Ejercicios React

[**1. Software de base 2**](#_l4raksjmxsrk)

[Plugins de VSC 2](#_f1sjqfvzvgag)

[Comandos utiles 2](#_cu1yne99dw5e)

[**2. Setup local de un CRUD preexistente 2**](#_e7v0mbdo6rel)

[Ejercicios para el caso presentado 7](#_j49nkc1lxjsd)

[**3. Challenges técnicos 7**](#_8vw7nz2521a3)

[3.a) Chat con efectos 7](#_cv3psom9fais)

[3.b) State as a Snapshot 7](#_axyi4h3upg2d)

[3.c) Pasar una función como prop 8](#_gjjt89n9ovb2)

[3.d) JSX in depth 8](#_dh0u5xdkraan)

[**4. Proyectos de cero 8**](#_4jci9zu401xc)

[4.a) Tic-tac-toe to hangman 8](#_6n7znj9c170o)

[4.b) CRUD 3 capas 8](#_stkmlftsed6n)

[4.c) React con servicios REST y CORS 8](#_9xd2pt65r3b0)

[**5. Setup con vite 9**](#_m086d8tnfi7t)

[**Anexo I. Arrancar al dia siguiente 10**](#_khnuyiazl1au)

[**Anexo II. React Chrome developer toolbar 11**](#_k33m5bllls7)

# 

# 1. Software de base

React es un framework de lado cliente pero usaremos diferentes productos de servidor para los endpoints. Inicialmente usaremos NodeJS: <https://nodejs.org/en/download> (en el store corporativo buscar por “node”)

## Plugins de VSC

Existen varios plugins de React para VSC sin embargo no añaden mucha funcionalidad, vamos a instalar el marcado como “starred” en este listado.

## 

## Comandos utiles

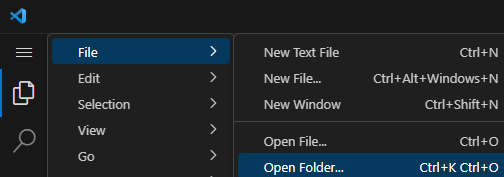
| **Utilidad** | **Comando** |
| --- | --- |
| Actualizar npm (para instalaciones antiguas) | **npm install -g npm** |
| Especificar la ruta donde npm se baja dependencias | **npm install --prefix ../../node\_modules** |

# 2. Setup local de un CRUD preexistente

Arrancar Visual Studio Code (VSC)

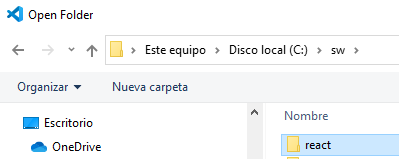


Especificar la carpeta que queremos abrir

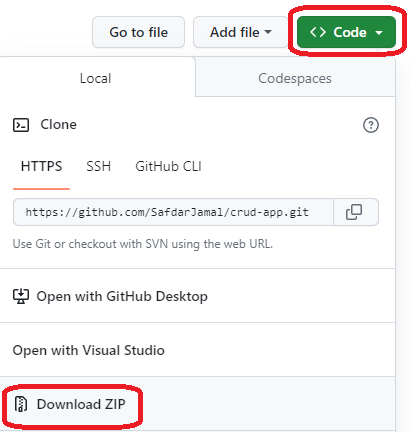


Vamos a usar algo similar a lo siguiente (una ruta corta, sin espacios en blanco, directamente en el disco principal)

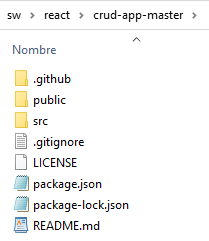
**c:/sw/react**

****

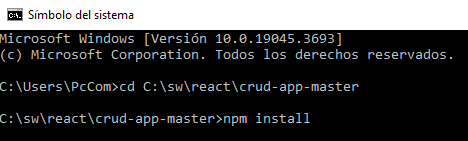
Descargar [este repositorio](https://github.com/SafdarJamal/crud-app/) de GitHub, para descargarlo hay un botón de descarga que se visualiza si pulsamos en el botón verde:



Al descomprimirlo debería quedar así (no es obligatorio pero es importante usar rutas cortas en desarrollo, no usar **/Desktop**, ni **/Users** ni nada de eso)



Esta aplicación corre sobre NodeJS, abrir una ventana cmd para ejecutar NPM (npm es el gestor de paquetes y nodeJS el servidor que arrancamos)



**npm install**

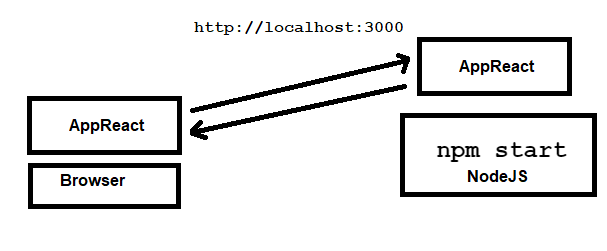
**npm start**

| Nota: Si al arrancar el servidor da un fallo puede ser por multiples motivos, en el PC del formador se quejaba del soporte SSL    Para evitar el problema podemos poner opciones al servidor de NodeJS  **set NODE\_OPTIONS=--openssl-legacy-provider** |
| --- |

