420-15D-FX TP3 A23

Travail pratique #3 : GareNoTicket

Pondération : 40%
Travail à faire individuellement
Date de remise : le **Jeudi 7 décembre 2023** avant minuit

1. Contexte

Ce travail pratique porte sur le framework **Vue.js** et 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.
La partie *frontend* en Vue.js fonctionnera en local et n'a pas besoin d'être hébergée.
L'API devra respecter un jeu de tests automatisés.

Les technologies suivantes sont utilisées : Vue.js, Node.js, Express.js, MongoDB et Mongoose.

Vous devrez utiliser un gestionnaire de version (Git) pour gérer votre projet. Vous devez faire des commits régulièrement et utiliser des messages de commit explicites. Vous pouvez faire un seul dépôt pour le *frontend* et le *backend* ou deux dépôts séparés.

2. Description

Le stationnement autour du Cegep est limité à une heure et la plupart des étudiants doivent déplacer leur voiture régulièrement pour éviter de recevoir une contravention.

Un étudiant a eu l'idée de créer une application qui lui permet de payer un valet pour déplacer sa voiture toutes les heures.

L'objectif de ce travail pratique est donc de développer une application qui permet au valet de gérer les déplacements des voitures et aux étudiants (utilisateurs) de localiser leur voiture et de payer le valet.

Cette application sera un prototype et toutes les fonctionnalités d'une vraie application ne seront pas implémentées. Par exemple, il n'y aura pas de système de paiement réel.

3. Technologies à utiliser

Liste des technologies à utiliser pour réaliser l'application:

- Une partie du *backend* Node.js avec Express.js est fourni et vous devez le compéter.
- Le *frontend* doit utiliser **Vue.js** version 3.

- Le *frontend* et le *backend* doivent-être dans deux projets différents et communiquer via des requêtes HTTP (utilisant le modèle **REST**).
- Vous pouvez utiliser un *framework* CSS si vous le souhaitez.
- Leaflet 1.9 pour la carte.

4. Fonctionnalités

Il faut prévoir un menu pour pouvoir accéder aux différentes fonctionnalités de l'application suivant le rôle de l'utilisateur (utilisateur ou valet) et suivant si l'utilisateur est connecté ou pas.

4.1 Authentification

L'utilisateur doit pouvoir s'inscrire sur le site en saisissant un nom, un courriel, un mot de passe et la confirmation du mot de passe.

4.1.1 Inscription

Un formulaire avec les informations suivantes :

- un nom
- un courriel
- un mot de passe
- confirmation du mot de passe

Si la création est effectuée sans erreur, on redirige vers la page pour se connecter.

S'il y a une erreur, le formulaire doit toujours être affiché.

Le valet est un utilisateur comme les autres mais il faudra mettre manuellement le champ **isValet** à **true** dans la base de données.

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

Pensez à valider les informations des formulaires "**côté frontend et côté backend**".

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.

La connexion redirige vers la page "Ma place" pour les utilisateurs et vers la page "Valet" pour les valets.

S'il y a une erreur, le formulaire doit toujours être affiché.

L'utilisateur doit également pouvoir se déconnecter du site.

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

4.2 Gestion du profil

4.2.1 Utilisateurs

Les utilisateurs doivent pouvoir visualiser et modifier leur profil. Cela inclut des informations telles que le nom d'utilisateur, le courriel, et les informations de la voiture (marque, modèle, couleur, plaque d'immatriculation).

L'utilisateur doit pouvoir supprimer son compte. Une confirmation doit être demandée avant de supprimer le compte.

Un utilisateur ne doit pas pouvoir modifier son profil lorsque sa voiture est stationnée.

4.2.2 Valet

Le valet doit pouvoir visualiser et modifier son profil. Cela inclut des informations telles que le nom d'utilisateur, le courriel et le tarif appliqué pour chaque déplacement de voiture.

4.3 Gestion des voitures

4.3.1 Utilisateurs (page *Ma place*)

Les utilisateurs doivent pouvoir stationner leur voiture, visualiser sa position actuelle sur une carte et la récupérer.

Si la voiture n'est pas stationnée, la carte doit afficher la localisation de l'utilisateur. L'utilisateur doit pouvoir déplacer le marqueur sur la carte pour indiquer la position de sa voiture si jamais ce n'est pas assez précis.
Un bouton "Je laisse ma voiture" doit afficher le texte suivant:

"Veuillez vérifier que votre voiture est bien stationnée à l'endroit indiqué sur la carte, ou déplacer le marqueur sur la position de votre voiture."

Puis un bouton "Je confirme" permet d'enregistrer la position de la voiture et son heure de stationnement dans la base de données.

