Développement d’une API

Piiquante

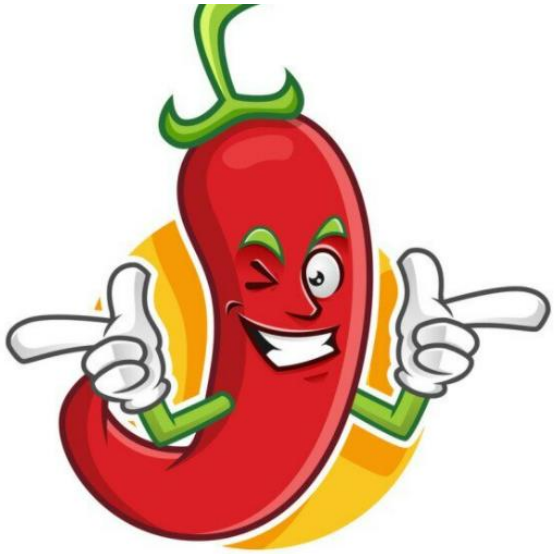
La mission

La mission

- Développer l'API de gestion des utilisateurs et des sauces.
- La partie frontend est déjà développée.
- La partie backend à développer consiste à
  - Gérer les utilisateurs
  - Gérer les sauces crées par chaque utilisateur et permettre de liker ou disliker chaque sauce.

La mission en détails

- Permettre la création de nouveaux utilisateurs
- Permettre le login des utilisateurs
- Pour chaque utilisateurs, créer, modifier ou détruire des sauces
- Pour tous: ajouter un like ou dislike sur chaque sauce
- Etre conforme à la sécurité

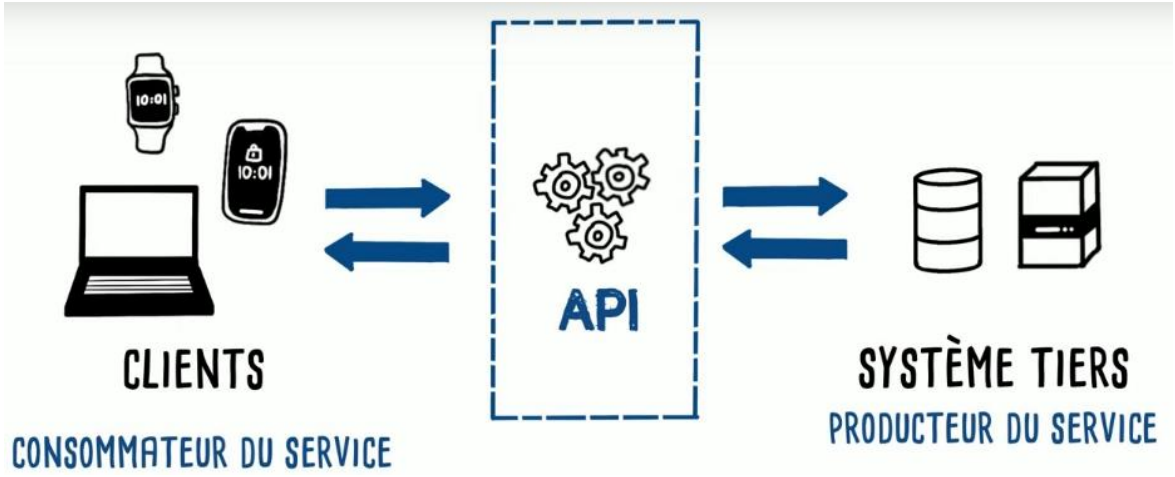


Les outils

Définition: Une API qu'est-ce que c'est ?

- **API: (**Application Programming Interface) signifie en français : « Interface de Programmation d'Application».

- une interface de programmation permet d'accéder à des services fournis par un système tiers.



NodeJS

- Le développement de l'API se fait avec NodeJS et Express

- Node permet d'écrire toutes les tâches côté serveur, en JavaScript, telles que la logique métier, la persistance des données et la sécurité.

- Express est un environnement de développement reposant sur Node, qui facilite la création et la gestion des serveurs Node.

Format des données: JSON

- Tous les objets stockés sont au format JSON: JavaScript Object Notation qui est le format le plus simple pour transmettre les données.
  - **Json.parse()** transforme le json en objet java script.

Base de données MongoDB

- Base de données noSQL
- Pas de schéma prédéfini des données.
- Format des données: BSON: JSON binaire.
- Facilement accessibles en java script.

