ACTIVIDAD DE REACT Y VITE

Andres Felipe Betancur Castañeda

Luis Fernando Sánchez

SENA, servicio nacional de aprendizaje

ADSO

FT-3223875

2025

Introducción

Durante el desarrollo de las evidencias prácticas del programa de **Análisis y Desarrollo de Software del SENA**, realicé una serie de actividades enfocadas en fortalecer mis conocimientos y habilidades en el área del **desarrollo front-end moderno**. Estas actividades me permitieron trabajar con tres herramientas muy importantes que se usan hoy en día en la industria del desarrollo web: **Vite, React y npm**.

Al inicio, me enfoqué en la **configuración del entorno de trabajo**, instalando y aprendiendo a usar **Node.js**, **npm** y **Visual Studio Code**, con el objetivo de preparar correctamente la base para empezar a crear proyectos. Después, comencé a trabajar con **Vite**, que me pareció una herramienta muy práctica y moderna para crear aplicaciones en React de manera rápida y organizada. Gracias a esto, pude entender mejor la estructura de un proyecto, los archivos principales y los comandos necesarios para ejecutarlo y compilarlo correctamente.

Una de las partes más importantes fue la **creación de mi hoja de vida modular (CV)** en React. En esta etapa aprendí a aplicar conceptos como la **componentización y la anidación**, dividiendo el proyecto en varias secciones independientes: perfil, educación, experiencia, habilidades, entre otras. Esto me ayudó a comprender cómo se puede organizar mejor el código y cómo los componentes hacen que el proyecto sea más fácil de mantener y reutilizar. Además, usé **Git y GitHub** para guardar los avances de mi trabajo y subirlos al repositorio, lo cual me permitió practicar el manejo de versiones de una forma más profesional.

Por último, trabajé con **npm**, aprendiendo cómo funciona para instalar, actualizar y administrar los paquetes dentro de un proyecto. A través de la práctica, entendí la importancia del archivo `package.json` y cómo mantener actualizadas las dependencias para evitar errores y garantizar la estabilidad del proyecto.

En general, estas actividades me ayudaron a unir la teoría con la práctica, entendiendo mejor cómo se aplican las herramientas modernas en el desarrollo web actual. También me sirvieron para mejorar mi autonomía, mi lógica de programación y mi capacidad para trabajar en proyectos reales con **JavaScript y React**.

GUIA DE APRENDIZAJE

3.1. Actividad 1: no hay evidencias

3.2. Actividad 2:

- **¿Que es un gestor dependencias de codigo?**

Un gestor de dependencias de código es una herramienta que se encarga de administrar automáticamente las librerías, módulos o paquetes que un proyecto necesita para funcionar correctamente. Su función principal es facilitar al programador la instalación, actualización y eliminación de esas dependencias sin tener que hacerlo manualmente. Estos gestores usan archivos especiales (como *package.json* o *requirements.txt*) donde se registran todas las librerías requeridas, sus versiones y configuraciones, lo que permite mantener el proyecto ordenado, reproducible y compatible entre distintos entornos. En pocas palabras, garantizan que el código del proyecto tenga siempre las herramientas externas necesarias para ejecutarse sin errores.

- **¿Qué es npm?**

npm (siglas de *Node Package Manager*) es el gestor de dependencias oficial de Node.js, y se utiliza para instalar, compartir y administrar librerías o paquetes de JavaScript. Su función principal es facilitar el trabajo de los desarrolladores al permitirles agregar fácilmente código creado por otros (como frameworks, herramientas o módulos) a sus proyectos, sin tener que programarlo desde cero. Además, npm cuenta con un enorme repositorio en línea donde se almacenan millones de paquetes disponibles para descargar. En resumen, npm es una herramienta esencial en el desarrollo con JavaScript porque automatiza la instalación y gestión de las dependencias de un proyecto.

- **¿Para qué se utiliza principalmente npm?**

npm se utiliza principalmente para **instalar, administrar y compartir paquetes o dependencias de JavaScript** que un proyecto necesita para funcionar. Gracias a npm, los desarrolladores pueden agregar fácilmente librerías externas (como *React*, *Express* o *Vue*), mantenerlas actualizadas y controlar sus versiones desde un mismo lugar. También permite ejecutar scripts o comandos que automatizan tareas dentro del proyecto, como iniciar el servidor o compilar el código. En pocas palabras, npm se usa para **gestionar todo el software adicional que ayuda al desarrollo y funcionamiento de proyectos en JavaScript y Node.js.**

- **¿Qué es el versionado semántico?**

El versionado semántico (o Semantic Versioning) es un sistema que se usa para asignar números de versión a los programas o librerías de forma que indiquen claramente los cambios realizados entre una versión y otra. Se representa con tres números, por ejemplo: 1.4.2, donde cada número tiene un significado específico:

- 1 → versión mayor (MAJOR): cambia cuando hay modificaciones grandes o incompatibles con versiones anteriores.
- 4 → versión menor (MINOR): cambia cuando se agregan nuevas funciones que no rompen la compatibilidad.
- 2 → versión de parche (PATCH): cambia cuando se corrigen errores o se hacen ajustes pequeños.

