

Développement web & API

Qu'est-ce qu'une API/API REST

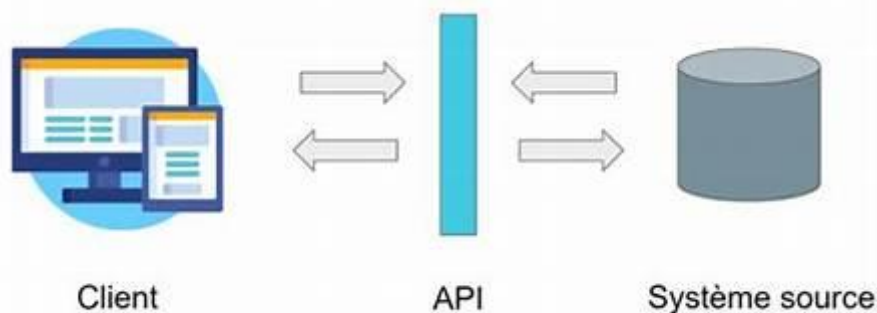
Une API (Application Programming Interface) est ce qui nous permettra de faire communiquer un client et un serveur (Le client désigne la machine qui va chercher une ressource sur une api l'api est coté serveur). L'API fait l'intermédiaire et « contrôle/réglemente » la communication.

Une API REST (Representational State Transfer), standard utilisé pour construire des APIs, s'appuie sur le protocole http.

Ce qui la caractérise :

- Elle est stateless : les deux parties communicantes n'ont pas à savoir sur qui elles communiquent ;
- Elle possède une architecture client-serveur constituée de clients, de serveurs et de ressources, avec des requêtes gérées via http (donc des requêtes conformes au protocole http) ;
- La gestion des ressources (qui sont identifiées par des URL et manipulées dans les requêtes) ;
- La représentation des ressources (souvent) dans le fichier json ;
- L'utilisation des status code (cause du suivi du protocole http).

Dans une API, le client est celui qui demande une ressource à l'API, tandis que l'API est côté serveur et fournit les ressources demandées.



L'API REST avec CRUD et OAuth

Les requêtes HTTP envoyées par le client contiennent des en-têtes (headers) qui sont des métadonnées associées à la requête. Parmi les en-têtes mentionnés ici, nous avons :

- Accept : Indique le format de réponse souhaité par le client.

- Authorization : Permet au serveur de vérifier si le client a le droit d'effectuer la requête.
- Body : Contient le corps de la requête, généralement utilisé dans les requêtes de type POST, PUT, ou PATCH pour envoyer des données au serveur.

Les requêtes HTTP les plus courantes sont :

- GET : pour récupérer des données(données dans l'URL).
- POST : pour créer de nouvelles ressources(données dans le corps de la requête (req.body)).
- PATCH : pour mettre à jour partiellement une ressource.
- DELETE : pour supprimer une ressource (identifiant dans l'URL (req.params)).
- PUT : pour mettre à jour entièrement une ressource (données à mettre à jour dans le corps, identifiant dans l'URL (req.body + req.params)).

Ceci est l'utilisation des opérations CRUD (Create, Read, Update, Delete). En utilisant les méthodes HTTP standardisées pour les opérations CRUD, l'API permet aux clients d'interagir de manière cohérente avec les ressources.

Chaque requête API est associée à un "endpoint", qui est l'URL spécifique permettant d'accéder à une ressource. Les clients peuvent accéder à ces endpoints en utilisant les méthodes HTTP appropriées pour accomplir leurs tâches.

OAuth est un protocole utilisé pour sécuriser l'accès aux ressources d'une API, en autorisant des applications tierces à accéder aux données d'un utilisateur sans nécessiter ses identifiants de connexion complets. Voici le fonctionnement :

Lorsqu'une application tierce souhaite accéder aux ressources d'un utilisateur, elle le redirige vers le fournisseur d'identité pour demander l'autorisation.

L'utilisateur est ensuite invité à donner son consentement pour permettre à l'application tierce d'accéder à ses ressources.

Une fois que l'utilisateur a donné son consentement, le fournisseur d'identité génère un jeton d'accès (le token(jsonwebtoken)) sécurisé. Le token représente les droits d'accès accordés à l'application tierce.

L'application tierce peut maintenant utiliser le token pour accéder aux ressources du fournisseur d'identité au nom de l'utilisateur, sans avoir besoin de demander les identifiants de connexion de l'utilisateur.

OAuth offre un moyen sûr d'autoriser l'accès aux ressources, car les identifiants de connexion de l'utilisateur ne sont jamais partagés avec l'application tierce. Le token a une durée de validité limitée et peut être révoqué à tout moment si nécessaire, renforçant ainsi la sécurité de l'accès aux ressources.

