SERVICIO NACIONAL DE APRENDIZAJE – SENA

Construcción de software integrador de tecnologías orientadas a servicios.

Competencia:

Desarrollo WEB – ReactJS o AngularJS.

Ficha:

3223875

Aprendiz:

Evelin Maria Moreno Freyles

Instructor:

Luis Fernando Sánchez

Medellín-Antioquia

04/11/2025

TABLA DE CONTENIDO

INTRODUCCIÓN

El desarrollo de aplicaciones web modernas requiere no solo conocimientos en programación, sino también una gestión eficiente de las herramientas y librerías necesarias para construir software de calidad. La correcta administración de dependencias y el uso de gestores como npm son fundamentales para garantizar la estabilidad, escalabilidad y mantenimiento de los proyectos en JavaScript.

Este documento tiene como objetivo conceptualizar los procesos de gestión de dependencias, el versionado semántico y el uso de herramientas de control de versiones, así como mostrar de manera práctica la instalación y configuración de un proyecto utilizando React y Vite. Con este análisis, se busca fortalecer las competencias en desarrollo web, fomentar buenas prácticas de programación y promover la organización y trazabilidad del código, aspectos esenciales en el ámbito académico y profesional.

3.2 Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje:

GA1-220501096-03-AA1-EV02-Gestion de dependencias de código según lenguaje de programación seleccionado (JavaScript).

Definición de los siguientes términos sobre los procesos de gestión de dependencias en JavaScript:

¿Qué es un gestor dependencias de código?

Un gestor de dependencias de código es una herramienta utilizada para descargar, instalar, actualizar y administrar de manera automática las librerías o paquetes externos que un proyecto requiere para su correcto funcionamiento.

En términos simples, cuando un programa necesita componentes desarrollados por terceros (como frameworks, módulos o librerías), el gestor de dependencias se encarga de:

- **Localizar** las dependencias necesarias en un repositorio (por ejemplo: npm, pip, Maven, entre otros).
- **Instalarlas** en el proyecto con la versión adecuada.
- **Actualizar** dichas dependencias cuando sea necesario.
- **Prevenir** conflictos entre versiones de librerías.
- **Registrar** las dependencias en un archivo de configuración (como package.json o requirements.txt).

Ejemplos de gestores según el lenguaje de programación:

- **JavaScript / Node.js:** npm, yarn
- **Python:** pip, poetry
- **Java:** Maven, Gradle
- **PHP:** Composer
- **.NET (C#):** NuGet

En conclusión, los gestores de dependencias permiten optimizar el trabajo del desarrollador**,** facilitando la organización del proyecto y garantizando que todas las librerías necesarias estén disponibles, actualizadas y correctamente configuradas.

¿Qué es npm?

npm (Node Package Manager) es el gestor de dependencias oficial de Node.js**,** utilizado para **administrar, instalar y actualizar paquetes o librerías de JavaScript** que permiten incorporar funcionalidades al proyecto sin necesidad de desarrollarlas desde cero.

Entre sus principales funciones se incluyen:

- **Descargar paquetes** desde el repositorio oficial de npm (https://www.npmjs.com).
- **Instalar dependencias** necesarias para el proyecto mediante comandos simples.
- **Registrar** todas las dependencias y scripts del proyecto en un archivo de configuración (package.json).
- **Publicar paquetes propios** para que puedan ser utilizados por otros desarrolladores.

Ejemplo de uso:

npm install express

Este comando instala la librería **Express**, ampliamente utilizada para crear servidores web en Node.js.

En conclusión, npm facilita la gestión de librerías externas en proyectos de JavaScript, contribuyendo a que el código sea más organizado, modular y reutilizable.

¿Para qué se utiliza principalmente npm?

npm se utiliza principalmente para gestionar librerías y paquetes en proyectos de JavaScript**.** Su función principal es facilitar la instalación, actualización y organización de dependencias**,** permitiendo que los desarrolladores:

1. **Agreguen funcionalidades** a sus aplicaciones sin programarlas desde cero, usando paquetes ya existentes.
2. **Mantengan ordenadas las dependencias** del proyecto mediante el archivo package.json.
3. **Actualicen y gestionen versiones** de las librerías para evitar conflictos.
4. **Publiquen sus propios paquetes** para compartirlos con otros desarrolladores.

En pocas palabras**:** npm permite que los proyectos de JavaScript sean más eficientes, modulables y fáciles de mantener**.**

¿Qué es el versionado semántico?

El **versionado semántico** (o Semantic Versioning, abreviado como **SemVer**) es un sistema para asignar números de versión a software o librerías de manera estructurada y predecible.

Se basa en un formato de **tres números**:

MAJOR.MINOR.PATCH

Donde:

- **MAJOR**: se incrementa cuando se realizan cambios incompatibles que rompen la compatibilidad con versiones anteriores.
- **MINOR**: se incrementa al agregar nuevas funcionalidades de forma compatible con versiones anteriores.
- **PATCH**: se incrementa al corregir errores o bugs sin afectar la funcionalidad existente.

Ejemplo:

2.5.1

- 2 → versión mayor
- 5 → versión menor
- 1 → corrección de errores

En resumen, el versionado semántico ayuda a los desarrolladores y usuarios a entender rápidamente el tipo de cambios que se han hecho en una versión y a gestionar dependencias de manera segura**.**

¿Como está especificado el Versionado Semántico?

El **Versionado Semántico** está especificado mediante la convención **SemVer (Semantic Versioning 2.0.0)**, la cual define un formato y reglas claras para asignar números de versión al software en tres números:

MAJOR.MINOR.PATCH.

Cada número tiene un propósito específico:

- **MAJOR:** se incrementa cuando se introducen cambios incompatibles con versiones anteriores.
- **MINOR:** se aumenta al añadir nuevas funcionalidades compatibles.
- **PATCH:** se modifica al corregir errores sin alterar el funcionamiento del software.

