

# FullStack Developer

---

React – Proyecto vite-react y despliegue

Ya estuvimos viendo como manejar el consumo de apis y el dom con react-router-dom, esto ya nos permite armar cualquier tipo de proyecto que escojamos, pero así como en su momento vimos que tenemos librerías para JavaScript Vanilla, también existen un número importante de librerías para React que vamos a instalar con “npm”



## Recordemos que es NPM

**NPM** (Node Package Manager) es el gestor de paquetes predeterminado para el entorno de ejecución de **Node.js**. Fue diseñado originalmente para ayudar a los desarrolladores a compartir y reutilizar código de manera eficiente, facilitando la instalación, actualización y administración de dependencias de un proyecto.

## ¿Qué es NPM?

NPM es una herramienta de línea de comandos que permite gestionar librerías, frameworks y otros módulos desarrollados por la comunidad o por el propio equipo del proyecto. Esto significa que, en lugar de tener que reinventar la rueda en cada proyecto, podés instalar paquetes que ya han sido desarrollados y probados por otros, acelerando el proceso de desarrollo y mejorando la calidad del software.

Vamos a instalar algunas librerías para ver como acoplarlas a un proyecto de react.

En este ejemplo para ahorrar tiempo usaremos un proyecto que estuvimos trabajando en clases anteriores

App World Cup 2022 Football

typewriter-effect

2.21.0 • Public • Published a year ago

Readme

Code Beta

2 Dependencies

60 Dependents

48 Versions

TypewriterJS v2

NO BUILDS

A simple yet power

Install

> npm i typewriter-effect

Repository

github.com/tameemsafi/typewriterjs

Homepage

github.com/tameemsafi/typewriterjs#r...

Weekly Downloads

Empecemos instalando

```
npm i typewriter-effect
```

Empecemos instalando “typewriter-effect”, con esta librería podemos realizar un efecto como si estaríamos escribiendo sobre nuestro sitio web.

Implementación de typewriter-effect, un ejemplo básico

```
import Typewriter from 'typewriter-effect';

const TypewriterExample = () => {
  return (
    <Typewriter
      options={{
        strings: ['Hola', 'Hola Mundo!', 'Bienvenidos a React!'],
        autoStart: true,
        loop: true,
        delay: 75,
      }}
    />
  );
};

export default TypewriterExample;
```

## Explicación del código:

- strings**: Es un array de cadenas de texto que aparecerán con el efecto de máquina de escribir.
- autoStart**: Hace que la animación comience automáticamente cuando el componente se monta.
- loop**: Si está en true, el texto se repetirá indefinidamente.
- delay**: Controla la velocidad de la animación (en milisegundos) entre cada letra escrita.

## Personalización adicional:

- pauseFor**: Puedes agregar pausas específicas después de que se complete una cadena de texto.
- deleteSpeed**: Controla la velocidad a la que el texto es "borrado".
- cursor**: Puedes cambiar el estilo del cursor de la animación (por defecto es un |).

```
<Typewriter
  options={{
    strings: ['Hola', 'Hola Mundo!', 'Bienvenidos a React!'],
    autoStart: true,
    loop: true,
    delay: 75,
    deleteSpeed: 50,
    pauseFor: 2000,
    cursor: '_',
  }}
/>
```

En el desarrollo de nuestra aplicación, nosotros podemos optar por muchas librerías similares para animar textos y otros elementos, cada una con sus propias características y ventajas. Librerías como **Typewriter-Effect**, **React-Typical** o **React Spring**, por ejemplo, nos ofrecen diferentes formas de implementar animaciones tipo máquina de escribir, transiciones fluidas o efectos de texto en loop.

La elección de la librería depende de nuestras necesidades específicas: si buscamos simplicidad y rapidez, **Typewriter-Effect** es ideal; si necesitamos animaciones más complejas y personalizables, **React Spring** o **Framer Motion** pueden ser opciones más adecuadas. Lo importante es que contamos con una amplia gama de herramientas que podemos ajustar a los requerimientos de nuestro proyecto, optimizando tanto el rendimiento como la experiencia visual de la aplicación.

