NodeJS API - A4 - Qualité et sécurité

01. Séparation entre serveur et App

Actuellement, notre application est bien séparé en différente couches. Cependant, le point d'entrée lui reste trop emmêlé. Il est temps de séparer le serveur de l'application en elle même.

Pour cela, créez un fichier `app.js`. Toute la logique d'instance de l'application, parsing des body ainsi que les routeurs/contrôleurs devront s'y trouver.

Votre fichier `server.js` lui sera toujours l'entrée, mais importera maintenant l'app. Cela sera utile pour gérer les tests

Si vous lancez votre code, rien ne devrait avoir changé.

02. Toujours avec son casque.

Sans protection

Partons en exploration. Exécutez une requêtes à votre API sur n'importe quelle route GET standard avec Postman ou cUrl. Regardez la partie Headers de la réponse.

```
x-powered-by : Express
content-type: application/json; charset=utf-8
content-length : 30
etag : W/"1e-V1HGUXWZj+5HhtdtXCbw2c6FoAE"
date : Tue, 03 Feb 2026 10:08:09 GMT
connection : close
```

Notez les headers qui commencent par `X-`. Ce sont des custom headers qui peuvent dévoiler des infos sensibles sur votre application. Ici `x-powered-by : Express` indique qu'on utilise Express ce qui peut aider un potentiel attaquant.

Comme le recommande l'OWASP, une fondation à but non lucratif qui œuvre pour améliorer la sécurité du Web, l'en-tête **X-Powered-By** devrait être supprimé. En effet, il ne faut jamais fournir aux attaquants des informations sur votre stack technologique. Sinon, ils pourraient exploiter ces informations pour tirer parti de vulnérabilités connues dans ce framework ou cette bibliothèque.

Nous pourrions analyser tous nos header grâce à Security Headers online project mais ce n'est pas le but (en plus, il faudrait rendre accessible sur le web votre API via des outils comme ngrok). Mais le réseau de l'IUT ne nous le permettrais pas.

Ajoutons un casque

Tout d'abord, vous devez ajouter **Helmet.js** aux dépendances de votre projet. Installez le package npm **helmet** avec la commande suivante :

```
npm i helmet
```

Votre fichier **package.json** contiendra désormais **helmet** dans le champ **dependencies**.

Configuration de Helmet dans Express

L'intégration de Helmet dans votre application Node.js avec Express est simple. En cas de problème, référez-vous au guide officiel.

Dans votre fichier `app.js` , importez Helmet avec la commande suivante :

```
const helmet = require("helmet")
```

Ensuite, enregistrez Helmet dans votre application Express comme ci-dessous avant vos routeurs :

```
app.use(helmet())
```

Gardez à l'esprit que `helmet()` n'est rien d'autre qu'un middleware Express. Plus précisément, la fonction principale `helmet()` est un wrapper autour de 15 sous-middlewares. Ainsi, en enregistrant `helmet()`, vous ajoutez 15 middlewares Express à votre application.

Notez que chaque middleware se charge de définir un en-tête de sécurité HTTP spécifique.

Retestez l'appel à votre route. Vos headers devraient être multiplié par 3. Si vous avez fait le test avec Security Headers, celui-ci sera beaucoup plus agréable à regarder.

On ne vas pas rentrer ici dans tous les détails des différents Headers mais je vous fourni un document annexe qui vous donnera plus d'infos (en anglais) Configuring security headers in Helmet

03. Générer une documentation

Installation

Comme toujours dans le monde du JS, encore un npm install !

```
npm i swagger-ui-express swagger-jsdoc
```

Rendez vous dans votre `app.js` pour ajouter l'import des dépendances.

```
const swaggerJsdoc = require("swagger-jsdoc");
const swaggerUi = require("swagger-ui-express");
```

Puis avant votre export de votre app, ajoutez la config suivante :

```
const options = {
  definition: {
    openapi: "3.1.0",
    info: {
      title: "MMI Express API with Swagger",
      version: "0.1.0",
      description:
        "TP de NodeJS. Apprentissage 05",
      license: {
        name: "MIT",
        url: "https://spdx.org/licenses/MIT.html",
      },
      contact: {
        name: "Author",
        url: "https://google.com",
        email: "info@email.com",
      },
    },
    servers: [
      {
        url: "http://localhost:3000",
```

```
      },
    ],
  },
  apis: ["./routes/*.js"],
};
const specs = swaggerJsdoc(options);
app.use(
  "/api-docs",
  swaggerUi.serve,
  swaggerUi.setup(specs, { explorer: true })
);
```

Vous pouvez bien sur adapter certains paramètres. Ensuite lancez votre application et rdv sur `http://localhost:3000/api-docs`

Vous devriez avoir une doc vide avec les infos fournie dans la config. Il est temps d'afficher des infos o/

Allons dans notre fichier `/routes/hello.js`. Si vous n'avez pas changé l'options apis dans la config, c'est ici que swagger va aller chercher les infos.

Première étate, nous allons définir notre modèle. Pour faire simple, nous n'allons que documenter le modèle `Person` et la route `/hello/person`. Ajoutez ce commentaire et actualisez la page de la doc

```
/**
 * @swagger
 * components:
 *   schemas:
 *     Person:
 *       type: object
 *       required:
 *         - age
 *         - name
 *       properties:
 *         name:
 *           type: string
 *           description: Person's name
 *         age:
 *           type: int
 *           description: Person's Age
```

```
 *       example:
 *         name: Reno Jackson
 *         age: 65
 */
```

Une fois analysé, vous pouvez actualiser votre doc. L'exemple devrait apparaitre. En cas d'erreur, il est possible que l'indentation soit mal faite. Nous écrivons actuellement du YAML dans des commentaires donc l'indentation est importante. Mais il nous manque encore la route ! Il faut donc ajouter un deuxième commentaire.

```
/**
 * @swagger
 * tags:
 *   name: Person
 *   description: Person's actions
 * /hello/person:
 *   post:
 *     summary: Return a Person's body
 *     tags: [Person]
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             $ref: '#/components/schemas/Person'
 *     responses:
 *       200:
 *         description: The Person.
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/Person'
 *       500:
 *         description: Some server error
 *
 */
```

Avec tout cela, n'hésitez pas à regarder la doc officielle (Open API et Swagger). Vous devriez pouvoir tester votre route depuis la doc comme sur Postman. Pour être plus confiant sur l'écriture de votre YAML, vous pouvez utiliser un outil tel que Swagger Editor Vous pouvez voir en live le résultat puis l'importer dans votre projet.

Attention, le code actuel est petit donc votre doc ne prends que peu de place. Cela ne sera pas le cas pour les gros projet et il faudra peut être revoir l'architecture de la doc ;).

04. Et après ?

Plein de sujets n'ont pas le temps d'être abordé mais si le dev back end NodeJS vous intéresse voici d'autres sujets de qualité/sécurité :

- Journalisation avec Sentry ou Winston
- Error Handling général
- Typage avec Zod ou même passage à TypeScript au lieu de JS