Reglas adicionales:

- Se pueden agregar **etiquetas prelanzamiento** (por ejemplo 1.0.0-alpha) para versiones en desarrollo.
- Se pueden usar **metadatos de compilación** (por ejemplo 1.0.0+build2025) para información adicional que no afecta la compatibilidad.
- Los incrementos deben reflejar claramente el **impacto de los cambios** en los usuarios y dependencias del software.

En conclusión, el Versionado Semántico está estructurado para comunicar de manera precisa el tipo de cambios realizados en una versión, facilitando la gestión de dependencias y la actualización segura de software.

¿Qué son las dependencias locales?

Las dependencias locales son aquellas librerías o módulos instalados directamente dentro del proyecto en el que se están utilizando, en lugar de estar disponibles de forma global en el sistema.

En otras palabras, se guardan en una carpeta específica del proyecto (por ejemplo, node_modules en Node.js), y **solo afectan ese proyecto**, sin interferir con otros.

Características principales:

- Se instalan dentro del directorio del proyecto.
- Se registran en el archivo de configuración (como package.json).
- Permiten que el proyecto funcione igual en cualquier equipo, siempre que se instalen las mismas dependencias.

Ejemplo (con npm):

```
npm install express
```

Esto instala la librería **Express** como una dependencia local del proyecto actual.

En resumen, las dependencias locales garantizan que cada proyecto tenga sus propias versiones de librerías, evitando conflictos y asegurando la reproducibilidad del entorno de desarrollo.

¿Qué son las dependencias de desarrollo?

Las dependencias de desarrollo son aquellas librerías o herramientas necesarias solo durante el proceso de creación y prueba del s oftware, pero no son requeridas cuando la aplicación se ejecuta en producción**.**

Estas dependencias ayudan en tareas como la compilación, el testeo, la depuración o el análisis del código.

Ejemplos comunes:

- **Frameworks de prueba:** Jest, Mocha.
- **Herramientas de compilación:** Babel, Webpack.
- **Analizadores o linters:** ESLint, Prettier.

Ejemplo (con npm):

npm install jest --save-dev

El modificador --save-dev indica que la librería se usará solo durante el desarrollo.

En resumen, las dependencias de desarrollo permiten automatizar y mejorar el proceso de construcción del software, sin afectar la ejecución del programa final.

¿Qué son las dependencias globales?

Las **dependencias globales** son aquellas librerías o paquetes instalados de forma general en el sistema operativo**,** y no dentro de un proyecto específico.

Esto significa que pueden ser utilizadas por cualquier proyecto o desde cualquier ubicación del sistema, sin necesidad de instalarlas nuevamente.

Características principales:

- Se instalan una sola vez en el sistema.
- Están disponibles para todos los proyectos.
- Son útiles para herramientas que se usan frecuentemente, como frameworks, gestores o utilidades de línea de comandos.

Ejemplo (con npm):

npm install -g nodemon

El modificador -g indica que la instalación es **global**, y el paquete estará disponible para todos los proyectos en el equipo.

En resumen, **las dependencias globales** facilitan el acceso a herramientas compartidas**,** pero deben usarse con precaución, ya que diferentes versiones pueden generar incompatibilidades entre proyectos**.**

¿Qué es el archivo package.json , que apartados tiene y cuál es su utilidad?

El archivo **package.json** es un **documento de configuración esencial en los proyectos de Node.js**, que contiene la información básica del proyecto y las dependencias necesarias para su funcionamiento.

Este archivo permite a **npm** (Node Package Manager) **gestionar, instalar y mantener** todas las librerías, scripts y metadatos del proyecto de manera organizada.

Principales apartados del package.json:

1. **name:** nombre del proyecto o paquete.
2. **version:** versión actual del proyecto (siguiendo el versionado semántico).
3. **description:** breve descripción del proyecto.
4. **main:** archivo principal que sirve como punto de entrada (por ejemplo, **index.js**).
5. **scripts:** comandos personalizados que automatizan tareas (por ejemplo, iniciar el servidor o ejecutar pruebas).
6. **dependencies:** lista de librerías necesarias para ejecutar el proyecto.
7. **devDependencies:** librerías usadas solo durante el desarrollo.
8. **author:** nombre del creador o responsable del proyecto.
9. **license:** tipo de licencia bajo la cual se distribuye el proyecto.

Utilidad:

El archivo **package.json** facilita la gestión del proyecto**,** ya que permite:

- Instalar todas las dependencias con un solo comando (**npm install**).

- Mantener organizada la información del proyecto.
- Automatizar tareas mediante scripts.
- Garantizar que todos los desarrolladores trabajen con las mismas versiones de librerías.

En síntesis, el **package.json** es el núcleo de configuración de un proyecto en Node.js**,** imprescindible para su mantenimiento, distribución y correcta ejecución.

¿Qué es el archivo package-lock.json y que utilidad tiene?

El **archivo package-lock.json** es un archivo que genera automáticamente **npm** cuando se instalan dependencias en un proyecto de Node.js. Su función principal es **registrar las versiones exactas** de todas las librerías instaladas, incluyendo sus dependencias internas.

Utilidad principal:

- **Garantiza la estabilidad del proyecto**, asegurando que todos los desarrolladores usen las mismas versiones de las dependencias.
- **Permite reproducir el entorno de instalación**, evitando errores por diferencias de versiones.
- **Optimiza la instalación**, ya que npm puede usar la información del archivo para reconstruir rápidamente las dependencias.

En conclusión, el package-lock.json **asegura la consistencia y confiabilidad** del proyecto, evitando conflictos entre versiones y garantizando que el software funcione igual en todos los entornos.

¿Qué es la carpeta node_modules en un proyecto de npm?

La **carpeta node_modules** es el directorio donde **npm instala todas las dependencias** (librerías y módulos) que un proyecto de Node.js necesita para funcionar.

Características principales:

- Contiene **todas las librerías descargadas** junto con sus dependencias internas.
- Se **crea automáticamente** cuando se ejecuta el comando **npm install.**
- No debe modificarse manualmente, ya que su contenido es gestionado por npm.
- Generalmente **no se incluye en los repositorios** (como GitHub), ya que puede regenerarse fácilmente a partir del archivo **package.json**.

