React Router DOM - Gestion des Routes

Introduction : Pourquoi le Routage ?

Le Problème des Applications Web Modernes

Dans les années 2000, chaque page web était un fichier HTML séparé :

- index.html → Page d'accueil
- contact.html → Page contact
- about.html → Page à propos

Changement de page = Nouvelle requête serveur + Rechargement complet

L'Arrivée des Single Page Applications (SPA)

Avec React, on crée des **applications monopages** :

- Une seule page HTML (index.html)
- Le JavaScript change le contenu dynamiquement
- Pas de rechargement de page
- Expérience utilisateur fluide

Problème : Comment gérer différentes "pages" sans rechargement ?

PARTIE 1 : Installation et Configuration

1.1 Installation

Avec npm

```
npm install react-router-dom
```

Avec yarn

```
yarn add react-router-dom
```

1.2 Les Deux Versions

// Version 6 (moderne - recommandée)

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
```

// Version 5 (ancienne)

```
import { BrowserRouter, Switch, Route, Link } from 'react-router-dom';
```

Dans ce cours, nous utiliserons React Router v6 (la dernière version).

PARTIE 2 : Les Concepts de Base

2.1 BrowserRouter - Le Conteneur Principal

BrowserRouter : Fournit le contexte de routage à toute l'application. Doit englober toute l'app.

```
// index.js ou App.js
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>,
  document.getElementById('root')
);
```

2.2 Routes et Route - Définir les Chemins

Routes : Conteneur qui regroupe toutes les routes
Route : Définit un chemin spécifique et le composant à afficher

```
import { Routes, Route } from 'react-router-dom';

function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </div>
  );
}
```

2.3 Link - Navigation sans Rechargement

Link : Similaire à <a href> mais sans rechargement de page. Maintient l'état de l'application.

```jsx
import { Link } from 'react-router-dom';

function Navigation() {
  return (
    <nav>
      <Link to="/">Accueil</Link>
      <Link to="/about">À propos</Link>
      <Link to="/contact">Contact</Link>
    </nav>
  );
}
```

PARTIE 3 : Exemple Pratique

3.1 Structure de Base

On import les modules necessaires

```
// App.js
import { Routes, Route } from 'react-router-dom';
import Navigation from './components/Navigation';
import Home from './pages/Home';
import About from './pages/About';
import Contact from './pages/Contact';
import User from './pages/User';
import NotFound from './pages/NotFound';

function App() {
```

3.1 Structure de Base

```
function App() {
  return (
    <div className="App">
      <Navigation />

      <main>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/contact" element={<Contact />} />
          <Route path="/user/:id" element={<User />} />
          <Route path="*" element={<NotFound />} />
        </Routes>
      </main>
    </div>
  );
}
```

3.2 Composant Navigation

```jsx
// components/Navigation.js
import { Link, NavLink } from 'react-router-dom';

function Navigation() {
  return (
    <nav className="navigation">
      <div className="logo">
        <Link to="/">MonSite</Link>
      </div>

      <ul className="nav-links">
        <li>
          <NavLink
            to="/"
            className={({ isActive }) =>
              isActive ? 'active' : ''
            }
```

3.2 Composant Navigation

```
      >
        Accueil
      </NavLink>
    </li>
    <li>
      <NavLink
        to="/about"
        className={({ isActive }) =>
          isActive ? 'active' : ''
        }
      >
        À propos
      </NavLink>
    </li>
    <li>
      <NavLink
        to="/contact"
        className={({ isActive }) =>
```

3.2 Composant Navigation

```
            </NavLink>
          </li>
          <li>
            <NavLink
              to="/contact"
              className={({ isActive }) =>
                isActive ? 'active' : ''
              }
            >
              Contact
            </NavLink>
          </li>
        </ul>
      </nav>
    );
}
```

3.3 Différence Link vs NavLink

Link : Navigation simple
NavLink : Navigation avec styles actifs

```
// Avec NavLink, on peut savoir si le lien est actif
<NavLink
  to="/about"
  style={({ isActive }) => ({
    color: isActive ? 'red' : 'blue'
  })}
>
  À propos
</NavLink>
```

PARTIE 4 : Routes Dynamiques et Paramètres

4.1 Routes avec Paramètres

