Instalar

React developer tools: https://chromewebstore.google.com/u/1/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=es

Primera tarea: instalar node y yarn, ¿Qué es cada cosa?
Node: https://nodejs.org/en
Yarn: https://classic.yarnpkg.com/lang/en/docs/install/#windows-stable

Extensiones para html
Auto close tag
Simple React snippets
ES7 + React/Redux/React-native snippets

¿Qué es un snippet?
¿Qué es un React Native?

Cd directorio/react
yarn create vite
nombre del proyecto: proyecto-gif-rodrigo-creando
framework: react
variant: javascript

Abrir la carpeta creada en VS Code

¿Cómo funcionan los directorios en una aplicación de React?
En una aplicación de React, los directorios y archivos se organizan de manera que faciliten la gestión del código y su mantenibilidad. Aquí se presenta una descripción general de cómo se suelen organizar los directorios en una aplicación de React:

1. **Carpeta src/**: Esta carpeta contiene el código fuente de la aplicación. Aquí es donde se encuentra la mayor parte del código que se escribirá.
2. **Carpeta components/**: Esta carpeta contiene los componentes de React de la aplicación. Un componente es una pieza reutilizable de la interfaz de usuario. En esta práctica la carpeta “components” estaría dentro de “src”.
3. **Carpeta pages/**: En esta carpeta se suelen almacenar las vistas principales y los componentes que solo se utilizan en esa página.
4. **Carpeta contexts/**: Aquí se pueden almacenar los contextos de React, que permiten compartir datos a través del árbol de componentes sin tener que pasarlos manualmente a través de las props.

5. **Carpeta hooks/**: Esta carpeta puede contener los hooks personalizados. Los hooks son una característica de React que permite utilizar el estado y otras características de React sin escribir una clase.
6. **Carpeta utils/**: Aquí se pueden almacenar funciones de utilidad que se utilizan en toda la aplicación.
7. **Carpeta services/**: Esta carpeta puede contener servicios, como llamadas a la API.
8. **Carpeta routes/**: Aquí se pueden definir las rutas de la aplicación.

Es importante mencionar que esta es solo una forma de organizar los directorios en una aplicación de React. La estructura exacta puede variar dependiendo de las necesidades del proyecto.

Más información:
https://es.legacy.reactjs.org/docs/faq-structure.html

¿Qué es un archivo .jsx?
JSX es una extensión de la sintaxis de JavaScript que permite escribir estructuras de HTML en el mismo archivo que el código de JavaScript

Instalar los módulos de node en la carpeta del proyecto:
1. En la terminal, cambiar de directorio para estar en el del proyecto
2. Ejecutar yarn dev
3. Abrir el navegador con el proyecto ejecutándose. ¿Qué es el estado de una aplicación?

El "estado" de una aplicación se refiere a la información que la aplicación está manteniendo en un momento dado. Esta información puede incluir datos ingresados por el usuario, decisiones tomadas por la aplicación, resultados de operaciones y más.

Por ejemplo, si estás utilizando una aplicación de comercio electrónico, el estado de la aplicación podría incluir los artículos que has añadido a tu carrito de compras, tu historial de búsqueda, si estás conectado o no, y otros detalles similares.

Es importante manejar correctamente el estado de la aplicación para asegurar una buena experiencia de usuario. Por ejemplo, si el estado no se maneja correctamente, un usuario podría perder su carrito de compras en una aplicación de comercio electrónico cada vez que actualiza la página.

TODO: MODIFICAR EL MAIN.JSX para empezar a personalizar

Para empezar a personalizar, en la carpeta src se borrarán todos los archivos excepto el main.jsx.

Crear un archivo que se llame “ProyectoGif.jsx”.

