**PROYECTO GALERIA DE IMÁGENES**

Para esta sección se va a crear una aplicación de galería de imágenes con React:

Esta es la línea de comando que se utiliza para inciar el proyecto.

npx create-react-app --template cra-template-pwa

**LA API**

La API que se utilizará en esta sección se llama Unsplash.

Es necesario crear una cuenta

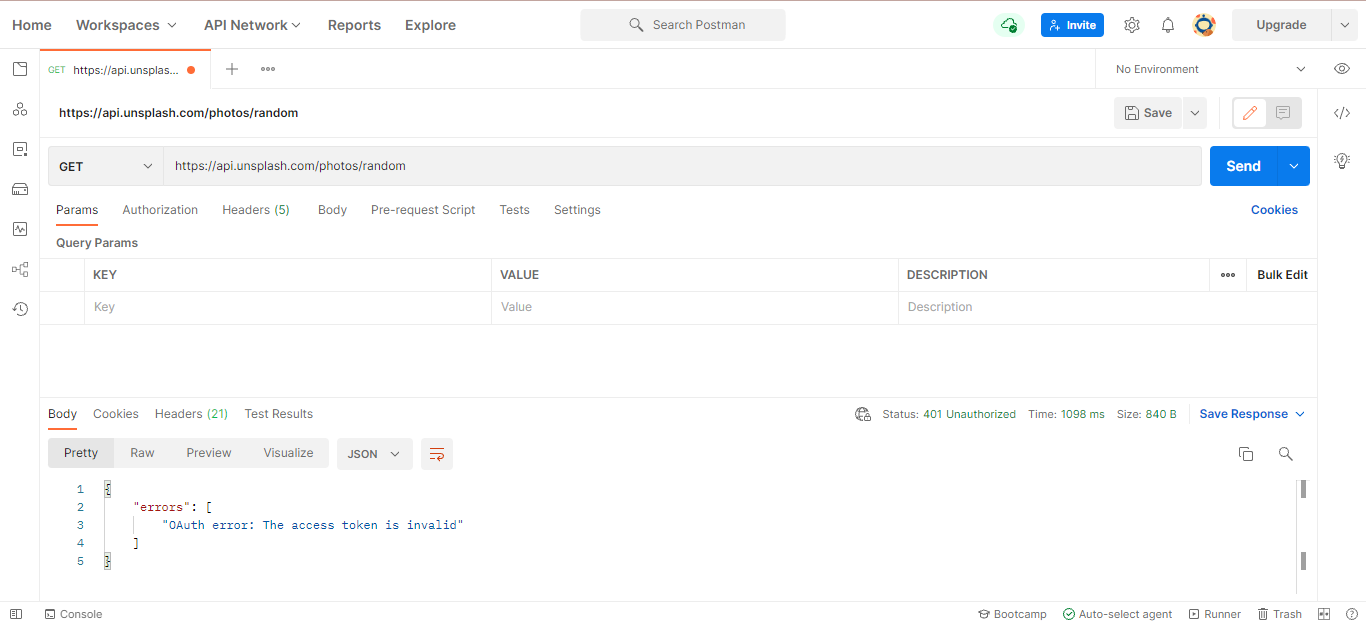
Agustin Stringa [stringaagustin1@gmail.com](mailto:stringaagustin1@gmail.com) Strii nachogordo123

**REALIZANDO PETICIONES DESDE LA WEB**

Para probar los resultados de la API se va a utilizar un servicio web llamado POSTMAN

Como inicio es necesario registrarse (en mi caso con Google) y luego ir a la sección de Workspaces

Una vez en esta sección, se pueden realizar peticiones agregando una nueva Request Desde el lugar que tiene el signo +.



Desde el desplegable se selecciona el tipo de peticion, y en el input se introduce la ruta.

**AUTORIZACIONES**

Para utilizar correctamente el servicio, la API nos provee de una key para utilizar en las consultas que se realicen.

Estas pueden enviarse por HEADERS o por URL.

Por ejemplo:

<https://api.unsplash.com/photos/?client_id=q0dmuecvQTkzovHYv3I7p6f-ohyGOejR_KI7POJJL8E>

esto nos permitiría realizar una petición de fotos. En el parámetro client\_id de GET se introduce la key que nos da la API.