Le marqueur ne peut plus être déplacé quand la voiture est stationnée.

Quand l'utilisateur a récupéré sa voiture, il clique sur un bouton "J'ai récupéré ma voiture". Cela permet de signaler que la voiture n'est plus stationnée.

Les boutons ne doivent pas être actifs s'ils ne sont pas nécessaires.

Un bouton pour centrer la carte sur la position de la voiture doit être affiché quand la voiture est stationnée.

Si un utilisateur cherche sa voiture pendant qu'elle est en cours de déplacement par un valet, un message doit s'afficher "Votre voiture est en cours de déplacement". Il ne peut pas récupérer sa voiture tant que le valet ne l'a pas stationnée.

Si l'utilisateur a un solde à payer supérieur à 20$, il ne peut plus utiliser le service tant qu'il n'a pas payé. Un message doit s'afficher "Vous devez payer votre solde avant de pouvoir utiliser le service".

Il peut bien sûr visualiser la position de sa voiture et la récupérer mais il ne peut plus la stationner.

4.3.2 Valet (page *Valet*)

Le valet doit pouvoir visualiser sa position (marqueur rouge) et la position de toutes les voitures stationnées sur une carte (marqueurs bleus).

Un bouton permet de centrer la carte sur la position du valet.

Un tableau doit afficher toutes les voitures stationnées avec les informations suivantes:

- nom du propriétaire de la voiture
- marque
- modèle
- plaque d'immatriculation
- couleur
- temps restant avant la fin du stationnement
- un bouton pour centrer la carte sur la position de la voiture
- un bouton pour ouvrir la page qui permet de déplacer la voiture (voir ci-dessous)

Le temps restant doit être affiché en secondes et **doit être mis à jour toutes les secondes**.

Page qui permet de déplacer la voiture

La carte doit afficher la position de la voiture (maqueur bleu) et la position du valet (marqueur rouge).

Le valet doit pouvoir déplacer une voiture stationnée en cliquant sur un bouton "Je déplace la voiture". Cela permet de signaler que la voiture n'est plus stationnée et qu'elle est en cours de déplacement.

Le marqueur bleu disparait de la carte quand la voiture n'est plus stationnée.

Le marqueur rouge peut être déplacé sur la carte pour indiquer la position de la voiture si jamais la géolocalisation n'est pas assez précise.

Un bouton "Validation du stationnement" permet de signaler que la voiture est stationnée à l'endroit indiqué sur la carte. Cela permet de mettre à jour la position de la voiture dans la base de données et son heure de stationnement.

Les boutons ne doivent pas être actifs s'ils ne sont pas nécessaires.

Chaque déplacement est facturé au propriétaire de la voiture. Le tarif est défini par le valet dans son profil.

4.3.3 Calcul du temps restant

Le temps restant est calculé en fonction de l'heure de stationnement. Le temps de stationnement est limité à **une heure**.

Cependant, le stationnement est gratuit entre 12h00 et 13h30. Si une voiture est stationnée entre 11h00 et 12h30, le temps restant doit être calculé pour un déplacement de la voiture à 13h30.

Le stationnement est également gratuit entre 17h00 et 9h00. Si une voiture est stationnée après 16h00, le temps restant n'est pas calculé car on suppose que l'étudiant va récupérer sa voiture avant le lendemain 9h00. Dans ce cas, le tableau des voitures stationnées affiche "Demain" pour la colonne temps restant.

Si la voiture est stationnée avant 9h00, le temps restant est calculé pour un déplacement de la voiture à 10h00.

4.4 Historique des transactions (page *Transaction*)

Les utilisateurs doivent pouvoir visualiser l'historique de leurs transactions, y compris le coût et le statut de paiement de chaque transaction.

La page comporte trois parties:

- le montant à payer avec un bouton pour payer. Le système de paiement n'est pas implémenté. Un clic sur le bouton paye la somme et la met à zéro. Les informations de la page doivent être mises à jour automatiquement. Le montant à payer est la somme de tous les montants des déplacements non payés. Le statut des déplacements payés doit être mis à jour automatiquement.
- une liste "historique des factures" avec les informations suivantes:
  - date
  - montant payé
- Un liste "historique des déplacements" avec les informations suivantes:
  - date
  - montant à payer
  - statut (payé ou non)

5. Validation des données

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

- Les messages d’erreur doivent être proche des champs en erreur.
- Voir les tableaux suivants pour les contraintes spécifiques de validation.

