**CRESPIN Mathieu, DIAW Abdoul, BESNARD Clément**

TP 4 : Questions

Question 1 :

Il ne faut pas commettre de informations d'identification ou de credentials sur git car cela peut entraîner une exposition accidentelle de ces informations sensibles. Si vous commettez des credentials dans un dépôt git et que vous le poussez sur un serveur public (comme GitHub), n'importe qui pourra accéder à ces informations et les utiliser à mauvais escient. Cela peut entraîner des conséquences graves, telles que des violations de données ou des accès non autorisés à des systèmes sensibles. Il est donc crucial de protéger les credentials en évitant de les committeer dans les dépôts git.

Question 2 :

Il est important d'avoir des configurations différentes en fonction de l'environnement pour des raisons de sécurité et de personnalisation. Par exemple, une application web peut utiliser des bases de données différentes pour le développement, les tests et la production. En utilisant des configurations différentes pour chaque environnement, on peut s'assurer que l'application fonctionne correctement et que les paramètres sont adaptés aux exigences spécifiques de chaque environnement.

Question 3 :

Une solution qui fonctionne bien mais qui présente des problèmes de maintenabilité peut être difficile à gérer à long terme. Les problèmes peuvent inclure une complexité accrue, une documentation insuffisante, une dépendance excessive à l'égard d'une seule personne ou d'un petit groupe de personnes, ou une rigidité qui rend difficile l'évolution de la solution. Pour résoudre ces problèmes, il est important de simplifier la conception, documenter soigneusement le code et les processus, encourager la collaboration et la diversité des compétences, et rendre la solution aussi flexible et évolutive que possible.

Question 5 :

Il existe plusieurs stratégies de gestion d'état disponibles pour les applications Vue.js, chacune avec ses avantages et ses inconvénients. Voici un tableau de comparaison entre certaines des stratégies les plus courantes :

| Stratégie de gestion d'état | Avantages | Inconvénients |
| --- | --- | --- |
| État local (composant) | Simple et facile à mettre en œuvre | Pas adapté aux applications complexes ou à grande échelle |
| État partagé (événements et props) | Flexible et réutilisable | Peut être difficile à déboguer et à maintenir |
| État réactif (Vue.observable) | Réactif et facile à mettre en œuvre | Pas adapté aux applications complexes ou à grande échelle |
| État global (Vuex) | Centralisé, réutilisable et facile à déboguer | Plus complexe à mettre en œuvre et à maintenir |
| État externe (API ou base de données) | Persistant et facile à partager entre applications | Latence et dépendance à l'égard d'une source externe |

En général, les stratégies d'état local et réactif sont les plus simples à mettre en œuvre, mais elles ne sont pas adaptées aux applications complexes ou à grande échelle. Les stratégies d'état partagé et global offrent plus de flexibilité et de réutilisabilité, mais elles peuvent être plus difficiles à déboguer et à maintenir. Enfin, les stratégies d'état externe offrent une persistance et une facilité de partage entre applications, mais elles peuvent être soumises à une latence et à une dépendance à l'égard d'une source externe.

Il est important de choisir la stratégie de gestion d'état la plus appropriée en fonction de la complexité et de la taille de l'application, ainsi que des exigences en matière de persistance et de partage des données. Dans certains cas, il peut être judicieux de combiner plusieurs stratégies pour répondre aux besoins de l'application. Par exemple, on peut utiliser l'état local pour gérer les données spécifiques à un composant, l'état partagé pour transmettre des données entre composants, et l'état global pour gérer les données partagées entre différentes parties de l'application.

Question 6 :

Je ne recommanderais pas de gérer exclusivement l'état avec des stores et de ne pas dépendre des props et provide, car cela peut introduire une complexité inutile et une dépendance excessive à l'égard d'une seule source de données. Je recommanderais plutôt d'utiliser une combinaison de stratégies de gestion d'état, en fonction des besoins spécifiques de l'application. Les stores peuvent être utiles pour gérer les données partagées entre différentes parties de l'application, tandis que les props et provide peuvent être utilisés pour gérer les données spécifiques à un composant. En utilisant une approche flexible et adaptable, on peut s'assurer que l'application reste facile à comprendre, à déboguer et à maintenir.

Question 7 :

Il n'y a pas de différence de performance significative entre <router-link to="/conversations">Conversations</router-link> et <a href="/conversations">Conversation</a>. Cependant, il y a une différence de fonctionnalité et de comportement entre les deux.

<router-link> est un composant Vue Router qui crée un lien cliquable qui active une route donnée lorsqu'il est cliqué. Il offre des fonctionnalités supplémentaires telles que la mise en surbrillance active du lien lorsque la route est active, et la préservation de l'historique de navigation lors de la navigation entre les routes.

En revanche, <a> est un élément HTML standard qui crée un lien cliquable vers une URL donnée. Il ne fournit pas les fonctionnalités supplémentaires offertes par <router-link>, telles que la mise en surbrillance active et la préservation de l'historique de navigation.

Par conséquent, je recommanderais d'utiliser <router-link> à la place de <a> pour les liens de navigation dans une application Vue Router, car il offre des fonctionnalités supplémentaires et un comportement plus prévisible. Cependant, il n'y a pas de différence de performance significative entre les deux, donc le choix dépendra des besoins spécifiques de l'application.