Si se tienen los snippets creados, usar “rafc” (React Arrow Function Componet):

Se crea un encabezado dentro de un fragmento y se borra la importación ya que en las versiones más recientes de React no es necesario hacerlo.

```
export const ProyectoGif = () => {
  return (
    <>
        <h1>ProyectoGif</h1>
    </>

  )
}
```

¿Qué es una función de flecha?

```
// Función tradicional
function (a){
  return a + 100;
}

// Función de flecha
(a) => a + 100;
```

El siguiente código crea una aplicación React que renderiza el componente ProyectoGif en un elemento con el id ‘root’ en la página HTML. <React.StrictMode> se utiliza para envolver el componente ProyectoGif y ayudar a detectar problemas potenciales en la aplicación durante el desarrollo. El componente ProyectoGif es el que está en el archivo anterior:

```
// Importamos la biblioteca de React
import React from 'react'

// Importamos la biblioteca ReactDOM, que proporciona métodos específicos del DOM para
React
import ReactDOM from 'react-dom/client'

// Importamos un componente llamado ProyectoGif desde otro archivo en el mismo directorio
import { ProyectoGif } from './ProyectoGif'

// Creamos un nuevo contenedor de raíz React en el elemento con el id 'root'
// y renderizamos el componente ProyectoGif en él
```

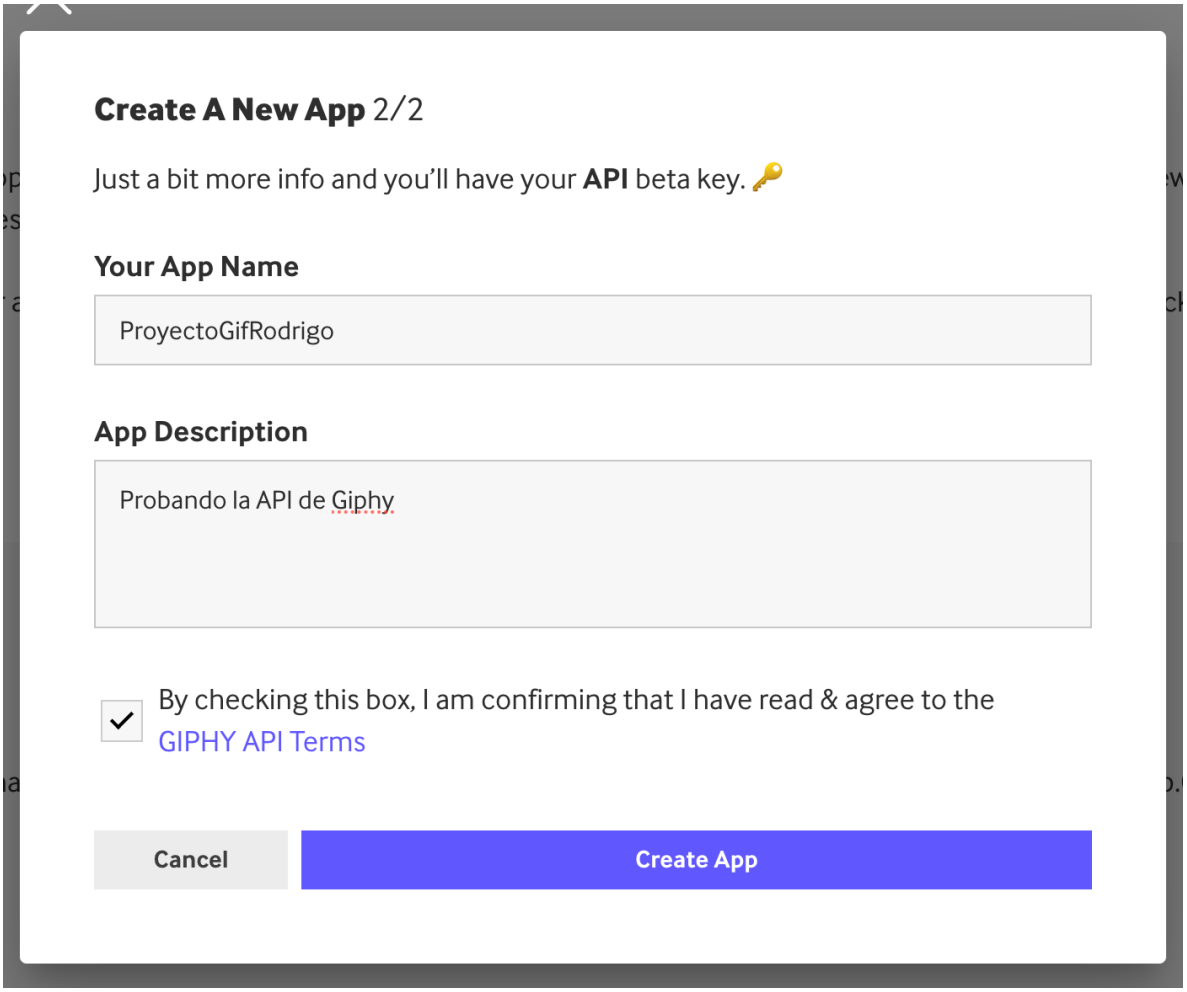
```
ReactDOM.createRoot(document.getElementById('root')).render(
  // React.StrictMode es un componente especial de React que comprueba si tu aplicación
tiene problemas potenciales durante el desarrollo
  <React.StrictMode>
     {/* Aquí es donde se renderiza el componente ProyectoGif */}
    <ProyectoGif />
  </React.StrictMode>,
)
```

