DOCUMENTATION TECHNIQUE

BILL–CUTTING BACKEND

3PROJ

RYAN DORDAIN / SERAPHIN DUBAIL / TRISTAN TOURBIER / THOMAS LAPERE

Table des matières

1- Choix des technologies : 2
1.1- Avantage des technologies utilisé : 2
2. Au démarrage 3
3. Fonctionnement global du backend de Bill Cutting 3
4. L'API 4
4.1- Architecture RESTful 4
4.2- Organisation du code 4
5. Bases de données 5
6. La sécurité 7
6.1- Mots de passe utilisateur 7
6.2- Les tokens JWT 8
6.3- Le chiffrements AES 8
7. L'Oauth 9

1- Choix des technologies :

Pour le Back-End de l'application Bill Cutting nous avons opté pour le Framework Express.js. Nous utilisons aussi deux bases de données : PostgreSQL via l'ORM[1] Sequelize et MongoDB via l'ODM[2] Mongoose. Nous utilisons aussi Minio pour le stockage des fichiers.

1.1- Avantage des technologies utilisé :

Node.js est un environnement d'exécution de javascript côté serveur. C'est un environnement asynchrone ce qui permet de gérer plusieurs clients en même temp. Il est aussi très performant en étant économe en ressource et mis à jour régulièrement. Il supporte aussi très bien les websockets ce qui permet de pouvoir intégrer des messagerie en temp réel facilement.

Express.js possède plusieurs avantages qui nous ont convaincu de l'utiliser. Tout d'abord c'est un Framework node.js utilisant donc javascript. Il est minimaliste, rapide et simple a prendre en main. C'est une technologie qui a déjà fait ses preuves et c'est l'un des Framework backend node.js les plus populaire. Il est régulièrement mis a jour ce qui limite les failles de sécurité. En plus de cela la documentation est très fournie et la communauté qui l'utilise est très présente sur les forums ce qui permet d'avoir beaucoup de cas pratique et de résolutions d'erreur.

PostgreSQL est un système de gestion de base de données relationnelle (SGBDR) open source. Qui réponds aux normes SQL. Il est très utiliser du fait de ses performances élevé ainsi que de sa scalabilité (c'est-à-dire qu'il peut gérer une augmentation d'utilisateur ou une augmentation de charge de travail). Postgres est utile pour stocker des éléments dépendant d'autre élément.

MongoDB est un système de gestion de base de données NoSQL orienté document. Il permet de stocker des structures de données clé/valeur, semblables à des documents JSON, dans des collections. Ces documents n'ont pas de schéma fixe et peuvent avoir des structures différentes au sein d'une même collection. Mongo est utile pour stocker des informations indépendantes qui ne seront que rarement modifié.

---

[1] ORM (Object-Relational Mapping) : outil qui permet à un programme informatique d'utiliser des objets pour interagir avec une base de données relationnelle.
[2] ODM (Object-Document Mapping) : outil permettant à un programme informatique d'utiliser des objets pour interagir avec une base de données orientée documents.

Minio est système de stockage d'objet compatible avec les bucket S3 d'Amazon. Il est très utilisé du fait de ses performance élevé et de sa très grande scalabilité. On s'en sert surtout pour stocker des objets non structurés, comme du texte, des images, des vidéos ou des pdfs et autres fichiers.

2. Au démarrage

Afin de pouvoir exécuter le Backend vous aurez besoin de crée un fichier .env contenant tous les mot de passe, clef de chiffrements, URL d'application et autre variable. Ceci nous permet d'augmenter la sécurité de l'application en faisant en sorte que les informations privées de production restent privées et indépendantes de ceux de pre-production ou de développement.
Ce fonctionnement nous permet aussi de modifier simplement les valeurs de ces variables utiliser a de nombreux endroit dans le code.
Cela peut être utile par exemple si nous changeons le nom de l'application ou les mots de passe des services.
Pour conclure les fichiers .env augmente la maintenabilité et la sécurité de l'application.

Vous devez crée ce fichier a l'emplacement : /BackEnd/Express/.env en vous basant sur le fichier .env_model situé ici : /BackEnd/Express/.env_model .