Le développement

Dialogue avec l'API

- Pour communiquer avec l'API, il faut lui envoyer des requêtes respectant le protocole HTTP.
- Avec:
  - Une méthode:
    - **GET** : permet de **récupérer** des ressources
    - **POST** : permet de **créer** ou **modifier** une ressource
    - **PUT** : permet de **modifier** une ressource
    - **DELETE** : Permet de **supprimer** une ressource
  - Une URL
  - Des données

Le CRUD

- Le CRUD est un acronyme pour les actions sur des données stockées
  - CREATE : Créer
  - Read : Lire
  - Update : Mettre à jour
  - Delete : Supprimer

- L'API de piiquante est développée pour réaliser chacune de ces actions pour les utilisateurs et les produits.

Les models

Données définies sous forme de schéma de données:

Utilisateurs:

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

Les models

Données définies sous forme de schéma de données:

Sauces:

```
//Creation schema pour les sauces

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

Les routes

- Les routes indiquent l'URL à utiliser dans la requête à envoyer à l'API par le frontend .
- Chaque route correspondra à une action décrite dans le controller.

Enregistrement et login des Utilisateurs:

```
//routes POST pour signup et login envoyé par le frontend
router.post("/signup", userCtrl.signup);
router.post("/login", userCtrl.login);
```

Les routes

- Les routes pour les sauces . Chaque route est associée à une action:
  - POST: Création d'une sauce
  - PUT: modification d'une sauce donnée par son id
  - GET: Récupération de toutes les sauces
  - GET et id: récupération d'une sauce

```
router.post("/", auth, multer, sauceCtrl.createSauce);
router.put("/:id", auth, multer, sauceCtrl.modifySauce);

router.get("/", auth, sauceCtrl.getAllSauce);
router.get("/:id", auth, sauceCtrl.getOneSauce);

router.delete("/:id", auth, sauceCtrl.deleteSauce);
```

Les routes

- Les routes pour les like . Chaque route est associée à une action:
  - POST:
    - Id de la sauce contenu dans la route.
    - Like/dislike dans le controller

```
//Route pour le like
router.post("/:id/like", auth, sauceCtrl.createLike);
```

Les controllers

Les controllers permettent d'associer une action à chaque route.

- Controllers pour les users:
  - Export du modèle de user

```
//modele users
const User = require("../models/User");
```

- Fonctions:
  - POST: signup pour l'enregistrement
    - Avec chiffrement du mot de passe
  - POST Login: pour le login
    - Vérification du passwd
    - Retour d'un token au frontend

Les controllers

- Controllers pour les sauces:
  - Export du modèle de la sauce
  - Chaque fonction correspond à un verbe
    - Retour d'un code d'erreur
    - Ou de bonne exécution
- Fonctions:
    - POST: exports.createSauce : création d'une sauce
    - GET: exports.getAllSauce: récupération de toutes les sauces
    - GET avec id : exports.getOneSauce: récupération d'une sauce
    - PUT: exports.modifySauce : modification d'une sauce
    - DELETE: exports.deleteSauce : Suppression d'une sauce

```
const Sauce = require("../models/Sauce");
```

Les controllers

- Controllers pour les like:

- Traitement du like ou dislike envoyé par le frontend
  - Route avec id de la sauce
  - Valeur du like dans les paramètres.
    - 1: like
    - -1 : dislike
    - 0: suppression du like ou dislike

- Fonction:
  - POST: exports.createLike

La gestion des images : multer

- Il est possible de gérer les images pour les stocker dans le disque.

```
const storage = multer.diskStorage({
  destination: (req, file, callback) => {
    //1ier argument = la callback      You, last week • Ajout middleware/multer-config.js
    callback(null, "images"); // null: pas d'erreur  images= nom du dossier de stockage
  },

