

420-15D-FX TP3 A24 partie2

Travail pratique #3 : Partie 2 Vue.js - Niyoro

Pondération : 20%

Travail à faire individuellement

Date de remise : le **Vendredi 13 décembre 2024** avant minuit

1. Contexte

Ce travail pratique porte sur le framework **Vue.js**. Vous devez réaliser la partie *frontend* de l'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.).

Cette application utilise l'API que vous avez développée dans le travail pratique 3 partie 1. Vous aurez des petits ajustements à faire sur l'API pour qu'elle fonctionne correctement avec l'application Vue.js.

2. Description

L'objectif de ce travail pratique est de permettre aux utilisateurs de visualiser, ajouter, modifier et supprimer des items. Les utilisateurs peuvent également réagir à un item en choisissant parmi les réactions suivantes : J'aime 👍, Informatif 💡, Drôle 😂, Inapproprié 🚫.

Vous pouvez utiliser ces émojis pour les réactions ou en choisir d'autres. L'utilisateur ne peut réagir qu'une seule fois à un item.

Les utilisateurs peuvent visualiser les items des autres utilisateurs. Les items privés ne sont visibles que par l'utilisateur qui les a créés.

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:

- Vue.js version 3
- Vue Router
- Pinia
- Leaflet (pour les cartes)
- Highlight.js ou une autre librairie de votre choix (pour la coloration syntaxique)

Vous devez utiliser des **composants** Vue.js et plusieurs *store* Pinia pour structurer votre application.

4. Fonctionnalités

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

L'avatar et le pseudo de l'utilisateur connecté doivent être affichés dans le menu. L'avatar doit être un lien vers la page de profil de l'utilisateur.

4.1. Authentification

4.1.1. Inscription (route `/signup`)

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.

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.

Le formulaire d'inscription doit afficher les erreurs de validation pour chaque champs à proximité de celui-ci avec un style bien visible.

4.2.2. Connexion (route `/login`)

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

Un utilisateur déconnecté peut voir les items mais ne peut faire aucune action.

Le formulaire de connexion doit afficher les erreurs de validation pour chaque champs à proximité de celui-ci avec un style bien visible.

4.3. Page d'accueil (route `/`)



La page d'accueil doit afficher les items sous forme de liste classée par date de création décroissante (les plus récents en premier).

Les items épinglés doivent apparaître en premier dans la liste. Ils doivent également être triés par date de création décroissante.





Un sélecteur en haut de la page doit permettre :

- s'il est coché, d'afficher seulement les items (publics et privés) de l'utilisateur **connecté**.
- s'il est décoché, d'afficher tous les items publics de tous les utilisateurs.

Chaque item doit être dans une carte avec les informations suivantes :

- titre
- url (sur le titre si elle existe).
- contenu (utilisation de la coloration syntaxique) s'il existe.
- privé ou public (icône cadenas  ou globe .
- un lien détail (permalien).
- tags (chaque tag doit être un lien pour afficher les items associés à ce tag).
- créé par (pseudo de l'utilisateur qui a créé l'item). Ce champ doit être un lien vers la page de profil de l'utilisateur.
- date et heure de création au format `jj mois aaaa hh:mm` (ex: 06 novembre 2024 à 12h30).

Si l'utilisateur est le créateur de l'item, il doit pouvoir pour chaque item :

- épingler ( si épinglé ou  si non épinglé)
- modifier 
- supprimer 

Un clic sur l'icône de la punaise doit épingler ou désépingler l'item en temps réel. Le changement d'icône doit être visible immédiatement.

Cependant, quand l'utilisateur épingle un item, celui-ci n'a pas besoin de se positionner en haut de la liste en temps réel. Il doit être épinglé lors du rechargement de la page.

4.3.1 Filtrage des items (route `/:tag`)

Quand l'utilisateur clique sur un tag, la liste des items doit être filtrée pour afficher seulement les items associés à ce tag.

Utilisez la même page que pour la page d'accueil.

La route doit être mise à jour pour afficher le tag sélectionné dans l'URL. (ex: `/tag1`)

4.4. page détail d'un item (route `/item/:id`)

La page détail d'un item doit afficher les mêmes informations que la page d'accueil avec ces fonctionnalités supplémentaires :

- les réactions associées à l'item avec le nombre de réactions pour chaque type de réaction doivent être affichées.
- l'utilisateur connecté doit pouvoir ajouter une réaction à l'item ou supprimer sa réaction. Il ne peut pas réagir plusieurs fois à un même item.
- le nombre de réactions pour chaque type de réaction doit être mis à jour en temps réel.
- affichage d'une carte si les coordonnées géographiques sont renseignées avec un marqueur à la position des coordonnées.

4.5. Page ajout d'un item (route `/add`)

L'utilisateur doit pouvoir ajouter un item en saisissant les informations suivantes :

- titre
- url
- contenu
- tags (séparés par un espace)
- latitude
- longitude
- privé ou public

Un seul champ de saisie pour les tags est suffisant. Les tags doivent être séparés par un espace. Ils seront envoyés à l'API sous forme de tableau.

Les tags seront ajoutés à la liste des tags existants s'ils n'existent pas déjà dans la base de données.

Le formulaire doit afficher les erreurs de validation pour chaque champs à proximité de celui-ci avec un style bien visible.

4.6. Page Tags (Seulement pour les administrateurs) (route `/tags`)

L'administrateur doit pouvoir visualiser la liste tags existants. Il peut également supprimer ou modifier un tag à partir de cette liste.

Si un tag est supprimé, il doit être retiré de tous les items qui lui sont associés (déjà fait côté API en principe).

L'administrateur ne peut pas ajouter de tag.

4.7. Page Profil (route `/profile/:id?`)

Utiliser la même page pour le profil de l'utilisateur connecté et pour le profil des autres utilisateurs.

La page de profil doit afficher les informations suivantes :

- avatar
- pseudo
- nom et prénom
- date d'inscription

On ne peut rien mettre à jour sur cette page.

5. Validation des données

Les validations des données sont les mêmes que celles de l'API. Vous devez les reproduire côté *frontend* pour éviter les erreurs de validation côté *backend*.

Cependant, si une erreur de validation est retournée par l'API, elle doit être affichée de façon claire et visible à l'utilisateur en haut du formulaire.

6. Hébergement

L'API doit être hébergée sur un serveur distant et accessible en ligne pendant au moins deux semaines après la remise du travail.

L'application Vue.js ne doit pas être hébergée sur un serveur distant.

7. 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. Utilisez des fonctions pour éviter la duplication de code.
- Portez une attention particulière à la qualité de l'interface.

8. À remettre

- Votre projet doit être remis sur Léa au format zip.
- Il doit comporter dans des dossiers séparés :
 - le code de l'**application Vue.js** avec un fichier `readme.md` avec les informations suivantes :

- **Url du dépôt Git** sur Gitlab.
- Ajoutez des informations que vous jugerez utile de fournir.
- Le code de l'API si jamais l'hébergement n'est pas fonctionnel au moment de la correction.
- N'oubliez pas de supprimer les dossiers `node_modules`.

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