Utilidad:

La carpeta node_modules permite que el proyecto **tenga acceso a todas las librerías necesarias** para su ejecución, manteniéndolas organizadas y separadas de otros proyectos.

En resumen, node_modules es la **carpeta que almacena físicamente las dependencias instaladas** por npm, garantizando que el proyecto funcione correctamente con los módulos requeridos.


Cheat Sheet – Comandos npm básicos
1. Inicializar un proyecto npm
npm init
Crea un archivo package.json con la configuración del proyecto (nombre, versión, dependencias, etc.)
Tip: Usa npm init -y para crearlo automáticamente con valores por defecto.
2. Instalar dependencias locales
npm install nombre-paquete
Instala una librería solo para este proyecto (se guarda en node_modules).
Se agrega automáticamente al apartado "dependencies" en el package.json.
3. Instalar dependencias de desarrollo
npm install nombre-paquete --save-dev
Instala dependencias solo necesarias durante el desarrollo (como nodemon, eslint, etc.).
Se agregan al apartado "devDependencies".
4. Instalar dependencias globales
npm install -g nombre-paquete
Instala un paquete para usarlo desde cualquier proyecto o terminal.
Ejemplo: npm install -g nodemon
5. Visualizar dependencias instaladas
npm list
Muestra las dependencias locales instaladas.
Usa npm list -g --depth=0 para ver las globales sin mostrar las subdependencias.
6. Instalar una versión específica
npm install nombre-paquete@versión
Permite instalar una versión concreta.
Ejemplo: npm install express@4.18.2

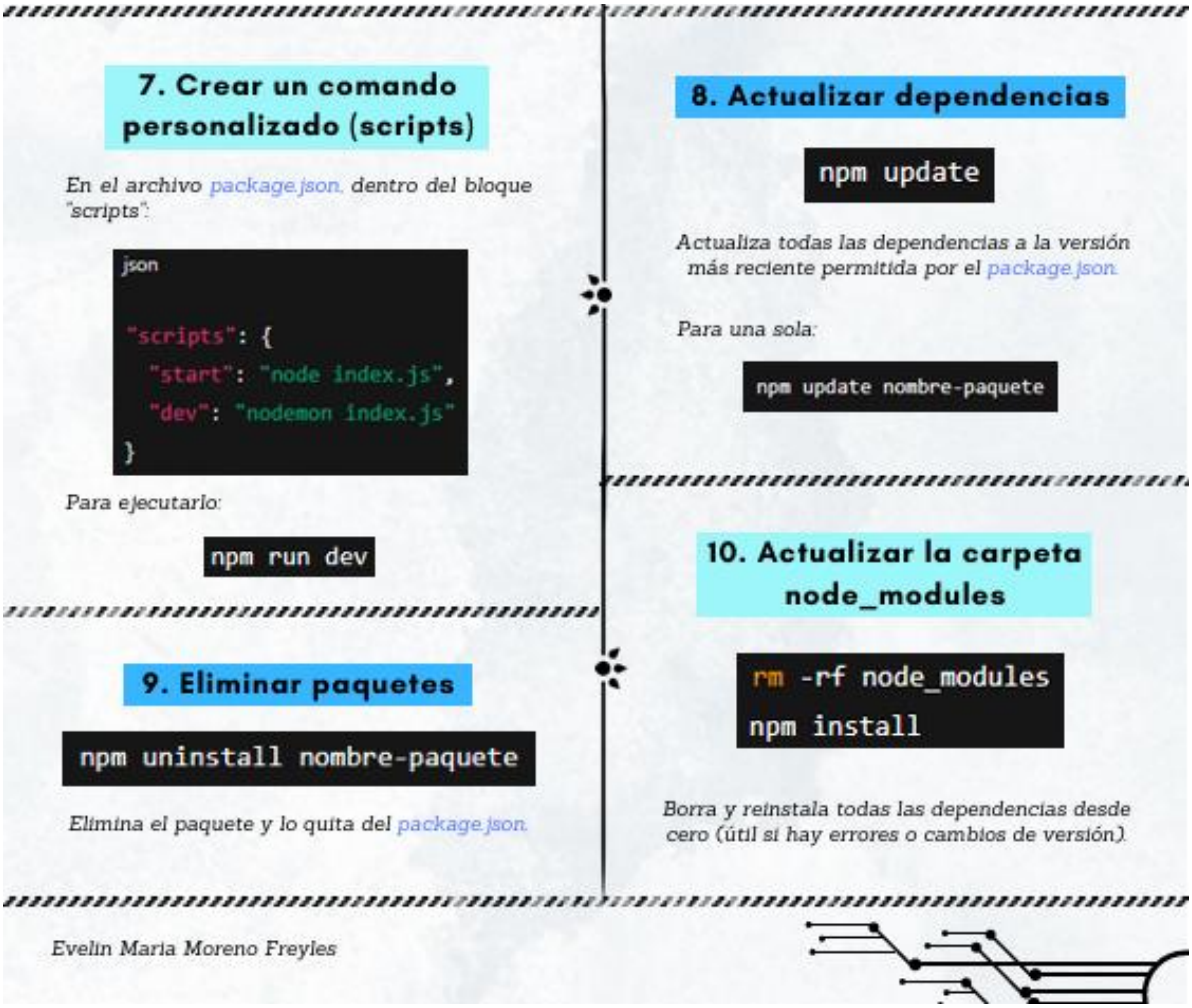

Ilustración 1 Cheat Sheet

3.3 Actividades de apropiación:

GA1-220501096-03-AA1-EV03 Manejo Práctico del gestor de dependencias NPM.