# PrimeReact

**PrimeReact** es una colección completa de componentes de **UI** (interfaz de usuario) para aplicaciones **React**. Es una librería muy utilizada porque ofrece una amplia variedad de componentes listos para usar, como botones, tablas, formularios, calendarios, gráficos y muchos más. Todos estos componentes están diseñados para ser altamente personalizables y fáciles de integrar.

Lo que distingue a **PrimeReact** es su enfoque en la accesibilidad, el rendimiento y su conjunto de temas prediseñados, que facilitan el desarrollo de aplicaciones con una apariencia profesional sin demasiado esfuerzo. También ofrece integración con **PrimeIcons**, que es un conjunto de íconos optimizados para funcionar perfectamente con los componentes.



## ¿Por qué usar PrimeReact?

- **Amplia variedad de componentes:** Ofrece más de 80 componentes como botones, cuadros de diálogo, menús, tablas, selectores de fecha, entre otros.
- **Temas personalizables:** Incluye temas preconstruidos como "Luna", "Rhea" o "Saga", y permite personalizar completamente el diseño según nuestras necesidades.
- **Compatible con mobile y responsive:** Los componentes están optimizados para adaptarse bien en dispositivos móviles y ser responsive.

## 1. Instalar PrimeReact

Ejecutamos el siguiente comando para agregar **PrimeReact** a nuestro proyecto de React:

```
npm install primereact primeicons
```

## 2. Agregar un tema de PrimeReact

PrimeReact utiliza un sistema de temas para el estilo de los componentes. Podemos elegir un tema predeterminado como parte de la instalación. Por ejemplo, si queremos usar el tema "Saga", necesitamos importarlo en nuestro archivo principal src/App.js:

```
import Layout from "../Layout"
import { BrowserRouter as Router, Routes, Route } from "react-router-dom"
import Home from "../pages/Home"
import Resultados from "../pages/Resultados"
import 'bootstrap/dist/css/bootstrap.min.css'
import 'bootstrap/dist/js/bootstrap.bundle.min.js'
import 'primereact/resources/themes/saga-blue/theme.css'; // Tema de PrimeReact
import 'primereact/resources/primereact.min.css'; // Estilos de los componentes
import 'primeicons/primeicons.css'; // Íconos

function App() {

  return (
    <Router>
      <Layout>
        <Routes>
          <Route path="/" element={<Home/>} />
          <Route path="/partidos" element={<Resultados/>} />
        </Routes>
      </Layout>
    </Router>
  )
}

export default App
```

Podemos elegir otros temas simplemente cambiando el archivo de tema en la ruta. Por ejemplo:

- saga-blue
- arya-orange
- vela-green
- luna-amber

### 3. Uso de componentes

Una vez instalado, ya podemos empezar a utilizar los componentes de **PrimeReact**. A continuación, un ejemplo simple usando un botón de PrimeReact:

```
import { Button } from 'primereact/button';

const Example = () => {
  return (
    <div>
      <Button label="Click Me" icon="pi pi-check" />
    </div>
  )
}

export default Example
```



✓ Click Me

#### 4. Documentación y personalización

Para aprovechar al máximo **PrimeReact**, nosotros podemos consultar su documentación oficial, que contiene ejemplos detallados de cada componente, así como opciones de personalización para adaptarlos a nuestras necesidades. También es posible personalizar los temas si queremos que coincidan con el estilo único de nuestra aplicación.

Con **PrimeReact**, tenemos una herramienta poderosa y flexible para crear interfaces de usuario completas y sofisticadas en aplicaciones React de manera rápida y eficiente.

Para crear una **tabla dinámica** en **React** que permita mostrar datos y actualizarse de manera interactiva, podemos utilizar la librería **PrimeReact**, que incluye componentes avanzados para tablas, como DataTable, con soporte para paginación, filtrado, ordenación y más.

