import { useState } from "react"

import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";

import { faChevronRight,faChevronLeft } from "@fortawesome/free-solid-svg-icons";

**function** Slideshow({pictures}){

**const** [index, setIndex] = useState(0);

**const** handleNext = () **=>**{

        setIndex((prevIndex) **=>** {

            if(prevIndex + 1 >= pictures.length){

                return 0

            }else{

                return prevIndex + 1

            }

        })

    }

**const** handlePrev =  () **=>**{

        setIndex((prevIndex) **=>**{

            if(prevIndex === 0){

                return pictures.length -1

            }else{

                return prevIndex - 1

            }

        }

    )

    }

    return(

        <>

            <img src={pictures[index]} alt="logement" />

            {pictures.length > 1 && (

            <>

                <FontAwesomeIcon icon={faChevronLeft} onClick={handlePrev}/>

                <FontAwesomeIcon icon={faChevronRight}  onClick={handleNext}/>

            </>

            )}

        </>

    )

}

export default Slideshow

A bien revoir et apprendre :

* fonctions handleNext et handlePrev , la logique comment y arriver
* pictures[index] comprende ce tableau

.find() est une **méthode JavaScript** des tableaux.

Elle sert à **trouver** (le premier) élément qui **correspond à une condition**.

**Pourquoi .find() et pas .filter() ?**

* .find() → retourne **un seul objet**
* .filter() → retourne un **tableau de tous les objets** qui matchent

Dans notre cas, on veut **UN logement**, donc .find() est parfait.

“Je cherche dans mon tableau de logements celui dont l’id correspond à l’id dans l’URL, et je le stocke dans currentLogement.”

Excellente question. Voici une explication **claire et directe**, pour que tu comprennes **comment** et surtout **pourquoi penser comme ça** en React :

### 📌 Pourquoi faire ça :

<Collapse title="Équipements">

<ul>

{findLogement.equipments.map((item, index) => (

<li key={index}>{item}</li>

))}

</ul>

</Collapse>

Parce que :

1. **findLogement.equipments** est un **tableau de chaînes de caractères** (ex: ["Frigo", "WiFi", "Douche"]).
2. Pour afficher chaque élément de ce tableau à l’écran, tu dois **le parcourir**.
3. En React, **on parcourt un tableau avec .map()**.
4. Tu veux afficher une **liste d’éléments HTML** → tu utilises <ul> (liste non ordonnée) + <li> (chaque ligne).
5. La key={index} est nécessaire pour que React suive chaque élément (important en rendering dynamique).

### 🤔 Comment "penser" comme ça :

Quand tu veux **afficher plusieurs données** venant d’un tableau :

1. **Demande-toi : est-ce un tableau ?** → Oui ? alors sûrement .map().
2. **Veux-tu afficher chaque élément dans un tag HTML ?** → Oui ? alors tu crées une structure comme :
3. array.map((item) => <div>{item}</div>)
4. **Si c’est une liste → <ul> + <li>**.

### 🧠 En résumé :

**Quand tu dois afficher plusieurs éléments venant d’un tableau, pense .map().**  
Et si c’est une vraie liste (à puces), utilise <ul><li>.

Souhaites-tu maintenant une explication rapide sur children ?

Tu suis les bonnes pratiques React pour :

* useParams → récupérer l’id dans l’URL
* .find() → localiser le bon logement dans le JSON
* Slideshow → passer les images
* Collapse → afficher dynamiquement la description et les équipements avec des composants réutilisables
* mapping d’un tableau (equipments) → propre, avec une key sur chaque <li>

### ✅ Ce que tu fais bien :

* Tu sépares bien la logique (récupération de données) et l'affichage.
* Tu utilises des composants enfants (Collapse, Slideshow), ce qui rend ton code modulaire.
* Tu respectes la hiérarchie HTML (ul → li).
* Tu ne rends pas des composants si les données ne sont pas prêtes (en supposant que tu fais des vérifs ailleurs ou que le JSON est fiable).