Travail pratique 1 - Création d'un service Web en REST sur Node.js connectée à une base de données infonuagique.

Objectifs

- Créer un service Web en REST utilisant Node.js et Express.js servant à fournir des données à un site Web de critiques de films.
- Utiliser adéquatement une base de données infonuagique sur Firestore de Google Cloud Platform
- Utiliser adéquatement les requêtes HTTP GET, POST, PUT et DELETE ainsi que les codes de statut HTTP.
- Gérer les erreurs et retourner les messages adéquats.
- Déployer le service Web sur Render à partir d'un dépôt GitHub (gratuit).

Consignes

Vous devez créer un service Web en REST utilisant Node.js et Express.js servant à fournir des données à un site Web de critiques de films. Nous ne créerons pas le site Web, mais nous allons créer le service Web qui fournira les données au site Web. Le service Web doit être déployé sur Render à partir d'un dépôt GitHub.

1. Connectez-vous au dépôt GitHub Classroom que je vous ai fourni: https://classroom.github.com/a/tO9SSXZi
2. Créer un serveur Web avec Node.js et Express.js.
3. Créer toutes les routes nécessaires pour répondre aux besoins du site Web. (voir plus bas) et assurez-vous de retourner les bons codes de statut HTTP et de gérer les erreurs ainsi que les cas où les données ne sont pas trouvées.
4. Créer une base de données sur Firestore de Google Cloud Platform et y ajouter les données nécessaires.
5. Créer un fichier `.env` et un `db-config.json` contenant les informations de connexion à la base de données. **Ne pas ajouter ce fichier au dépôt GitHub. Vous devrez déposer ce fichier sur Omnivox et sur Render**
6. Créer un fichier `.gitignore` pour ignorer le fichier `.env`, le fichier de configuration de la base de données et le dossier `node_modules`.
7. Créer un dossier public contenant une page index.html. Rédigez la documentation de votre API dans cette page. La documentation doit contenir les liens vers les différentes routes et donner des explications sur les différents paramètres nécessaires s'il y a lieu.
8. Faîtes une `pull request` lorsque votre projet est complet et déployer votre application sur Render.

Les routes

Votre serveur doit minimalement gérer les routes suivantes:

Routes publiques

Films

- `GET /api/films` - Retourne la liste de tous les films. Vous devez être en mesure de trier les films par année, par titre et par réalisateur de façon ascendante ou descendante à l'aide de paramètres dans l'URL. (Ex: /api/films?tri=annee&ordre=asc)
- `GET /api/films/:id` - Retourne le film ayant l'identifiant `:id`.
- `POST /api/films` - Ajoute un nouveau film à la liste de films. Le corps de la requête doit contenir les informations du film à ajouter. Le film doit être ajouté à la base de données et l'identifiant du film doit être retourné.
- `PUT /api/films/:id` - Modifie le film ayant l'identifiant `:id`. Le corps de la requête doit contenir les informations du film à modifier. Le film doit être modifié dans la base de données.
- `DELETE /api/films/:id` - Supprime le film ayant l'identifiant `:id` de la base de données.

Utilisateurs

- `POST /api/utilisateurs/inscription` - Ajoute un nouvel utilisateur à la liste des utilisateurs. Le corps de la requête doit contenir les informations de l'utilisateur à ajouter. Le mot de passe doit être encrypté avant d'être ajouté à la base de données. L'identifiant de l'utilisateur doit être retourné. Assurez-vous de ne pas créer deux utilisateurs avec le même courriel. Retournez `false` avec un message d'erreur dans ce cas précis.
- `POST /api/utilisateurs/connexion` - Connecte un utilisateur. Le corps de la requête doit contenir les informations de l'utilisateur à connecter. Le mot de passe doit être comparé avec celui dans la base de données. Si le mot de passe est valide, votre API retournera `true` sinon elle retournera `false` avec un message d'erreur approprié..

Les entités

L'entité `film` doit contenir minimalement les informations au format suivant:

```
 {
        "titre": "Alien - Le 8ème passager",
        "genres": ["Horreur", "Science-fiction"],
        "description":
            "Un vaisseau spatial perçoit une transmission non-identifiée comme un
signal de détresse. Lors de son atterrissage, l'un des membres de l'équipage est
attaqué par une mystérieuse forme de vie, ils réalisent rapidement que son cycle de
vie vient seulement de commencer.",
        "annee": "1979",
        "realisation": "Ridley Scott",
        "titreVignette": "alienle8emepassager.jpg",
        "commentaires": [
            {
                "auteur": "Simon",
                "texte": "Un classique du cinéma d'horreur."
            },
            {
```

```
                "auteur": "Maxime",
                "texte": "J'en fait encore des cauchemars."
            }
        ]
    },
```

L'entité `utilisateur` doit contenir minimalement les informations au format suivant:

```
{
        "courriel": "admin@email.com",
        "mdp": "12345",
    },
```

Je vous ai fourni un fichier modèle de données pour chaque entité que vous pouvez utiliser au besoin

Critères d'évaluation

Le travail compte pour 25% de la note finale.

Utilisation adéquate des éléments du langage /15

- Il est possible de récupérer, ajouter, modifier et supprimer des films.
- Les données récupérées contiennent l'id de l'élément
- Il est possible de trier les films par année, par titre et par realisateur de façon ascendante ou descendante à l'aide de paramètres dans l'URL.
- Il est possible de créer un utilisateur et de se connecter.
- Les mots de passe sont encryptés avant d'être ajoutés à la base de données.
- Les données sont bien validées avant d'être ajoutées à la base de données.
- Les routes publiques sont toutes accessibles et retournent les bons codes de statut HTTP et les bonnes données et gère les cas où les données ne sont pas trouvées et/ou les erreurs.
- L'API utilise des paramètres dynamiques dans les routes. (Ex: /api/films/:id)
- L'API utilise des paramètres dans l'URL. (Ex: /api/films?tri=annee&ordre=asc)

Optimisation et qualité du code /5

- Le code est bien structuré et facile à lire.
- Le code est exempt d'erreurs.
- Le code est optimisé, sans répétition et est exempt de code inutile.
- Le code est bien commenté.
- Les fonctions et variables sont nommées de façon adéquate.
- Le code est bien indenté.
- Chaque route est documentée dans la page index.html. La documentation contient les liens vers les différentes routes ainsi que les paramètres au besoin.

Respect des consignes /5

- Le projet respecte le devis.
- Le projet est déposé sur GitHub et déployé sur Render.com
- Le fichier `.env`, le fichier `db-config.json` et les infos de connexion sont déposés sur Omnivox

Bonus /1.25

- Il est possible de modifier et supprimer un utilisateur

Remise

Vous devez déposer sur Omnivox:

- un dossier zip contenant votre projet **sans le dossier node_modules**.
- le fichier db-config.json
- le fichier .env
- le nom et le mot de passe de l'utilisateur pour tester la connexion,
- le lien vers votre application Render

Une pénalité de 5% par jour de retard sera appliquée

Plagiat

Il s'agit d'un travail individuel. Vous pouvez utiliser les notes où la documentation. Vous devez citez vos sources si vous empruntez du code de plus de 3 lignes sur Internet. L'utilisation d'outils de génération de code par intelligence artificielle est interdite (Ex: Copilot, ChatGPT). Tout plagiat entrainera une note de 0 pour le travail.