Communication avec une API

fetch, axios et React Query

- fetch et axios
- React Query (TanStack Query)

fetch et axios

Appels API : fetch et axios

fetch natif

```
async function getUsers() {
  const res = await fetch('/api/users', {
    method: 'GET',
    headers: { 'Content-Type': 'application/json', 'Authorization': `Bearer ${token}` }
  });
  if (!res.ok) throw new Error(`HTTP ${res.status}`);
  return res.json();
}

async function createUser(data) {
  const res = await fetch('/api/users', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  });
  if (!res.ok) throw new Error(`HTTP ${res.status}`);
  return res.json();
}
```

axios

```
npm install axios
```

```
import axios from 'axios';

// Instance configurée
const api = axios.create({
  baseURL: 'https://api.example.com',
  timeout: 10000,
```

```
  headers: { 'Content-Type': 'application/json' }
});

// Intercepteur pour le token
api.interceptors.request.use(config => {
  const token = localStorage.getItem('token');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});

// Intercepteur pour les erreurs
api.interceptors.response.use(
  response => response.data,
  error => {
    if (error.response?.status === 401) logout();
    return Promise.reject(error);
  }
);

// Utilisation
const users = await api.get('/users');
const user = await api.post('/users', { nom: "Hugo" });
await api.put('/users/1', { nom: "Hugo P." });
await api.delete('/users/1');
```

React Query (TanStack Query)

React Query (TanStack Query)

La meilleure solution pour gérer l'état serveur : cache automatique, refetch, états de chargement/erreur, pagination, optimistic updates...

```
npm install @tanstack/react-query
```

Configuration

```
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';

const queryClient = new QueryClient({
  defaultOptions: {
    queries: { staleTime: 1000 * 60 * 5 }, // 5 minutes
  },
});

<QueryClientProvider client={queryClient}>
  <App />
</QueryClientProvider>
```

useQuery — lecture de données

```
import { useQuery } from '@tanstack/react-query';

function Users() {
  const { data, isLoading, isError, error, refetch } = useQuery({
    queryKey: ['users'],          // Clé de cache unique
    queryFn: () => api.get('/users'),
    staleTime: 1000 * 60,         // 1 minute avant refetch
  });

  if (isLoading) return <Spinner />;
```

```
  if (isError) return <p>Erreur : {error.message}</p>;

  return <ul>{data.map(u => <li key={u.id}>{u.nom}</li>)}</ul>;
}

// Avec paramètre (refetch automatique quand userId change)
const { data: user } = useQuery({
  queryKey: ['user', userId],
  queryFn: () => api.get(`/users/${userId}`),
  enabled: !!userId, // N'exécute pas si userId est null
});
```

useMutation — écriture de données

```
import { useMutation, useQueryClient } from '@tanstack/react-query';

function CreateUser() {
  const queryClient = useQueryClient();

  const mutation = useMutation({
    mutationFn: (newUser) => api.post('/users', newUser),
    onSuccess: () => {
      // Invalider le cache pour forcer un refetch
      queryClient.invalidateQueries({ queryKey: ['users'] });
    },
    onError: (error) => {
      console.error("Erreur création :", error);
    },
  });

  return (
    <button
      onClick={() => mutation.mutate({ nom: "Hugo" })}
      disabled={mutation.isPending}
    >
      {mutation.isPending ? "Création..." : "Créer"}
    </button>
  );
}
```