420-15D-FX TP3 A24

Travail pratique #3 : Partie 1 API - Niyɔrɔ

Pondération : 30%
Travail à faire individuellement
Date de remise : le **Vendredi 29 novembre 2024** avant minuit

1. Contexte

Ce travail pratique porte sur la complétion d'une **API REST** de base. La documentation devra être réalisée avec **Postman**.
L'API devra être hébergée sur un serveur distant.
L'API devra respecter un jeu de tests automatisés.
La partie *frontend* en Vue.js sera réalisée plus tard dans la partie 2.

2. Description

L'objectif de ce travail pratique est de concevoir une API pour une application de partage permettant aux utilisateurs de sauvegarder et d'organiser des contenus intéressants trouvés en ligne (sites web, extraits de code, recettes, lieux, etc.).

Les utilisateurs pourront rendre ces items publics ou les partager de manière privée avec leurs amis.

Un système de réactions sera également implémenté pour permettre aux utilisateurs d'exprimer leurs opinions sur les contenus partagés.

Cette application sera un prototype et toutes les fonctionnalités d'une vraie application ne seront pas implémentées.

3. Technologies à utiliser

Liste des technologies à utiliser pour réaliser l'application:

- Base de données : MongoDB
- Backend : Node.js, Express.js
- ORM : Mongoose
- Tests : Postman

Le projet inclura un jeu de tests automatisés qui validera la conformité des routes et des opérations CRUD demandées.

4. Fonctionnalités

4.1 Authentification

L'utilisateur doit pouvoir s'inscrire sur le site en saisissant:

- un pseudo
- un nom
- un prénom
- un courriel
- un mot de passe
- une confirmation du mot de passe

Les utilisateurs administrateurs auront le champ `is_admin` à `true`. On modifie ce champ directement dans la base de données.

Le champ `is_active` est à `true` par défaut.

Le mot de passe doit être crypté en utilisant la librairie **bcrypt**.

L'authentification doit être gérée grâce à un **jeton JWT**.

Le jeton d'authentification doit être valide pour une période de 24h. À tout moment, si le jeton est expiré ou que l'utilisateur n'est pas connecté et qu'il tente d'accéder à une page pour laquelle il doit l'être, alors celui-ci est redirigé vers la page de connexion.

4.1.2 Connexion

L'utilisateur doit pouvoir se connecter sur le site en saisissant un courriel et un mot de passe.

Un utilisateur déconnecté a seulement accès aux pages d'inscription et de connexion.

4.2 Profils

4.2.1 Utilisateurs

Les utilisateurs doivent pouvoir visualiser et modifier leur profil.

Les modifications possibles sont :

- le nom
- le prénom
- le courriel
- le mot de passe
- l'avatar

L'utilisateur doit pouvoir supprimer son compte. Lorsqu'un utilisateur supprime son compte, tous les items et réactions associés à cet utilisateur doivent également être supprimés.

L'avatar sera créé à partir du `pseudo` de l'utilisateur en utilisant le site Robohash (https://robohash.org/).
Ce doit être une URL de type `https://robohash.org/:pseudo`

4.2.2 Administrateur

L'administrateur doit pouvoir visualiser la liste des utilisateurs et modifier les informations de chaque utilisateur. Il peut également supprimer ou rendre inactif un utilisateur. Un utilisateur inactif ne peut plus se connecter.

L'administrateur doit pouvoir visualiser la liste des items et tags. Il peut également supprimer un item ou un tag.

4.3 Items

Un utilisateur doit pouvoir ajouter, modifier, supprimer et visualiser ses items (individuel ou liste).

Il peut également visualiser les items (individuel ou liste) qui ne sont pas privés des autres utilisateurs. Prévoir une route pour afficher tous les items publics.

Un item est composé des champs suivants :

- titre obligatoire
- url
- contenu
- latitude
- longitude
- privé ou public
- épinglé ou non
- permalien obligatoire
- tags (choix de tags existants ou création de nouveaux tags s'ils n'existent pas)
- créé par (référence à l'utilisateur qui a créé l'item)
- date de création
- date de mise à jour

Les items peuvent être publics ou privés. Les items privés ne sont visibles que par l'utilisateur qui les a créés.
Les items peuvent être épinglés. Les items épinglés apparaissent en premier quand on récupère la liste des items d'un utilisateur.