Vamos a modificar nuestra tabla en el componente Resultados.jsx

```
const Resultados = () => {

  const { data, loading, error } = useFetchData('http://localhost:8000/api/partidos')

  return (
    <table class="table">
      <thead>
        <tr>
          <th scope="col">#</th>
          <th scope="col">Goles Local</th>
          <th scope="col">Equipo Local</th>
          <th scope="col">Goles Visitante</th>
          <th scope="col">Equipo Visitante</th>
        </tr>
      </thead>
      <tbody>
        {data.map(resultado =>
          <tr key={resultado.id}>
            <th scope="row">={resultado.id}</th>
            <th>={resultado.goles_local}</th>
            <td>
              {resultado.equipo_local.nombre}</td>
            <td>{resultado.goles_visitante}</td>
            <td>{resultado.equipo_visitante.nombre}</td>
          </tr>
        )}
      </tbody>
    </table>
  )
}

export default Resultados
```





Ahora vamos a crear la lógica necesaria para activar el dark-mode en nuestro sitio. Utilizaremos un helper que se encargará de modificar todos los elementos del DOM que sean necesarios, mientras que en React solo manejaremos el estado de la variable.

```
import { useState, useEffect } from 'react';
import { SelectButton } from 'primereact/selectbutton';
import toggleThemeClasses from '../helpers/toggleThemeClasses';

const ThemeSwitcher = () => {
  const [theme, setTheme] = useState('light'); // Estado del tema (claro u oscuro)

  // Opciones para el SelectButton (Modo Claro/Oscuro)
  const themeOptions = [
    { label: 'Light', value: 'light' },
    { label: 'Dark', value: 'dark' }
  ];

  // Cambiar la clase del body según el tema seleccionado
  useEffect(() => {
    toggleThemeClasses(theme)
  }, [theme]);

  return (
    <div className="theme-switcher">
      <SelectButton value={theme} options={themeOptions} onChange={(e) => setTheme(e.value)} />
    </div>
  );
};

export default ThemeSwitcher;
```

Con este código anterior manejamos el estado y con el helper todo el manejo del DOM

```
// Helper para cambiar las clases del modo claro/oscuro usando classList
const toggleThemeClasses = (theme) => {
  const isDarkMode = theme === 'dark';

  // Cambiar clases del body
  document.body.classList.toggle('bg-dark', isDarkMode);
  document.body.classList.toggle('text-light', isDarkMode);
  document.body.classList.toggle('bg-light', !isDarkMode);
  document.body.classList.toggle('text-dark', !isDarkMode);

  // Cambiar clases del header
  const header = document.querySelector('header');
  console.log(header)
  if (header) {
    header.classList.toggle('bg-dark', isDarkMode);
    header.classList.toggle('text-light', isDarkMode);
    header.classList.toggle('bg-light', !isDarkMode);
    header.classList.toggle('text-dark', !isDarkMode);
  }

  // Nav Links
  const navLinks = document.querySelectorAll('.nav-link');
  navLinks.forEach((link) => {
    link.classList.toggle('text-white', isDarkMode);
    link.classList.toggle('text-dark', !isDarkMode);
  });
};

export default toggleThemeClasses
```

# Pasando a producción

Para pasar la aplicación a producción usando **Render.com**, veamos esta guía paso a paso sobre cómo desplegar una aplicación web, ya sea un frontend (como React) o un backend (como Node.js, Django, Laravel, etc.). Es decir, esta guía nos va a servir a futuro con los temas que veremos.

## Crear una cuenta en Render

1. Vamos a [Render.com](https://render.com).

2. Iniciamos sesión con GitHub.

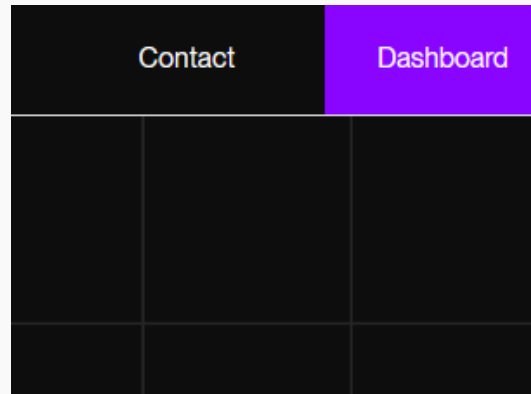
1. **Render** se integra directamente con GitHub para facilitar el proceso de despliegue.