Express et node.js

Express est un framework web qui facilite la création d'APIs et d'applications web en utilisant Node.js et JavaScript. Voici les deux briques fondamentales qui caractérisent Express :

- les routes :

Les routes dans Express définissent les endpoints de l'API. Chaque route est associée à une URL spécifique et à une méthode HTTP (GET, POST, PUT, DELETE, etc.). Lorsqu'un client envoie une requête HTTP à une URL donnée, Express compare cette URL avec les routes définies pour trouver celle qui correspond à la requête. Une fois qu'une route correspondante est trouvée, la fonction de gestionnaire associée à cette route est exécutée pour gérer la requête et renvoyer une réponse au client.

Voici comment se présente les routes (exemple dans mon api)

```
10 router.get('/events', isAuthenticated, eventController.getEvents);
11
```

- le middleware :

Les middleware sont des fonctions intermédiaires qui sont exécutées avant que la requête n'atteigne la route finale. Les middleware peuvent effectuer des actions spécifiques, telles que l'authentification, la validation des données, l'enregistrement de journaux, la compression, etc. Les middleware peuvent être globaux, appliqués à toutes les routes, ou spécifiques à certaines routes.

Exemple de middleware de mon api :

```
3
4 app.use('/auth', authRoutes);
5
6 app.use('/events', eventRoutes);
7
8
```

App.use est la méthode d'express utilisée pour monter un middleware.

'/auth' est l'url du middlewre eventRoutes

Les middlewares permettent de séparer et d'organiser les routes de l'application.

En combinant ces deux briques, on peut créer des APIs Express flexibles. Les routes définissent les points d'accès à l'API et les middlewares permettent de gérer des actions communes avant d'atteindre la route finale. Cette approche facilite la création d'APIs, évolutives et bien structurées.

Node.js est un environnement d'exécution qui permet de créer des applications côté serveur en javascript. Met à disposition outils et modules pour faciliter le développement.

Pour initier un projet node :

- npm init

à la suite de cette commande, un fichier « package.json » est créé, avec les informations du projet initié.

Les dépendances sont gérées dans le dossier nodemodules, qui apparaît à la suite de l'installation de celle-ci avec la commande npm install/ npm -D

Pour installer express :

- npm install express

Les "contrôleurs" sont des services qui gèrent les interactions avec la base de données. Les "routes" font l'association entre les URIs (uniform resource locators)--> permet d'identifier les différents types de ressources disponibles) et les requêtes HTTP, en définissant les points d'accès de l'API.

Nodemon est une dépendance de développement, utilisé pour redémarrer automatiquement le serveur chaque fois qu'un changement est sauvegardé dans les fichiers source.

Pour installer cette dépendance : npm install nodemon --save-dev

La gestion des erreurs est possible avec un middleware d'erreurs, en voici un exemple :

```
app.use((error, req, res, next) => {  
  console.log(error);  
  const status = error.statusCode || 500;  
  const message = error.message;  
  const data = error.data;  
  res.status(status).json({ message: message, data: data });  
});
```

mongoose

Le middleware gestionnaire d'erreurs capture les erreurs survenues lors du traitement des requêtes, puis renvoie une réponse JSON contenant le message d'erreur et les données associées, avec un code de statut HTTP approprié.

Status Codes (Codes de statut)

Les status codes sont inclus dans la réponse HTTP renvoyée par le serveur en réponse à la requête du client. Ils indiquent le résultat de manière standardisée. Voici des status codes couramment utilisés :

200 OK : Indique que la requête du client a été traitée avec succès et que la réponse contient les données demandées.

201 Created : Indique que la requête du client a été traitée avec succès et qu'une nouvelle ressource a été créée sur le serveur.

400 Bad Request : Indique que la requête du client est incorrecte ou mal formée, et que le serveur ne peut pas la traiter.

401 Unauthorized : Indique que l'accès à la ressource demandée nécessite une authentification et que l'utilisateur n'est pas correctement authentifié ou n'a pas les autorisations nécessaires.

404 Not Found : Indique que la ressource demandée n'a pas été trouvée sur le serveur.

Autres outils, technologies et développement

Pour utiliser les APIs, des outils tels que Postman ou Insomnia peuvent être utilisés pour envoyer des requêtes et recevoir des réponses.

MERN (MongoDB, Express, React, Node) et LAMP (Linux, Apache, MySQL, PHP) sont piles logicielles couramment utilisées pour développer des applications.

