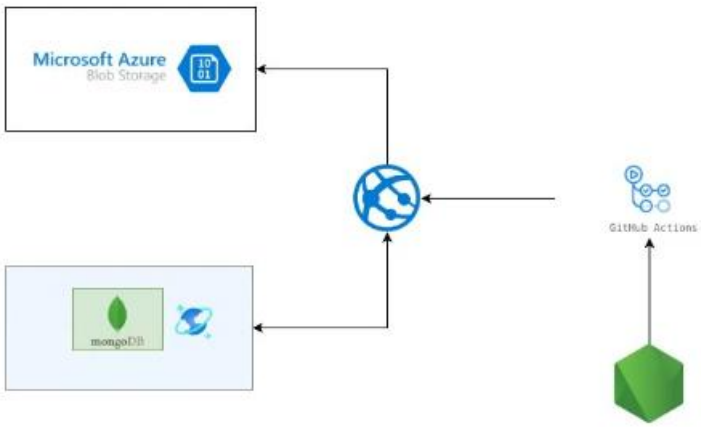
Documentation

Dans le cadre du cours Developing Solutions for Microsoft Azure nous avons réalisé un projet en nous servant des services Azure. Nous avons réalisé une plateforme de médias sociaux à l'aide d'Azure qui permet la création et la gestion du contenu via des API. Notre plateforme se concentre sur la partie backend pour les réseaux sociaux.

Architecture de la plateforme

Entre les différents choix qui s'offraient à nous pour la construction de l'architecture de notre plateforme, nous avons opté pour Node.js pour sa facilité d'utilisation dans le développement mais aussi la simplification du flux de travail. Node.js nous permet également de gérer les événements asynchrone lors des nombreuses connexions sur la plateforme.

L'architecture de notre application repose donc sur Node.js. Nous nous servons de GitHub Actions qui utilisent un fichier pour effectuer le déploiement vers l'APP Service qui est service Microsoft Azure. Une connexion vers la base de données MongoDB est utilisé à des fins du projets. Celle-ci communique avec l'APP Service qui se sert de l'espace de stockage Microsoft.



Les différentes fonctions de la plateforme

La plateforme regorge différentes fonctionnalités dont : l'authentification de l'utilisateur, la gestion du contenu grâce à l'ajout et la suppression de contenu (texte), l'ajout et la suppression des commentaires, l'ajout des médias (des images, des vidéos, des audios, et autres types de fichiers).

L'authentification

Tout utilisateur a la possibilité de consulter le contenu disponible sur la plateforme. Seuls les utilisateurs inscrits ont la possibilité de rajouter du contenu sur la plateforme. Grâce au système d'authentification les utilisateurs peuvent s'inscrire.

Un système de hiérarchisation des utilisateurs a été mis en place pour accorder des différents droits. Il s'agit ici des différents rôles attribués aux utilisateurs leur accordant le droit de supprimer un autre utilisateur, des contenus, des médias.

La gestion des contenus

Une fois connecté, l'utilisateur a la possibilité de rajouter du contenu de type texte, audio, vidéo, photo et autres types de formats. L'utilisateur a également la possibilité de modifier et de supprimer le contenu publié sur la plateforme.

Documentation technique

Le code JavaScript représente un ensemble de fonctions permettant de gérer les utilisateurs dans une application, notamment la récupération de tous les utilisateurs, la mise à jour des informations d'un utilisateur et la suppression d'un utilisateur. Les opérations sont sécurisées à l'aide de JSON Web Tokens (JWT).

Dépendances

Le code utilise les modules Node.js suivants :

- **require('../models/user')**: Importe le modèle utilisateur.
- **require('dotenv').config()**: Charge les variables d'environnement à partir du fichier **.env**.
- **const jwt = require('jsonwebtoken')**: Utilise la bibliothèque JWT pour gérer les tokens.

Fonctions de la classe user

1. getAllUsers

- **Description**: Récupère tous les utilisateurs de la base de données.
- **Méthode HTTP**: GET
- **Endpoint**: **/users**
- **Réponse réussie**: Retourne un tableau d'utilisateurs.
- **Réponse d'erreur**: Retourne un statut 500 en cas d'échec.

2. updateUser

- **Description**: Met à jour les informations d'un utilisateur spécifié.
- **Méthode HTTP**: PUT
- **Endpoint**: **/users/update**
- **Paramètres de requête**: Les mises à jour des informations de l'utilisateur.
- **Réponse réussie**: Retourne un message de succès et les informations mises à jour de l'utilisateur.
- **Réponse d'erreur**: Retourne un statut 401 si le token JWT est manquant ou malformé, un statut 404 si l'utilisateur n'est pas trouvé, et un statut 500 en cas d'autres erreurs.