## Conectar tu repositorio de GitHub

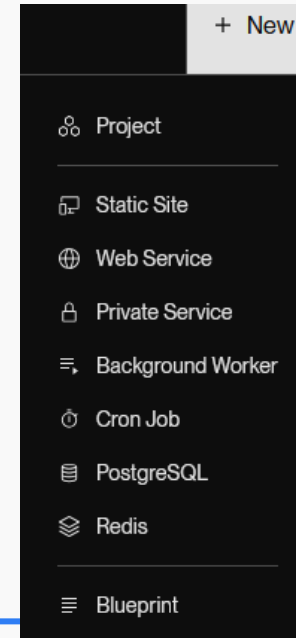
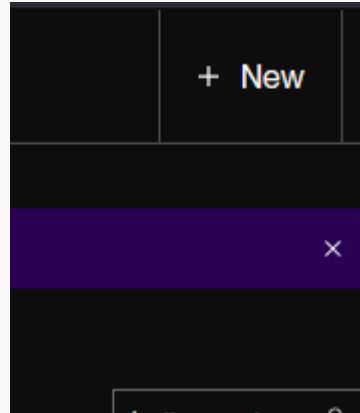
1. Una vez que hayas iniciado sesión, deberás conectar tu cuenta de **Render** con tu repositorio de **GitHub**.

2. Desde el tablero de Render, haz clic en el botón "New" y selecciona "Web Service" para desplegar una aplicación web.

Hacemos click en Dashboard



Hacemos click en New  
y luego en Static Site



Va a tardar un momento para que carguen los proyectos de GitHub y luego aparecerá completa esta vista