ORM (Object-Relational Mapping) et ODM (Object-Document Mapping) sont des outils qui permettent de manipuler la base de données avec du code plutôt qu'avec des requêtes SQL brutes.

Pour déboguer une application, on peut utiliser des fonctions telles que `console.log()` pour afficher des messages dans la console du navigateur ou du serveur.

MVC (modèle-vue-contrôleur) est un modèle d'architecture logicielle. Ce modèle permet d'avoir une certaine organisation dans un projet. Sépare le projet en plusieurs fichiers, à savoir le modèle (qui gère les données et les règles associées), la vue (représentent les données structurées renvoyées au client) et le contrôleurs (traite les demandes des clients).

Une Promise en JavaScript est un objet utilisé pour représenter une valeur qui peut être disponible immédiatement, plus tard ou jamais. Elles sont utilisées pour gérer les opérations asynchrones. (une promise est commenté dans le code)

Mon API REST : « gestionnaire » d'évènements

La consigne : « Un simple CRUD, une architecture qui fait intervenir routage et controller, une connexion a une base de donnees, la creation d'un middleware, un systeme de login par echange de JWT simple »

Après l'étude de chacun des fichiers mis à disposition sur teams, j'ai pu m'en servir pour réaliser ma propre api. API qui permet d'ajouter, de récupérer des évènements etc...

L'api se compose de différents dossiers :

- controllers : contient les controllers (le « cerveau » de l'api) responsable de la logique métier de l'api. Ils assurent la coordination entre les requêtes entrantes, les modèles de données et les règles métier, et ils génèrent les réponses appropriées pour les renvoyer aux clients.
- middleware : contient middleware d'authentification « s'occupe » de la sécurité
- models : contient les fichiers qui définissent les schémas de données utilisés par l'api. Les modèles décrivent comment les informations doivent être organisées et stockées.
- routes : contient les fichiers qui définissent les routes. Chaque route est associée à une ou plusieurs actions spécifiques.
- Et le fichier app.js : point d'entrée principal de l'api, initialise et configure toute l'application en utilisant les différents composants définis dans les dossiers controllers, middleware, models, et routes. Dans ce fichier, on importe les dépendances nécessaires, configurez votre serveur, définissez les middleware, les routes, et on lie les contrôleurs aux différentes actions de l'api.

sans compter le nodemodules et les fichiers json.

On reconnait ainsi le modèle MVC.

Etant une application back-end, l'api récupère des infos mais doit les stocker. Pour cela j'ai créer une base de données et je l'ai connectée à mon application.

J'y ai créer deux collections : « users » et « events ».

Pour que vous puissiez la tester, vous trouverez dans le dossier les fichiers json de cette base de données.

Url de connexion:

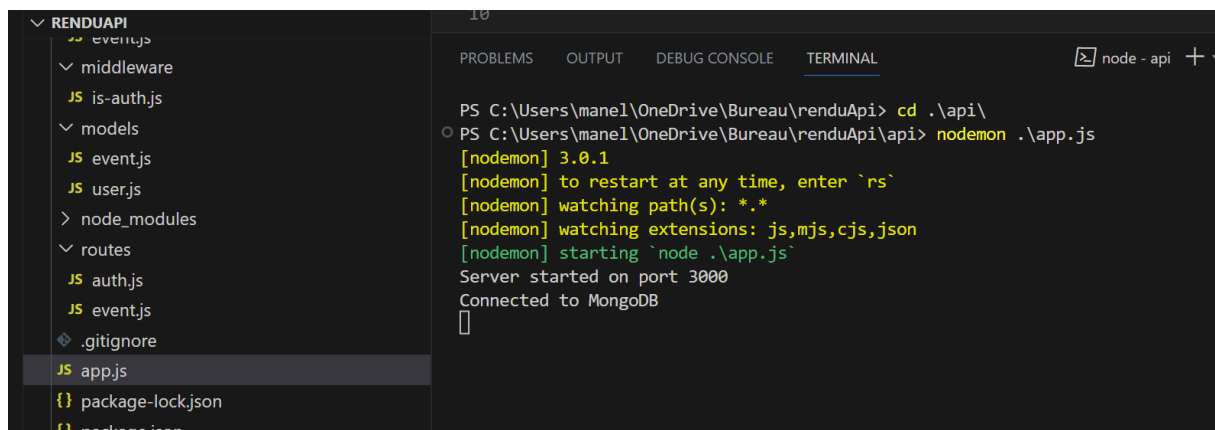
mongodb://127.0.0.1:27017/dbapi

Garder l'adresse 127.0.0.1 et ne pas remplacer par localhost

27017: correspond au port utilisé par mongodb et vérifier que le port utilisé par mongodb est bien celui-ci. Sinon remplacer par le port utilisé.