  //2ieme argument  : nom de fichier à utiliser
  filename: (req, file, callback) => {
    const name = file.originalname.split(" ").join("_"); //nom d'origine avec blancs remp
    const extension = MIME_TYPES[file.mimetype]; //ajout de l'extension en ft du dossier
    callback(null, name + Date.now() + "." + extension); // appel callback : null : pas d
    //nom + date+ extension
  },
});
```

La sécurité

Sécurité - OWASP

- **OWASP: Open Web Application Security Project**
- (**OWASP**) est une communauté en ligne travaillant sur la sécurité des applications Web.
- Elle publie des recommandations de sécurisation WEB

Conseils de sécurité

- Utilisation de hash pour les passwd
- Utilisation de token pour authentifier les utilisateurs.
- Utilisation du fichier .env pour stocker les données sensibles comme les clés secrètes et les identifiants de connexion à la base de données.
- Vérifier les informations dans le server

sécurité – Les mots de passe

- **Hash des mots de passe**
  - Pour plus de sécurité les mots de passe ne doivent pas être stockés en clair.
  - Signup: les passwd sont enregistrés hashé
  - Login:
    - bcrypt vérifie le mot de passe envoyé par l'utilisateur avec le hash enregistré
    - Si correct renvoi du TOKEN au frontend

Protection des données – Les middleware

- Gestion d’authentification des utilisateurs pour protéger les routes sensibles.
- Création d'un middleware "auth.js" pour vérifier l'utilisateur.
- Chaque requête demandant une modification ou suppression d'un objet sera validée par le middleware d'authentification.
- Il est indispensable de vérifier que l’utilisateur qui demande une action est bien le propriétaire de la sauce.
    - Recherche du propriétaire de la sauce dans la base de données
    - Vérification du token d’authentification

Les tests

Tests d'acceptation

- Chaque développement doit être accompagné d'un plan de test.
- On distingue:
  - Les tests nominaux: fonctionnement normal
  - Les tests d'erreur: fonctionnement quand une erreur est commise ou une action non demandée est effectuée
  - Les tests aux limites quand on utilise le produit au maximum de sa capacité.

- Pour réaliser le plan de test :
  - Pour chaque page, test du fonctionnement normal
  - Test des fonctionnements en erreur

Tests d'acceptation – Ajout/modification/suppression

- Test:
  - SIGNUP
  - LOGIN

  - CREATION
  - MODIFICATION
  - SUPPRESSION

Tests d'acceptation – like et dislike

- Tester:
  - Ajout/suppression de like