## You are deploying a Static Site

Source Code

Git Provider

Public Git Repository

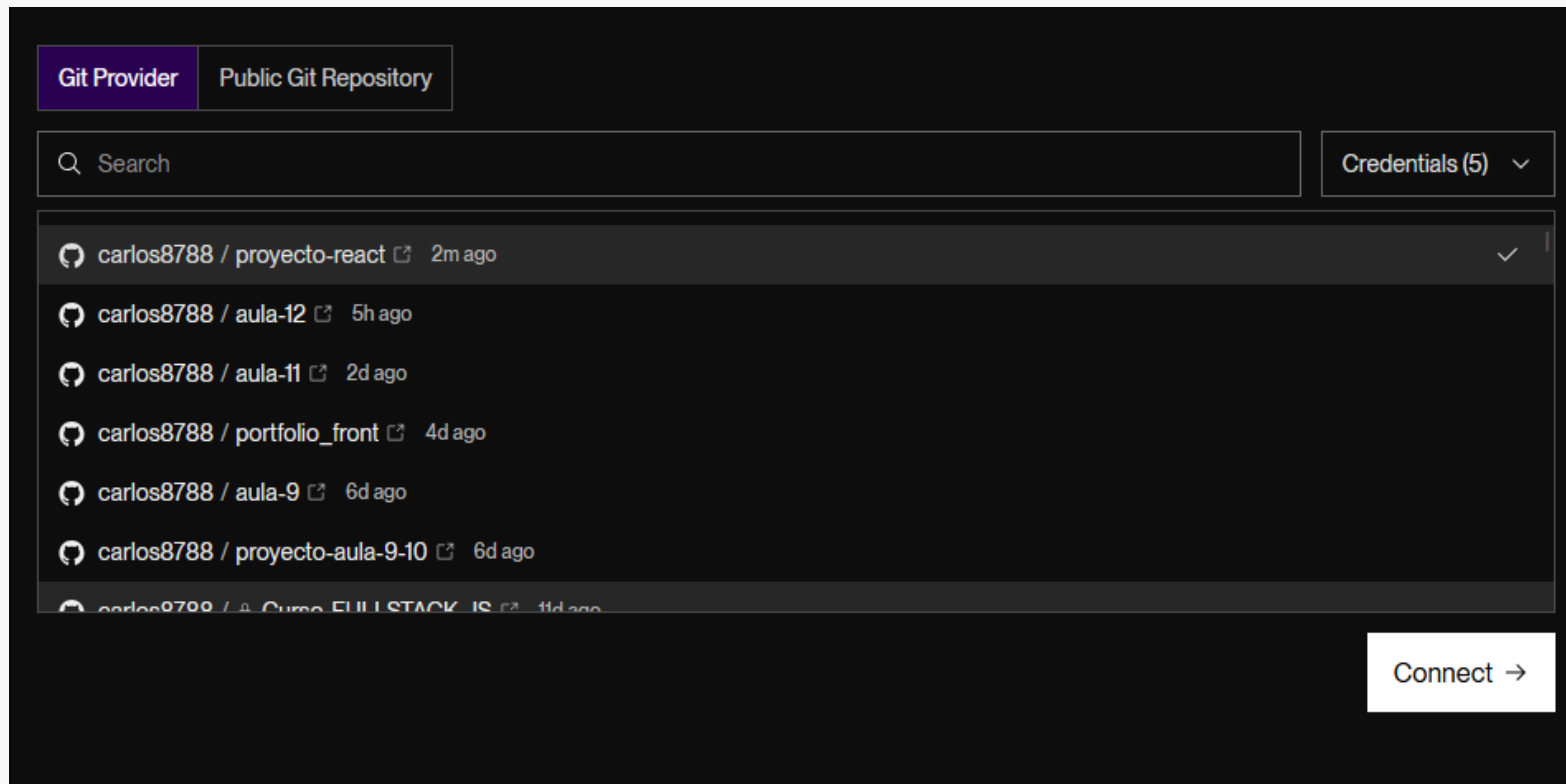
Search

Credentials (5) ▾

- carlos8788 / proyecto-react 1m ago
- carlos8788 / aula-12 5h ago
- carlos8788 / aula-11 2d ago
- carlos8788 / portfolio\_front 4d ago
- carlos8788 / aula-9 6d ago
- carlos8788 / proyecto-aula-9-10 6d ago
- carlos8788 / Cump FULLSTACK JS 1d ago

Connect →

Una vez que elijamos el proyecto, aparecerá habilitado el botón Connect, lo presionamos.



Vamos a pasar a un panel de configuración, en este caso nos centraremos en lo esencial para que funcione correctamente

**Source Code**

carlos8788 / proyecto-react • 3m agoEdit


**Name**

A unique name for your static site.

proyecto-react

**Project** Optional

Add this static site to a [project](#) once it's created.

**Create a new project to add this to?**

You don't have any projects in this workspace. [Projects](#) allow you to group resources into environments so you can better manage related resources.

+ Create a project

**Branch**

The Git branch to build and deploy.

master

**Root Directory** Optional

If set, Render runs commands from this directory instead of the repository root. Additionally, code changes outside of this directory do not trigger an auto-deploy. Most commonly used with a [monorepo](#).

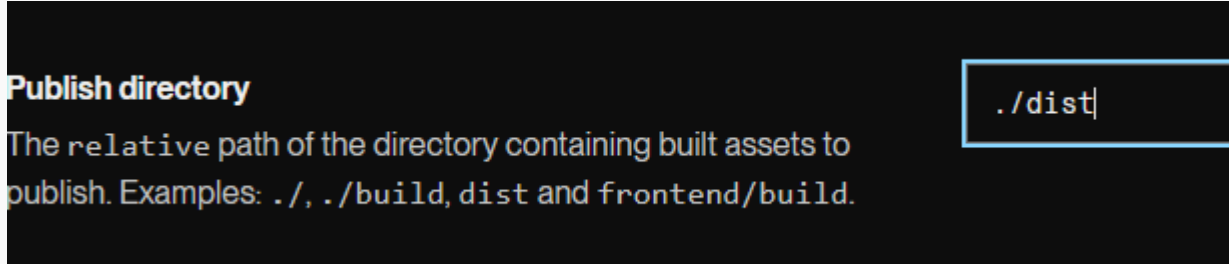
e.g. src

**Build Command**

Render runs this command to build your app before each deploy.