Les items doivent être associés à un utilisateur. Un utilisateur peut avoir plusieurs items.
Quand on récupère un item, on doit aussi récupérer les réactions associées à cet item.

Le permalien est un champ unique pour chaque item. Il est généré automatiquement à partir du titre de l'item + un identifiant unique (UUID par exemple).
Il permet d'accéder à l'item via une URL de type `/item/:permalink` quand on utilisera vue.js.

4.4 Réactions

Les utilisateurs doivent pouvoir réagir à un item en choisissant parmi les réactions suivantes :

- 1-J'aime
- 2-Informatif
- 3-Drôle
- 4-Inapproprié

Seules les valeurs numériques sont enregistrées en base de données.

Les réactions doivent être associées à un utilisateur et à un item. Un utilisateur **ne peut pas réagir plusieurs fois** à un même item.

Les réactions doivent être datées. Une réaction ne peut pas être modifiée. Elle peut seulement être supprimée. Seul l'utilisateur qui a fait la réaction peut la supprimer.

Les réactions doivent être visibles par tous les utilisateurs. Les réactions doivent être affichées quand un utilisateur visualise un item.

L'administrateur n'a pas besoin de gérer les réactions.

4.5 Tags

Quand un utilisateur crée un item, il peut lui associer un ou plusieurs tags. Les tags sont enregistrés dans une collection à part et sont associés à un ou plusieurs items.

L'utilisateur peut supprimer la référence à un tag dans un item mais pas le tag lui-même (quand il met à jour un item).

S'il n'existe pas déjà dans la base de données, le tag sera créé pour l'associer à un item.

Les tags doivent donc être uniques.

Seul l'administrateur peut effectuer toutes les autres opérations CRUD sur les tags. Si un tag est supprimé, il doit être retiré de tous les items qui lui sont associés.

5. Validation des données

5.1. Utilisateur

Champ	Contrainte(s)
Pseudo	String, Entre 1 et 50 caractères Requis N'est pas déjà utilisé
Prénom	String, Entre 1 et 50 caractères Requis
Nom	String, Entre 1 et 50 caractères Requis
Courriel	String, Est un courriel N'est pas déjà utilisé Requis
Mot de passe	String, Doit contenir un minimum 6 caractères Requis
Confirmation du mot de passe	String, Égal au mot de passe Requis
is_active	Boolean False par défaut
is_admin	Boolean False par défaut
avatar	String, URL de l'avatar

5.2 Item

Champ	Contrainte(s)
title	String, maximum 100 caractères Requis
url	String, URL pour les sites et articles
content	String, contenu optionnel
latitude	Number, latitude géographique
longitude	Number, longitude géographique
private	Boolean, indique si l'item est privé
sticky	Boolean, indique si l'item est épinglé
permalink	String, permalien unique pour l'item Requis
tags	String, tableau de références aux tags
created_by	String, référence à l'utilisateur qui a créé l'item Requis

Champ	Contrainte(s)
created_at	Date, date de création
updated_at	Date, date de dernière mise à jour

5.3 Tag

Champ	Contrainte(s)
name	String, nom du tag Requis N'est pas déjà utilisé Pas plus de 50 caractères
created_at	Date, date de création
updated_at	Date, date de dernière mise à jour

5.4 Réaction

Champ	Contrainte(s)
type	enum ("J'aime", "Informatif", "Drôle", "Inapproprié"), type de réaction Requis
user_id	String, référence à l'utilisateur qui a fait la réaction Requis
item_id	String, référence à l'élément faisant l'objet de la réaction Requis
created_at	Date, date de création
updated_at	Date, date de dernière mise à jour

6. API

L'API doit être développée avec **Express.js**.

Vous devez compléter le code qui est fourni. Vous pouvez modifier le code existant sauf pour les **routes**. Vous aurez probablement besoin de plusieurs `middleware` pour vérifier si l'utilisateur est administrateur, s'il est authentifié, s'il est propriétaire de la ressource, etc.