```
▼[{_id: "624c0c77d5f63ae981bdc881", userId: "624ab3e0c5282bb93200a2be", name: "Jonquilles",…},…]
  ▼0: {_id: "624c0c77d5f63ae981bdc881", userId: "624ab3e0c5282bb93200a2be", name: "Jonquilles",…}
      description: "aux chataignes"
      dislikes: 0
      heat: 3
      imageUrl: "http://localhost:3000/images/Jonquille.png1649353912849.png"
      likes: 1
      mainPepper: "chataignes"
      manufacturer: "Marcel"
      name: "Jonquilles"
      userId: "624ab3e0c5282bb93200a2be"
      usersDisliked: []
    ▶ usersLiked: ["624ab3e0c5282bb93200a2be"]
      __v: 0
      _id: "624c0c77d5f63ae981bdc881"
```

Tests d’erreur – avec POSTMAN

- Le frontend ne permet pas de faire des tests d’erreur poussés .
  - Exemple: Suppression d’une sauce par un utilisateur non propriétaire.

  - Tests DELETE et PUT:
    - Utilisation de POSTMAN avec:
      - TOKEN du user connecté
      - requête: DELETE

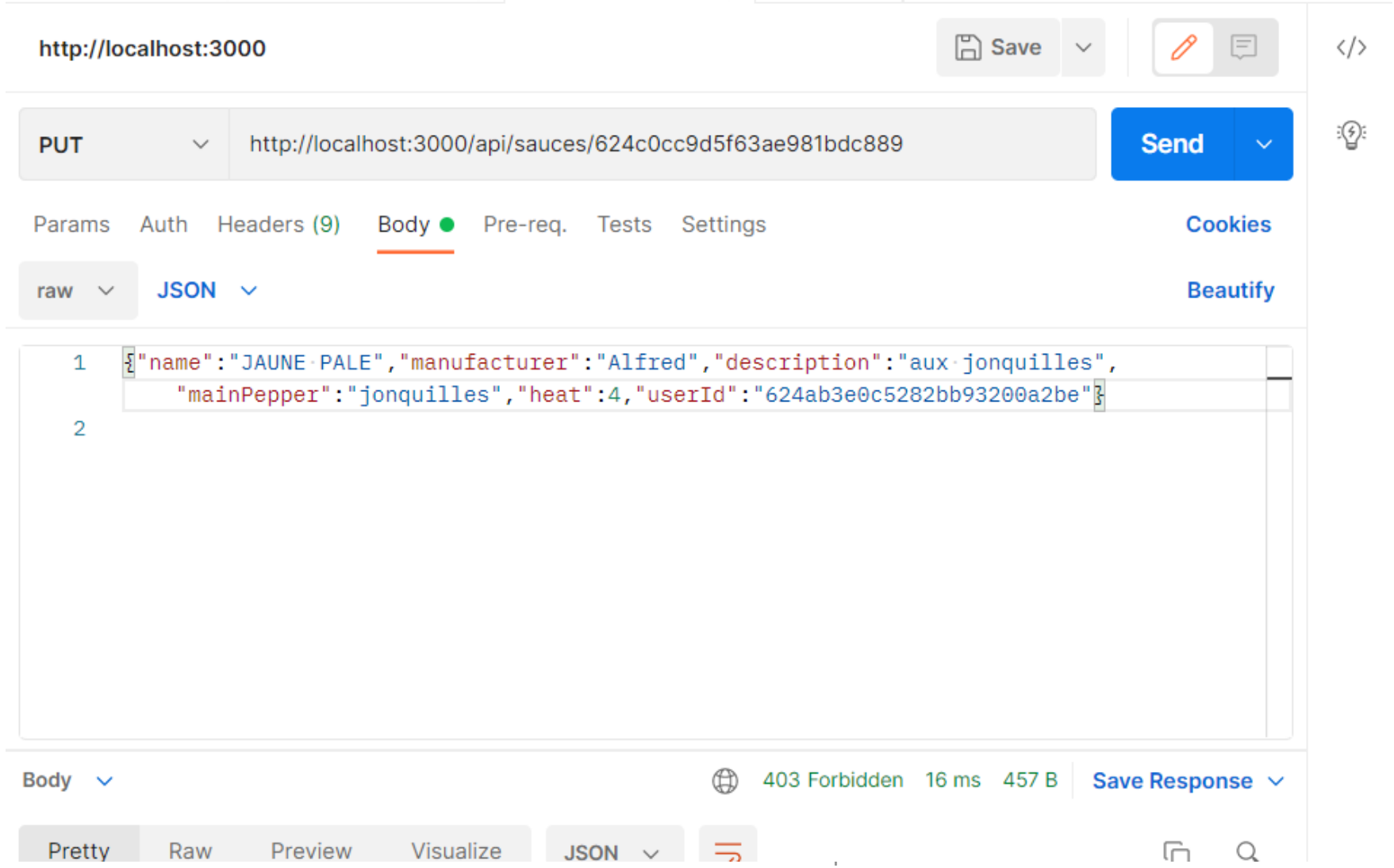
Tests d’erreur – avec POSTMAN

- Exemple: falsification du body en mettant le userId du user loggué
- Et on tente de modifier une sauce qui ne nous appartient pas:

->Retour erreur 403

- **Modification refusée**

Tests d’erreur – avec POSTMAN



conclusion

Conclusion

L’API de piiquante a été implémentée en respectant la sécurité

Le code a été testé:

- A partir du frontend
- A partir de l'outil POSTMAN pour les requetes non accessibles par le frontend.

Liens utiles

- Définition API en 4 mn
- NodeJs: https://nodejs.org/
- MongoDB: https://www.mongodb.com

Installations

Titre	Installation	A partir de	Explication
npm init	npm init	backend	
nodemon	npm install –g nodemon	backend	nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.
express	npm install express –save	backend	Express est un environnement de développement reposant sur Node, qui facilite la création et la gestion des serveurs Node.
mongoose	npm install mongoose	backend	Interface avec MongoDB
validator	npm install --save mongoose-unique-validator	backend	Evite duplication dans la base de données
bcrypt	npm install --save bcrypt	backend	Cryptage du passwprd
basic-auth	`npm install --save basic-auth`	backend	Vérification authentification
jsonwebtoken	npm install --save jsonwebtoken	backend	Création d'un token pour l'utilisateur
multer	npm install --save multer	backend	téléchargement des fichiers
dotenv	`npm install dotenv --save`	backend	Gestion du fichier pour les variables d'environnement sensibles comme user/passwd ou clés secrétes.
postman	https://www.postman.com/downloads/		Vérification des requetes POSTMAN pour un server local

Glossaire

- API: Application Programming Interface
- CRUD: Create Read Update Delete
- Framework: infrastructure de développement
- Hash: Un hash est le résultat de l'application d'une fonction de hachage cryptographique. Pour vérifier la validité du mot de passe d'un utilisateur, il suffit de calculer le hash du mot de passe envoyé et de comparer à la valeur stockée en base de données.
- JSON: JavaScript Object Notation.
- HTTP: HyperText Transfert Protocol
- NodeJS: est une plateforme logicielle libre en JavaScript, orientée vers les applications réseau évènementielles
- REST: Representational State Transfer