En resumen, el versionado semántico permite entender fácilmente la magnitud y el tipo de cambios en un software, ayudando a mantener la estabilidad y compatibilidad de los proyectos.

• Como está especificado el Versionado Semántico.

El **versionado semántico** se especifica con el formato **MAJOR.MINOR.PATCH** (por ejemplo, **1.4.2**). Cada número indica el tipo de cambio: **MAJOR** para modificaciones incompatibles, **MINOR** para nuevas funciones compatibles y **PATCH** para correcciones de errores o ajustes menores.

• Qué son las dependencias locales.

Las **dependencias locales** son las librerías o paquetes que se **instalan directamente dentro de un proyecto** y se **guardan en su propia carpeta**, normalmente en node_modules. Solo afectan a ese proyecto específico y no al resto del sistema. Esto permite que cada proyecto tenga las versiones exactas de las dependencias que necesita, sin interferir con otros proyectos que usen las mismas librerías pero en versiones diferentes.

• Qué son las dependencias de desarrollo

Las **dependencias de desarrollo** son aquellas librerías o paquetes que se **utilizan solo mientras se está creando o probando un proyecto**, pero **no son necesarias cuando la aplicación ya está en funcionamiento**. Se usan, por ejemplo, para tareas como compilar el código, hacer pruebas o verificar errores. En npm, estas dependencias se instalan con el comando npm install --save-dev.

• **Qué son las dependencias globales**

Las **dependencias globales** son los paquetes o librerías que se **instalan en todo el sistema**, no solo dentro de un proyecto específico. Esto permite usarlas desde cualquier carpeta o proyecto sin tener que volver a instalarlas cada vez. Se suelen usar para herramientas o comandos que se necesitan de forma general, como **npm**, **nodemon** o **typescript**. En npm, se instalan con el comando npm install -g.

• **¿Que es el archivo package.json y que apartados tiene y cual es su utilidad?**

El **archivo package.json** es el documento principal de un proyecto en **Node.js**, donde se guarda su información, configuración y dependencias. Contiene apartados como **name**, **version**, **description**, **scripts**, **dependencies** y **devDependencies**. Su utilidad es **administrar y describir el proyecto**, permitiendo instalar y ejecutar fácilmente todas sus librerías y comandos.

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

El archivo package-lock.json se crea automáticamente cuando se instalan dependencias con npm. Su función es guardar las versiones exactas de cada paquete y sus subdependencias para que, al instalar el proyecto en otro lugar, se obtengan las mismas versiones y el programa funcione igual. En resumen, garantiza la estabilidad y consistencia del proyecto entre distintos entornos.

• **que es la carpeta node_modules en un proyecto de npm.**

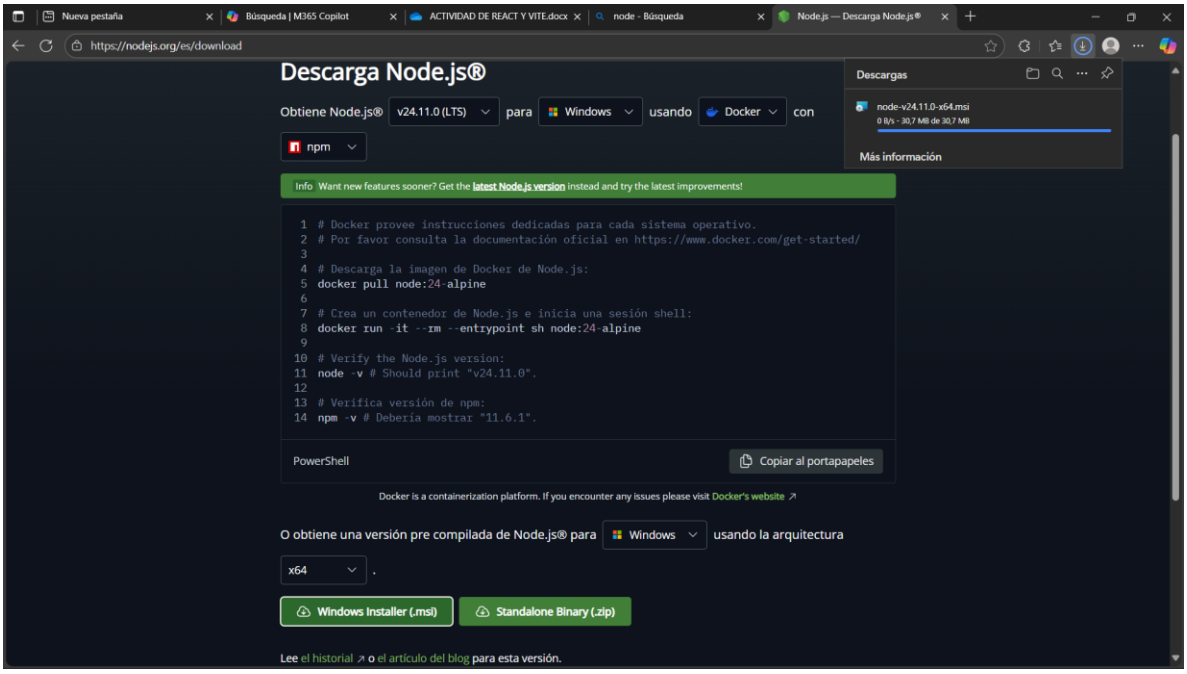
La carpeta node_modules es donde npm guarda todas las dependencias (librerías y módulos) que un proyecto necesita para funcionar. Cada vez que instalas un paquete con npm install, este se descarga y se almacena ahí. En resumen, es la carpeta que contiene el código de todas las librerías instaladas que usa tu proyecto.