Podemos probar el funcionamiento de la aplicación escribiendo en Google Chrome la siguiente URL

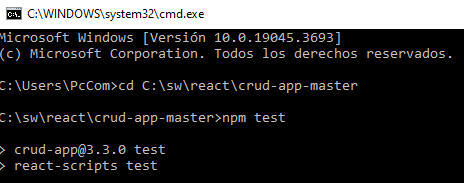
<http://localhost:3000>

Al acceder desde el navegador se descargan los “activos” web del proyecto de React que se ejecutan sobre el navegador (el servidor solo se usa para “servir” los recursos al cliente)

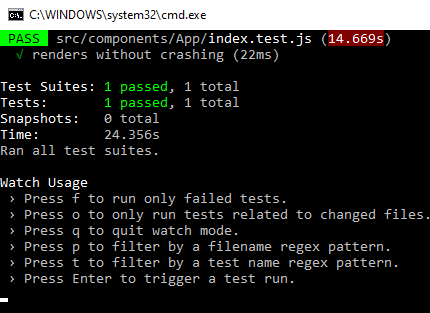


Vamos a ejecutar también el test que viene con el proyecto

**npm test**

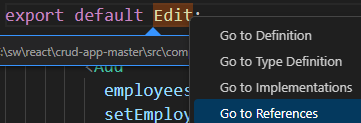


Deberíamos tener éxito en el único test existente

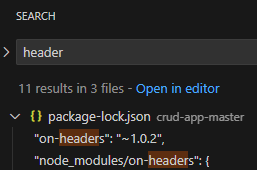


## Ejercicios para el caso presentado

1. Usar VSC para navegar el proyecto usando CTRL, evitar usarlo en tipos nativos de React o el elementos que no sean parte de la jerarquía o composición de React del propio ejemplo (es decir, queremos navegar los componentes Type, Edit, Table,...)
2. Usa “Botón derecho → “Go to references” para encontrar uso de components en otros components



1. Para algunas referencias que VSC no descubre automáticamente tenemos que usar Search



1. Identificar dónde está programada la prueba
2. Por último diagramar todo el caso práctico usando la herramienta que desees (Visio, Paint, Gimp, StarUML, Diagrams.net,...)

# 3. Challenges técnicos

## 3.a) Chat con efectos

Analizar cada caso del uso de efectos presentados en estos ejemplos y diagramarlos individualmente

<https://react.dev/reference/react/useEffect#examples-connecting>

## 3.b) State as a Snapshot

Vamos a estudiar en detalle el caso de state como un snapshot que se crea durante la fase de render, esto se entiende mejor sí se compara con el diagrama de ciclo de vida que vimos en teoría.

<https://react.dev/learn/state-as-a-snapshot>

## 3.c) Pasar una función como prop

Las lambdas se pueden pasar como parámetros para que se ejecuten “a posteriori”, analizar el caso presentado en este link

<https://react.dev/reference/react/Children#calling-a-render-prop-to-customize-rendering>

## 3.d) JSX in depth

Solucionar estos retos y analizar la utilidad del fallo (como de fiable es la descripción del problema en relación al problema real) en cada caso

<https://react.dev/learn/javascript-in-jsx-with-curly-braces#challenges>

# 4. Proyectos de cero

## 4.a) Tic-tac-toe to hangman

El tic-tac-toe (3 en raya) es el [tutorial básico de react](https://react.dev/learn/tutorial-tic-tac-toe). Vamos a programar el proyecto entero, y después vamos a usarlo como ejemplo para montar nuestro propio juego del ahorcado. Iremos copiando trozos de código y adaptándolo para montar el juego completo

¿Qué métodos “de negocio” debería tener la clase que implemente la funcionalidad del juego?

Intenta programar de cero el caso práctico (existen varias soluciones en Internet pero no las uses para la implementación).

## 4.b) CRUD 3 capas

Monta un crud de 3 capas desde cero sobre un ámbito de negocio que te resulte interesante (videoclub, gestor de videojuegos, música), no hace falta montar backend, se puede simular con JSONs en el lado cliente.

* FÁCIL: Podemos hacer enfoque sustractivo: usar el proyecto de la práctica 1 para adaptarlo a nuestro propio
* DIFÍCIL: Podemos hacer enfoque aditivo: Empezar de cero (o de un scaffolding muy básico) y vamos añadiendo a base de trozos de código

## 4.c) React con servicios REST y CORS

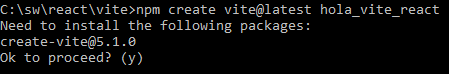
Fuente: <https://www.stackhawk.com/blog/react-cors-guide-what-it-is-and-how-to-enable-it/#enable-cors-on-server-side>

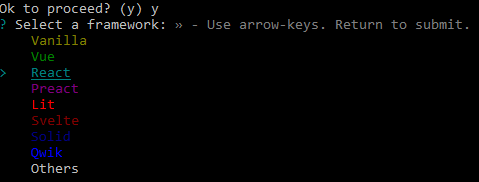
CORS no es realmente un problema de lado de cliente pero vamos a programar una aplicación React que sirva de front a una aplicación faceless implementada con soporte CORS.