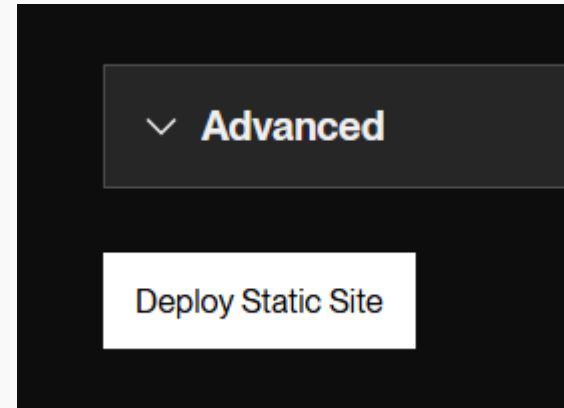
\$ npm install; npm run build

Solamente vamos a completar el campo "Publish directory" con `./dist`



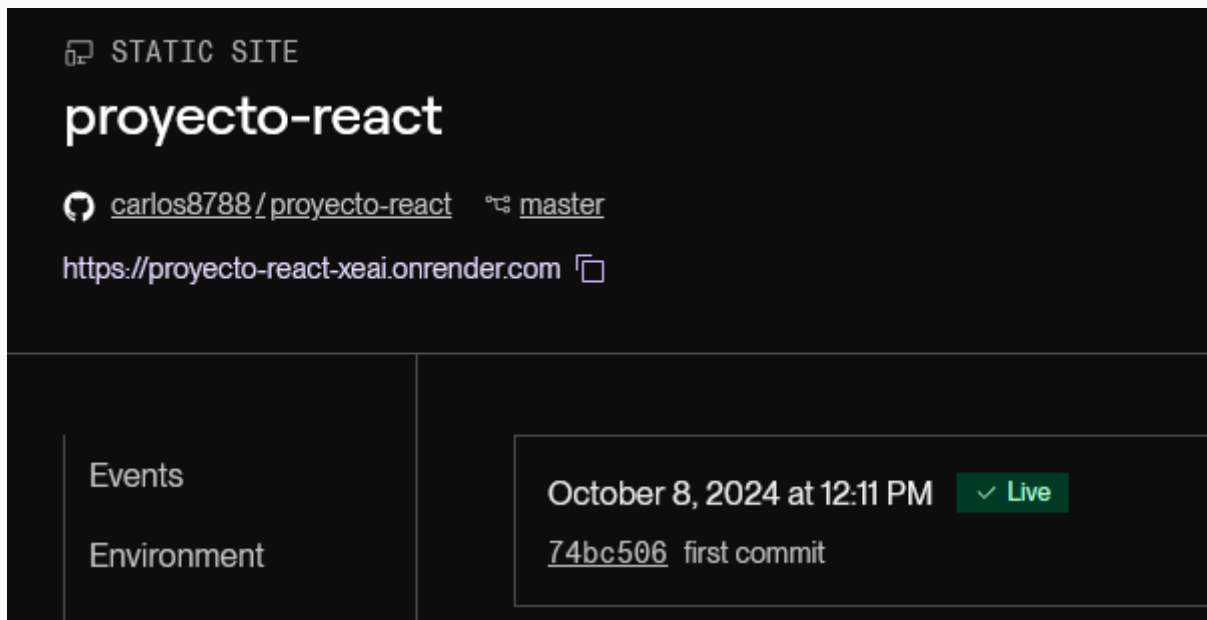
Ya que cuando vite hace el proceso de building, almacena esto en la carpeta dist/

Y para finalizar presionamos en Deploy Static Site





Vamos a ver una consola que hace el proceso automático de instalar las dependencias y demás



Una vez que el proceso termine nos aparecerá la URL y además un cartel con la fecha y un recuadro verde indicando que la app está funcionando correctamente

# PROYECTO REACT A PRESENTAR

Vas a crear un proyecto de React siguiendo todas las cosas que vimos, como sugerencia, te recomiendo hacer una app de tu portfolio, debido a que el backend que veremos será orientado a este. Eso no limita a que intentes hacer otro tipo de app, lo único que se pide es que por lo menos incluya los temas vistos, modularizar la app, react-router-dom y alguna librería de tu agrado. Por último, desplegarla en render.com (aclaración, el proyecto debe estar subido a un repositorio de GitHub como público).