GUÍA DE INSTALACIÓN DE NODEJS

Instalación de NodeJs y NPM como finalizar su configuración

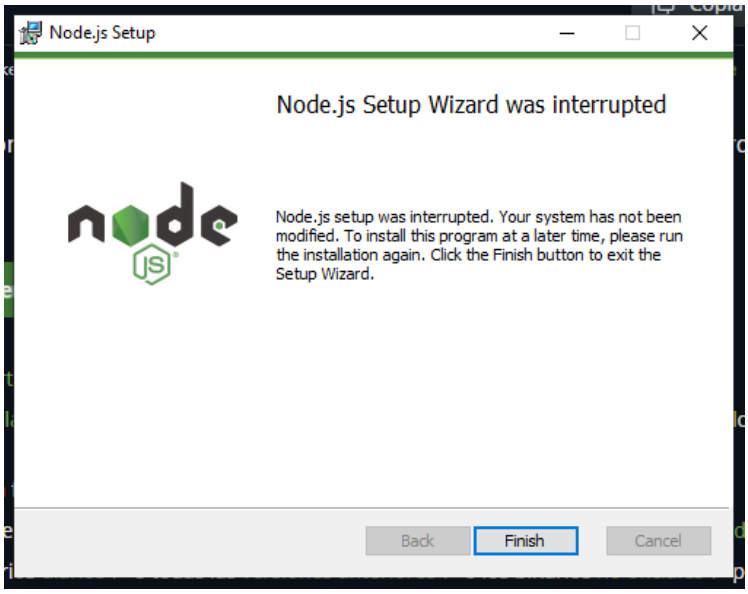

Ilustración 2 Descarga de Node.js (Elaboración propia)

Verificación de la instalación

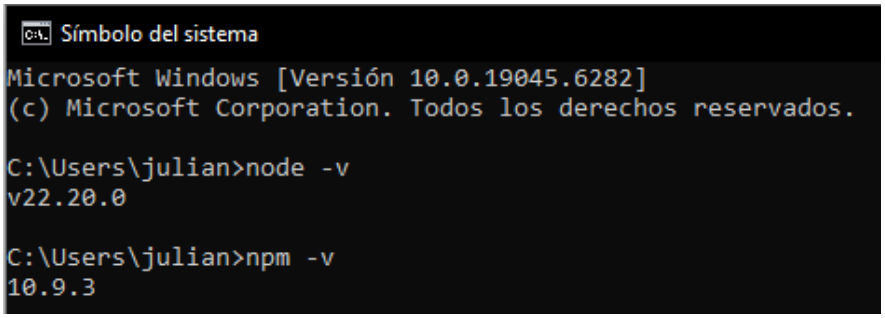

Ilustración 3 Verificación de Descargas y Versiones (Elaboración propia)

MANEJO DE DEPENDENCIAS EN NPM

- Inicialización de un Proyecto npm

- Crear una Carpeta para el Proyecto

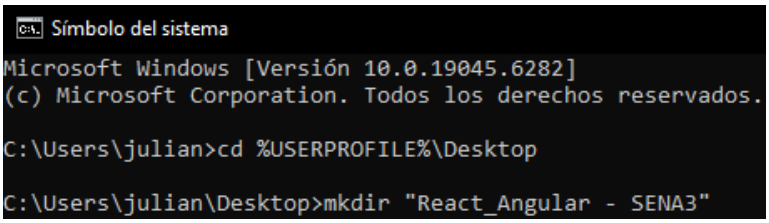

Ilustración 4 Creación de carpeta (Elaboración propia)

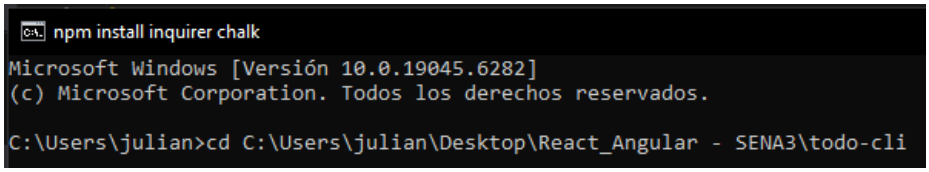

Ilustración 5 Crear Carpeta "todo-cli" (Elaboración propia)

Inicializar un Proyecto npm

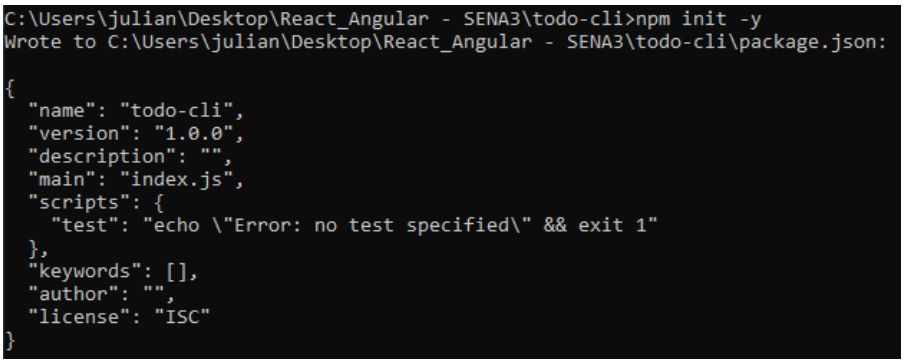

Ilustración 6 (Elaboración propia)

1. Instalación de Dependencias

- Instalación de Dependencias Locales

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm install inquirer chalk

added 37 packages, and audited 38 packages in 27s

5 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Ilustración 7 Instalación dependencias locales (Elaboración propia)

- Instalación de Dependencias de Desarrollo

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm install --save-dev jest eslint
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache
if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and power
ful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

added 362 packages, and audited 400 packages in 50s

64 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Ilustración 8 Instalación dependencias desarrollo (Elaboración propia)

- Instalación de Dependencias Globales

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm install -g nodemon

added 29 packages in 3s

4 packages are looking for funding
  run `npm fund` for details
```

Ilustración 9 Intalación dependecias globales (Elaboración propia)

1. Gestión de Paquetes

- Visualizar Paquetes Instalados

Para listar las dependencias locales instaladas:

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm list --depth=0
todo-cli@1.0.0 C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli
+-- chalk@5.6.2
+-- eslint@9.38.0
+-- inquirer@12.10.0
`-- jest@30.2.0
```

