2023

RÉCAPITULATIF DE LA FORMATION : REACT QUERY

WORKFLOW ENGINE PROJECT

Réalisé par :KHOUDRAJI OUIAM

SOMMAIRE

React Query : est une bibliothèque JavaScript qui facilite la gestion des requêtes et de la mise en cache des données dans les applications React. Elle offre une approche déclarative et intuitive pour gérer l'état des données côté client, avec un support intégré pour les requêtes asynchrones, la mise en cache, la pagination et plus encore.

1-Instalation de React Query :

- Utilisez la commande **npm install react-query** ou **yarn add react-query** pour installer React Query dans votre projet.
- Importez React Query dans votre fichier principal :
  import { QueryClientProvider, QueryClient } from 'react-query'

2-Configuration :

Initialisez React Query dans votre application en utilisant le composant <QueryClientProvider> et le <QueryClient> :

Créez une instance de QueryClient :

```
const queryClient = new QueryClient().
```

Enveloppez votre application avec le QueryClientProvider pour fournir l'instance de QueryClient à tous les composants React :

```
ReactDOM.render(
  <QueryClientProvider client={queryClient}>
    <App />
  </QueryClientProvider>,
  document.getElementById('root')
);
```

3-Requêtes de base :

Le hook useQuery est utilisé pour effectuer des requêtes côté client et gérer les résultats de manière réactive.

Etapes à suivre:

Importez useQuery depuis React Query :

```
import { useQuery } from 'react-query'.
```

Utilisez useQuery dans votre composant pour effectuer une requête :

```
const { data, isLoading, isError } = useQuery('myData', () =>
  fetch('https://api.example.com/data').then((res) => res.json())
);
```

- *Le premier argument passé à useQuery est une clé unique pour identifier la requête. Dans l'exemple, la clé est 'myData'.*
- *Le deuxième argument est une fonction de rappel asynchrone qui effectue la requête. Dans cet exemple, nous utilisons fetch pour récupérer les données depuis une API.*
- *Le hook retourne un objet contenant les propriétés suivantes :*
  - ***data:** les données renvoyées par la requête.*
  - ***isLoading:** un booléen indiquant si la requête est en cours de chargement.*
  - ***isError:** un booléen indiquant si la requête a rencontré une erreur.*
- *React Query gère automatiquement la mise en cache des résultats de la requête. Lorsque vous appelez useQuery avec la même clé de requête, il vérifie s'il a déjà les données en cache et les renvoie immédiatement. Il met également à jour automatiquement les données en cache lorsque la requête est réussie.*

4-Mise en cache :

Par défaut, React Query met en cache automatiquement les résultats des requêtes.

Utilisez le hook useQuery avec la même clé de requête pour récupérer les données mises en cache :

```
const { data } = useQuery('myData', () => fetchCachedData());
```

5-Gestion des mutations :

Le hook useMutation est utilisé pour effectuer des mutations côté client, telles que la création, la mise à jour ou la suppression de données.

Etapes à suivre:

Importez useMutation depuis React Query :

```
import { useMutation } from 'react-query'.
```

Utilisez useMutation pour effectuer des mutations côté client :

```
const mutation = useMutation((data) =>
  fetch('https://api.example.com/data', {
    method: 'POST',
    body: JSON.stringify(data),
  })
);
```

```
// Appel de la mutation
mutation.mutate({ /* données à envoyer */ });
```

- *Le premier argument passé à useMutation est une fonction qui effectue la mutation côté client. Dans cet exemple, nous utilisons fetch pour envoyer les données à une API via une requête POST.*
- *Le hook retourne un objet mutation qui contient diverses propriétés et méthodes utiles pour gérer la mutation :*
    - ***mutate:** une fonction qui déclenche la mutation. Vous pouvez lui passer les données à envoyer à la mutation.*
    - ***isLoading:** un booléen indiquant si la mutation est en cours d'exécution.*
    - ***isError:** un booléen indiquant si la mutation a rencontré une erreur.*
    - ***data:** les données renvoyées par la mutation en cas de succès.*
- *React Query gère automatiquement les états de chargement, d'erreur et de succès pour vous, ce qui facilite la gestion des mutations côté client.*