Pasando por headers:

Desde el servicio web **postman** se puede agregar un elemento desde el menú HEADERS de la consulta y agregar la key-> Authorization con el valor Client-ID /APIKEY

Desde el código fetch, puede hacerse de esta manera:

  fetch(urlFotos, {

    method: "GET",

    headers: {

      Authorization: `Client-ID ${apiKey}`,

    },

  })

El método **fetch** permite como segundo parámetro un objeto con las configuraciones editables.

**CONFIGURACIONES DEL PROYECTO**

Las principales configuraciones hasta el momento son:

* Cambiar el componente principal app.js por app.jsx (modificar esto en el index.js)
* Además, se eliminan los archivos index.css y App.css

**RENDERIZANDO VARIAS IMÁGENES DESDE UNA PETICION**

El principio de funcionamiento de la operación es, manejar un componente padre con un state y un effect, que renderice varios elementos hijos.

El resultado de la petición será un array de objetos que contendrá datos de cada imagen.

La idea de usar un array es para renderizar su funcion **map** y que al ser un state incial vacio, no genere un error como lo haría en caso de manejarse un state con un object vacio.

CODIGO:

Contamos con 2 componentes, Cards.jsx (padre) y Card.jsx (hijos)

Este es el código del componente padre:

import React, { useState, useEffect } from "react";

import Card from "./Card";

const Cards = () => {

  const urlfotos = "https://api.unsplash.com/photos/";

  const apiKey = "q0dmuecvQTkzovHYv3I7p6f-ohyGOejR\_KI7POJJL8E";

  const [images, setImages] = useState([]);

  const peticion = async () => {

    const res = await fetch(urlfotos, {

      method: "GET",

      headers: {

        Authorization: `Client-ID ${apiKey}`,

      },

    });

    const data = await res.json();

    setImages(data);

  };

  useEffect(() => {

    peticion();

  }, []);

  return (

    <>

      {images.map((imagen) => (

        <Card

          src={imagen.urls.small}

          alt={imagen.alt\_description}

          key={imagen.id}

        />

      ))}

    </>

  );

};

export default Cards;

como se ve, lo único resaltable es el código de la funcion petición (la encargada de recibir los datos de la API)

EL componente empezaría su “ciclo de vida” con un state = [] array vacío, pero luego, gracias al useEffect ejecutaría la funcion **petición()** y cambiaría ese state por el array de objetos traídos desde el servicio.

Una vez modificado el estado, se renderizaría nuevamente el componente, modificando el resultado de su **return** ahora recorriendo un array con datos con el método **.map()**

CODIGO DEL COMPONENTE CARD

import React from "react";

import PropTypes from "prop-types";

const Card = ({ src, alt }) => {

  return (

    <div className="card" style={{ width: "18rem" }}>

      <img src={src} className="card-img-top" alt={alt} />

      <div className="card-body">

        <h5 className="card-title">Card title</h5>

        <p className="card-text">

          Some quick example text to build on the card title and make up the

          bulk of the card's content.

        </p>

        <a href="#" className="btn btn-primary">

          Go somewhere

        </a>

      </div>

    </div>

  );

};

Card.propTypes = {

  img: PropTypes.string,

};

export default Card;

este componente recibe por props los datos de cada elemento del array de objetos de state y los renderiza sobre este componente.

Además, las propTypes están configuradas para recibir solo cadenas de texto.

**HACIENDO EL BUSCADOR**

Para hacer el buscador, se crea un componente independiente que maneje el input y el label que constituyen el buscador.

Este es el código del mismo:

const Buscador = ({ input, setInput }) => {

*//pasar valor del input al state*

  const handleSubmit = (e) => {

    e.preventDefault();

    setInput(document.querySelector('input[type="text"]').value);

  };

  return (

    <div className="p-3">

      <form onSubmit={handleSubmit}>

        <label htmlFor="input-query">

          {" "}

          Buscar: <input type="text" name="input-text" id="input-query" />

        </label>

      </form>

    </div>

  );

};

Como se ve, se recibe el **state** input y la función **setInput** para manejar el estado y así llamar a la petición() cada vez que este cambie.

Para que esto funcione, ahora el state pasa a ser del componente Container y es “pasado” como props a cada uno de los componentes que le utilicen.

const Containers = () => {

*//estado para keyword de componente buscador*

  const [input, setInput] = useState("");

  return (

    <div className="container border p-3">

      <Buscador input={input} setInput={setInput}></Buscador>

      <Botones input={input} setInput={setInput}></Botones>

      <Cards input={input} />

    </div>

  );

};

En este caso nuestro componente Botones y Buscador, modifican el state, por tanto deben recibirle como props.