```
// Dans App.js
<Routes>
  <Route path="/users/:userId" element={<UserProfile />} />
  <Route path="/products/:productId" element={<ProductDetail />} />
  <Route path="/category/:categoryId/:subcategoryId" element={<Category />} />
</Routes>
```

Syntaxe : `:nomDuParamètre`

4.2 Récupérer les Paramètres

useParams() : Hook qui retourne un objet avec tous les paramètres de l'URL

```jsx
// pages/UserProfile.js
import { useParams } from 'react-router-dom';

function UserProfile() {
  const { userId } = useParams(); // Récupère le paramètre

  return (
    <div>
      <h1>Profil de l'utilisateur {userId}</h1>
      {/* Affiche "Profil de l'utilisateur 123" si URL = /users/123 */}
    </div>
  );
}
```

PARTIE 5 : Navigation Programmatique

5.1 useNavigate - Navigation dans le Code

```
import { useNavigate } from 'react-router-dom';

function LoginForm() {
  const navigate = useNavigate();
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();

    try {
      // Simuler une connexion
      const response = await loginAPI(username, password);
```

5.1 useNavigate - Navigation dans le Code

```
    if (response.success) {
      // Rediriger vers le dashboard
      navigate('/dashboard');

      // Ou avec remplacement (pas d'historique)
      // navigate('/dashboard', { replace: true });
    }
  } catch (error) {
    console.error('Erreur de connexion:', error);
  }
};
```

5.1 useNavigate - Navigation dans le Code

```
return (
  <form onSubmit={handleSubmit}>
    <input
      value={username}
      onChange={(e) => setUsername(e.target.value)}
      placeholder="Nom d'utilisateur"
    />
    <input
      type="password"
      value={password}
      onChange={(e) => setPassword(e.target.value)}
      placeholder="Mot de passe"
    />
    <button type="submit">Se connecter</button>

    <button
```

5.1 useNavigate - Navigation dans le Code

```
        placeholder="Mot de passe"
      />
      <button type="submit">Se connecter</button>

      <button
        type="button"
        onClick={() => navigate('/register')}
      >
        Créer un compte
      </button>
    </form>
  );
}
```

5.2 Navigation avec État

```
// Page d'envoi
const navigate = useNavigate();

const handlePayment = () => {
  const orderDetails = {
    items: cartItems,
    total: calculateTotal(),
    date: new Date()
  };

  navigate('/confirmation', {
    state: orderDetails
  });
};
```

5.2 Navigation avec État

```
// Page de confirmation
import { useLocation } from 'react-router-dom';

function Confirmation() {
  const location = useLocation();
  const orderDetails = location.state;

  return (
    <div>
      <h1>Commande confirmée !</h1>
      <p>Total: ${orderDetails.total}</p>
      {/* ... */}
    </div>
  );
}
```

5.3 Navigation Avancée

```
const navigate = useNavigate();

// Aller en arrière
const goBack = () => {
  navigate(-1); // -1 = retour, -2 = deux pages en arrière
};

// Aller en avant
const goForward = () => {
  navigate(1); // 1 = avant
};

// Naviguer avec options
const goHome = () => {
  navigate('/', {
    replace: true,     // Remplace l'entrée actuelle dans l'historique
    state: { from: 'login' } // Passe des données
  });
};
```

PARTIE 6 : Routes Imbriquées (Nested Routes)

6.1 Concept des Routes Imbriquées

Les routes imbriquées permettent d'afficher plusieurs composants en même temps :

- /dashboard → Layout + Dashboard
- /dashboard/users → Layout + Dashboard + Users
- /dashboard/settings → Layout + Dashboard + Settings

6.2 Exemple : Tableau de Bord

```jsx
// App.js
<Routes>
  <Route path="/dashboard" element={<DashboardLayout />}>
    <Route index element={<DashboardHome />} />
    <Route path="users" element={<Users />} />
    <Route path="settings" element={<Settings />} />
    <Route path="reports" element={<Reports />} />
  </Route>
</Routes>
```

6.3 Composant DashboardLayout