5.1. Utilisateur

Champ	Contrainte(s)
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 du mot de passe	Égal au mot de passe Requis
isValet	Booléen False par défaut

5.2 Voitures

Champ	Contrainte(s)
Marque	Entre 1 et 50 caractères Requis
Modèle	Entre 1 et 50 caractères Requis

Champ	Contrainte(s)
Couleur	Entre 3 et 50 caractères Requis
Plaque d'immatriculation	6 caractères Requis

5.2. Autres validations

Champ	Contrainte(s)
Tarif et prix	Entier positif, enregistré en cents

6. API

L'api doit être réalisée avec **Express.js**.

Vous devez compléter le code qui est fourni. Vous pouvez modifier le code existant sauf pour 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'authentification doit être gérée avec un jeton JWT.

Vous devez vous assurer qu'un utilisateur ne peut pas effectuer des actions sur des ressources qui ne lui appartiennent pas.

Vous devez respecter les notions vues en classe :

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

7. Frontend

La partie *front* doit être codée avec le *framework* Vue.js et respecter les bonnes pratiques vues en cours.

Vous devez utiliser **au moins** un *component*.

Vous pouvez utiliser *Pinia* (ou *Vuex*) pour gérer l'état de l'application.

Pour chaque action de l’utilisateur ou du valet qui le nécessite (modifier le profil, connexion / déconnexion, stationner ou récupérer la voiture, etc.), il faut faire apparaître un **message de confirmation ou d’erreur**.

Si un utilisateur non connecté tente d'accéder à une page qui nécessite une authentification, il doit être redirigé vers la page pour se connecter.

- Vous devez afficher une page 404 si un utilisateur authentifié tente d'accéder à une ressource qui n'existe pas.
- Vous devez respecter les bonnes pratiques vues en cours.
- 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 (Firefox...).

8. Tests

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

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@gare.ca, user2@gare.ca...)
- 1 valet (valet@gare.ca).
- au moins 3 factures payées par utilisateur.
- au moins 3 déplacements non payés par utilisateur.
- 1 voiture en cours de déplacement.
- 1 voiture stationnée à 9h00.
- 1 voiture stationnée à 11h45.
- 1 voiture stationnée à 16h30.
- 1 voiture non stationnée.

Tous les mots de passe des utilisateurs et du valet doivent être **123456**.

9. HATEOAS

Le principe HATEOAS doit être implémenté pour la route get('/user/').

Il faut ajouter dans la réponse le lien pour la route suivante :

- delete('/user/')

L'application vue.js doit utiliser ce lien quand elle en a besoin.

10. 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.

11. Hébergement

L'API doit être hébergée sur un serveur distant.

12. 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.

13. À remettre

- Votre projet doit être remis sur Léa au format zip.
- Il doit comporter dans des dossiers séparés :
  - le code de l'**API**
  - le code **Vue.js**
  - un export des requêtes *Postman* au format JSON pour **les tests**
  - un fichier d'export de la **base de données** au format Mondodb (si jamais le *seed* ne fonctionne pas)
- Dans le dossier zip, vous devez ajouter une page de présentation du travail avec
  - **L'url de la documentation** *Postman*.
  - **Url du dépôt Git** sur Gitlab.
  - Ajoutez des informations que vous jugerez utile de fournir.
- L'API doit être **hébergée sur un serveur distant** et accessible pendant deux semaines après la date de la remise

14. Annexes

14.1 Base de données

Vous devez ajouter les champs `updatedAt` et `createdAt` à toutes les collections.

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

14.1.1. Voiture

- marque
- modele
- couleur
- plaque
- valet: id du valet qui déplace la voiture
- isParked: true si la voiture est stationnée
- isMoving: true si la voiture est en cours de déplacement
- latitude
- longitude
- timeToLeave: heure à laquelle la voiture doit être déplacée

14.1.2. Utilisateur

- courriel
- mot de passe
- nom
- isValet: true si l'utilisateur est un valet
- price: prix du déplacement si l'utilisateur est un valet
- voiture: id de la voiture de l'utilisateur

14.1.3 historique

- price: prix du déplacement
- isPaid: true si le déplacement est payé
- userId: id de l'utilisateur
- valetId: id du valet

14.1.4 facture

- price: prix total de la facture
- userId: id de l'utilisateur

14.2 Barème de correction

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

- Git / Base de données / Architecture / Seeds: **10%**
- API: **20%**
- Vue.js: **40%**
- Tests / Documentation Postman / Hébergement / HATEOAS: **20%**
- 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.

14.3 Extension location-guard

Il est conseillé d'utiliser une extension pour votre navigateur telle que location-guard pour simuler différents emplacements géographiques.

Cela peut être utile pour tester comment votre application réagit à différents emplacements, sans avoir à vous déplacer physiquement.

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