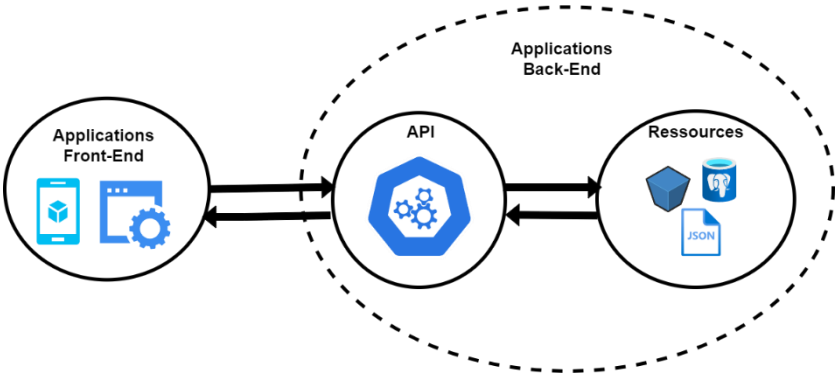
3. Fonctionnement global du backend de Bill Cutting

Le backend de Bill Cutting est divisé en deux parties :

- Une API en Express.js
- Des ressources stockées dans des bases de données et via Minio

Pour établir un lien entre le frontend utilisé par le client et ses informations, nous avons créé une API. Cette API agit comme un intermédiaire entre les ressources et les différentes applications frontend. Grâce à cela, nous pouvons gérer qui peut voir certaines informations et accéder à certaines données.
Ainsi, les bases de données et Minio ne sont pas directement accessibles depuis internet. Il est nécessaire de passer par l'API, ce qui augmente la sécurité et limite les risques de fuites de données.
Le fait d'utiliser une API permet de normaliser les accès aux données via des URL.

Grace a cela nous pouvons avoir plusieurs Frontend sans devoir reprogrammer à chaque fois la façons dont ces applications accèdes aux informations de Bill Cutting.



4. L'API

4.1- Architecture RESTful

Notre API est dite RESTful (Representational State Transfer) car elle utilise les caractéristiques clef de ce style d'architecture logicielle :

- Utilisation des méthodes HTTP comme GET/POST/PUT/DELETE
- Indépendance des requêtes, chaque requête doit contenir toutes les informations nécessaires, le serveur ne garde pas d'information sur l'état du client entre les requêtes
- Les différentes fonctionnalités sont accessibles via des URL uniques, chaque endpoint gère une fonctionnalité et une seule.
- Les données échanger entre le client et le serveur sont standardisé, ici nous utilisons le format JSON

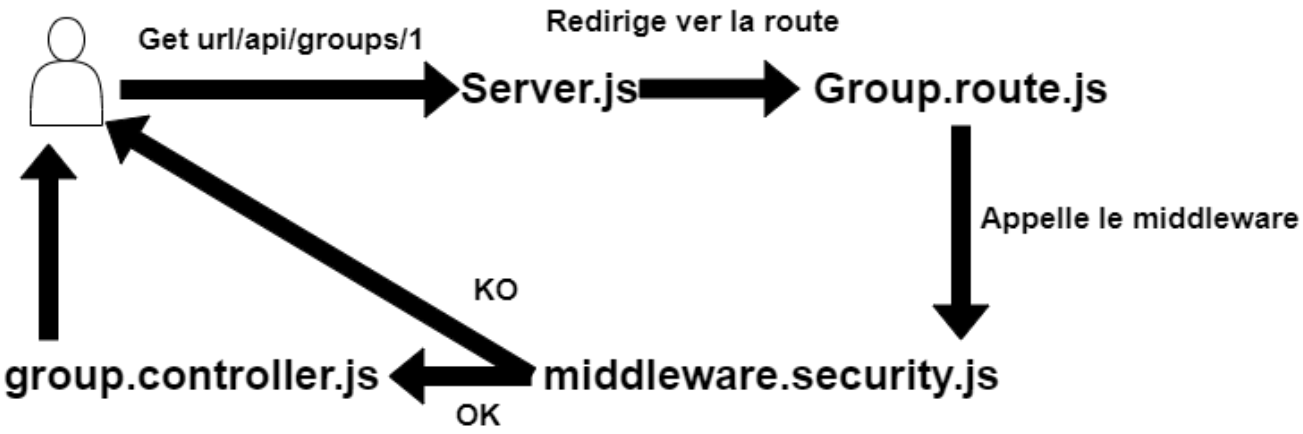
4.2- Organisation du code

Nous avons séparé notre code Express.js en différents fichier et nous les avons organisés dans différents dossiers afin de pouvoir avoir un code source propre et maintenable facilement. Voici les principaux éléments de cette organisation :