```jsx
// layouts/DashboardLayout.js
import { Outlet, Link } from 'react-router-dom';

function DashboardLayout() {
  return (
    <div className="dashboard">
      <aside className="sidebar">
        <h2>Tableau de bord</h2>
        <nav>
          <ul>
            <li><Link to="/dashboard">Accueil</Link></li>
            <li><Link to="/dashboard/users">Utilisateurs</Link></li>
            <li><Link to="/dashboard/settings">Paramètres</Link></li>
            <li><Link to="/dashboard/reports">Rapports</Link></li>
          </ul>
        </nav>
```

6.3 Composant DashboardLayout

```
            <li><Link to="/dashboard/users">Utilisateurs</Link></li>
            <li><Link to="/dashboard/settings">Paramètres</Link></li>
            <li><Link to="/dashboard/reports">Rapports</Link></li>
          </ul>
        </nav>
      </aside>

      <main className="dashboard-content">
        {/* Outlet affiche les routes enfants */}
        <Outlet />
      </main>
    </div>
  );
}
```

Outlet : Composant qui sert de "place" où les routes enfants seront rendues.

6.4 Route Index

```
<Route index element={<DashboardHome />} />
```

Route index : S'affiche quand on est exactement sur le chemin parent (ici /dashboard)

PARTIE 7 : Routes Protégées et Authentification

7.1 Vérification d'Authentification

```
// hooks/useAuth.js
import { useState, useEffect } from 'react';

function useAuth() {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Vérifier si l'utilisateur est connecté
    const token = localStorage.getItem('token');

    if (token) {
      // Récupérer les infos utilisateur
```

7.1 Vérification d'Authentification

```
    if (token) {
      // Récupérer les infos utilisateur
      fetchUserData(token).then(userData => {
        setUser(userData);
        setLoading(false);
      });
    } else {
      setLoading(false);
    }
  }, []);
```

7.1 Vérification d'Authentification

```
  const login = async (credentials) => {
    const response = await loginAPI(credentials);
    localStorage.setItem('token', response.token);
    setUser(response.user);
  };

  const logout = () => {
    localStorage.removeItem('token');
    setUser(null);
  };

  return { user, loading, login, logout };
}
```

7.2 Composant ProtectedRoute

```jsx
// components/ProtectedRoute.js
import { Navigate, useLocation } from 'react-router-dom';
import { useAuth } from '../hooks/useAuth';

function ProtectedRoute({ children }) {
  const { user, loading } = useAuth();
  const location = useLocation();

  if (loading) {
    return <div>Chargement...</div>;
  }

  if (!user) {
    // Rediriger vers login, mais sauvegarder l'emplacement actuel
    return <Navigate to="/login" state={{ from: location }} replace />;
  }

  return children;
}
```

7.3 Utilisation dans les Routes

```
// App.js
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/login" element={<Login />} />
  <Route path="/register" element={<Register />} />

  <Route
    path="/dashboard"
    element={
      <ProtectedRoute>
        <DashboardLayout />
      </ProtectedRoute>
    }
  >
    <Route index element={<DashboardHome />} />
    <Route path="profile" element={<Profile />} />
    <Route path="settings" element={<Settings />} />
  </Route>
</Routes>
```

7.4 Page de Login avec Redirection

```jsx
// pages/Login.js
import { useState } from 'react';
import { useNavigate, useLocation } from 'react-router-dom';
import { useAuth } from '../hooks/useAuth';

function Login() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const navigate = useNavigate();
  const location = useLocation();
  const { login } = useAuth();

  // Récupérer la page d'origine (si existante)
  const from = location.state?.from?.pathname || '/dashboard';
```

7.4 Page de Login avec Redirection

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setError('');

  try {
    await login({ email, password });
    // Rediriger vers la page d'origine
    navigate(from, { replace: true });
  } catch (err) {
    setError('Email ou mot de passe incorrect');
  }
};
```

7.4 Page de Login avec Redirection

```
return (
  <div className="login-page">
    <h1>Connexion</h1>
    <form onSubmit={handleSubmit}>
      <input
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Email"
        required
      />
      <input
        type="password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        placeholder="Mot de passe"
        required
```

7.4 Page de Login avec Redirection

```
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          placeholder="Mot de passe"
          required
        />
        {error && <p className="error">{error}</p>}
        <button type="submit">Se connecter</button>
      </form>
      <p>
        Pas de compte ? <Link to="/register">S'inscrire</Link>
      </p>
    </div>
  );
}
```

