Travail pratique #3

Vue.js et API REST

Objectif

Le but de ce projet est de créer une application en utilisant une API en Express.js pour le backend et Vue.js pour le frontend.

Contexte

Vous devez réaliser une application permettant de gérer vos tâches et projets à l'aide de tableaux, de listes et de cartes. Les utilisateurs peuvent créer des tableaux pour différents projets ou catégories, et à l'intérieur de chaque tableau, ils peuvent créer des listes pour organiser les tâches par étapes ou par catégories. Chaque liste contient des cartes qui représentent des tâches individuelles ou des éléments de projet, et les utilisateurs peuvent déplacer ces cartes d'une liste à l'autre pour suivre la progression de chaque tâche.

Modalités

- Ce travail doit être fait **individuellement**
- Cette partie du travail est sur 100 points et vaut pour **40% de la note finale**.
- Date de remise : **Voir la date de remise sur MS Teams**

Fonctionnement de l'application

L'utilisateur doit pouvoir s'inscrire sur le site en saisissant un nom d'utilisateur, courriel et mot de passe et la confirmation du courriel. Le mot de passe doit être crypté en utilisant la librairie **bcrypt.**

L'utilisateur doit pouvoir se connecter en saisissant son nom d'utilisateur et mot de passe. 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 page pour laquelle il doit l'être, alors celui-ci est **redirigé vers la page de connexion**. L'utilisateur doit également pouvoir **se déconnecter du site**.

À l'arrivée sur la page d'accueil, si l'utilisateur n'est pas connecté alors l'application affiche la page de connexion permettant à l'utilisateur de se connecter ou de cliquer sur un lien lui permettant de s'inscrire. Une fois l'utilisateur connecté, l'application affiche la liste des tableaux de l'utilisateur **du plus récent au moins récent.**

Tableaux

- L'utilisateur pour créer un nouveau tableau en cliquant sur le bouton "Ajouter un tableau" et en saisissant un nom. Il peut y avoir plusieurs tableaux avec le même nom.
- L'utilisateur peut consulter les détails d'un tableau en cliquant sur un tableau et supprimer un tableau en cliquant sur l'icône de suppression.

Détails d'un tableau

- La page affichant les détails d'un tableau doit contenir les éléments suivants :
- Nom du tableau
    - Filtre permettant d'affiche les cartes dont la date limite répond aux critères suivants (multichoix) :
        - Aucune date limite
        - En retard
        - Échéance demain
    - Listes du tableau.
        - Titre de la liste4

    - Cartes de la liste
        - Titre de la carte
        - Date limite : Ne s'affiche que si la date est présente. S'affiche en **rouge si la date est dépassée**, en **jaune si l'échéance est dans moins de 24h** et en **vert sinon**.

- L'utilisateur peut créer une nouvelle liste en cliquant sur le bouton "Ajouter une liste" et en saisissant un titre. Il peut modifier le titre d'une liste en cliquant sur le titre**. La modification est appliquée lorsque l'utilisateur appuie sur le bouton "Entrée"**. Il est possible de supprimer une liste en cliquant sur l'icône de suppression.
- L'utilisateur peut créer une nouvelle carte en cliquant sur le bouton "Ajouter une carte" et en saisissant un titre. Lorsque l'utilisateur clique sur une carte, **une fenêtre modale** doit s'afficher permettant à l'utilisateur de modifier la carte (titre, description et date et heure limite). **La modification est appliquée seulement si l'utilisateur clique sur le bouton "Enregistrer"**. Il est possible de supprimer une carte en cliquant sur l'icône de suppression.
- Il doit être possible pour l'utilisateur de **glisser et déposer une carte d'une liste à une autre**. Il doit également être possible de **modifier l'ordre d'affichage des cartes en glissant et déposant une carte dans une même liste**. La ou les listes doivent être mises à jour suite à cette action.

1. Backend (API)

L'API doit être réalisée avec Express.js. Vous devez compléter le code fourni avec l'énoncé. **Vous pouvez modifier le code existant, mais pas les routes. Vous pouvez cependant, ajouter des middlewares dans les routes.**

Toutes les routes définies dans le dossier "routes" **doivent être implémentées même si elles ne sont pas toutes utilisées dans l'application.**

L'utilisateur doit être authentifié pour effectuer toutes les actions sur l'API à **l'exception de la connexion et de la création d'un compte.**

L'authentification doit être gérée avec un **jeton JWT**.

