Tours Component - Custom React Component

Overview

Tours component is a versatile and reusable component that allows adding a presentation to a web page. It is a good way to add the necessary information as succinctly as possible, with the possibility to display more or less information, depending on your preferences. We have also included a *loading component* that displays the message "Loading..." until the data is loaded from the database, to avoid the unsightly appearance of an empty page. The Tours component, which represents the entire display on the page, is made up of several single *Tour components* - this represents each tour together with all the information related to it.

Usage

The tour component is extremely versatile, being composed of several individual tour components that can adapt to any type of application, be it news, car presentations, events or even sales.

Start

First it is necessary, to run this project we need to run the *"npm start"* command, which if run successfully, navigates to port http://localhost:3000. If there is already running on that port, we can choose to change it and access the one displayed in the terminal, and when we change anything at the application level and save, the page will refresh automatically.

Integrate

To be able to use this component in the react app, you need to add the Tour.js file from the /src/Tour.js path, and it also needs to be imported into App.js, as follows:

```
import React, { useState, useEffect } from 'react'
import Loading from './Loading'
import Tours from './Tours'
```

```
import Tour from './Tour';
```

The use of the component in the code is done through both <Tour> and <Tours> tags.

In the *Tour componen*t - implemented in Tour.js, all the necessary details of a tour are saved, more precisely, an article that stores the information, the image, the price and the description. It is used inside the second component, namely the *Tours component*, implemented in Tours.js, where all available tours are returned.

```
<div>
  {tours.map((tour) => {
    return <Tour key={tour.id} {...tour} removeTour={removeTour} />;
  })}
</div>
```

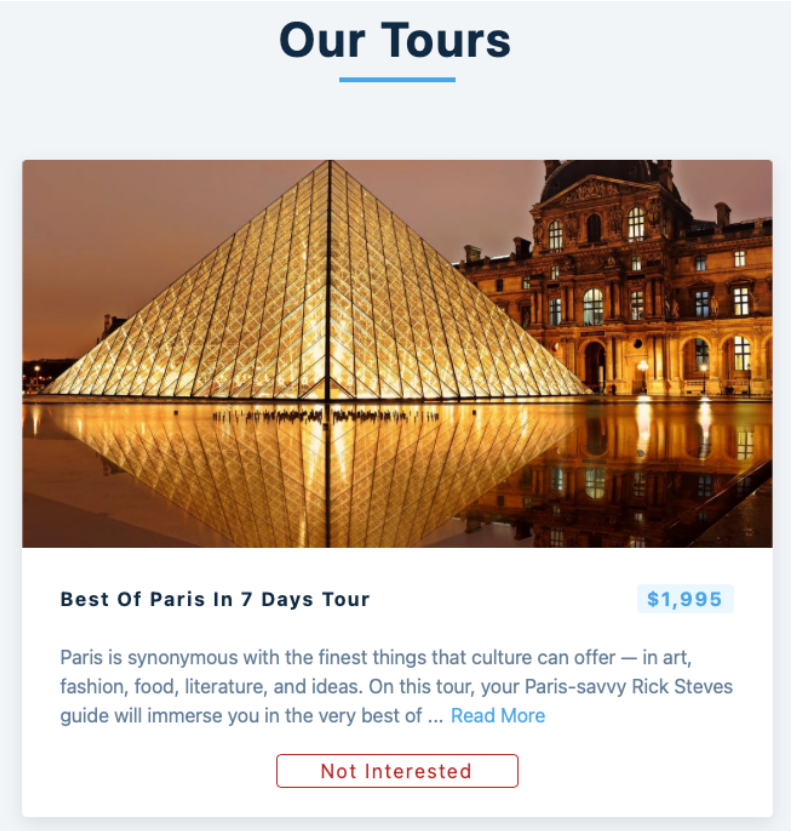
The *Tours component*, which contains both all available tours and the remove function that allows deleting tours that are not of interest, which is used within App.js.

```
return (
  <main>
    <Tours tours={tours} removeTour={removeTour} />
  </main>
)
```