Ilustración 10 Lista dependencias locales (Elaboración propia)

Para listar las dependencias globales:

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm list -g --depth=0
C:\Users\julian\AppData\Roaming\npm
`-- nodemon@3.1.10
```

Ilustración 11 Lista dependencias globales (Elaboración propia)

- **Instalación de una Versión Específica de un Paquete**

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm install chalk@4.1.0

changed 1 package, and audited 400 packages in 4s

64 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Ilustración 12 Instalación versión especifica (Elaboración propia)

2. Scripts y Actualizaciones

- **Crear la Estructura de Directorios del Proyecto**

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>mkdir src

C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>touch src/index.js
```

Ilustración 13 Creación de carpetas (Elaboración propia)

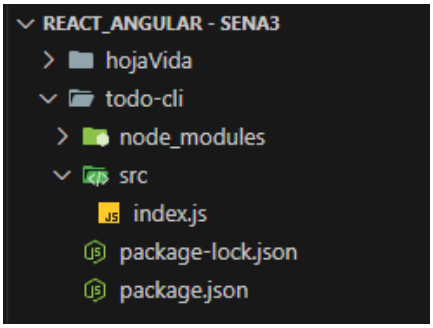

Ilustración 14 Estructura de archivos (Elaboración propia)

- **Crear un Comando (Script) en el Proyecto**

Se puede definir scripts personalizados en el **package.json.** Por ejemplo, para ejecutar mi aplicación:

```
Depurar
"scripts": {
  "start": "node src/index.js"
},
```

Ilustración 15 (Elaboración propia)

Ahora, podemos ejecutar la aplicación con:

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm start

> todo-cli@1.0.0 start
> node src/index.js
```

Ilustración 16 Ejecutar Aplicación (Elaboración propia)

Incluimos el uso de nodemon:

```
"scripts": {
  "start": "node src/index.js",
  "dev": "nodemon src/index.js"
}
```

Ilustración 17 Incluir nodemon

```
  "scripts": {
    "start": "node src/index.js",
    "dev": "nodemon src/index.js"
  },
```

Ilustración 18 (Elaboración propia)

Ahora, podemos ejecutar la aplicación con:

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm run dev

> todo-cli@1.0.0 dev
> nodemon src/index.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/index.js`
[nodemon] clean exit - waiting for changes before restart
```

Ilustración 19 Ejecutacón de Aplicación (Elaboración propia)

index.js

todo-cli > src > index.js > ...

```
// src/index.js

console.log("¡¡Bienvenido a mi TODO CLI - SENA3!");

// Ejemplo: mostrar una lista simple de tareas
const tareas = ["Estudiar Node.js", "Practicar React", "Aprender Los Comandos Github", "Subir proyecto al GitHub"];
console.log("Mis tareas pendientes son:");
tareas.forEach((t, i) => console.log(`${i + 1}. ${t}`));
```

Ilustración 20 Script (Elaboración propia)

```
[nodemon] starting `node src/index.js`
¡¡Bienvenido a mi TODO CLI - SENA3!
Mis tareas pendientes son:
1. Estudiar Node.js
2. Practicar React
3. Aprender Los Comandos Github
4. Subir proyecto al GitHub
[nodemon] clean exit - waiting for changes before restart
```

Ilustración 21 (Elaboración propia)

ACTUALIZAR DEPENDENCIAS

1. Actualizar Dependencias Locales

a. Verificar qué dependencias están desactualizadas

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm outdated
Package  Current  Wanted  Latest  Location             Depended by
chalk      4.1.0   4.1.2   5.6.2  node_modules/chalk   todo-cli
eslint    9.38.0  9.39.0  9.39.0  node_modules/eslint  todo-cli
```

Ilustración 22 Verificación Dependencias Desactualizadas (Elaboración propia)

b. Actualizar todas las dependencias locales

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm update

added 22 packages, removed 28 packages, changed 11 packages, and audited
394 packages in 33s

63 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Ilustración 23 Actualizar Dependencias Locales (Elaboración propia)

c. Actualizar una dependencia específica a la última versión compatible

```
npm update nombre-paquete
```

Ilustración 24 Actualizar Dependencias Especificas

Solo actualizará ese paquete.

d. Actualizar una dependencia a la última versión absoluta

A veces queremos forzar la actualización a la última versión **disponible**, incluso si es una nueva versión mayor (puede incluir breaking changes):

```
npm install nombre-paquete@latest
```

Ilustración 25 Instalación npm paquetes (Elaboración propia)

Ejemplo:

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm install chalk@latest

added 22 packages, changed 1 package, and audited 416 packages in 4s

64 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Ilustración 26 (Elaboración propia)

e. Actualizar todos los paquetes a la última versión (usando ncu)

Para actualizar a la última versión absoluta (rompiendo los rangos), podemos usar la herramienta npm-check-updates (ncu):

```
npm install -g npm-check-updates
ncu -u       # Actualiza los rangos de versión en package.json
npm install  # Instala las nuevas versiones
```

Ilustración 27 Actualización paquetes última versión (Elaboración propia)

Advertencia: Esto puede introducir incompatibilidades. Revisa el changelog de cada dependencia.

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm install -g npm-check-updates

added 1 package in 3s

C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>ncu -u
Upgrading C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli\package.json
[====================] 4/4 100%

 eslint  ^9.38.0  →  ^9.39.0

Run npm install to install new versions.

C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm install

up to date, audited 416 packages in 3s

64 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Ilustración 28 (Elaboración propia)

2. Actualizar Dependencias Globales

a- Ver qué paquetes globales están desactualizados

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm outdated -g --depth=0
```