Vous devez vous assurer qu'un utilisateur ne peut pas effectuer des actions sur des tableaux, listes ou cartes qui ne lui appartiennent pas.

Vous devez respecter les notions vues en classe :

- Codes HTTP adéquats (200, 201,403,404,409...)
- Gestion des erreurs
- Utilisation de middleware (authentification, validation, etc.)

2. Frontend

- Le frontend doit être développé en utilisant **Vue.js**. Vous devez respecter la maquette filaire.
- Les **vues**, les **routes** et les **composants** suivants **doivent être inclus** dans l'application :
  - **Vues** :
    - **AccueilView** : Affiche les tableaux d'un utilisateur et permet la création d'un nouveau tableau.
    - **TableauView** : Affiche les détails d'un tableau et permet la modification du tableau, la création et modification des listes ainsi que la création et modification des cartes.
    - **ConnexionView** : Affiche le formulaire de connexion. S'il y a une erreur, le formulaire doit toujours être affiché avec les messages d'erreurs. Sinon, on redirige à la page d'accueil.
    - **InscriptionView** : Affiche le formulaire de création d'un compte. S'il y a une erreur, le formulaire doit toujours être affiché avec les messages d'erreurs. Sinon, on redirige vers la page de connexion.
  - **Composants** :

- **Tableau** : Affiche le nom d'un tableau et l'icône de suppression. Permet de rediriger vers la vue affichant son détail lorsque l'utilisateur clique sur le tableau.
- **Liste** : Affiche le titre de la liste avec l'icône de suppression et ses cartes. Permet de modifier le titre de la liste et de supprimer la liste. Permet également de créer une carte et de glisser-déposer une carte dans une autre liste ou dans la même liste.
- **Carte** : Affiche le titre, l'icône de suppression et la date limite (si présente). Affiche la fenêtre modale pour la modification de la carte que l'utilisateur clique sur la carte. La fenêtre modale doit afficher tous les champs d'une carte.

- **Routes** :
  - **/** : Affiche la page AccueilView
  - **/connexion** : Affiche la page de connexion
  - **/inscription** : Affiche la page d'inscription
  - **/tableaux/:tableauId** : Affiche le détail d'un tableau

- Vous devez afficher une page **404** si un utilisateur authentifié tente d'accéder à une ressource qui n'existe pas et une page **403** si l'utilisateur authentifié tente d'accéder à un tableau qui ne lui appartient pas.
- Vous devez respecter les bonnes pratiques vues en classe
- La mise en page doit bien s'adapter à divers types d'appareils (responsive).
- Toutes les pages produites dans le cadre de ce travail doivent s'afficher correctement avec les navigateurs modernes (Chrome et Firefox).

3. Validation des données

Vous devez effectuer la validation des données autant du côté **frontend et backend.**

- Les messages d'erreur doivent être proches des champs en erreur.
- Voir les tableaux suivants pour les contraintes de validation :

3.1 Utilisateur

Champs	Contraintes
Nom	Entre 3 et 50 caractères Requis
Courriel	Est un courriel N'est pas déjà utilisé Pas plus de 50 caractères Requis
Mot de passe	Doit contenir un minimum 6 caractères Requis
Confirmation mot de passe	Identique au mot de passe Requis

3.2 Tableau

Champs	Contraintes
Titre	Au moins 1 caractère Requis
Date de création	Requis

3.3 Liste

Champs	Contraintes

Titre	Au moins 1 caractère Requis

3.4 Carte

Champs	Contraintes
Titre	Au moins 1 caractère Requis
Description	
Date limite	Date et heure

Tests

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

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

La base de données **doit être dans le même état après l'exécution des tests**. Vous devez ajouter une route à l'API permettant de supprimer les données et ajouter vos données de tests. 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.

Vos jeux d'enregistrement doivent avoir au minimum :

- Deux utilisateurs.