- Un dossier Configurations, qui va contenir toutes les configurations de connexion aux bases de données et a minio.
- Un dossier Models qui va contenir tous nos objets utiliser par les ORM/ORD afin d'initialisé et interagir facilement avec les bases de données.
- Un dossier Routes qui va contenir les différents fichiers contenant les Endpoint de l'api. Cela va permettre de diriger les requêtes utilisateurs ver leur middleware et fonctionnalité associé
- Un dossier Middlewares. Les middlewares sont des fonctions qui se mettent avant la logique métier et qui vont permettre par exemple d'autoriser l'accès a tel utilisateur ou alors redimensionner une image.
- Un dossier Controllers qui contient tous les contrôleurs et donc la fonctionnalité associé a cet endpoint, cela peut être crée un utilisateur, crée une depense ou envoyer un ticket de caisse dans le Bucket défini pour etc...

- Le fichier server.js. C'est là que va être initialisé express, que les routes vont être défini et associer à leur fichier, pareil pour les modeles etc, c'est ce qui va relier tous les morceaux de code entre eux.

Voici donc le chemin d'une requête pour accéder aux informations d'un groupe :



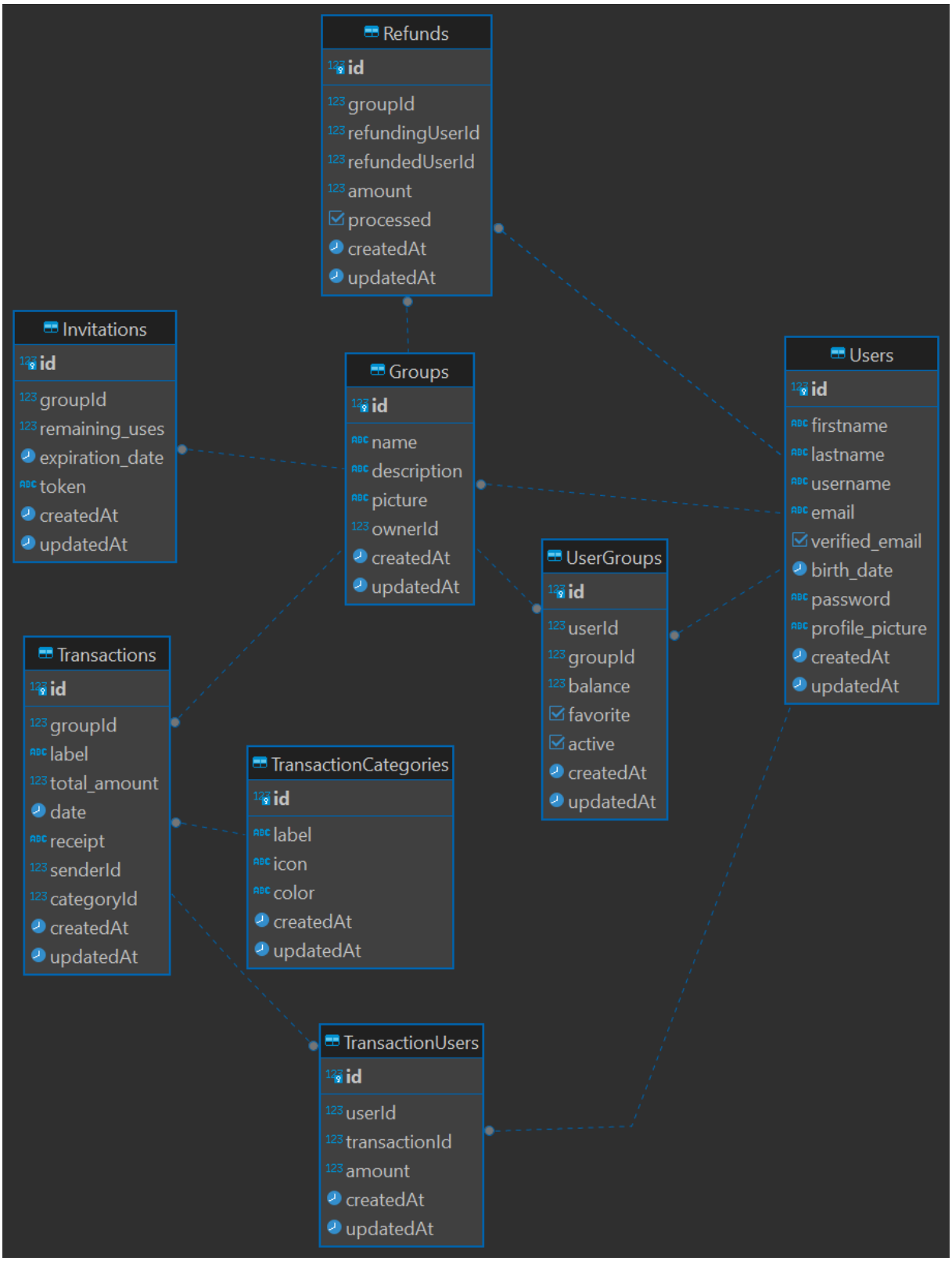
L'utilisateur envoie une requête GET a l'url /api/groups/1.
Elle va arriver au fichier server.js, être envoyer au bon fichier route qui va l'envoyer au middleware qui vérifie si l'utilisateur est connecté.
Si il est connecté la requête est envoyer au contrôleur associé, sinon un code d'erreur est envoyer au client directement.
Une fois la requête arrivé au contrôleur il va s'occuper d'aller chercher les informations et les envoyer au client en JSON

