Code Review NodeJs Express

Voici le compte rendu de la revue de code que j'ai fais sur le projet node.js.

Sommaire

database-connection.js 3 remarques

helpers/chantier.js 7 remarques

routes/chantier.js 4 remarques

helpers/user.js 4 remarques

token-utils.js 5 remarques

random-utils.js 2 remarques

index.js 6 remarques

db.sql 10 remarques

générales 6 remarques

database-connection.js

- Les identifiants de la base de données dont le mot de passe sont en clair dans le code ce qui n'est pas une bonne pratique. En effet, le mot de passe devrait se trouver dans le fichier .env pour qu'il ne soit pas push sur GitHub.
- Dans la fonction query les paramètres passés ne sont pas vérifiés, ainsi un acteur malveillant pourrait injecter du code pour pouvoir récupérer des données sensibles de la base de données (surtout que dans le cas de ce projet le mot de passe n'est pas hasher dans la base de données).
- Je pense qu'exporter la constante `connection` n'est pas une bonne pratique. En effet, je pense qu'il serait préférable de seulement exporter `authenticate` et `query` puisque `connection` n'est jamais réutilisé autre part dans le projet. De plus, je pense qu'il est préférable qu'une seule fonction avec des vérifications et des gestions d'erreur puisse établir une connexion à la base de données.

helpers/chantier.js

- Dans `findChantier` on pourrait modifier la requête SQL par `SELECT * FROM chantier where numero = ?` avec en paramètre de la fonction le numéro du chantier que l'on souhaite rechercher. Il n'y aura jamais deux chantier avec le même numéro puisque c'est la clef primaire de la table chantier et que la valeur est auto-incrémenté à chaque nouvelle ligne insérée. Le résultat sera donc toujours mono ligne. De plus, utiliser cette méthode est plus optimisé, car elle permet de ne pas récupérer toute la

table à chaque requête pour seulement après effectuer le traitement dessus. (Bonus : on pourrait remplacer l' * par les champs qui nous intéressent vraiment pour être encore plus optimisé.).

- Il semble que dans `findChantier` la gestion d'erreur ai été oublié. On devrait rajouter `throw new PersonnalError('Error when querying', e);`.
- Toujours dans `findChantier`, il y a un paramètre inutile que l'on peut supprimer
- Dans le `updateChantier` la concaténation d'un chaîne de caractères comme dans la fonction pourrait mener à une injection sql. En effet, il serait préférable d'utiliser la syntaxe avec les ? qui permet une vérification des valeurs ?
- Le nom de la fonction `putRandomChantier` n'est pas très conventionnel puisque cette fonction sert à insérer des éléments dans la table. Pour une méthode de peuplement dans ce genre, on devrait utiliser le nom `createRandomChantier`.
- Pour la fonction `putRandomChantier` il serait préférable de passer les valeurs en paramètre au lieu de devoir à chaque fois modifier les valeurs dans le fichier à la main.
- Les fonctions de ce fichier ne vérifient pas si l'utilisateur qui exécute ces requêtes possède les droits pour effectuer ces actions. Pour cela, on aurait pu utiliser les méthodes créées dans le fichier `token-utils.js` pour vérifier si l'utilisateur est bien connecté et si son token jwt est encore valable.

routes/chantier.js

- Le contenu saisit par l'utilisateur n'est jamais vérifié pour voir s'il correspond au format attendu, cela pourrait permettre me prévenir d'attaques par injection SQL ou bien tout simplement de vérifier l'intégrité des données.
- La gestion des codes de retour renvoyé en cas d'erreur n'est pas fait correctement. En effet, il n'es pas très utile de retourner un code d'erreur 500 puisqu'il n'indique presque rien au programmeur utilisant l'API. Si les codes retournés faisaient référence à une erreur précise le développeur serait plus apte à résoudre les problèmes. C'est aussi un gain de temps précieux lorsqu'on utilise une api.
- Il serait préférable d'utiliser un **PUT** à la place du **POST** pour la mise à jour d'un chantier puisque POST est conventionnellement utilisé pour ajouter une des données dans une base de données.
- Idem pour l'endpoint `/delete/:numero` il serait plus logique de faire un delete à la place d'un get puisque la fonction est de supprimer un chantier.

helpers/user.js

- Le mot de passe n'est pas hashé avant d'être inséré dans la base de données ce qui n'est vraiment pas sécurisé puisque le mot de passe est en clair. Il faudrait par exemple utiliser un algorithme de chiffrement tel que SHA256 avant de stocker le mot de passe.
- La gestion d'erreur n'est pas présente, il est donc plus difficile de débugger le code.
- On pourrait utiliser un égal au lieu du `like` puisque l'on connais le contenu des chaînes que l'on va comparer. Cela permettrait de minimiser les risques de résultats inattendus.