¿Qué es un componente?
¿Puede haber un componente dentro de otro componente?
¿A qué se refiere “renderizar”?
¿Qué es el DOM?

Empezar a consumir la API de Giphy creando una cuenta en:

https://developers.giphy.com/

Seleccionar “Create an app” (puede estar en la documentación) y después se selecciona la opción API y seguir los pasos:



Para obtener la API Key se puede acceder desde:

https://developers.giphy.com/dashboard/

¿Qué es una API? ¿A qué se refiere "consumir una API"? ¿Qué es una API Key?

¿Qué es lo que se va a construir?

```
export const ProyectoGif = () => {
  return (
    <>
      {/* Título */}
      <h1>ProyectoGif</h1>

      {/* Input */}
```

```
        {/* Listado de Gifs */}
          {/* Git Item */}
    </>

  )
}
```

Agregar el CSS generado en un archivo nuevo llamado “styles.css” que se colocará en src:

```
* {
    font-family: Helvetica, Arial, sans-serif;
    background-color: rgb(234, 234, 234);
  }

  body {
    padding: 60px;
  }

  input {
    background-color: white;
    border-radius: 5px;
    border: 1px solid rgb(97, 32, 158);
    color: black;
    font-size: 1.2rem;
    outline: none;
    padding: 10px 15px;
    width: 100%;
  }

  h2 {
    font-size: 1.5rem;
  }

  h3 {
    font-size: 3rem;
    margin-bottom: 5px;
  }

  .card-grid {
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: center;
  }
```

```css
.card {
  align-content: center;
  align-items: center;
  background-color: white;
  border-radius: 10px;
  box-shadow: 0px 3px 5px rgba(0, 0, 0, 0.03);
  display: flex;
  flex-direction: column;
  height: 0%;
  justify-content: center;
  margin-bottom: 20px;
  margin-right: 20px;
  overflow: hidden;
}

.card p {
  background-color: white;
  flex: 1;
  font-size: 1.5rem;
  margin-top: 5px;
  padding: 5px 20px 0px 20px;
  text-align: center;
}

.card img {
  width: 100%;
}
```

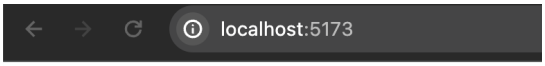
Añadir la importación en el main.jsx y notar la diferencia en la página:

```jsx
// Importamos la biblioteca de React
import React from 'react'
// Importamos la biblioteca ReactDOM, que proporciona métodos específicos del DOM para React
import ReactDOM from 'react-dom/client'
// Importamos un componente llamado ProyectoGif desde otro archivo en el mismo directorio
import { ProyectoGif } from './ProyectoGif'
//Importamos los estilos que se encuentran en src. ¿Qué significaba "./"?
import './styles.css'
```

¿Qué significaba "./"?