Dbapi correspond à la bdd.

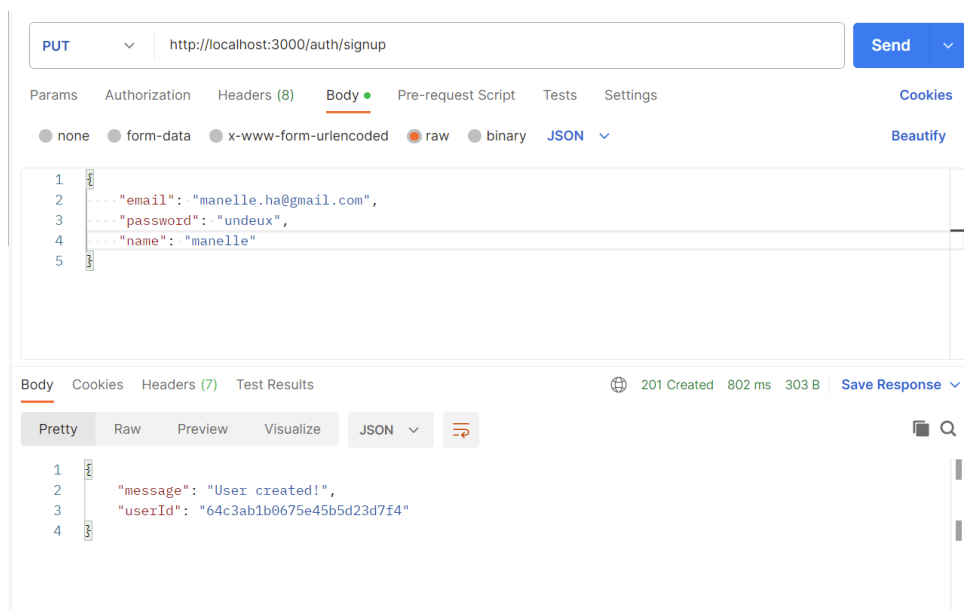
Une fois l'api terminé et la connexion à la base de données établie, j'obtiens cela dans le terminal :



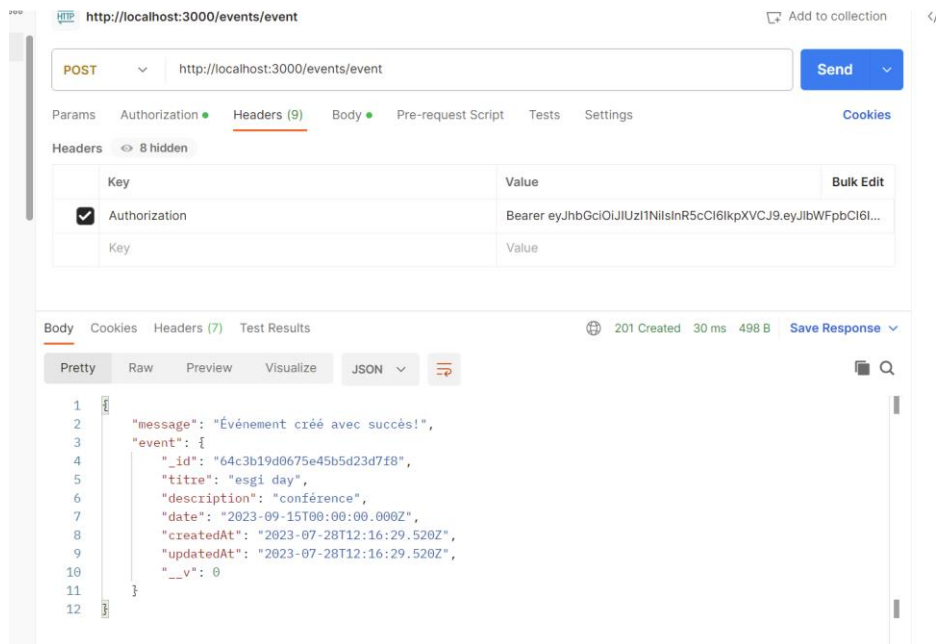
```
PS C:\Users\manel\OneDrive\Bureau\renduApi> cd .\api\  
PS C:\Users\manel\OneDrive\Bureau\renduApi\api> nodemon .\app.js  
[nodemon] 3.0.1  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node .\app.js`  
Server started on port 3000  
Connected to MongoDB  
█
```

J'ai utilisé postman pour tester mon api et envoyer des requêtes. On peut visualiser que les requêtes get dans le navigateur.