Ilustración 29 Paquetes Globales Desactualizados (Elaboración propia)

b- Actualizar todos los paquetes globales

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm update -g

changed 276 packages in 28s

50 packages are looking for funding
  run `npm fund` for details
```

Ilustración 30 Actualización Paquetes Globales (Elaboración propia)

Actualiza todos los paquetes globales a sus últimas versiones **compatibles** según sus rangos.

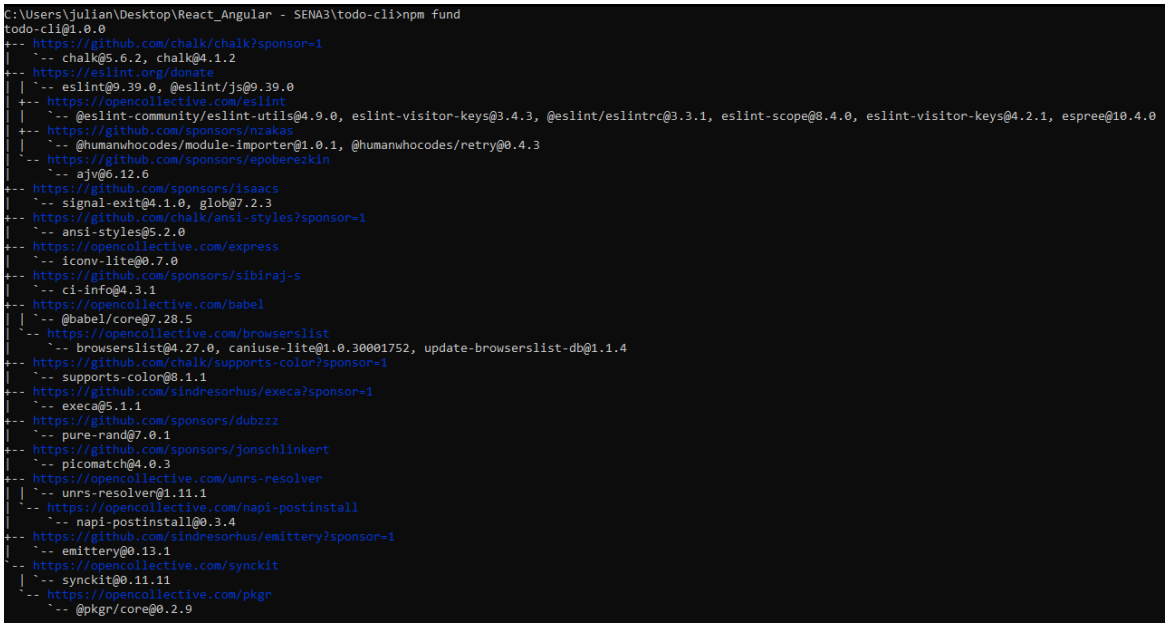

Ilustración 31 (Elaboración propia)

1. **Actualizar un paquete global específico a la última versión**

```
npm install -g nombre-paquete
```

Ilustración 32 Actualización Paquete Global Especifico (Elaboración propia)

O para la última versión absoluta:

```
npm install -g nombre-paquete@latest
```

Ilustración 33 Actualización Ultima Versión (Elaboración propia)

Ejemplo:

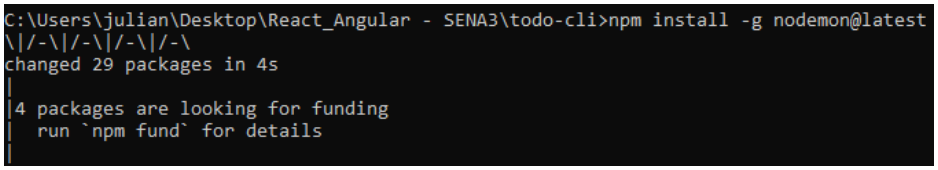

Ilustración 34 Ejemplo (Elaboración propia)

a. Limpieza y Mantenimiento

- Eliminar Paquetes

Para desinstalar un paquete y eliminarlo del package.json:(docs.npmjs.com)

```
npm uninstall nombre-del-paquete
```

Ilustración 35 Desinstalación Paquete (Elaboración propia)

Por ejemplo, para desinstalar **chalk**:

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm uninstall chalk
\|/-\|/-
removed 1 package, and audited 415 packages in 4s
-
-63 packages are looking for funding
-  run `npm fund` for details
-
found 0 vulnerabilities
-
```

Ilustración 36 Ejemplo Desinstalación (Elaboración propia)

- Actualizar node_modules

Si deseamos reinstalar todas las dependencias desde cero:

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>rm -rf node_modules

C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm install
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out
lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more com
prehensive and powerful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

added 414 packages, and audited 415 packages in 36s

63 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Ilustración 37 Reiniciar Todas las Dependencias (Elaboración propia)

Sección de Práctica

Ejercicio: Implementa una funcionalidad básica en tu aplicación TODO-CLI que permita agregar tareas.

Paso 1: Instalar inquirer

```
C:\Users\julian\Desktop\React_Angular - SENA3\todo-cli>npm install inquirer

up to date, audited 415 packages in 5s

63 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Ilustración 38 Instalación Inquirer (Elaboración propia)

Paso 2: Editar tu archivo src/index.js

index.js | package.json

todo-cli > src > index.js > ...