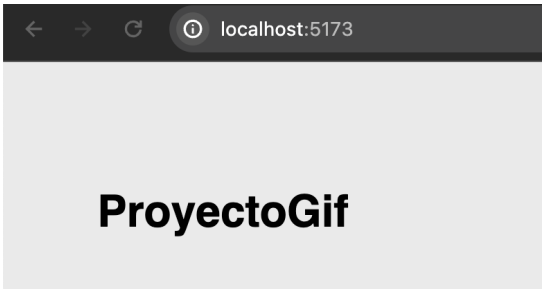
¿Cómo notar que el estilo sí cambió?

Antes



ProyectoGif

Después



¿Qué es hot loader en React?

Usando el Hook useState (función que permite agregar estado de React a los componentes funcionales).

```
export const ProyectoGif = () => {

  //Usando el Hook useState (función que permite agregar estado de React a los componentes funcionales)
  //Para evitar el posible error de Javascript por no definir un valor inicial, se le pone un arreglo que tenga 'One Punch' como valor inicial
  const [categories, setCategories] = useState(['One Punch'])

  return (
```

`const [categories, setCategories] = useState()` es una forma de crear una variable de estado llamada `categories` y una función para actualizar esa variable, `setCategories`, en el componente.

Corrigiendo el error: En la consola del navegador se puede ver el siguiente error:

```
2 Uncaught ReferenceError: useState is not defined  ProyectoGif.jsx?t=1701064754173:20
    at ProyectoGif (ProyectoGif.jsx?t=1701064754173:20:39)
    at renderWithHooks (react-dom.development.js:16305:18)
    at mountIndeterminateComponent (react-dom.development.js:20074:13)
    at beginWork (react-dom.development.js:21587:16)
    at HTMLUnknownElement.callCallback2 (react-dom.development.js:4164:14)
    at Object.invokeGuardedCallbackDev (react-dom.development.js:4213:16)
    at invokeGuardedCallback (react-dom.development.js:4277:31)
    at beginWork$1 (react-dom.development.js:27451:7)
    at performUnitOfWork (react-dom.development.js:26557:12)
    at workLoopSync (react-dom.development.js:26466:5)

▶ The above error occurred in the <ProyectoGif> component:  react-dom.development.js:18687

    at ProyectoGif (http://localhost:5173/src/ProyectoGif.jsx?t=1701064754173:20:39)

Consider adding an error boundary to your tree to customize error handling behavior.
Visit https://reactjs.org/link/error-boundaries to learn more about error boundaries.

Uncaught ReferenceError: useState is not defined  @react-refresh:278
    at ProyectoGif (ProyectoGif.jsx?t=1701064754173:20:39)
    at renderWithHooks (react-dom.development.js:16305:18)
    at mountIndeterminateComponent (react-dom.development.js:20074:13)
```

El error se corrige añadiendo la importación a useState en la primera línea del código y actualizando la página:

```
import { useState} from 'react';
```

En una nueva carpeta llamada “hooks”, se creará el archivo “useFetchGifs.js” que tendrá un hook personalizado llamado “useFetchGifs.js” que se utiliza para obtener imágenes GIF de una categoría específica. Cuando se monta el componente, se llama a la función getImages, que obtiene las imágenes GIF de la categoría y las guarda en el estado images. También establece isLoading en falso para indicar que se han cargado las imágenes. El hook devuelve un objeto con images e isLoading:

```
// Importamos useEffect y useState de React
import { useEffect, useState } from 'react';
// Importamos la función getGifs de helpers
import { getGifs } from '../helpers/getGifs';

// Definimos un hook personalizado llamado useFetchGifs que toma una categoría como argumento
export const useFetchGifs = ( category ) => {
```

```
    // Creamos una variable de estado images y una función para actualizarla setImages
    const [images, setImages] = useState([]);
    // Creamos una variable de estado isLoading y una función para actualizarla
setIsLoading
    const [isLoading, setIsLoading] = useState( true );

    // Definimos una función asíncrona getImages
    const getImages = async() => {
        // Obtenemos nuevas imágenes de la categoría con la función getGifs
        const newImages = await getGifs( category );
        // Actualizamos images con las nuevas imágenes
        setImages(newImages);
        // Establecemos isLoading en falso
        setIsLoading(false);
    }

    // Usamos useEffect para llamar a getImages cuando se monta el componente
    useEffect( () => {
        getImages();
    }, []);

    // Devolvemos un objeto con images e isLoading
    return {
        images,
        isLoading
    }

}
```