5. Bases de données

Nous avons deux bases de données, une SQL (PostgreSQL) et une NO-SQL (MongoDB). Ce choix était important pour nous car nous voulions utiliser les méthodes de stockage les plus adapté à chaque situation.

Les bases de données SQL sont très utiles et adapter pour stocker des données structurées avec un schéma définis à l'avance et possédant de forte relation avec d'autre élément de la base de données. Elle nous permet d'effectuer des requêtes complexe afin de retourner des informations complète facilement.
Nous stockons par exemple dedans les utilisateurs, leurs groupes et leur transaction. Ainsi en une seule requête nous sommes capables d'avoir toutes les transactions de l'utilisateur X passée dans le groupe Y.

Voici le schémas la base de données SQL:



Les bases de données NO-SQL quand a elles peuvent stocker des données semi-structuré ou non structuré. Elles peuvent aussi traiter un grand nombre de donnée sans sacrifié de performance. Nous l'avons donc utilisé pour sauvegarder les messages, les notifications ainsi que les méthodes de payement des utilisateurs.

Les messages et les notifications vont être très nombreuses et ont besoin d'être accessible presque instantanément.
Les méthodes de payement n'ont pas de schéma de données fixe, en effet un RIB n'as pas le même schéma qu'un nom d'utilisateur Paypal.

C'est donc pour ces spécificités que nous avons utilisé de manière complémentaire des bases de données SQL et NO-SQL.

6. La sécurité

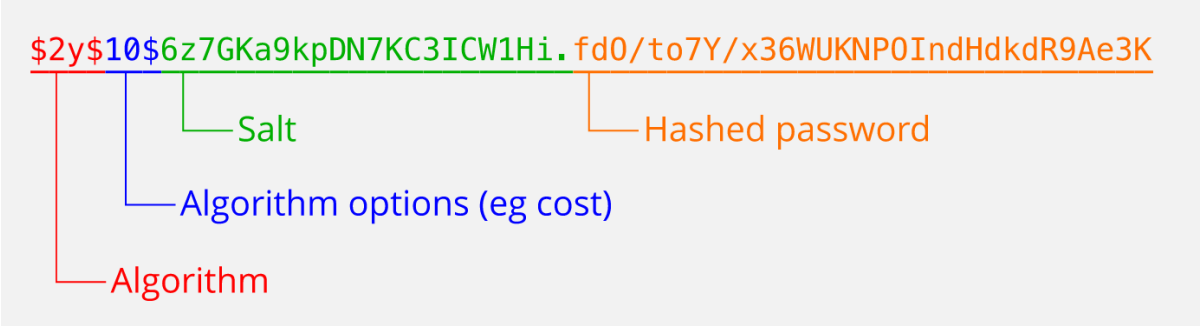
6.1- Mots de passe utilisateur

Les mots de passe utilisateur sont l'une des informations les plus sensibles que nous stockons. Nous n'autorisont que des mots de passe constitué d'au moins 8 caractère, contenant des minuscules, des majuscules, des chiffres et deux caractère spéciaux.