3. deleteUser

- **Description**: Supprime un utilisateur spécifié.
- **Méthode HTTP**: DELETE
- **Endpoint**: **/users/delete**
- **Paramètres de requête**: Les informations de l'utilisateur à supprimer (username ou email).
- **Réponse réussie**: Retourne un message de succès et les informations de l'utilisateur supprimé.

- **Réponse d'erreur**: Retourne un statut 401 si le token JWT est manquant ou malformé, un statut 403 si l'utilisateur n'est pas un administrateur, un statut 404 si l'utilisateur n'est pas trouvé, et un statut 500 en cas d'autres erreurs.

Sécurité

Les fonctions **updateUser** et **deleteUser** utilisent des tokens JWT pour s'assurer que seuls les utilisateurs authentifiés peuvent effectuer ces actions. De plus, la fonction **deleteUser** nécessite que l'utilisateur soit un administrateur pour supprimer d'autres utilisateurs.

Variables d'Environnement

- **process.env.secret**: Clé secrète utilisée pour signer et vérifier les tokens JWT.

Notes

- Le code utilise des structures **try-catch** pour gérer les erreurs et renvoyer des réponses appropriées en cas d'échec des opérations.
- Les erreurs sont également affichées dans la console pour faciliter le débogage.
- La gestion des erreurs est détaillée dans les réponses JSON renvoyées.

Fonctions de la classe authentication

Fonction register

La fonction **register** permet l'inscription d'un nouvel utilisateur dans le système.

- **Paramètres:**
    - **req** (Object): Requête HTTP contenant les données de l'utilisateur à enregistrer.
    - **res** (Object): Réponse HTTP renvoyée au client.
- **Fonctionnement:**

1. Vérifie si l'e-mail fourni est déjà associé à un utilisateur existant.
2. Si un utilisateur existe déjà, renvoie une réponse d'erreur avec le statut 400.
3. Hache le mot de passe fourni avant de l'enregistrer dans la base de données.
4. Crée un nouvel objet utilisateur avec le nom d'utilisateur, l'e-mail et le mot de passe haché.
5. Enregistre le nouvel utilisateur dans la base de données.
6. Renvoie une réponse JSON avec un message de succès et les détails de l'utilisateur enregistré.

- **Retour:**
    - Object JSON avec les propriétés suivantes :
        - **message** (String): Message de succès ou d'erreur.
        - **user** (Object): Détails de l'utilisateur enregistré.

Fonction login

La fonction **login** permet à un utilisateur existant de se connecter au système.

- **Paramètres:**
    - **req** (Object): Requête HTTP contenant les informations de connexion de l'utilisateur.
    - **res** (Object): Réponse HTTP renvoyée au client.
- **Fonctionnement:**

1. Recherche l'utilisateur dans la base de données en utilisant l'e-mail fourni.
2. Si aucun utilisateur n'est trouvé, renvoie une réponse d'erreur avec le statut 400.
3. Compare le mot de passe fourni avec le mot de passe enregistré dans la base de données.
4. Si les mots de passe ne correspondent pas, renvoie une réponse d'erreur avec le statut 401.
5. Génère un token d'authentification avec l'ID de l'utilisateur.
6. Ajoute le token d'authentification dans l'en-tête de la réponse.
7. Renvoie une réponse JSON avec un message de succès.

- **Retour:**
    - Object JSON avec les propriétés suivantes :
        - **message** (String): Message de succès ou d'erreur.
        - **Authorization** (String): Token d'authentification dans l'en-tête de la réponse.

Remarques

- Ces fonctions utilisent les modules externes **bcrypt** pour le hachage des mots de passe et **jsonwebtoken** pour la gestion des tokens.
- Les erreurs sont gérées de manière à renvoyer des réponses appropriées avec des codes d'erreur HTTP.

Fonctions de la classe content

Fonction createContent

La fonction **createContent** permet la création d'un nouveau contenu dans le système.

- **Paramètres:**
    - **req** (Object): Requête HTTP contenant les données du contenu à créer.
    - **res** (Object): Réponse HTTP renvoyée au client.
- **Fonctionnement:**

1. Extraction du token d'authentification de l'en-tête **Authorization**.
2. Vérification de la présence et du format correct du token dans l'en-tête **Authorization**.
3. Extraction du token en supprimant le préfixe "Bearer ".

4. Validation et extraction de l'ID de l'utilisateur à partir du token en utilisant **jsonwebtoken**.
5. Création d'un nouvel objet **Content** avec le titre, le texte et l'ID de l'utilisateur.
6. Sauvegarde du contenu dans la base de données.
7. Renvoie une réponse JSON avec un message de succès et les détails du contenu créé.

- **Retour:**
    - Object JSON avec les propriétés suivantes :
        - **message** (String): Message de succès ou d'erreur.
        - **content** (Object): Détails du contenu créé.

Fonction getAllContent

La fonction **getAllContent** permet de récupérer tous les contenus existants dans le système.