Probar en postman la API:

https://developers.giphy.com/docs/api/endpoint/#search

¿Qué es mapear datos?

Se crea la carpeta “helpers” con el archivo “getGifs.js” que tendrá el api key creado en la página de Giphy. Esta función getGifs se utiliza para obtener imágenes GIF de una categoría específica de la API de Giphy. Hace una petición a la API de Giphy con la categoría y un límite de 10 imágenes, extrae los datos de la respuesta, mapea los datos a un nuevo arreglo de objetos con el id, el título y la URL de cada imagen, y devuelve este arreglo. El parámetro “&q” viene de la inicial de la palabra “query” que se refiere a lo que vas a buscar. Recuerda que la API se obtiene de www.developers.giphy.com/dasboard

```
// Definimos una función asíncrona getGifs que toma una categoría como argumento
export const getGifs = async( category ) => {
```

```
    // Construimos la URL para la API de Giphy con la categoría y un límite de 10
imágenes
    const url =
`https://api.giphy.com/v1/gifs/search?api_key=EscribeAquíTuPropiaAPIKey&q=${ category
}&limit=10`;
    // Hacemos una petición a la API de Giphy
    const resp = await fetch( url );
    // Extraemos los datos de la respuesta en formato JSON
    const { data } = await resp.json();

    // Mapeamos los datos a un nuevo arreglo de objetos con id, título y URL de cada
imagen
    const gifs = data.map( img => ({
        id: img.id,
        title: img.title,
        url: img.images.downsized_medium.url
    }));

    // Devolvemos el arreglo de objetos
    return gifs;
}
```

Antes de trabajar con los componentes, el archivo ProyectoGif.jsx irá llamando a cada uno

```
import { useState } from "react";
import { AddCategory, GifGrid } from "./components";

export const ProyectoGif = () => {
  //Usando el Hook useState (función que permite agregar estado de React a los
componentes funcionales)
  //Para evitar el posible error de Javascript por no definir un valor inicial, se le
pone un arreglo que tenga 'One Punch' como valor inicial
  const [categories, setCategories] = useState(["Dragon ball gt"]);
  // Definimos una función onAddCategory que toma un argumento newCategory
  const onAddCategory = (newCategory) => {
    // Si newCategory ya está en la lista de categorías, no hacemos nada y salimos de la
función
    if (categories.includes(newCategory)) return;
    // Si newCategory no está en la lista de categorías, la agregamos al principio de la
lista
    setCategories([newCategory, ...categories]);
  };

  // Devolvemos el JSX que se renderizará
  return (
    <>
```

```
      <h1>ProyectoGifRodrigo</h1>
      {/* // Renderizamos el componente AddCategory y le pasamos una función que se
activará cuando se agregue una nueva
      categoría */}
      <AddCategory onNewCategory={(value) => onAddCategory(value)} />
      {/* // Mapeamos las categorías a componentes GifGrid, cada uno con una clave única
y una categoría */}
      {categories.map((category) => (
        <GifGrid key={category} category={category} />
      ))}
    </>
  );
};
```

Se crea una carpeta llamada “components” que estará dentro de “src” y contendrá cada uno de los componentes a trabajar:

