**NEXTJS**

1. **Les routes**
   1. **Templates**

Dans le NextJs, il y’a un quelque chose de très intéressant, appelé **template,** Lorsque vous créez un fichier template.tsx dans un projet Next.js, ce comportement peut s'expliquer par la manière dont Next.js utilise les Layouts et les Templates. Voici pourquoi cela pourrait se produire et comment corriger ce comportement si nécessaire.

* **Comprendre les fichiers template.tsx dans Next.js**

Dans Next.js 13 et au-delà, lorsque vous utilisez l'App Router, un fichier nommé template.tsx dans un répertoire spécifique est utilisé comme un React Server Component pour fournir un "template" de rendu pour toutes les routes imbriquées dans ce répertoire.

app/

dashboard/

template.tsx

page.tsx

settings/

page.tsx

template.tsx dans dashboard/ sera appliqué pour toutes les routes sous dashboard/ (par exemple, /dashboard, /dashboard/settings).

Si vous avez un fichier template.tsx dans le répertoire racine (app/template.tsx), il sera utilisé comme template global pour toutes les routes, ce qui peut expliquer pourquoi il se "lance" dans tous les composants.

* **Solution : Limiter l’application des templates**

Si vous ne voulez pas que le fichier template.tsx soit appliqué globalement, assurez-vous de :

* + - Positionner le fichier template.tsx uniquement dans les répertoires spécifiques où il est requis.

Par exemple, si vous souhaitez qu'il s'applique uniquement aux routes sous dashboard/, placez-le dans app/dashboard/template.tsx.

* + - Supprimer le fichier template.tsx du répertoire racine (app/template.tsx) s'il existe.
* **Différence entre layout.tsx et template.tsx**
  + - **layout.tsx** : Garde l'état persistant entre les changements de pages dans son scope (comme un véritable layout).
    - **template.tsx** : Crée une nouvelle instance pour chaque navigation (utile pour éviter le partage non désiré d'état ou de composants).

La différence entre layout.tsx et template.tsx réside dans leur comportement face au rendu des enfants (ou sous-routes) et la manière dont ils gèrent l'état et la persistance.

* + - **layout.tsx** : Garde l'état persistant entre les navigations au sein de son scope.
    - **template.tsx** : Crée une nouvelle instance à chaque navigation, ce qui est utile lorsque vous ne voulez pas conserver d'état ou si vous avez besoin d’un rendu propre pour chaque navigation.

**Exemple avec une démonstration concrète**

**STRUCTURE DES FICHIERS**

app/

layout.tsx # Layout global pour toutes les routes

dashboard/

layout.tsx # Layout spécifique pour le tableau de bord

template.tsx # Template spécifique pour certaines navigations

page.tsx # Page Dashboard principale

settings/

page.tsx # Page des paramètres sous Dashboard

1. **layout.tsx - Comportement persistant**
2. export default function DashboardLayout({ children }: { children: React.ReactNode }) {
3. return (
4. <div>
5. <nav>
6. <h2>Navigation persistante</h2>
7. <ul>
8. <li><a href="/dashboard">Home</a></li>
9. <li><a href="/dashboard/settings">Settings</a></li>
10. </ul>
11. </nav>
12. <main>{children}</main>
13. </div>
14. );
15. }

Ce layout.tsx affiche une navigation persistante pour toutes les sous-routes de /dashboard (comme /dashboard et /dashboard/settings).

Lors de la navigation entre les sous-pages, le composant parent ne se démonte pas, ce qui signifie que la barre de navigation et son état restent inchangés.

1. **template.tsx - Nouveau rendu à chaque navigation**
2. // app/dashboard/layout.tsx
3. import React from "react";
4. // app/dashboard/template.tsx
5. import React from "react";
6. export default function DashboardTemplate({ children }: { children: React.ReactNode }) {
7. console.log("Template re-rendered");
8. return (
9. <div style={{ border: "1px solid blue", padding: "10px" }}>
10. <h1>Dashboard Template</h1>
11. {children}
12. </div>
13. );
14. }

* Ce template.tsx ajoute un comportement unique pour chaque navigation.
* À chaque fois que l'utilisateur navigue entre /dashboard et /dashboard/settings, le template est recréé, ce qui déclenche le re-rendu et réinitialise tout état local ou UI qui lui est associé.
* Cela est utile si vous voulez éviter que l'état persiste entre les navigations (par exemple, un formulaire).

**Quand utiliser quoi ?**

* **layout.tsx** : Pour tout ce qui doit rester constant, comme une barre de navigation, un footer, ou des éléments qui partagent un état.
* **template.tsx** : Pour des sous-routes où un nouveau rendu ou une réinitialisation complète est souhaitée. Par exemple, des formulaires où vous voulez commencer avec un état vierge.

**1.2. PATHNAME**