- Je pense que le Promise de `findIncludeUser` et de `createUser` n'est pas nécessaire dans ce cas puisque la fonction query retourne déjà une promesse.

token-utils.js

- Le temps d'expiration du token est beaucoup trop long, il serait préférable de faire une méthode `refreshToken` pour permettre de fournir un nouveau token lorsque l'ancien est expiré. Le problème ici est que si le token est volé alors il pourra être utilisé pour une longue durée.
- Pas de gestion d'erreur donc toujours difficile à débugger.
- Dans le cas de `createToken` il serait préférable de créer une copie de l'objet user pour après pouvoir effectuer des modifications et non pas sur l'objet passé en paramètre et qui est retourné en fin de fonction
- Certaines méthodes comme `connectedUser`, `extractToken` et `verifyAndExtractToken` ne sont jamais appelés dans le projet. Ces fonctions sont donc inutiles et elles complexifient le code.
- L'algorithme de chiffrement HS256 est basé seulement sur une clef secrète, si celle-ci venait à leak ou est volé alors la sécurité est mise en jeu. On pourrait se baser sur un algo de chiffrement avec un système de clé publique/privés.

random-utils.js

- Les noms de variable ne sont pas très explicites, ce qui ne facilite pas la compréhension du code.
- On pourrait utiliser une autre façon pour générer une chaîne de caractère d'une longueur donnée.

index.js

- Les noms de variable ne sont pas très explicites, ce qui ne facilite pas la compréhension du code.
- Il y a cette chaîne de caractère qui est réutilisé à plusieurs endroits dans le code. L'utilisation d'une librairie est possible ou bien stockée une seule fois dans un fichier à part cette chaîne. Tous les fichiers souhaitant utiliser cette chaîne n'auraient qu'à l'importer.
- L'endpoint `/session/signup` n'est pas en `async` ce qui peut être bloquant.
- La variable c dans `/session/signup` est inutile puisqu'elle n'est jamais transmise à aucune fonction (elle est transmise, mais la fonction `createUser` n'utilise pas cette variable). Le code est donc complexifié et cela n'apporte rien.
- On pourrait simplifier un maximum les endpoints pour qu'il n'y ai que des appels à des fonctions ce qui permettrai de simplifier et de rendre le code plus lisible pour un programmeur arrivant sur le projet.
- Les codes de retour ne sont pas explicites (seulement des 500). Il faudrait faire une gestion d'erreur avec différents codes de retour en fonction des erreurs retournés. L'erreur 500 indique seulement une erreur côté serveur, mais n'apporte aucune précision sur la nature de celle-ci. Alors qu'il existe différents codes de retour pour différentes erreurs.

db.sql

- La table user et chantier utilise le type `CHAR` alors qu'il serait préférable d'utiliser le type `VARCHAR` ou `CHAR` prend plus de place en mémoire puisque c'est une taille fixe qui est allouée pour chaque valeur. L'utilisation de varchar est préférable puisqu'elle alloue la bonne taille à la valeur qu'on insert dans la table.
- Si on laisse les `CHAR` alors il faudrait augmenter la taille pour le champ `email` car 24 caractères ne suffisent pas. La taille maximale d'une adresse est de **320** il faudrait donc que ce champ puisse stocker une adresse mail de cette taille.
- Le champ `email_verification` est inutile puisque normalement, ce genre de vérification doit être fait en amont de l'insertion de données dans la bdd.
- Pour le champ `password`:
- La taille est bien trop courte puisqu'aujourd'hui, la CNIL recommande un mot de passe de minimum 12 caractères.
- Le mot de passe n'est pas hashé et il est stocké en clair dans la base de données. Il faudrait donc augmenter la taille disponible à par exemple 64 caractère pour pouvoir stocker un hash en SHA256.
- La table `user` ne contient pas de clef primaire. On pourrait ajouter une colonne id en clef primaire avec une valeur qui s'incrémenterai toute seule.
- Il est difficile d'identifier facilement un utilisateur puisqu'aucune colonne n'est unique. Deux utilisateurs peuvent avoir la même adresse email, car cette condition n'est vérifiée à aucun endroit dans le code.
- Dans la table `chantier` il n'y a pas de vérification pour antériorité de la date de début par rapport à la date de fin. On pourrait rajouter la contrainte `CHECK(date_debut<date_fin)`. On pourrait rajouter cette contrainte dans la table user pour vérifier la similarité des emails.
- Il n'y a aucune relation entre les deux tables il est donc difficile par exemple de savoir quels utilisateur est sur quel chantier. On pourrait mettre en contrainte des foreign key.
- On pourrait passer le champ `description` en `TEXT` pour plus de cohérence.
- On pourrait passer le champ `city_cp` en `UNSIGNED INT` puisque cette valeur ne peux pas être négative.

générales

- Je pense que l'utilisation d'un ORM pour la création de la base de données aurait été préférable. En effet, ce genre de structure permet une plus grande flexibilité, par exemple pour changer rapidement de base de données à la place de réécrire toutes les requêtes, il suffit juste de spécifier quel type de bdd on souhaite pour que la base se régénère toute seule. Il est aussi plus facile de rajouter des colonnes dans une table avec un ORM puisqu'il suffit de faire une migration et cela évite plein de problèmes.
- Il n'y a pas de requêtes préparées dans le projet. Comme certaines requêtes vont être exécuté de nombreuses fois, il serait préférable de faire des requêtes préparées. L'api sera plus performant et le temps de réponse à la base de données sera réduit.
- La connexion à la base de données n'est jamais fermée à la fin des requêtes pour libérer les ressources (même si dans le cas d'une petite API comme celle-ci ce genre de point peut être négligés).
- Il n'y a pas de test pour les endpoints ou bien même pour les fonctions dans le code.
- La documentation est très limité et présente seulement à certains endroits. On pourrai présenter le panel des codes de retour possible retournés par l'API par exemple.
- Il ne faut pas push le fichier .env sur GitHub puisque ce ficher est justement censé rester en local sur la machine de l'utilisateur (surtout ici avec la clé pour déchiffrer les tokens JWT)