mongoose	Un ORM pour MongoDB qui facilite la gestion des données
Mongoose-unique-validator	Un plugin pour Mongoose qui ajoute une validation d'unicité pour les données
Multer	Un module pour gérer les fichiers multimédia dans une application Express.js

Architecture



Models 1/2

Ils définissent les types de données, les relations entre les différentes entités et les règles de validation des données. Les modèles sont créés à l'aide d'une bibliothèque telle que Mongoose dans le cas de MongoDB

User.js

Définit le modèle pour les utilisateurs, incluant les informations d'identification telles que l'identifiant et le mot de passe.

Sauce.js

Définit un modèle de données pour les sauces piquantes. Il utilise la bibliothèque Mongoose pour définir le schéma de données qui décrit les différents champs requis pour une entrée de sauce.

Models 2/2

User.js

```
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, unique: true}
});

userSchema.plugin(uniqueValidator);

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

Sauce.js

```
const mongoose = require('mongoose');

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

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

Contrôleurs 1/2

Ils gèrent les requêtes HTTP reçues par l'API, en utilisant les modèles pour interagir avec la base de données. Ils définissent les logiques métiers et les réponses à envoyer en fonction des requêtes. Les contrôleurs permettent de séparer les différentes tâches de l'API en segments distincts, ce qui facilite la maintenance et l'évolution du code.

User.js

Il gère les fonctionnalités de l'inscription et de la connexion de l'utilisateur en utilisant les modules "bcrypt" pour hasher le mot de passe, "jsonwebtoken" pour générer le token JWT, et "dotenv" pour accéder à la clé secrète de l'environnement. Il utilise le modèle "user" pour créer un nouvel utilisateur et trouver un utilisateur existant par son email. En cas de succès de l'inscription ou de la connexion, un message de réponse et un token JWT sont renvoyés, sinon un message d'erreur est envoyé

Sauce.js

Il définit les fonctions suivantes :

- **getAllSauces** : Récupère toutes les sauces de la base de données
- **getOneSauce** : Récupère une sauce en particulier en fonction de son identifiant
- **createSauce** : Crée une nouvelle sauce dans la base de données
- **modifySauce** : Modifie une sauce existante dans la base de données
- **deleteSauce** : Supprime une sauce de la base de données
- **likeSauce** : Ajoute un like ou un dislike à une sauce

Contrôleurs 2/2

User.js

```
exports.login = (req, res, next) => {
    User.findOne({email: req.body.email})
    .then(user => {
        if (user === null) {
            res.status(401).json({message: 'Paire identifiant
        } else {
            bcrypt.compare(req.body.password, user.password)
            .then(valid => {
                if (!valid) {
                    res.status(401).json({message: 'Paire ide
                } else {
                    res.status(200).json({
                        userId: user._id,
                        token: jwt.sign(
                            {userId: user._id},
                            process.env.JWT_TOKEN,
                            { expiresIn: '24h'}
                        )
                    });
```

Sauce.js

```
exports.createSauce = (req, res, next) => {
    const sauceObject = JSON.parse(req.body.sauce);
    delete sauceObject._id;
    const sauce = new Sauce ({
        ...sauceObject,
        userId: req.auth.userId,
        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}))
};
```

Middlewares 1/2

Sont des fonctions intermédiaires qui s'exécutent entre la réception d'une requête et la délégation de cette requête à une route ou un contrôleur approprié. Ils peuvent être utilisés pour exécuter des tâches telles que la validation de la requête, l'authentification de l'utilisateur, la modification des données de la requête, etc. Ils peuvent être partagés entre plusieurs routes.

Auth.js

Il nous permet de vérifier la validité du token d'authentification envoyé dans la requête HTTP. Il utilise la bibliothèque "jsonwebtoken" pour décoder le token et vérifier qu'il a été signé avec la clé correcte. Si le token est valide, l'identifiant de l'utilisateur est ajouté à l'objet de requêtes sous la forme de "req.auth.userId". Si le token n'est pas valide, une erreur 401 est envoyée

Multer-config.js

Il traite le fichier en utilisant le package Multer. Il définit un stockage pour les fichiers images téléchargés et vérifie le type MIME des fichiers pour s'assurer qu'ils sont des images JPEG ou PNG. Les fichiers sont enregistrés dans le répertoire "images" et leur nom est modifié pour inclure la date actuelle afin d'éviter les conflits de nom de fichier.