Cómo inicializar un proyecto de npm	**Cómo instalar dependencias locales**	**Cómo instalar dependencias de desarrollo**
Se abre la terminal: Ctrl + ñ	Para instalar las dependencias locales se utiliza el siguiente comando: npm install paquete	Para instalar las dependencias de desarrollo se utiliza el siguiente comando: npm install paquete --save-dev
Creamos y entramos a una carpeta: mkdir mi-proyecto cd mi-proyecto	**Cómo instalar dependencias globales**	**Cómo visualizar las dependencias instaladas**
	Para instalar dependencias globales se utiliza: npm install -g paquete	Para ver dependencias locales: npm list Para ver dependencias globales: npm list -g --depth=0
Inicializa el proyecto con npm: npm init	**Cómo instalar una versión específica de un paquete**	**Cómo crear un comando (script)**
	npm install paquete@versión Ejemplo: npm install express@4.18.2	En el archivo package.json: "scripts": { "start": "node app.js" } Ejecutar: npm run start
Cómo actualizar dependencias	**Cómo eliminar paquetes**	**Cómo actualizar node_modules**
Se utiliza el comado npm update	Se utiliza el comando: npm uninstall paquete	Elimina la carpeta y reinstala todas las dependencias: rm -rf node_modules npm install

3.3. Actividad 3:

Bitacora, paso a paso

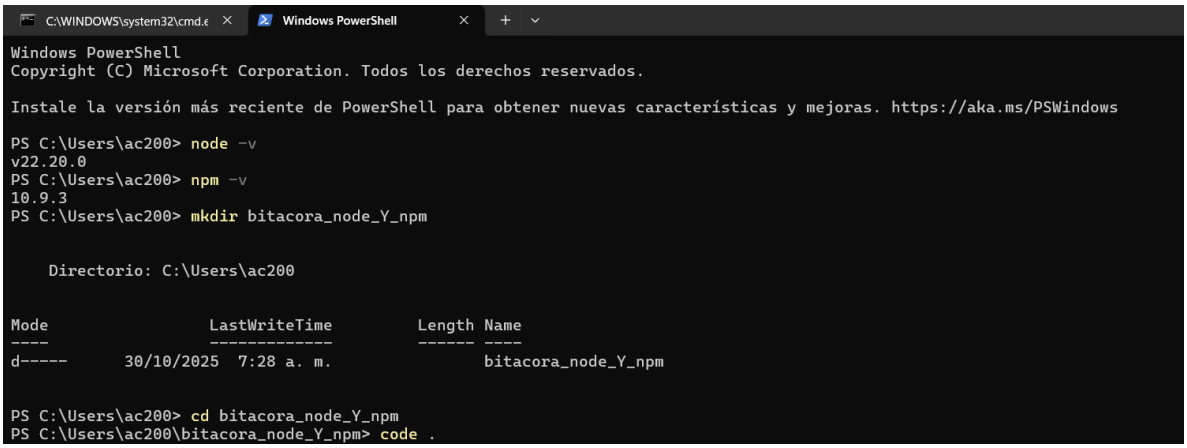
1. Instalación y configuración de Node.js y npm:
   - Aquí se hizo la instalación y configuración Node.js desde su sitio oficial como se indica en la guía



   - Luego de haberse instalado y configurado Node.js y npm, se hizo la comprobación en la consola de la versión e instalación con los siguientes comandos:

     node –v, npm –v

   - Se creo la carpeta de trabajo “bitacora_node_Y_npm” con el comando mkdir bitacora_node_Y_npm y se abrio en visual studio con el comando, code .



2. Inicialización del proyecto con npm:

- En donde estamos trabajando (cmd) ejecutamos el siguiente comando npm init -y que creo un archivo llamado package.js con los metadatos del proyecto

3. Instalación de dependencias:

- Para la Instalación de dependencias locales se utilizo el comando npm install express
- Para la Instalación de dependencias globales se utilizo el comando npm install -g npm-check-updates
- Para la Instalación de dependencias de desarrollo se utilizo el comando npm install nodemon --save-dev

```
PS C:\Users\ac200\bitacora_node_Y_npm> npm install express

added 68 packages, and audited 69 packages in 3s

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

found 0 vulnerabilities
PS C:\Users\ac200\bitacora_node_Y_npm> npm install nodemon --save-dev

added 27 packages, and audited 96 packages in 2s

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

found 0 vulnerabilities
PS C:\Users\ac200\bitacora_node_Y_npm> npm install -g npm-check-updates

added 1 package in 2s
```