- **Paramètres:**
    - **req** (Object): Requête HTTP.
- **Retour:**
    - Renvoie une réponse JSON contenant tous les contenus de la base de données.

Fonction getContentById

La fonction **getContentById** permet de récupérer un contenu spécifique en fonction de son ID.

- **Paramètres:**
    - **req** (Object): Requête HTTP contenant l'ID du contenu à récupérer.
    - **res** (Object): Réponse HTTP renvoyée au client.
- **Retour:**
    - Renvoie une réponse JSON contenant les détails du contenu spécifié par l'ID.

Fonction deleteContentById

La fonction **deleteContentById** permet de supprimer un contenu spécifique en fonction de son ID.

- **Paramètres:**
    - **req** (Object): Requête HTTP contenant l'ID du contenu à supprimer.
    - **res** (Object): Réponse HTTP renvoyée au client.
- **Retour:**
    - Renvoie une réponse JSON avec un message de succès ou une erreur, indiquant si le contenu a été supprimé avec succès ou s'il n'a pas été trouvé.

Remarques

- Les tokens d'authentification sont utilisés pour assurer que la création de contenu est effectuée par un utilisateur authentifié.

- Les erreurs sont gérées de manière à renvoyer des réponses appropriées avec des codes d'erreur HTTP.
- L'accès à certaines fonctionnalités, comme la suppression de contenu, peut nécessiter des autorisations spécifiques.

Fonctions de la classe media

Fonction uploadMedia

La fonction **uploadMedia** permet de télécharger un fichier média sur Azure Blob Storage et d'enregistrer les informations du média dans une base de données MongoDB.

- **Paramètres:**
    - **req** (Object): Requête HTTP contenant le fichier à télécharger.
    - **res** (Object): Réponse HTTP renvoyée au client.
- **Fonctionnement:**

1. Récupère le fichier à partir de la requête HTTP (**req.files.file**).
2. Génère un nom unique pour le blob en utilisant la date actuelle et le nom du fichier.
3. Utilise les informations d'identification Azure à partir des variables d'environnement pour créer un client BlobService.
4. Charge le fichier dans le conteneur Blob sur Azure Blob Storage.
5. Enregistre les informations du média dans la base de données MongoDB, y compris le nom du fichier, l'ID de l'utilisateur uploader et l'URL du blob.
6. Renvoie une réponse JSON avec un message de succès et les détails du fichier média enregistré.

- **Retour:**
    - Object JSON avec les propriétés suivantes :
        - **message** (String): Message de succès ou d'erreur.
        - **media** (Object): Détails du fichier média téléchargé.

Fonction getAllMedia

La fonction **getAllMedia** permet de récupérer tous les fichiers média enregistrés dans la base de données.

- **Paramètres:**
    - **req** (Object): Requête HTTP.
- **Retour:**
    - Renvoie une réponse JSON contenant tous les fichiers média enregistrés dans la base de données.

Remarques

- Les informations d'identification Azure (nom du compte, clé du compte) et le nom du conteneur sont récupérés à partir des variables d'environnement via **process.env**.
- Les erreurs sont gérées de manière à renvoyer des réponses appropriées avec des codes d'erreur HTTP.
- L'accès à la fonction **uploadMedia** peut nécessiter des autorisations spécifiques en fonction des besoins de votre application.

Fonctions de la classe comment

1. addComment

- **Description**: Ajoute un commentaire à un contenu spécifié.
- **Méthode HTTP**: POST
- **Endpoint**: **/comments/add**
- **Paramètres de requête**: Le texte du commentaire (**text**) et l'identifiant du contenu (**contentId**) provenant du corps de la requête.
- **Réponse réussie**: Retourne un message de succès et les informations du commentaire ajouté.
- **Réponse d'erreur**: En cas d'échec, renvoie un statut 500 avec un message d'erreur.

2. getCommentsByContentId

- **Description**: Récupère tous les commentaires associés à un contenu spécifique.
- **Méthode HTTP**: GET
- **Endpoint**: **/comments/:contentId**
- **Paramètres de requête**: L'identifiant du contenu (**contentId**) extrait de l'URL.
- **Réponse réussie**: Retourne un tableau contenant les commentaires associés au contenu spécifié.
- **Réponse d'erreur**: En cas d'échec, renvoie un statut 500 avec un message d'erreur.

Modèle de Commentaire

Le modèle de données des commentaires devrait avoir au moins les propriétés suivantes :

- **text**: Texte du commentaire.
- **contentId**: Identifiant du contenu auquel le commentaire est associé.

Notes

- Les blocs **try-catch** sont utilisés pour gérer les erreurs et renvoyer des réponses appropriées en cas d'échec des opérations.
- Les erreurs sont consignées dans la console pour faciliter le débogage.

Assurez-vous que le modèle de données **Comment** est correctement défini avec les propriétés nécessaires dans le fichier **../models/comment**.