PARTIE 8 : Routing Avancé

8.1 Routes Asynchrones (Lazy Loading)

```jsx
import { lazy, Suspense } from 'react';

// Chargement paresseux des composants
const Home = lazy(() => import('./pages/Home'));
const About = lazy(() => import('./pages/About'));
const Contact = lazy(() => import('./pages/Contact'));

function App() {
  return (
    <Suspense fallback={<div>Chargement...</div>}>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </Suspense>
  );
}
```

PARTIE 9 : Gestion des Erreurs

9.1 Page 404 (Not Found)

```jsx
// pages/NotFound.js
import { Link } from 'react-router-dom';

function NotFound() {
  return (
    <div className="not-found">
      <h1>404 - Page non trouvée</h1>
      <p>La page que vous cherchez n'existe pas.</p>
      <Link to="/">Retour à l'accueil</Link>
    </div>
  );
}
```

9.2 Route 404

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/about" element={<About />} />
  {/* Cette route attrape TOUTES les autres URLs */}
  <Route path="*" element={<NotFound />} />
</Routes>
```

RÉSUMÉ DES CONCEPTS CLÉS

1. BrowserRouter

- Fournit le contexte de routage
- Doit englober toute l'application

2. Routes et Route

- Routes : Conteneur pour toutes les routes
- Route : Définit un chemin et son composant

3. Navigation

- Link : Navigation sans rechargement
- NavLink : Link avec styles actifs
- useNavigate() : Navigation programmatique

4. Paramètres de Route

- Syntaxe : :parametre
- Récupération : useParams()

5. Routes Imbriquées

- Outlet : Affiche les composants enfants
- Routes enfants héritent du chemin parent

6. Authentification

- ProtectedRoute : Protège les routes privées
- Redirection vers login si non authentifié

7. Hooks Utiles

- useParams() : Paramètres d'URL
- useNavigate() : Navigation
- useLocation() : Informations sur l'URL actuelle
- useSearchParams() : Paramètres de recherche (?key=value)

HTTP, Verbes et Codes d'État

CHAPITRE 1 : Les Bases d'HTTP

1.1 Qu'est-ce que HTTP ?

HTTP (HyperText Transfer Protocol) est le **langage de communication** entre votre navigateur (Chrome, Firefox) et les serveurs web.

