Documentation du projet Link Up

Introduction

Link Up est une plateforme de réseaux sociaux conçue pour les créateurs de contenu. En exploitant la puissance d'Azure, cette application offre des fonctionnalités flexibles et évolutives pour la gestion de contenu, permettant une intégration transparente avec diverses interfaces front-end.

L'objectif est de fournir une solution backend robuste et sécurisée qui peut être utilisée pour construire des réseaux sociaux adaptés aux besoins spécifiques des créateurs et des développeurs.

Fonctionnalités principales

1. Authentification et Autorisation

- Authentification sécurisée des utilisateurs via JWT.
- Gestion des profils utilisateurs : public ou privé.
- Middleware de protection des routes RESTful, limitant l'accès aux utilisateurs authentifiés.

2. Gestion du Contenu

- CRUD (Create, Read, Update, Delete) complet pour les publications.
- Association de contenu à des utilisateurs authentifiés.

3. Gestion des Médias

- Upload de fichiers (images, vidéos) via Azure Blob Storage.
- Génération d'URL sécurisées pour accéder aux fichiers stockés.

4. API RESTful

- Endpoints sécurisés pour gérer les utilisateurs, les publications, et les fichiers.
- Documentation Swagger pour faciliter l'intégration avec des interfaces front-end.

Architecture du Projet

L'architecture backend est construite avec :

- **Node.js et Express** : pour la gestion des API RESTful.
- **Azure Cosmos DB** : pour le stockage des données utilisateurs et des publications.
- **Azure Blob Storage** : pour le stockage des fichiers multimédia.

API RESTful

Endpoints Utilisateurs

Méthode	Endpoint	Description	Authentification
POST	/users/register	Enregistrer un nouvel utilisateur.	❌ Non requise
POST	/users/login	Authentifier un utilisateur.	❌ Non requise

Endpoints Publications

Méthode	Endpoint	Description	Authentification
POST	/posts	Créer une nouvelle publication avec un fichier (optionnel).	✅ Requise
GET	/posts	Récupérer toutes les publications visibles.	✅ Requise
GET	/posts/me	Récupérer les publications de l'utilisateur connecté.	✅ Requise
GET	/posts/:id	Récupérer une publication spécifique par ID.	✅ Requise
PATCH	/posts/:id	Mettre à jour une publication spécifique par ID.	✅ Requise
DELETE	/posts/:id	Supprimer une publication spécifique par ID.	✅ Requise

Exigences Générales

Pour les requêtes POST et PUT des publications :

Le corps de la requête doit contenir :

- content : Une chaîne de caractères représentant le contenu à ajouter ou à mettre à jour.
- file : Un fichier au format binaire.

Pour les requêtes login et register :

Le corps de la requête doit contenir :

- username : Une chaîne de caractères représentant le nom d'utilisateur.
- password : Une chaîne de caractères représentant le mot de passe.

Documentation Swagger

Swagger est intégré pour documenter et tester les API RESTful. Accessible à l'URL suivante :

- Swagger Documentation

Déploiement sur Azure

Résumé des resources déployées

Ressource	Nom	Emplacement	Justification
Groupe de Ressources	LinkUpRG	West Europe	Centralise les ressources pour simplifier la gestion.
Cosmos DB	linkup-cosmosdb	France Central	Faible latence, mode Serverless adapté aux charges légères.
Base de Données	LinkUpDB	France Central	Stockage des données utilisateur et contenu.
Conteneurs Cosmos DB	Users, Posts	France Central	Organisation logique des données (utilisateurs, posts).
Compte Blob Storage	linkupstoragelma	West Europe	Stockage multimédia, configuration LRS pour résilience locale.
Conteneur Blob Storage	uploads	West Europe	Stockage des fichiers avec contrôle d'accès par SAS ou niveau d'accès blob.
App Service	linkuplma	West Europe	Hébergement du backend Node.js, plan tarifaire économique pour démarrage.

Récapitulatif des étapes

1. **Groupe de Ressources**
    - Créé un groupe **LinkUpRG** dans **West Europe** pour regrouper toutes les ressources.
2. **Azure Cosmos DB**
    - Créé une instance **linkup-cosmosdb** en **France Central**.
    - **Mode Serverless** : Idéal pour les projets légers, permet une facturation basée sur la consommation réelle.
    - Créé une base de données **LinkUpDB** avec les conteneurs suivants :
        - **Users** : Stocke les informations des utilisateurs.
        - **Posts** : Stocke les publications et leurs métadonnées.
3. **Azure Blob Storage**
    - Créé un compte de stockage **linkupstoragelma** en **West Europe**.
    - Paramètres :
        - Performances : **Standard**.
        - Réplication : **Stockage localement redondant (LRS)**.
        - Type : **StorageV2**.
    - Créé un conteneur **uploads** pour stocker les fichiers multimédias avec un niveau d'accès **Objet blob**.

4. **Azure App Service**
    - Créé un service **linkuplma** pour héberger l'API.
    - Configuration :
        - Système d'exploitation : **Linux**.
        - Plan tarifaire : **B1** (coût optimisé pour les petits projets).
        - Version de Node.js : **20 LTS**.
    - Lié le dépôt Git pour un déploiement continu.
    - Ajouté les variables d'environnement directement dans les paramètres de l'App Service.
5. **Configuration dans le Code**
    - Ajouté les informations nécessaires dans **config.js** :
        - **Cosmos DB** : endpoint, key, databaseId, containerId.
        - **Blob Storage** : accountName, accountKey, containerName.

Sécurité

Authentification JWT

- Les utilisateurs reçoivent un token JWT lors de l'inscription ou de la connexion.
- Les endpoints REST sont protégés par un middleware qui vérifie la validité du token.

Azure Blob Storage

- Utilisation d'URL SAS (Shared Access Signatures) pour sécuriser les accès aux fichiers multimédias.

Scalabilité et Performances

Recommandations pour l'Optimisation

1. **Cosmos DB** :
    - Ajouter des index sur les champs souvent recherchés.
    - Utiliser des partitions pour répartir les données efficacement.
2. **Blob Storage** :
    - Compresser les fichiers avant l'upload pour économiser de l'espace.
    - Configurer des URL SAS pour limiter la durée d'accès aux fichiers.
3. **Backend** :

- Utiliser Redis pour la mise en cache des requêtes fréquentes.
- Implémenter une pagination pour réduire les charges sur l'API.

4. **Azure App Service** :
   - Activer l'**autoscaling** pour gérer les pics de trafic.

Structure des Fichiers

```
LinkUp/
├── app.js                  # Point d'entrée principal de l'application
├── config.js               # Configuration pour Azure et JWT
├── cosmosClient.js         # Configuration et connexion à Cosmos DB
├── blobService.js          # Gestion du stockage Blob
├── routes/                 # Routes pour les API REST
│   ├── userRoutes.js       # Routes pour les utilisateurs
│   └──postRoutes.js        # Routes pour les publications
├── controllers/            # Contrôleurs de logique métier
│   ├── userController.js   # Gestion des utilisateurs
│   ├── postController.js   # Gestion des publications
├── middleware/             # Middleware (authentification)
│   ├── auth.js             # Middleware d'authentification
├── swagger.js              # Documentation Swagger pour les API
├── package.json            # Dépendances du projet
```

Liens Utiles

- Swagger Documentation
- Azure Cosmos DB
- Azure Blob Storage