```
// src/index.js

import inquirer from 'inquirer';

// si usas "type": "module" en package.json

// Si te da error por "import", usa la versión con require:
// const inquirer = require("inquirer");

let tareas = []; // Arreglo en memoria

async function agregarTarea() {
  const respuesta = await inquirer.prompt([
    {
      type: "input",
      name: "tarea",
      message: "📝 Escribe una nueva tarea:",
    },
  ]);
  tareas.push(respuesta.tarea);
  console.log("\n✅ Tarea agregada con éxito.\n");

  mostrarTareas();
}

function mostrarTareas() {
  console.log("📋 Lista de tareas:");
  tareas.forEach((tarea, index) => {
    console.log(`${index + 1}. ${tarea}`);
  });
  console.log("--------------------------\n");
}

async function main() {
  console.log("👋 Bienvenido a TODO-CLI - SENA3\n");
  await agregarTarea();
}

main();
```

Ilustración 39 Modificación Archivo index.js (Elaboración propia)

Paso final: Resultados en consola:

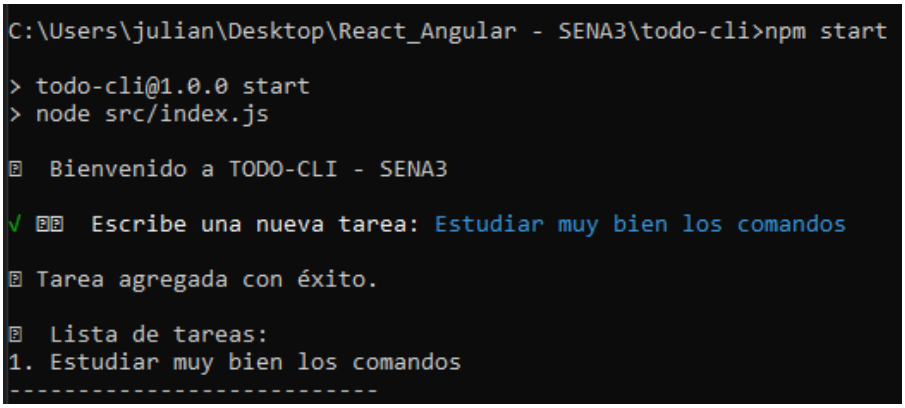

Ilustración 40 Resultado en Consola (Elaboración propia)

GA1-220501096-03-AA1-EV04-Tu primer laboratorio con React y Vite – Instalación, estructura y ejecución.

MI PRIMER LABORATORIO DE REACT + VITE

Ejemplo práctico

Paso 1: Verificar la instalación de Node.js.

```
C:\Users\julian\Desktop\React_Angular - SENA3>node -v
v22.20.0
```

Ilustración 41 Verificación Instalación Node.js (Elaboración propia)

Paso 2: Crear un nuevo proyecto con Vite.

```
C:\Users\julian\Desktop\React_Angular - SENA3>npm create vite@latest
Need to install the following packages:
create-vite@8.0.2
Ok to proceed? (y) y