Analogie simple :

Vous au restaurant : "Serveur, je voudrais une pizza"

Le serveur : "Voici votre pizza"

HTTP = le langage de cette conversation

1.1 Qu'est-ce que HTTP ?

Quand vous tapez google.com dans votre navigateur :

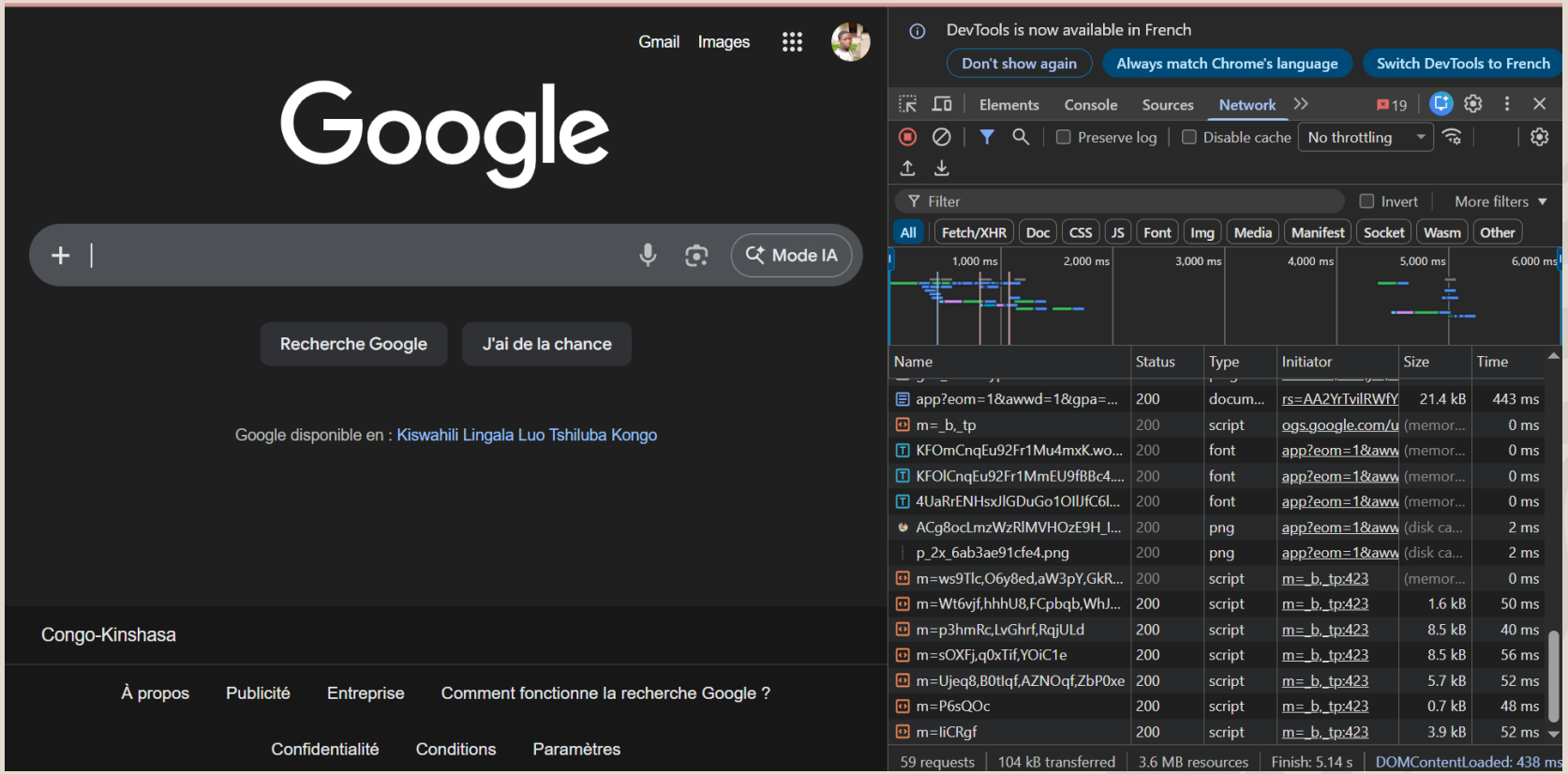
Votre navigateur dit :

- "GET / HTTP/1.1" (Donne-moi la page d'accueil)
- Le serveur Google répond : "200 OK" + le code HTML de Google
- Votre navigateur affiche Google

1.2 Comment voir HTTP en action ?

Dans Chrome/Firefox :

1. Ouvrez les DevTools (F12)
2. Allez dans l'onglet **Network** (Réseau)
3. Rafraîchissez une page
4. Vous voyez toutes les conversations HTTP !



CHAPITRE 2 : Les Verbes HTTP (Méthodes)

2.1 GET - Lire des données

Quand l'utiliser :

- Afficher une page web
- Récupérer des données d'une API
- Charger des images, CSS, JavaScript

2.1 GET - Lire des données

Caractéristiques :

- Ne modifie **jamais** les données sur le serveur
- Peut être mis en cache
- Visible dans l'URL (?page=2&search=test)

2.1 GET - Lire des données

```
// Avec fetch
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => response.json())
  .then(posts => {
    console.log('J\'ai reçu', posts.length, 'articles');
  });
```

2.1 GET - Lire des données

```
// Avec axios
axios.get('https://jsonplaceholder.typicode.com/posts')
  .then(response => {
    console.log('Statut:', response.status);
    console.log('Données:', response.data);
  });
```

2.1 GET - Lire des données

⚠️ Erreur courante :

```
// ❌ MAUVAIS : GET ne doit JAMAIS avoir de body
fetch('/api/search', {
  method: 'GET',
  body: JSON.stringify({ query: 'test' }) // INTERDIT !
});

// ✅ BON : Les paramètres vont dans l'URL
fetch('/api/search?query=test')
```

2.2 POST - Créer des données

Quand l'utiliser :

- Créer un nouvel utilisateur
- Envoyer un formulaire
- Uploader un fichier

2.2 POST - Créer des données

Caractéristiques :

- A un **body** avec les données
- Retourne souvent **201 Created**
- Crée une nouvelle ressource

2.2 POST - Créer des données