Lorsque l'on s'inscrit :

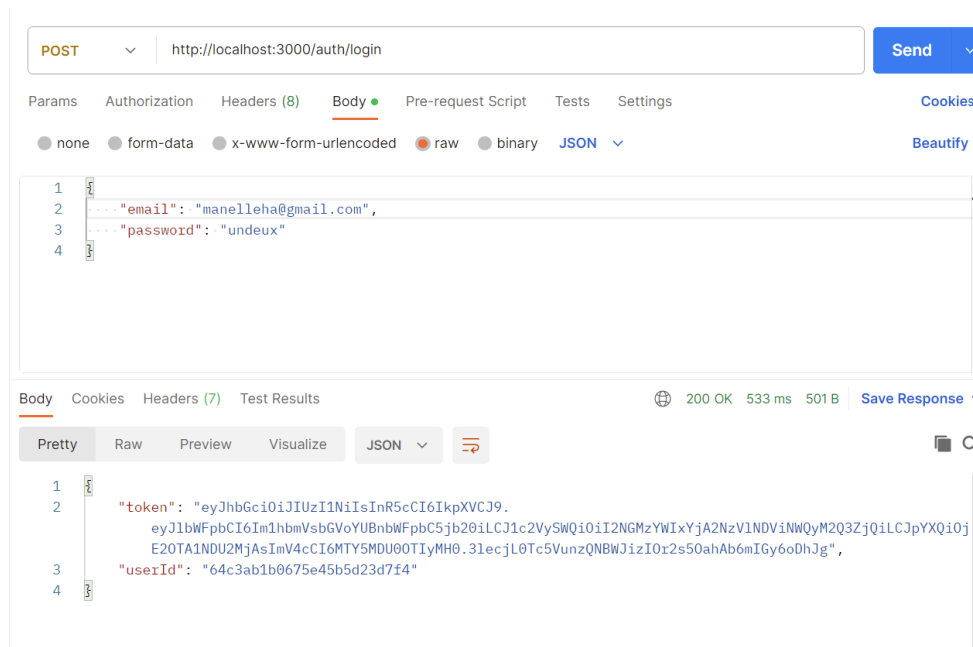


Lorsque l'on souhaite récupérer des évènements :



Au niveau de la sécurité :

Ici nous pouvons constater que le token est bien délivré lorsque l'on se connecte



Dans la base de données, le mot de passe est bien crypté :

+

ADD DATA

EXPORT DATA

1-1c

```
  _id: ObjectId('64c3ab1b0675e45b5d23d7f4')
  events: Array (empty)
  email: "manelleha@gmail.com"
  password: "$2a$12$EDQFhZ9fsMKcHgI6dcXkF0vC5dSmfJNu8x7jsJgykJERAQqBJe2UK"
  name: "manelle"
  __v: 0
```

Nous reconnaitrons l'utilisation d'opérations CRUD :

```
26   },
27   eventController.creerEvent
28 );
29
30 router.get('/event/:idEvent', isAuth, eventController.getEvent);
31
32 router.put(
33   '/event/:idEvent',
34   isAuth,
35   [
36     body('titre')
37       .trim()
38       .isLength({ min: 5 }),
39     body('description')
40       .trim()
41       .isLength({ min: 5 }),
42     body('date')
43       .isISO8601()
44   ]
45 );
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL node - api + - [] [] ...

Server started on port 3000

Grâce au module 'bcryptjs', le mot de passe inséré est crypté dans la base de données. Cela garantit une meilleure sécurité.

```
const { validationResult } = require('express-validator/check');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

const User = require('../models/user');

exports.signup = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    const error = new Error('Validation failed. ');
    error.statusCode = 422;
    error.data = errors.array();
    throw error;
  }
  const email = req.body.email;
  const name = req.body.name;
  const password = req.body.password;
  bcrypt
    .hash(password, 12)
    .then(hashPw => {
      const user = new User({
        email: email,
        password: hashPw,
```

SOURCES :

<https://developer.mozilla.org/fr/docs/Web/HTTP/Status>
<https://kinsta.com/fr/base-de-connaissances/qu-est-ce-que-node-js/>
<https://kinsta.com/fr/base-de-connaissances/qu-est-express-js/>
<https://www.ibm.com/fr-fr/topics/rest-apis>
<https://www.digitalocean.com/community/tutorials/workflow-nodemon-fr>
<https://welovedevs.com/fr/articles/debuter-avec-postman/>
<https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1445306-node-js-definition-simple-et-utilisation-pratique/>
<https://chat.openai.com/>
<https://talks.freelancerepublik.com/comprendre-utiliser-architecture-mvc/>

Autres sources :

- Les cours
- Aide d'un proche travaillant dans l'informatique