4. Manejo y actualización de dependencias:
   Utilice los siguientes comandos para

- Visualizar dependencias instaladas: npm list --depth=0.
- Actualizar una dependencia específica: npm install express@latest.
- Actualizar todas las dependencias: npm update.
- Revisar vulnerabilidades: npm audit fix.

```
PS C:\Users\ac200\bitacora_node_Y_npm> npm list --depth=0
bitacora_node_y_npm@1.0.0 C:\Users\ac200\bitacora_node_Y_npm
+-- express@5.1.0
`-- nodemon@3.1.10

PS C:\Users\ac200\bitacora_node_Y_npm> npm install express@latest

up to date, audited 96 packages in 1s

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

found 0 vulnerabilities
PS C:\Users\ac200\bitacora_node_Y_npm> npm update

up to date, audited 96 packages in 4s

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

found 0 vulnerabilities
PS C:\Users\ac200\bitacora_node_Y_npm> npm audit fix

up to date, audited 96 packages in 2s

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

found 0 vulnerabilities
```

5. **Creación de scripts personalizados y limpieza del proyecto**

- Se configuraron scripts dentro del archivo package.json (start y dev).
- Ejecución del Proyecto con npm run.
- Eliminación y reinstalación de node_modules: Remove-Item -Recurse -Force node_modules; npm install.

```
PS C:\Users\ac200\bitacora_node_Y_npm> npm run
Lifecycle scripts included in bitacora_node_y_npm@1.0.0:
  test
    echo "Error: no test specified" && exit 1
PS C:\Users\ac200\bitacora_node_Y_npm>
```

```
PS C:\Users\ac200\bitacora_node_Y_npm> Remove-Item -Recurse -Force node_modules; npm install

added 95 packages, and audited 96 packages in 1s

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

found 0 vulnerabilities
PS C:\Users\ac200\bitacora_node_Y_npm>
```

Bitacora, Taller practico con npm segun la guía “gestión de dependencias con npm”