Middlewares 2/2

Auth.js

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
    try {
        const token = req.headers.authorization.split(" ")[1];
        const decodedToken = jwt.verify(token, process.env.JWT_TOKEN);
        const userId = decodedToken.userId;
        req.auth = {userId};
        next();
    } catch (error) {
        res.status(401).json({ error });
    }
};      You, last week • #
```

Multer-config.js

```
const multer = require('multer');

const MIME_TYPES = {
    'image/jpg': 'jpg',
    'image/jpeg': 'jpg',
    'image/png': 'png'

};

const storage = multer.diskStorage({
    destination: (req, file, callback) => {
        callback(null, 'images')
    },
    filename: (req, file, callback) => {
        const name = file.originalname.split(' ').join('_');
        const extension = MIME_TYPES[file.mimetype];
        callback(null, name + Date.now() + '.' + extension);
    }
});

module.exports = multer({storage}).single('image');      You,
```

Routes 1/2

Elles sont des points d'entrée pour les requêtes HTTP. Elles définissent comment les données sont reçues et envoyées à l'application. Chaque route est associée à une fonction de contrôleur qui gère la logique de l'application. Elles sont définies pour les opérations CRUD (Create, Read, Update, Delete).

User.js

Les deux routes définies ici sont :

- POST"/signup" : qui la création d'un nouvel utilisateur
- POST"/login" : qui gère la connexion utilisateur

Sauce.js

Les routes utilise le middleware "auth" et "multer-config"

- GET"/" : récupère l'ensemble des sauces
- GET"/:id" : récupère une sauce en particulier
- POST"/" : Crée une nouvelle sauce
- PUT"/:id" : Modifie une sauce existante
- DELETE"/:id" : Supprime une sauce existante
- POST"/:id/like" : Ajoute un like /dislike sur une sauce existante

Routes 2/2

User.js

```
const express = require('express');
const router = express.Router();
const userCtrl = require('../controllers/user');

router.post('/signup', userCtrl.signup);
router.post('/login', userCtrl.login);

module.exports = router;        You, last week • #
```

Sauce.js

```
const express = require('express');
const router = express.Router();
const auth = require('../middleware/auth');
const multer =require('../middleware/multer-config');

const sauceCtrl = require('../controllers/sauce');

router.get('/', auth, sauceCtrl.getAllSauces);
router.get('/:id', auth, sauceCtrl.getOneSauce);
router.post('/', auth, multer, sauceCtrl.createSauce);
router.put('/:id', auth, multer, sauceCtrl.modifySauce);
router.delete('/:id', auth, sauceCtrl.deleteSauce);
router.post('/:id/like', auth, sauceCtrl.likeSauce);

module.exports = router;        You, last week • #
```

Mesures de sécurité

Plusieurs mesures de sécurité sont mises en place. Tout d'abord, l'authentification des utilisateurs est gérée à l'aide de token JWT, qui est vérifié à l'aide du middleware 'auth.js' lors de chaque requête qui nécessite une authentification.

Modules & Sécurité

Dotenv

Permet de sécuriser les information sensibles comme la connection à la base de donnée ainsi que le secret Token

Rate-Limite

Limite le nombre de requêtes(100) d'une source spécifique dans un intervalle de temps donné(15min) Pour éviter certaines attaques DDoS

Unique-validator

Vérifie l'unicité des données enregistrées dans la base de données

Helmet

Aide à renforcer la sécurité en définissant les en-têtes HTTP de sécurité appropriés

Bcrypt

Crypte les mots de passe avant de les stocker dans la base de données, protège contre les attaques de base de données

Merci !

Nous passons maintenant
à la partie questions,
discussion