```
// Avec fetch
fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json' // TRÈS IMPORTANT !
  },
  body: JSON.stringify({
    title: 'Mon premier article',
    body: 'Le contenu de mon article',
    userId: 1
  })
})
.then(response => {
  if (response.status === 201) {
    console.log('✅ Article créé avec succès !');
  }
  return response.json();
})
.then(newPost => {
  console.log('ID du nouvel article:', newPost.id);
});
```

2.2 POST - Créer des données

```
// Avec axios (plus simple)
axios.post('https://jsonplaceholder.typicode.com/posts', {
  title: 'Mon premier article',
  body: 'Le contenu de mon article',
  userId: 1
})
.then(response => {
  console.log('Article créé:', response.data);
});
```

2.3 PUT vs PATCH - La différence CRUCIALE

PUT = Remplacer TOUT

```
// Situation : Un utilisateur existe avec ces données
// { id: 1, name: 'Jean', email: 'jean@mail.com', city: 'Paris' }

// Requête PUT
fetch('/api/users/1', {
  method: 'PUT',
  body: JSON.stringify({
    name: 'Jean Modifié',
    email: 'jean.nouveau@mail.com'
    // ⚠️ OUBLIÉ : city n'est pas envoyé !
  })
});

// Résultat FINAL : city a DISPARU !
// { id: 1, name: 'Jean Modifié', email: 'jean.nouveau@mail.com' }
```

2.3 PUT vs PATCH - La différence CRUCIALE

PATCH = Modifier seulement certaines choses

```
// Même situation initiale
// { id: 1, name: 'Jean', email: 'jean@mail.com', city: 'Paris' }

// Requête PATCH
fetch('/api/users/1', {
  method: 'PATCH',
  body: JSON.stringify({
    email: 'jean.nouveau@mail.com'
    // ✅ Je change seulement l'email
  })
});

// Résultat FINAL : Seul l'email est changé !
// { id: 1, name: 'Jean', email: 'jean.nouveau@mail.com', city: 'Paris' }
```

2.3 PUT vs PATCH - La différence CRUCIALE

Règle d'or :

- PUT quand vous voulez remplacer TOUT
- PATCH quand vous voulez changer seulement certaines propriétés

2.4 DELETE - Supprimer des données

```
// Avec fetch
fetch('https://jsonplaceholder.typicode.com/posts/1', {
  method: 'DELETE'
})
.then(response => {
  if (response.status === 200 || response.status === 204) {
    console.log('✅ Article supprimé');
  }
});
```

2.4 DELETE - Supprimer des données

```
// Avec axios
axios.delete('https://jsonplaceholder.typicode.com/posts/1')
  .then(() => {
    console.log('✅ Article supprimé');
    // Mettre à jour l'interface
    setPosts(posts.filter(post => post.id !== 1));
  });
```

CHAPITRE 3 : Les Codes d'État HTTP

3.1 Catégories des codes

Code	Signification	Explication
1xx	Informatif	"Je réfléchis..."
2xx	Succès ✅	"Tout est bon !"
3xx	Redirection 🔀	"Va voir ailleurs"
4xx	Erreur client ❌	"C'est ta faute"
5xx	Erreur serveur 🔥	"C'est ma faute"

3.2 Les 2xx - Succès

- 200 OK - Tout va bien
- 201 Created - Ressource créée
- 204 No Content - Succès mais pas de données

3.3 Les 4xx - Erreurs du client

- 400 Bad Request - Requête mal formée
- 401 Unauthorized - Non authentifié
- 403 Forbidden - Pas les permissions
- 404 Not Found - Ressource introuvable

3.4 Les 5xx - Erreurs du serveur

- 500 Internal Server Error - Le serveur a planté
- 503 Service Unavailable - Serveur en maintenance

CHAPITRE 4 : APIs GRATUITES

1. JSONPlaceholder - Données de test

URL de base : https://jsonplaceholder.typicode.com

Routes disponibles (toutes supportent GET, POST, PUT, PATCH, DELETE)