En esta imagen cree una carpeta en mi consola y entre en la misma con el comando cd proyecto_vite, luego lo inicialice con npm init –y y esto creo el package.js

```
Windows PowerShell                                                Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\ac200> mkdir proyecto_vite


    Directorio: C:\Users\ac200


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        30/10/2025  8:11 a. m.                proyecto_vite


PS C:\Users\ac200> cd proyecto_vite
PS C:\Users\ac200\proyecto_vite> npm init -y
Wrote to C:\Users\ac200\proyecto_vite\package.json:

{
  "name": "proyecto_vite",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Luego utilice npm install para traer e instalar algunas dependencias en mi proyecto y herramientas como:

npm install inquirer chalk

→ Instala las librerías Inquirer (para menús interactivos en consola) y Chalk (para poner colores al texto).

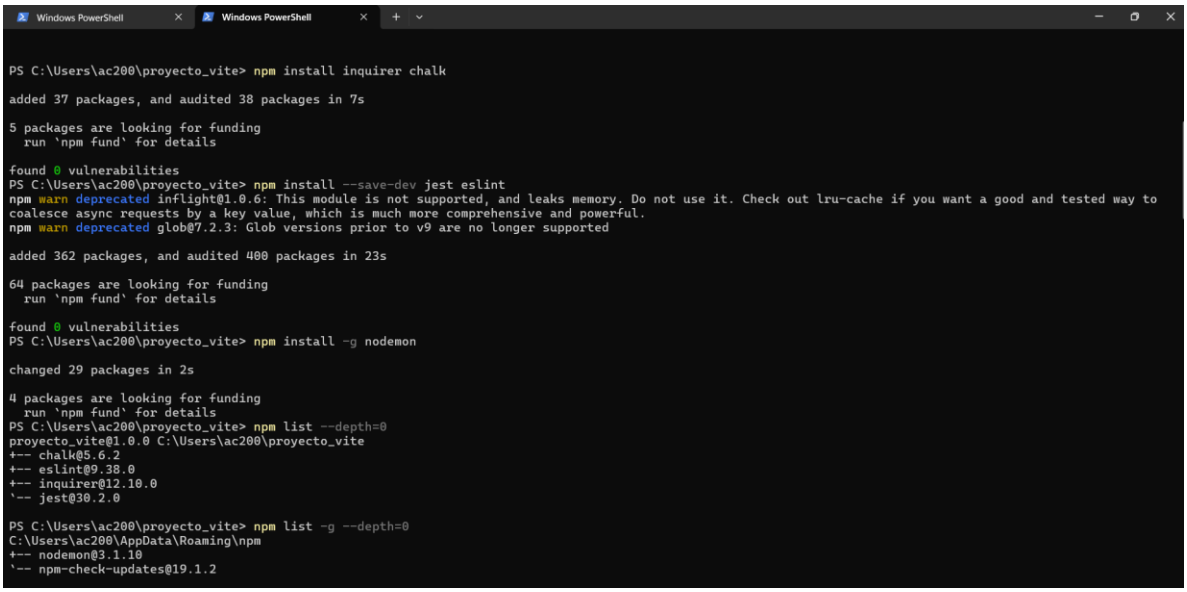
npm install --save-dev jest eslint

→ Instala Jest (para hacer pruebas automáticas) y ESLint (para detectar errores y mejorar tu código). Solo se usan en desarrollo.

npm install -g nodemon

→ Instala Nodemon globalmente, que reinicia tu app Node.js automáticamente cuando haces cambios.

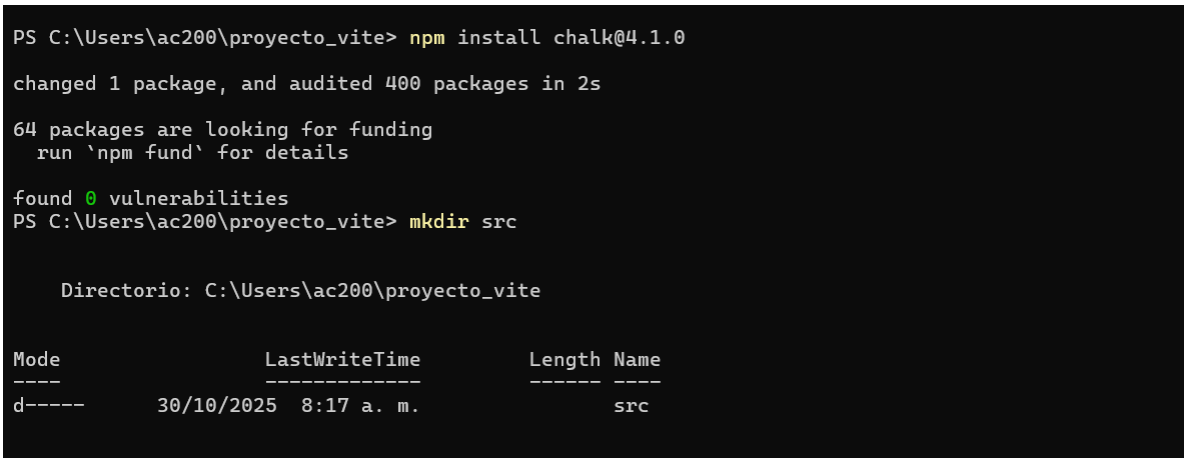
Y utlice este comando para mostrar las dependencias principales instaladas en mi proyecto (sin subdependencias) npm list --depth=0



Aquí instale una versión especifica de chalk con el comando

npm install chalk@4.1.0

y cree una carpeta llamda src con mkdir



A el inicio utilice el comando echo para crear un archivo en la carpeta src llamando index.js donde le escribí un pequeño código que dice que imprima “servidor funcionando correctamente”

Lance mi aplicación con el comando: npm start

Y luego levante un servidor local para probar mi app con el comando: npm run dev

npm outdated → muestra qué paquetes del proyecto están viejos.

npm update → actualiza los paquetes del proyecto a las versiones más nuevas compatibles.

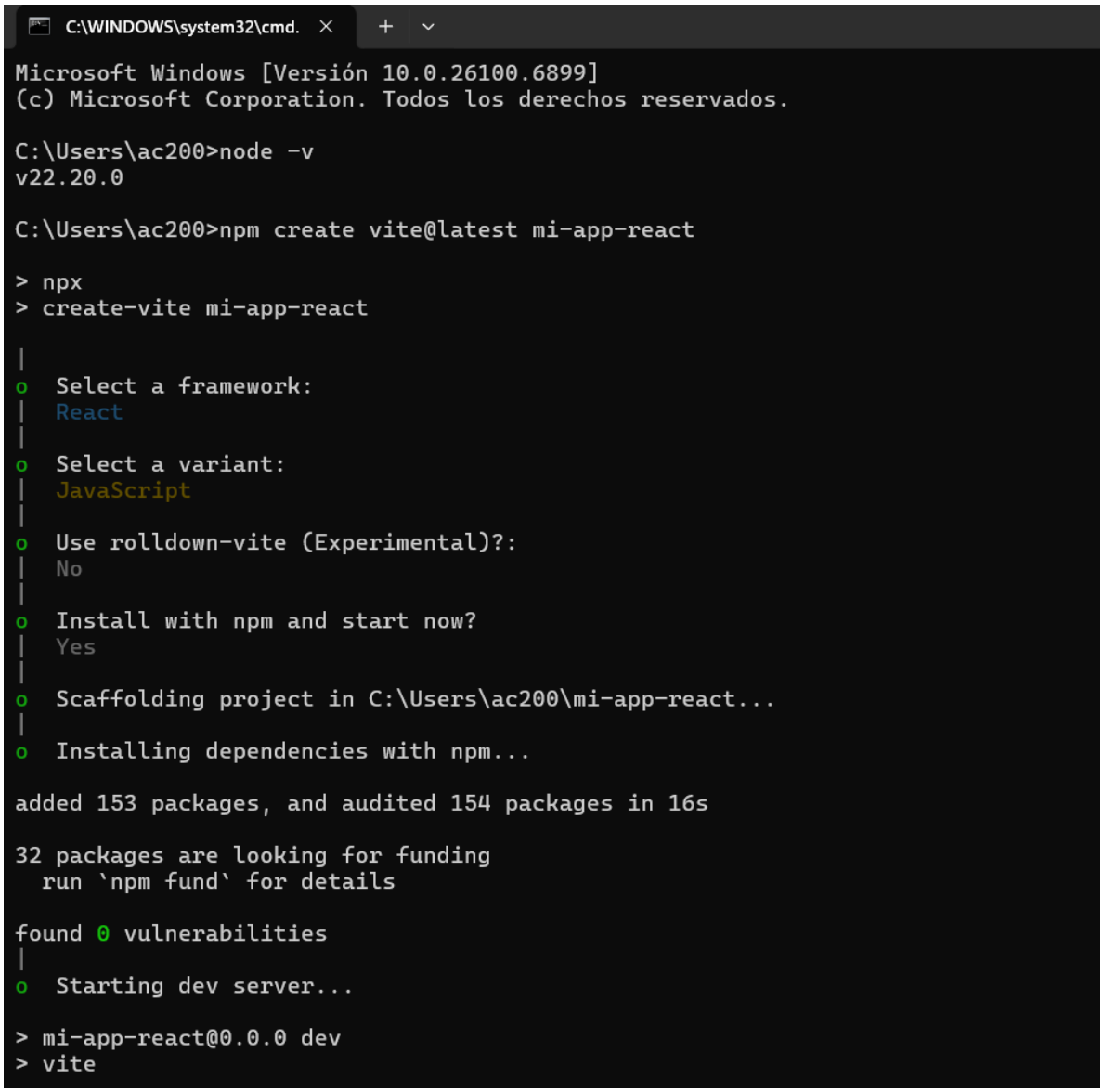
npm outdated -g --depth=0 → muestra qué paquetes globales están desactualizados.

