Travail pratique #2

API -REST

Objectif

Compléter une API REST de base et modifier la structure d'une base de données en utilisant les technologies suivantes : Node.js, Express.js, MongoDB et Mongoose.

Contexte

La clinique médicale HealHub souhaite développer une API REST afin de gérer les médecins, les patients et les rendez-vous entre les médecins et leurs patients. L'API est commencée, mais le développeur précédent n'a pas eu le temps de la terminer et de plus il a commis des **erreurs** !

Modalités

- Ce travail doit être fait **individuellement**
- Cette partie du travail est sur 100 points et vaut pour **30% de la note finale**.
- Date de remise : **Voir la date de remise sur MS Teams**

Fonctionnement de l'application

L'API doit permettre :

- De créer, de modifier, de consulté et de supprimer des médecins.
- De créer, de modifier, de consulté et de supprimer des patients.
    - D'ajouter ou supprimer des informations dans l'historique du dossier d'un patient
- De créer et de supprimer des rendez-vous
    - Chaque rendez-vous est d'une durée fixe de 30 minutes. Ainsi, pour créer un rendez-vous, il suffit de fournir la date et l'heure de début. La date et heure de fin sera calculée à partir de la date de l'heure de début. Attention, il ne doit pas y avoir de conflits lors de la création d'un rendez-vous.

Modèles

Voici les caractéristiques des différents modèles :

- Médecin (tous les champs sont obligatoires) :
    - Nom
    - Prénom
    - Téléphone
    - Courriel
    - Spécialité (ex. Cardiologue, Pédiatre, etc.)
- Patient (tous les champs sont obligatoires):
    - Nom
    - Prénom
    - Date de naissance
    - Téléphone
    - Courriel
    - Adresse
    - Code postal (vous n'avez pas à valider le format du code postal pour cette version de l'API)
    - Historique : tableau dont chaque élément est représenté par :
        - Information (requis)
        - Identifiant du médecin qui a écrit dans l'historique
        - Create_at : Date et heure à laquelle l'élément a été ajouté.
- Rendez-vous (tous les champs sont requis)
    - Identifiant du patient
    - Identifiant du médecin
    - Début (date et heure de début du rendez-vous)
    - Fin (date et heure de fin du rendez-vous)
    - Notes (Motif du rendez-vous)

Instruction

- Configurer l'environnement et les requêtes dans Postman avec les fichiers fournis :
    - HealHub.postman_collection.json
    - HealHub.postman_environment.json
- Compléter ou créer les modèles suivants :
    - medecin.js
    - patient.js
    - rendezvous.js
- Installer les modules de projet
- Démarrer le projet et exécuter la route http://localhost:3000/db/seed pour charger les collections dans la base de données ou exécuter le test **GET db/seed**.

Compléter le code fourni dans le fichier dans le dossier **HealHub** afin de respecter tous les éléments indiqués en annexe et que tous les tests configurés dans Postman aient été exécuté avec succès.

Tests

Le jeu de test fourni utilise la version initiale de la base de données. Si vous exécutez les tests après avoir modifié la base de données, il est normal que certains tests échouent. Vous devrez réinitialiser la base de données en exécutant le test **GET db/seed.**

Caractéristiques générales du code

- Respectez les règles de base de la programmation vues en classe.
    - Structure du code
    - Présentation du code, indentation, saut de ligne.
    - Nommage des fonctions et variables de manière intelligible et raisonnée.
- Vous devez commenter votre code en utilisant **JSDoc Generator**
- Porter une attention particulière à la qualité du code. Le code doit être le plus simple possible.

Remise

Vous devez remettre sur MS Team : Un fichier nommé "**InfosGitLab.txt**" contenant l'URL de votre dépôt ***GitLab***.

Assurez-vous de me donner accès à votre dépôt distant (mvezina@cegepgarneau.ca).

Précisions sur l'évaluation

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

Éléments	Critères de performance	Savoir-faire	Points
00SU Effectuer le développement d'applications Web transactionnelles			
1. Analyser le projet de développement de l'application.	1.1 Analyse juste des documents de conception. 1.2 Détermination correcte des tâches à effectuer.	Respect des exigences du document de conception	15
2. Préparer l'environnement de développement informatique.	2.1 Installation correcte de la plateforme de développement et du système de gestion de base de données de développement. 2.2 Installation correcte des logiciels et des bibliothèques.	Installer correctement la plateforme de développement Web et le système de gestion de base de données de développement. Installer les logiciels et bibliothèques appropriés	5
	2.3 Configuration appropriée du système de gestion de versions. 2.4 Importation correcte du code source.	Configurer le gestionnaire de versions (local et distant). Créer un clone d'un dépôt distant. Qualité et fréquences des commits	5
3. Préparer la base de données.	3.1 Création ou adaptation correctes de la base de données. 3.2 Insertion correcte des données initiales ou des données de tests. 3.3 Respect du modèle de données.	Créer une base de données ou bien importer et adapter une base de données. Insérer des données initiales ou des données de tests. Vérifier la correspondance de la base de données avec le modèle de données.	5