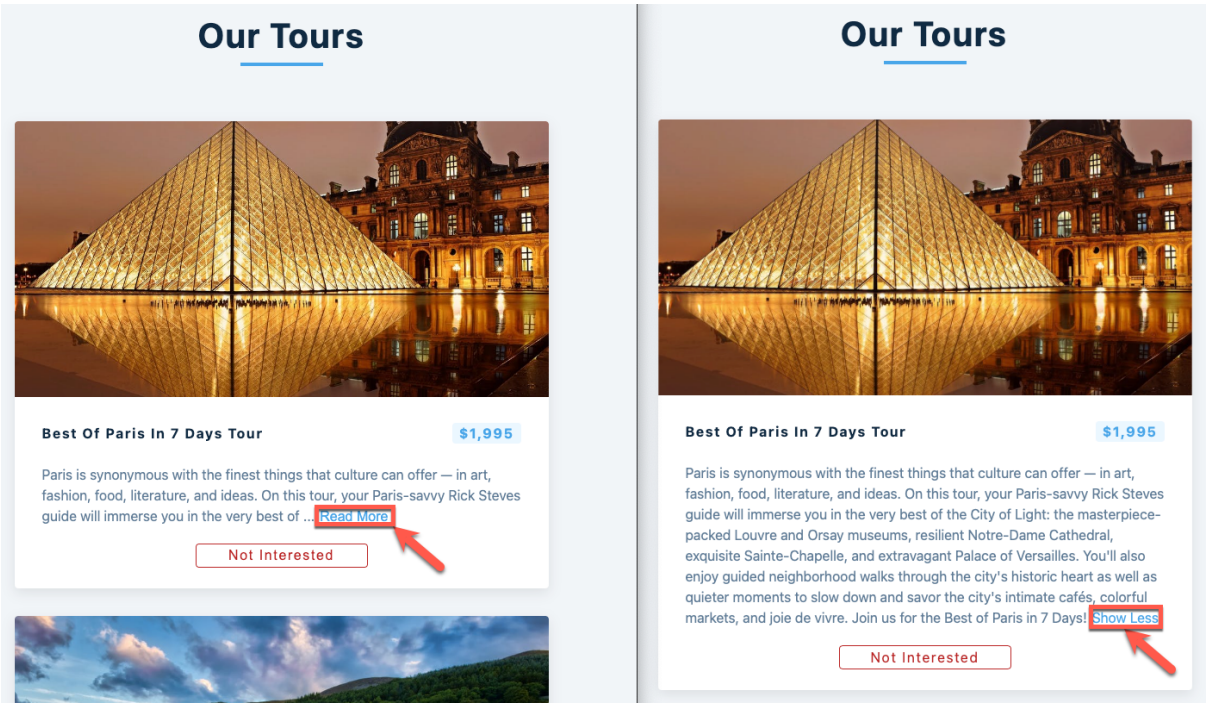
Description

The items appear visually displayed as an article containing the presentation image, price, name, information and the 'Not interested' button which allows you to delete the tour you are not interested in.

At the bottom is the *'Not interested' button* which allows you to delete a tour that is not interesting. This allows you to delete all the tours, but if you have deleted one of the tours that was of interest, then you can refresh the page and all the tours from the beginning will be reloaded.



Initially the space that is reserved for inspirations is set to a limit of 200 characters. By pressing the *'Read more' button* the information is completely filled in. If you wish to return to the initial, succinct presentation, you can press the *'Show less' button*.



Code

App.js

In order to avoid a blank white page, we have created a loading component, which displays a message "Loading..." until the page is loaded from the database. After the setLoading data has been loaded, it switches to false.

```
const fetchTours = async () => {
  setLoading( value: true)
  try {
    const response = await fetch(url)
    const tours = await response.json()
    setLoading( value: false)
    setTours(tours)
  } catch (error) {
    setLoading( value: false)
    console.log(error)
  }
}
```

When all tours are deleted, a "No tours left" message appears, and a refresh button reloads all tours.

```
if (tours.length === 0) {
  return (
    <main>
      <div className='title'>
        <h2>no tours left</h2>
        <button className='btn' onClick={() => fetchTours()}>
          refresh
        </button>
      </div>
    </main>
  )
}
```

Finally we encounter the favorable case where no previous error is present and the tours are loaded directly, at the moment the data is loaded.

Tour.js

Load all the parameters needed to create a tour. In the information part there is a readMore button that displays the entire text, which is initially limited to 200 characters. Also implemented here is the delete or "not interested" button that allows to delete a tour that is not interesting.

```
const Tour = ({ id, image, info, name, price, removeTour }) => {
  const [readMore, setReadMore] = useState( initialState: false);
  return (
    <article className="single-tour">
      <img src={image} alt={name} />
      <footer>
        <div className="tour-info">
          <h4>{name}</h4>
          <h4 className="tour-price">${price}</h4>
        </div>
        <p>
          {readMore ? info : `${info.substring(0, 200)}...`}
          <button onClick={() => setReadMore(!readMore)}>
            {readMore ? 'show less' : '  read more'}
          </button>
        </p>
        <button className="delete-btn" onClick={() => removeTour(id)}>
          not interested
        </button>
      </footer>
    </article>
  );
};

export default Tour;
```

Tours.js

In this component all tours available in the database are mapped. For each object it finds in the array, it retrieves a new component from the Tour. {...tour} passes everything the object needs to access, and the removeTour function is as shown above.

```
const Tours = ({ tours, removeTour }) => {
  return (
    <section>
      <div className="title">
        <h2>our tours</h2>
        <div className="underline"></div>
      </div>
      <div>
        {tours.map((tour) => {
          return <Tour key={tour.id} {...tour} removeTour={removeTour} />;
        })}
      </div>
    </section>
  );
};
```