```
PS C:\Users\ac200\proyecto_vite> echo 'console.log("Servidor funcionando correctamente 🚀");' | Out-File -Encoding utf8 src/index.js
PS C:\Users\ac200\proyecto_vite> npm start

> proyecto_vite@1.0.0 start
> node src/index.js

Servidor funcionando correctamente 🚀
PS C:\Users\ac200\proyecto_vite> npm run dev

> proyecto_vite@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`
Servidor funcionando correctamente 🚀
[nodemon] clean exit - waiting for changes before restart

npm outdated



[nodemon] restarting due to changes...
[nodemon] starting `node src/index.js`
Servidor actualizado y en funcionamiento
[nodemon] clean exit - waiting for changes before restart
PS C:\Users\ac200\proyecto_vite> npm outdated
Package  Current  Wanted  Latest  Location            Depended by
chalk      4.1.0   4.1.2   5.6.2  node_modules/chalk  proyecto_vite
PS C:\Users\ac200\proyecto_vite> npm update

added 22 packages, removed 26 packages, changed 5 packages, and audited 396 packages in 15s

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

found 0 vulnerabilities
PS C:\Users\ac200\proyecto_vite> npm outdated -g --depth=0
PS C:\Users\ac200\proyecto_vite>
```

Actividad 4: Proyecto mi-app-react

Paso 1:

- Abrí la consola y escribí el comando **node –v**, sirve para ver la versión de node que tengo
- Con el comando **npm install vite@latest mi-app-react**, genere la estructura base de una app react lista para personalizar llamado mi-app-react
  Me pregunto que framework requiero que es react y cual variante que es JavaScritpv, luego me pregunto que si quiera que fuera una versión experimental y elegí que no porque no quiero errores en esta misma



```
  VITE v7.1.12  ready in 2427 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Paso 2:

Instalación de dependencias

- Entre a mi proyecto o carpeta con el comando
  cd mi-app-react
- Y luego instale las dependencias npm con el comando
  Npm install

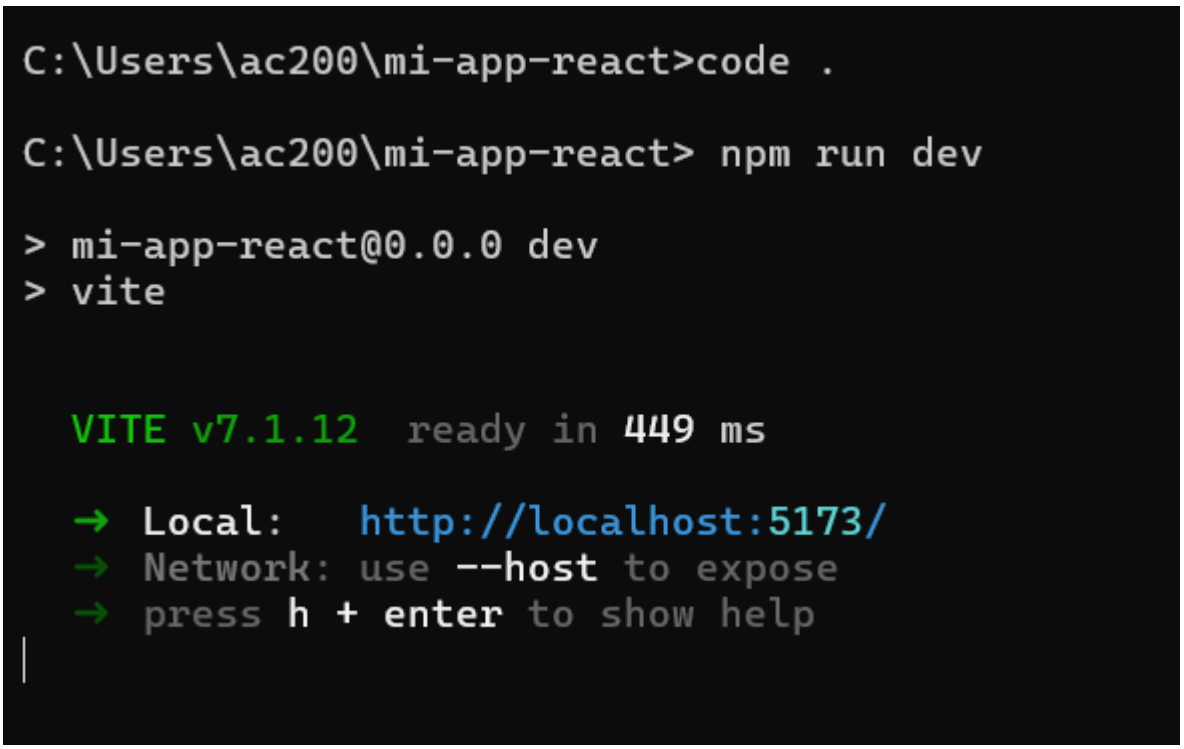
- Este paso descargo las librerías y módulos requeridos para ejecutar mi app

```
C:\Users\ac200>cd mi-app-react

C:\Users\ac200\mi-app-react>npm install

up to date, audited 154 packages in 1s

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

found 0 vulnerabilities

C:\Users\ac200\mi-app-react>
```

