Atelier 3 NextJS : CRUD

# Variable d’environnement

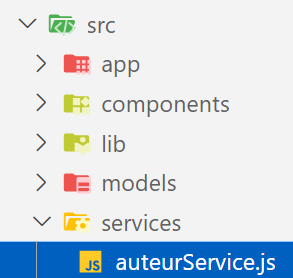


API\_URL = http://localhost:3000/api

# Services

Ensuite, sous /src créer le dossier services.

Dans lequel créer auteurService.js



const AUTEUR\_API="/auteurs/"

export const fetchAuteurs=async()=> {

const res = await fetch(process.env.API\_URL+AUTEUR\_API, { cache: 'no-store' })

const response = await res.json();

return **response .auteurs**;

}

export const fetchAuteurById=async(auteurId)=> {

    const res = await fetch(process.env.API\_URL+AUTEUR\_API+`${auteurId}`,{

        method: 'GET'

    });

    const response = await res.json();

    return **response.auteur** ;

}

export const deleteAuteur=async(auteurId) =>{

    const res = await fetch(process.env.API\_URL+AUTEUR\_API+`${auteurId}`,{

        method: 'DELETE'

    });

    const response = await res.json();

    return response;

}

export const addAuteur=async(auteur)=> {

    const res = await fetch(process.env.API\_URL+AUTEUR\_API, {

        method: 'POST',

        headers: {

        'Content-Type': 'application/json',

        },

        body: JSON.stringify(auteur),

    });

    const response = await res.json();

    return response;

}

export const editAuteur=async(auteur) =>{

    const res = await fetch(process.env.API\_URL+AUTEUR\_API+`${auteur.\_id}`, {

        method: 'PUT',

        headers: {

        'Content-Type': 'application/json',

        },

        body: JSON.stringify(auteur),

    });

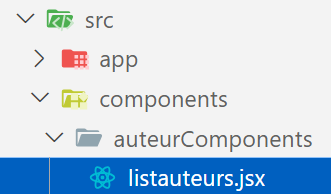
    const response = await res.json();

    return response;

}

# Affichage des auteurs

Sous /src/components/ créer le dossier auteurComponents puis créer le fichier listauteurs.jsx



Accéder au site<https://www.material-react-table.com/>

Puis Quick examples->Basic Example

Installer le composant material-react-table

npm install material-react-table

"use client"

import  { useMemo } from 'react';

import { MaterialReactTable } from 'material-react-table';

const Listauteurs = ({auteurs}) => {

  const columns = useMemo(

    () => [

    {

    accessorKey: 'nomauteur', //access nested data with dot notation

    header: 'NOM',

    size: 100,

    },

    {

    accessorKey: 'email',

    header: 'EMAIL',

    size: 100,

    },

    {

    accessorKey: 'numtel', //normal accessorKey

    header: 'TEL',

    size: 100,

    }

    ],

    [auteurs],

    );

return (

<div>

<MaterialReactTable columns={columns} data={auteurs} />;

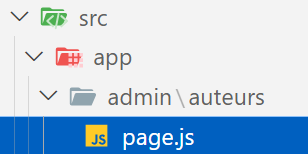
</div>

  )

}

export default Listauteurs

On va créer un dossier appelé auteurs sous /src/app/admin ayant page.js



import Listauteurs from '@/components/auteurComponents/listauteurs';

import {fetchAuteurs} from "@/services/auteurService"

const getAuteurs=async()=>{

const data=await fetchAuteurs()

return data;

}

const AuteurPage = async() =>{

    const auteurs=await getAuteurs()

  return (

   <div className="container">

      <Listauteurs auteurs={auteurs}/>

    </div>

  )

}

export default AuteurPage

Test dans le navigateur

<http://localhost:3000/admin/auteurs>

## 

# File conventions

Remarque : Pour tous ces fichiers de file convention, une fois le fichier créé et resté vide sans code il générera une erreur.

## error.js

Un fichier d'erreurs définit une limite d'erreur d'interface utilisateur pour un segment de route.

Il est utile pour détecter les erreurs inattendues qui se produisent dans les composants serveur et les composants clients et pour afficher une interface utilisateur de secours.

Sous /src/app/admin/auteurs créer le fichier error.js

'use client' // Error components must be Client Components

import { useEffect } from 'react'

export default function Error({ error, reset }) {

  useEffect(() => {

    // Log the error to an error reporting service

    console.error(error)

  }, [error])

  return (

    <div>

      <h2>Something went wrong!</h2>

      <button

        onClick={

          // Attempt to recover by trying to re-render the segment

          () => reset()

        }

      >

        Try again

      </button>

    </div>

  )

}

Un essai est de mettre une URL end point erronée dans le service dans la requête get puis d’essayer l’url : <http://localhost:3000/admin/auteurs>

On aura une page personnalisée d’erreur.

## 

## loading.js

Par défaut, ce fichier est un composant serveur - mais peut également être utilisé comme composant client via la directive "use client".

Sous /src/app/admin/auteurs créer le fichier loading.js

export default function Loading() {

    // Or a custom loading skeleton component

    return <p>Loading...</p>

  }

Loading n'accepte aucun paramètre.

Comme essai le temps de charger la page on peut voir dans l’url : <http://localhost:3000/admin/auteurs>

## 

## not-found.js

Le fichier not-found est utilisé pour restituer l'interface utilisateur lorsque la fonction notFound est lancée dans un segment de route. En plus de proposer une interface utilisateur personnalisée, Next.js renverra un code d'état HTTP 200 pour les réponses diffusées en continu et 404 pour les réponses non diffusées.

En plus de détecter les erreurs notFound() attendues, le fichier racine app/not-found.js gère également toutes les URL sans correspondance pour l'ensemble de votre application. Cela signifie que les utilisateurs qui visitent une URL qui n'est pas gérée par votre application verront l'interface utilisateur exportée par le fichier app/not-found.js.

Sous **/src/app/** créer le fichier not-found.js



import Link from 'next/link'

export default function NotFound() {

  return (

    <div>

      <h2>Not Found</h2>

      <p>Could not find requested resource</p>

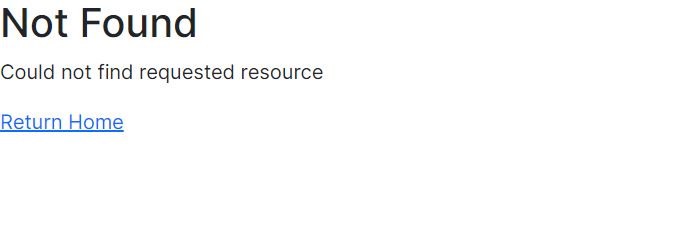
      <Link href="/">Return Home</Link>

    </div>

  )

}

Si on essaie n’importe quelle URL inexistante on aura :



# Suppression des auteurs