> npx
> create-vite

|
o  Project name:
|  mi-app-react
|
o  Select a framework:
|  React
|
o  Select a variant:
|  JavaScript
|
o  Use rolldown-vite (Experimental)?:
|  No
|
o  Install with npm and start now?
|  Yes
|
o  Scaffolding project in C:\Users\julian\Desktop\React_Angular - SENA3\mi-app-react...
|
o  Installing dependencies with npm...
```

Ilustración 42 Creación Nuevo Proyecto Vite (Elaboración propia)

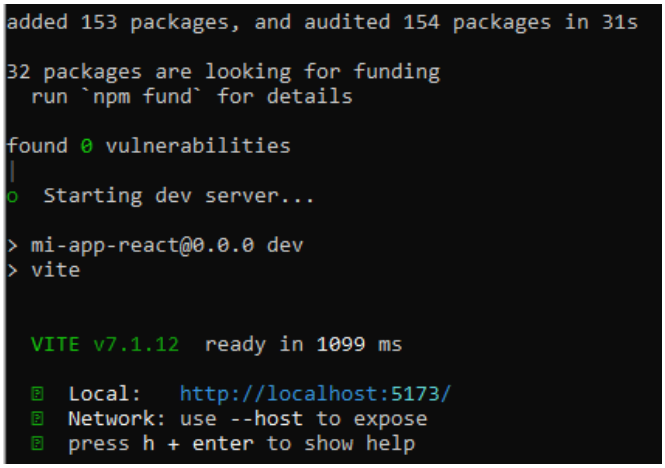

Ilustración 43 Verificación Instalación (Elaboración propia)

Paso 3: Instalar las dependencias.

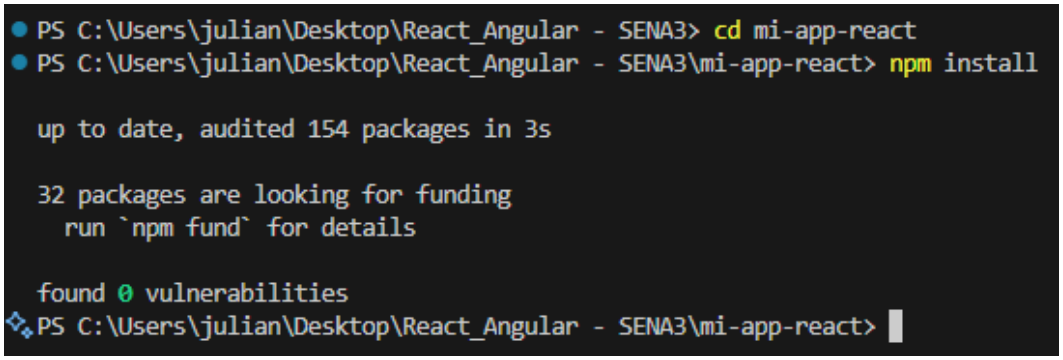

Ilustración 44 Instalación Dependencias (Elaboración propia)

Paso 4: Ejecutar el servidor de desarrollo.

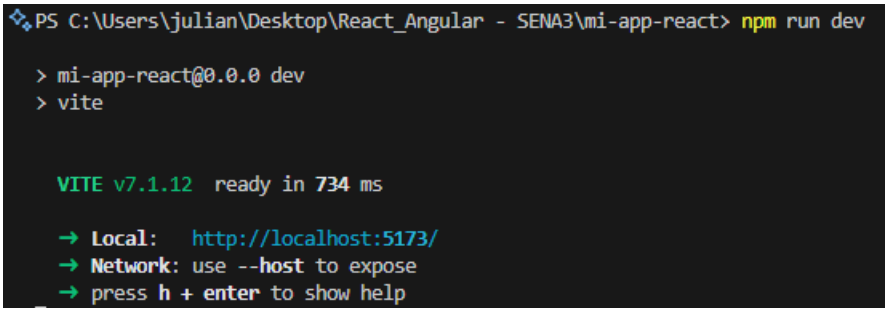

Ilustración 45 Ejecutar Servidor de Desarrollo (Elaboración propia)

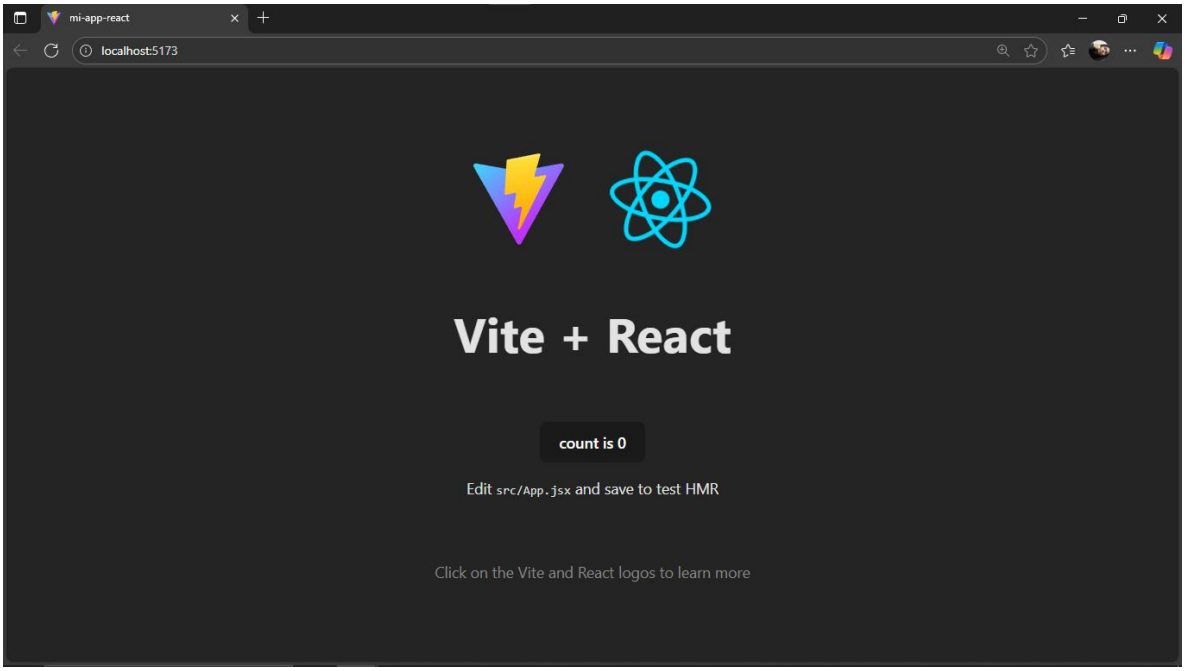

Ilustración 46 Servidor (Elaboración propia)

Paso 5: Explorar la estructura del proyecto.

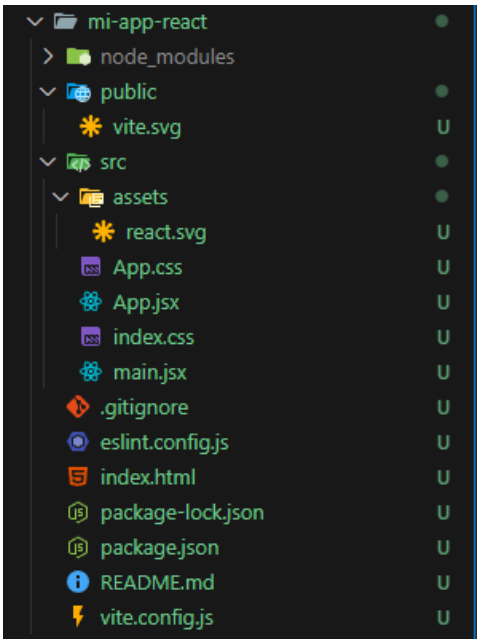

Ilustración 47 Estructura Proyecto (Elaboración propia)

Paso 6: Crear un componente React simple.

App.jsx U

mi-app-react > src > App.jsx > ...

```jsx
import './App.css';

import { useState } from 'react';

import viteLogo from '/vite.svg';

import reactLogo from './assets/react.svg';

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <h1>¡Hola, React con Vite!</h1>
      <div>
        <a href="https://vite.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}

export default App
```

Ilustración 48 Crear un componente React simple (Elaboración propia)

Ilustración 49 Servidor (Elaboración propia)

REFERENCIAS

npm, Inc. (s.f.). *npm documentation*. https://docs.npmjs.com

React. (s.f.). *React – A JavaScript library for building user interfaces*. https://reactjs.org

Semantic Versioning. (s.f.). *Semantic Versioning 2.0.0*. https://semver.org

Vite. (s.f.). *Vite: Next Generation Frontend Tooling*. https://vitejs.dev

freeCodeCamp.org. (2020, mayo 20). *Learn React in 1 Hour* [Video]. YouTube.

https://www.youtube.com/watch?v=Ke90Tje7VS0

Educated Show. (s.f.). *Guía de instalación de NodeJs*. Notion. https://educated-show-144.notion.site/Guia-de-instalaci-n-de-NodeJs-1f74671e02a180cd9284fd1143d69a3e?pvs=4

Educated Show. (s.f.). *Manejo de dependencias en npm*. Notion. https://educated-show-144.notion.site/Manejo-de-dependencias-en-npm-1f74671e02a180059863c5eb9af8229b?pvs=4

Educated Show. (s.f.). *Tu primer laboratorio de React + Vite* [Página web]. Notion.

https://educated-show-144.notion.site/Tu-primer-laboratorio-de-React-Vite-

1f74671e02a180d69676c1327e8ead6b?pvs=4

SENA. (2025). *Construcción de software integrador de tecnologías orientadas a servicios*.

Medellín-Antioquia: SENA.