Paso 3:

Ejecución del servidor

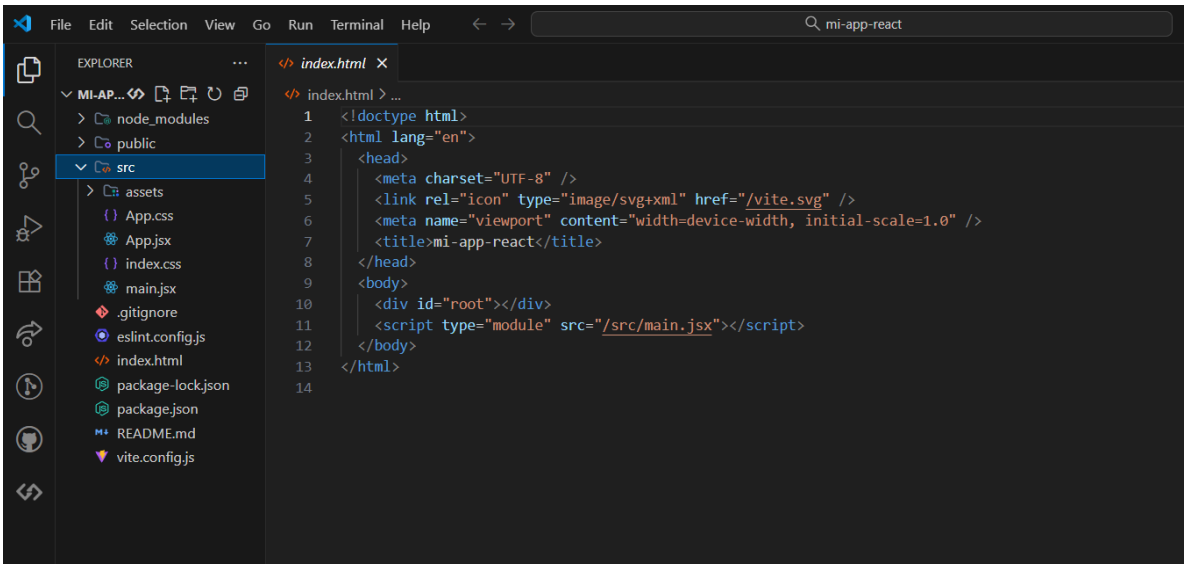
- Con el comando **code .** entre a mi carpeta donde tengo mi app en Visual Studio code
- Luego inicie el servidor local con el comando **npm run dev** donde podre ver la interfaz de mi app en el navegador