4. Programmer la logique applicative du service.	4.2 Programmation correcte de la réception des données d'entrée. 4.3 Choix approprié des clauses, des opérateurs, des commandes ou des paramètres dans les requêtes à la base de données. 4.4 Manipulation correcte des données de la base de données.	Programmer l'extraction (lecture et désérialisation) des données d'entrée. Respect du format de donnée JSON Choix et qualité des requêtes Programmer les traitements appropriés sur les données récupérées	15
	4.5 Programmation correcte de l'envoi des données de sortie. 4.6 Application rigoureuse des techniques de programmation sécurisée. 4.7 Respect des protocoles de communication et des formats d'échange de données.	Déterminer le code de statut et les entêtes selon les résultats de l'opération Déterminer les formats des données de sortie Transformer les résultats selon les formats de données déterminés Valider les données d'entrée au moment de la désérialisation Gestion des erreurs Respect de l'architecture REST Concevoir l'interface du service d'échange de données • Ressources • URI • méthodes, • formats de données • en-tête http • etc. Associer une méthode à l'opération à effectuer sur la ressource	45
		Caractéristiques générales • Qualité du code • Documentation du code	10
Pénalités • Retard. • Remise incorrecte du travail. • Français dans les interfaces • D'autres éléments selon le besoin.			

Annexe

Ressources REST

Pour toutes les requêtes PUT et POST, les **Header** doivent avoir Content-type : application/json

Pour toutes les réponses, les Header doivent avoir Content-type : application/json excepté pour la méthode DELETE.

Le code de statut doit être 404 si une ressource n'est pas trouvée. Il pourrait y avoir d'autre code d'erreur à retourner selon le besoin.

URI	Méthode	Description	Requête	Code de statut	Réponse
medecins	GET	Obtient tous les médecins. Paramètre **specialite** optionnel de type **string**: permet de filtrer les médecins sur leur spécialité		200	**Body** : Tableau d'objets Json médecin
medecins	POST	Crée un médecin	**Body** : Objet Json médecin	201	**Body** : Objet Json médecin. **Location** : Uri pour accéder au médecin créé.
medecins/:id	GET	Obtient la représentation d'un médecin ayant l'id "id".		200	**Body** : Objet Json médecin.

medecins/:id	PUT	Permet de modifier un médecin ayant l'id "id".	**Body** : Objet Json médecin	200	**Body** : Objet Json médecin.
medecins/:id	DELETE	Permet de supprimer un médecin ayant l'id "id".		204	
patients	GET	Obtient tous les patients.		200	**Body** : Tableau d'objets Json patient
patients	POST	Crée un patient	**Body** : Objet Json patient	201	**Body** : Objet Json patient. **Location** : Uri pour accéder au patient créé.
patients /:id	GET	Obtient la représentation d'un patient ayant l'id "id".		200	**Body** : Objet Json patient.
patients /:id	PUT	Permet de modifier un patient ayant l'id "id".	**Body** : Objet Json patient	200	**Body** : Objet Json patient.
patients /:id/historique	POST	Permet d'ajouter un objet à l'historique du patient ayant l'id "id".	**Body :** Objet Json {medecinId, information}	201	**Body:** Objet Json patient. **Location** : Uri pour accéder au patient créé.
patients /:id/historique/:id_historique	DELETE	Permet de supprimer l'élément ayant le id "id_historique" dans l'historique du patient ayant l'id "id		204	

patients /:id	DELETE	Permet de supprimer un patient ayant l'id "id".		204	
rendezvous	POST	Crée un rendez-vous	**Body** : Objet Json rendez-vous	201	**Body** : Objet Json rendez-vous. **Location** : Uri pour accéder au rendez-vous.
rendezvous/:id	GET	Permet d'obtenir le rendez-vous ayant comme l'id "id".		200	**Body:** Objet Json rendez-vous.
rendezvous/médecins/:id	GET	Obtient tous les rendez-vous du médecin ayant l'id "id". Paramètre **date** optionnel de type **date** : permet de filtrer les rendez-vous sur une date en particulier.		200	**Body:** Tableau d'objets Json rendez-vous
rendezvous/patients/:id	GET	Obtient tous les rendez-vous du patient ayant l'id "id". Paramètre **date** optionnel de type **date** : permet de filtrer les rendez-vous sur une date en particulier.		200	**Body:** Tableau d'objets Json rendez-vous
rendezvous/:id	DELETE	Permet de supprimer un rendez-vous ayant l'id "id".		204	
db/seed	GET	Permet de vider la base de données et ajouter les données initiales pour les tests		200	**Body:** Objets médecins, patients et rendez-vous créés.L