1. AddCategory.jsx. Este código muestra un título, un componente AddCategory que permite al usuario agregar nuevas categorías, y una lista de componentes GifGrid, cada uno correspondiente a una categoría. Cuando se agrega una nueva categoría a través del componente AddCategory, se llama a la función onAddCategory, que agrega la nueva categoría al principio de la lista de categorías (siempre que no esté ya en la lista).

```
// Importamos la función useState de React
import { useState } from 'react';

// Definimos un componente funcional llamado AddCategory
export const AddCategory = ({ onNewCategory }) => {

    // Creamos una variable de estado inputValue y una función para actualizarla
setInputValue
    const [ inputValue, setInputValue ] = useState('');

    // Definimos una función que se activa cuando cambia el valor del campo de entrada
    const onInputChange = ({ target }) => {
        // Actualizamos inputValue con el nuevo valor del campo de entrada
        setInputValue( target.value );
    }

    // Definimos una función que se activa cuando se envía el formulario
    const onSubmit = ( event ) => {
        // Prevenimos la recarga de la página
        event.preventDefault();
        // Verificamos que inputValue no esté vacío o solo contenga espacios en blanco
```

```
        if( inputValue.trim().length <= 1) return;

        // Limpiamos el campo de entrada
        setInputValue('');
        // Llamamos a la función onNewCategory con inputValue como argumento
        onNewCategory( inputValue.trim() );
    }

    // Devolvemos el JSX que se renderizará
    return (
        <form onSubmit={ onSubmit }>
            <input
                type="text"
                placeholder="Buscar gifs"
                // Vinculamos el valor del campo de entrada a inputValue
                value={ inputValue }
                // Cuando cambia el valor del campo de entrada, llamamos a onInputChange
                onChange={ onInputChange }
            />
        </form>
    )
}
```

2. GifGrid.jsx. Este componente GifGrid muestra un título con la categoría, un mensaje de carga si las imágenes están cargando, y una lista de componentes GifItem, cada uno correspondiente a una imagen. Usa el hook personalizado useFetchGifs para obtener las imágenes y el estado de carga.

```
// Importamos el componente GifItem
import { GifItem } from './GifItem';
// Importamos el hook personalizado useFetchGifs
import { useFetchGifs } from '../hooks/useFetchGifs';

// Definimos un componente funcional llamado GifGrid que toma una categoría como prop
export const GifGrid = ({ category }) => {

    // Usamos el hook useFetchGifs para obtener las imágenes y el estado de carga
    const { images, isLoading } = useFetchGifs( category );

    // Devolvemos el JSX que se renderizará
    return (
        <>
            {/* // Mostramos la categoría */}
            <h3>{ category }</h3>
            {/* // Si está cargando, mostramos un mensaje de carga */}
            {
                isLoading && ( <h2>Cargando...</h2> )
```

```
            }

            {/* // Creamos un div con la clase card-grid */}
            <div className="card-grid">
                {/* // Mapeamos las imágenes a componentes GifItem, cada uno con una
clave única y las propiedades de la imagen */}
                {
                    images.map( ( image ) => (
                        <GifItem
                            key={ image.id }
                            { ...image }
                        />
                    ))
                }

            </div>

        </>
    )
}
```

3. GifItem.jsx. Este componente GifItem muestra una imagen y su título. Toma title, url e id como props, aunque id no se utiliza en este componente. La imagen se muestra con src establecido a url y alt establecido a title, y el título se muestra en un elemento p.

```
// Definimos un componente funcional llamado GifItem que toma title, url e id como props
export const GifItem = ({ title, url, id }) => {

  // Devolvemos el JSX que se renderizará
  return (
    // Creamos un div con la clase card
    <div className="card">
        {/* // Mostramos una imagen con src establecido a url y alt establecido a title
*/}
        <img src={ url } alt={ title } />
        {/* // Mostramos el título de la imagen */}
        <p>{ title }</p>
    </div>
  )
}
```

Para que sea posible que el archivo “ProyectoGif.jsx” pueda importar los componentes, en la carpeta “componets” se creará “index.js” que tendrá las exportaciones:

```
export * from './AddCategory';
export * from './GifGrid';
export * from './GifItem';
```

Así quedaría la carpeta “components”