Toutes les routes définies dans le dossier *routes* doivent être implémentées.

Assurez-vous qu'un utilisateur ne puisse pas effectuer d'actions sur des ressources qui ne lui appartiennent pas.

Le champ **mot de passe** ne doit jamais apparaître dans les réponses.

Assurez vous d'utiliser un `middleware` pour gérer les erreurs.

Les concepts vus en classe doivent être respectés :

- Utilisation des codes HTTP appropriés (200, 201, 403, 404, etc.)
- Gestion des erreurs
- Utilisation de `middlewares` (authentification, validation, etc.)

7. Tests

Vous devez réaliser un jeu de tests automatisés avec *Postman* pour tester tous les cas qui concernent l'authentification et les utilisateurs / items (toutes les routes qui sont dans les fichiers `routes/auth.mjs` et `routes/item.mjs`).

La collection de test doit pouvoir être exécutée de manière séquentielle comme pour le TP2. Les requêtes doivent être **dynamiques et utiliser des variables de collection**.

La base de données **doit être dans le même état après l'exécution des tests**. Par exemple si vous créez un enregistrement, vous devez le supprimer.

Vous devez ajouter une route à l'API permettant de supprimer les données et d'ajouter vos données de test (seeds). Ainsi, vous devez ajouter cette route à vos requêtes Postman afin qu'elle puisse être exécutée à la fin de l'exécution de vos tests.

La base de données doit contenir au minimum :

- 5 utilisateurs (user1@niyoro.ca, user2@niyoro.ca...),
- 1 administrateur (admin@niyoro.ca),
- au moins 3 items par utilisateur, privés et publics, épinglés ou non,
- des tags pour les items (entre 0 et 3 tags par item),
- des réactions pour plusieurs items.

Tous les mots de passe doivent être **123123**.

8. HATEOAS

Afin d'alléger ce travail, le principe HATEOAS doit être implémenté seulement pour toutes les routes qui concernent les `tags`

9. Documentation

Vous devez réaliser la documentation de l'API avec *Postman*. Vous devez utiliser la fonctionnalité **Publish** de Postman pour publier la documentation sur le web.

10. Hébergement

L'API n'a pas beson d'être hébergée sur un serveur distant pour le moment mais il faudra le faire pour la partie 2.

11. Caractéristiques générales du code

- respectez les règles de base de la programmation vues en cours.
  - structure du code.
  - présentation du code, indentation, saut de ligne.
  - nommage des fonctions et variables de manière intelligible et raisonnée.
  - commentaires.
- une attention particulière sera portée à la qualité du code.
- le code doit être le plus simple et le plus lisible possible.
- Portez une attention particulière à la qualité de l'interface.

12. À remettre

- Votre projet doit être remis sur Léa au format zip.
- Il doit comporter dans des dossiers séparés :
  - le code de l'**API** avec un fichier `readme.md` avec les informations suivantes :
    - **L'url de la documentation** *Postman*.
    - **Url du dépôt Git** sur Gitlab.
    - Ajoutez des informations que vous jugerez utile de fournir.
  - un export des requêtes *Postman* au format JSON pour **les tests**.
  - un fichier d'export de la **base de données** au format Mondodb.

13. Annexes

13.1 Base de données

Structure suggérée de la base de données (vous pouvez la modifier) :

https://dbdiagram.io/d/Niyɔrɔ-6718091197a66db9a3e8f228

13.2 Barème de correction

Voici une ébauche du barème de correction. Il pourrait être modifié lors de la correction.

- Git / Base de données / Architecture / Seeds: **20%**
- API: **40%**
- Tests / Documentation Postman / HATEOAS: **30%**

- Caractéristiques générales: **10%**
  - Qualité du code
  - Documentation du code
  - Respect des principes enseigné dans le cours
  - Qualité des interfaces
- Pénalités
  - Retard.
  - Remise incorrecte du travail.
  - Français dans les interfaces.
  - D’autres éléments selon le besoin.

> S'il y a lieu, des précisions ou des ajustements vous seront donnés sur le canal Teams.