Afin de pouvoir les gérer de manière sécurisée et en respectant la RGPD nous les avons hasher [3] avec Bcrypt.

Bcrypt est une fonction de hashage basé sur l'algorithme blowfish. Il intègre un facteur de coup, c'est-à-dire le nombre de fois que blowfish est appliqué au mot de passe, dans notre application nous l'avons définie à 12.

La chaine de caractère qui résulte de Bcrypt contient le hash du mot de passe et un sel intégré. Le sel permet de ne pas donner le même résultat à chaque hashage. Cela rend Bcrypt insensible aux attaques de type rainbow tables[4]



Quand un utilisateur voudra se connecter nous allons hasher le mot de passe qu'il nous a fournis et le comparer au hash que nous avons stocké, si la comparaison est bonne alors il a fourni le bon mot de passe et peut se connecter, sinon on lui refuse l'accès.

---

[3] Hasher : générer un « hash ». Un hash est une chaîne de caractères de longueur fixe créée via une fonction mathématique non réversible. On ne peut donc pas retrouver la valeur à la base d'un hash.
[4] Attaque par rainbow table : une méthode de piratage qui utilise des tables pré-calculées de hashes et leurs chaînes de caractères d'origine pour inverser les fonctions de hachage et retrouver les mots de passe de bases.

6.2- Les tokens JWT

Afin qu'un utilisateur n'ai pas a donner son mot de passe des qu'il souhaite accéder a une nouvelle page nous utilisons des token JWT. Ces json web token contiennent trois parties :

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI1IiwiaWF0IjoxNzE2MDc3NDAxLCJleHAiOjE3MTYxNjM4MDF9.CBUHSSk-xXdcd8Lqf6xR0-PNmS12LnTBIGGWZLlpZWQ

La partie rouge est le header, il contient les informations sur le type de token et l'algorithme de signature.
La partie verte est la charge utile, c'est les informations que l'ont souhaite transférer.
La partie orange est la signature qui permet de vérifier que le token viens bien du serveur et n'as pas était modifier.

Quand un utilisateur se connecte il reçoit un token valable pendant une période déterminer, par exemple 24h. Ce token servira à identifier l'utilisateur. Quand il voudra accéder à un endpoint il enverra son token qui sera vérifier par le serveur afin de s'assurer que l'utilisateur est bien celui qu'il prétend être et qu'il a accès à la ressource demandé. Le token fait donc acte de certificat d'identification pendant la durée de sa validité.

Dans notre application nous ne stockons que trois choses dedans, l'identifiant de l'utilisateur, sont timestamp de création et son timestamp d'expiration.

6.3- Le chiffrements AES

Les informations bancaires comme les noms d'utilisateurs paypal et les RIB sont aussi des données sensibles. Nous ne pouvons cependant pas les hasher car nous avons besoin d'y accéder après les avoirs stocker.
Nous avons donc utilisé un algorithme de chiffrement très connus et très utiliser : l'AES. C'est une méthode de chiffrement symétrique qui utilise donc la même clef pour chiffrer et déchiffrer. Actuellement aucun moyen autre que le brute force ne permet de casser AES. Afin de renforcer cet algorithme nous créons un IV [5] différent à chaque fois et nous le stockons avec la valeur chiffrée.

[5] IV (Initialization Vector) : Paramètre utilisé dans certains algorithmes de chiffrement, semblable a un sel

7. L'Oauth

Pour faciliter le processus d'inscription aux nouveaux utilisateurs nous proposons un système d'Oauth via google. Ce système va permettre aux utilisateurs de se connecter via leur compte google.

Si l'utilisateur accepte la connexion, Google va nous envoyer quelque information privée comme le nom, prenom, email et nom d'utilisateurs. Nous allons par la suite crée un compte dans notre application avec ces informations.

Pareil pour la connexion ou nous allons identifier l'utilisateur via son email.

Cela augmente donc la fluidité et améliore l'expérience utilisateur.