- Chaque utilisateur doit avoir créé au moins deux tableaux ou plus.
- Chaque tableau doit contenir au moins deux listes ou plus.
- Chaque liste doit contenir au moins 2 cartes ou plus. Il doit y avoir des cartes avec une description des cartes avec aucune date limite, une date limite arrivant à échéance en moins de 24h et une date limite expirée.
- Tous les mots de passe des utilisateurs doivent être 123456.

Vous devez créer un "**Workspace" de type équipe** et m'ajouter à votre équipe (mvezina@cegepgarneau.ca)

HATEOAS

Le principe HATEOAS doit être implémenté pour toutes **les routes qui concernent les cartes.**

Documentation

Vous devez réaliser la documentation de l'API avec **Swagger** qui sera accessible avec la route **GET /api-docs**

Hébergement.

L'API doit être hébergée sur un serveur distant.

Caractéristiques générales du code

- Le code de l'api doit se trouver dans le **dossier "api"** et le code de l'application vue.js doit se trouver dans le **dossier "public".**
- 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.
- Porter une attention particulière à la qualité du code. Le code doit être le plus simple possible.
- Porter une attention à la qualité de l'interface.

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).

Votre dépôt doit contenir les éléments suivants :

- Le code le **l'API** (inclure le fichier .env avec les vraies informations)
- Le code **Vue.js** (inclure le fichier .env avec les vraies informations)
- Un export des requêtes Postman au format JSON pour **les tests**
- Un export des variables de **collections** et **d'environnement** de Postman au format JSON
- **l'API doit être hébergée sur un serveur distant et accessible pendant deux semaines après la date de la remise**

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.	Respect de l'architecture de l'application pour l'application l'api et l'application vue.js Installation des packages appropriés.	
	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	
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.	Respect des modèles de données pour la base de données MongoDb. Création de la route pour l'initialisation de la base de données avec des données de tests. Respect des jeux de données demandés.	

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. 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.	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 Déterminer le code de statut et les en-têtes 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 Gestion de l'authentification (JWT) Validation des données 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	20

		HATEAOS	
5. Programmer une application de test utilisant le service.	5.1 Récupération exacte de l'interface du service. 5.2 Utilisation appropriée du service. 5.3 Conversion appropriée des données fournies par le service en données exploitables par l'application de test.	Tests Postman Créer les requêtes HTTP appropriées afin d'interroger le service Web et vérifier les réponses (code HTTP, format, quantité, ordre, etc.) Désérialiser les données fournies par le service Web et vérifier le bon format. Effectuer des tests fonctionnels automatisés selon les spécifications.	15
6. Contrôler la qualité du service.	6.1 Application rigoureuse des plans de tests.		
7. Participer au déploiement du service.	7.1 Application correcte de la procédure de migration du service sur le serveur. 7.2 Application rigoureuse des mesures de sécurité.	Respecter la procédure de l'hébergeur Web pour la migration du service sur le serveur. Charger la base de données chez l'hébergeur. Configuration du service en mode Production vs Développement Configurer les mots de passe et les accès à la base de données et à l'hébergement Web.	
8. Rédiger la documentation.	8.1 Détermination correcte de l'information à rédiger.	Concevoir la documentation en respectant les normes	
00SU Effectuer le développement d'applications Web transactionnelles			
6. Programmer la logique applicative du côté client.	6.1 Manipulation adéquate des objets du modèle DOM. 6.2 Programmation appropriée d'appels asynchrones. 6.3 Programmation correcte des interactions entre l'interface Web et l'utilisatrice ou l'utilisateur.	Utiliser le Framework MVC côté client Vue.js Manipuler les objets du DOM avec Vue.js	40

	6.4 Utilisation systématique des techniques de validation de données des formulaires Web.	Programmer des appels asynchrones et récupérer les résultats. Adapter les interfaces graphiques en fonction des résultats des appels asynchrones. Mettre à profit la liaison des données bidirectionnelles. Utiliser les particularités de Vue.js pour valider les formulaires.	
Caractéristiques générales • Qualité du code • Documentation du code • Respect des principes enseigné dans le cours • Qualité des interfaces			10
Pénalités • Retard. • Remise incorrecte du travail. • Français dans les interfaces • D'autres éléments selon le besoin.			