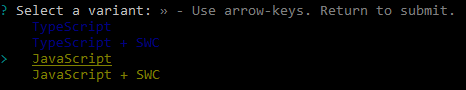
# 5. Setup con vite

**cd c:\sw\react\vite**

**npm create vite@latest hola\_vite\_react**



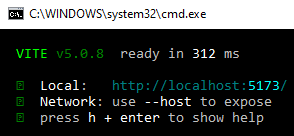




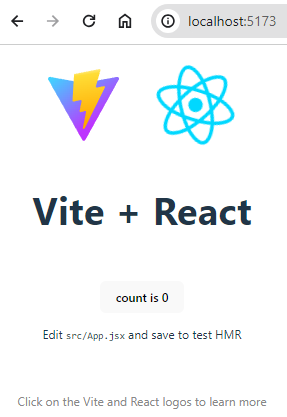
**npm install**

**npm run dev**

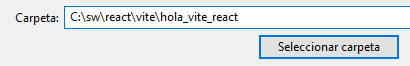
Una vez ejecutado debería salir la siguiente pantalla



Y la página plantillada por Vite debería ser accesible en el navegador



Vamos a arrancar VSC sobre la carpeta con el scaffolding de Vite

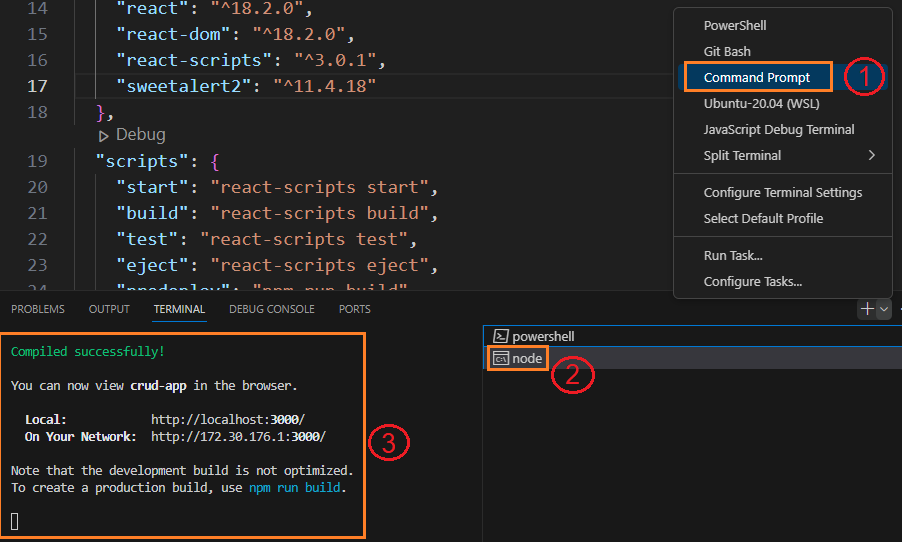


# Anexo I. Arrancar al dia siguiente

No todos los pasos que hacemos es necesario repetirlos al dia siguiente, solo el arranque del servidor (el npm install es el que puebla la carpeta node\_modules de librerías JS y ‘compila’ los “chunks js” derivados de nuestro código React)

| Abrir el CMD    **cd c:/sw/react/crud-app-master**  **set NODE\_OPTIONS=--openssl-legacy-provider**  **npm start** |
| --- |

Se puede lanzar también desde el propio VSC

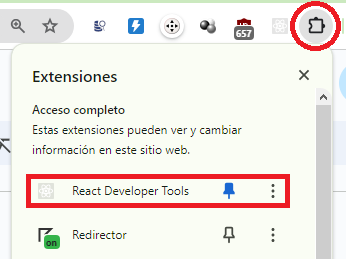
****

# Anexo II. React Chrome developer toolbar

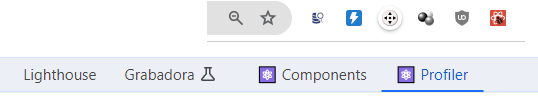
Al abrir el inspector se propone la descarga de un plugin de Chrome, Firefox o Edge que facilita la identificación de Components de React

El link de descarga es este: <https://react.dev/learn/react-developer-tools>

Una vez instalado el plugin del navegador correspondiente debemos activarlo



Y una vez activo veremos en el Developer Toolbar dos nuevas opciones: Components y Profiler



La pestaña de Components por ejemplo permite visualizar la página con resaltado del área ocupado por Components de React