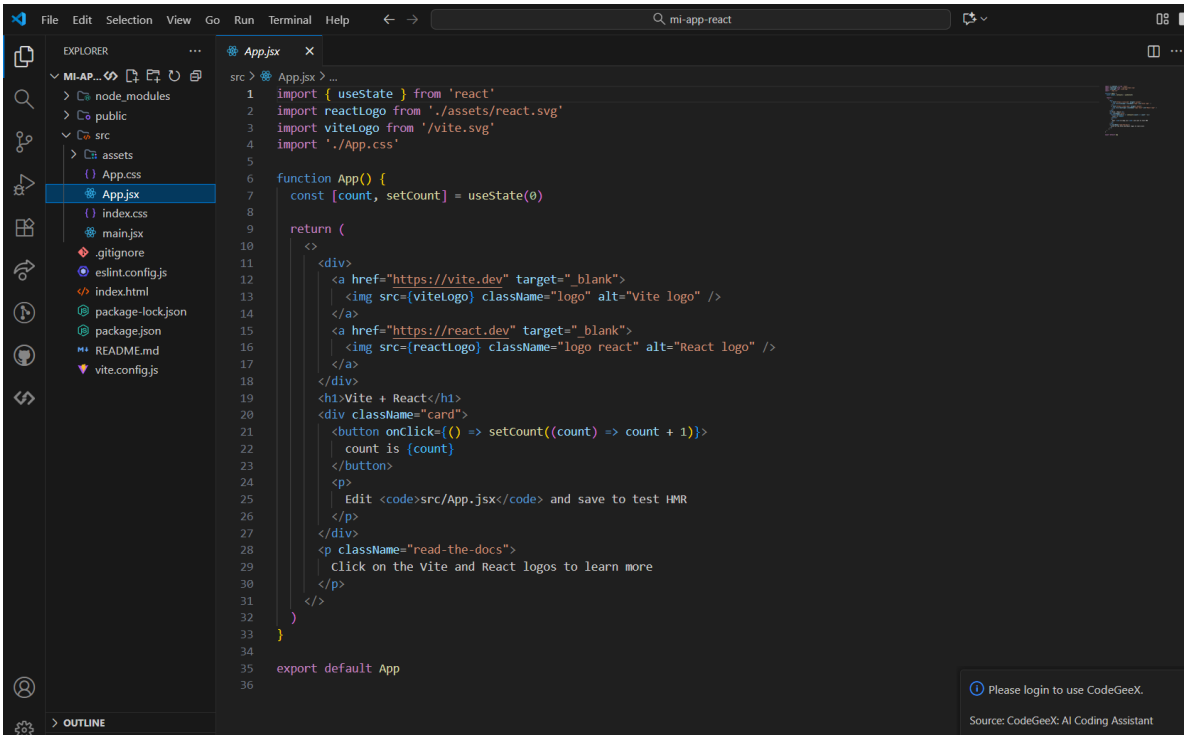


Paso 4:

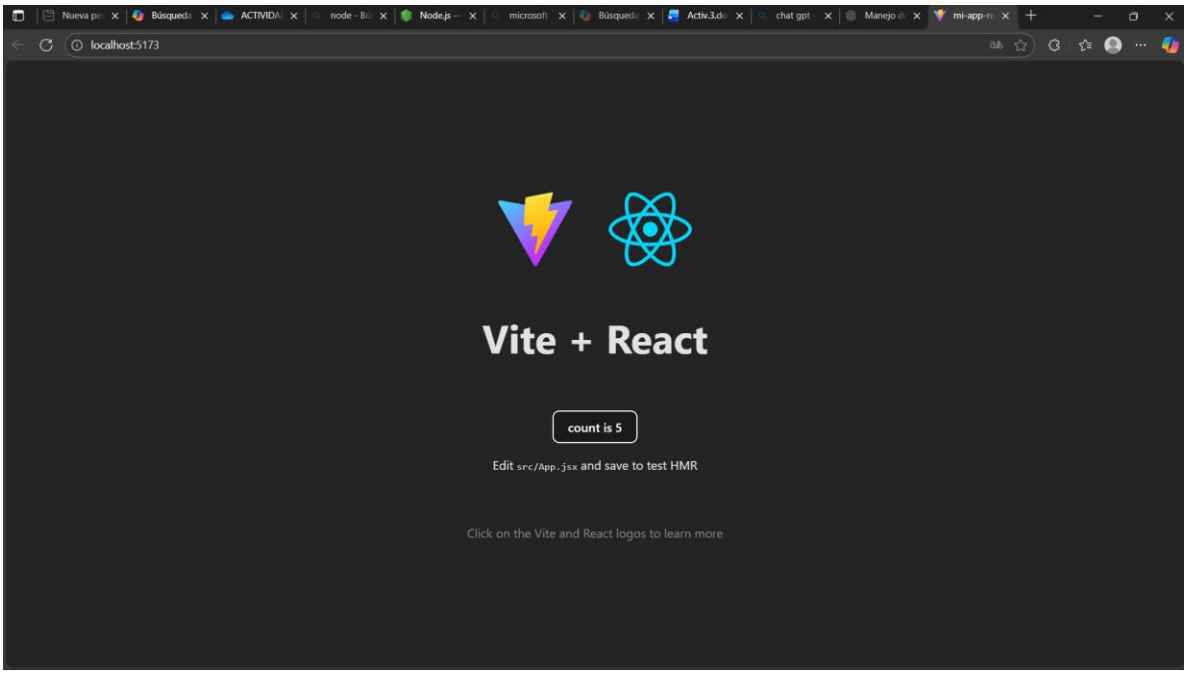
Exploración de la estructura del proyecto

- Aquí podemos ver las dependencias y todo lo que requiere y contiene nuestra aplicación





Vista de mi aplicación desde el servidor local



Resultados:

Nuestra aplicación fue exitosamente configurada con vite, mostrando:

- Servidor local donde se ejecuta nuestra aplicación, puerto 5173
- Estructura de archivos clara y organizada

ACTIVIDAD 5:

Instrucciones para ejecutar el proyecto

1. Verificar instalación de Node.js y npm

Antes de comenzar, asegúrate de tener instalados **Node.js** y **npm** en tu equipo.
Para comprobarlo, abre la terminal y escribe los siguientes comandos:

node -v
npm -v

Si ambos devuelven un número de versión, la instalación está correcta.

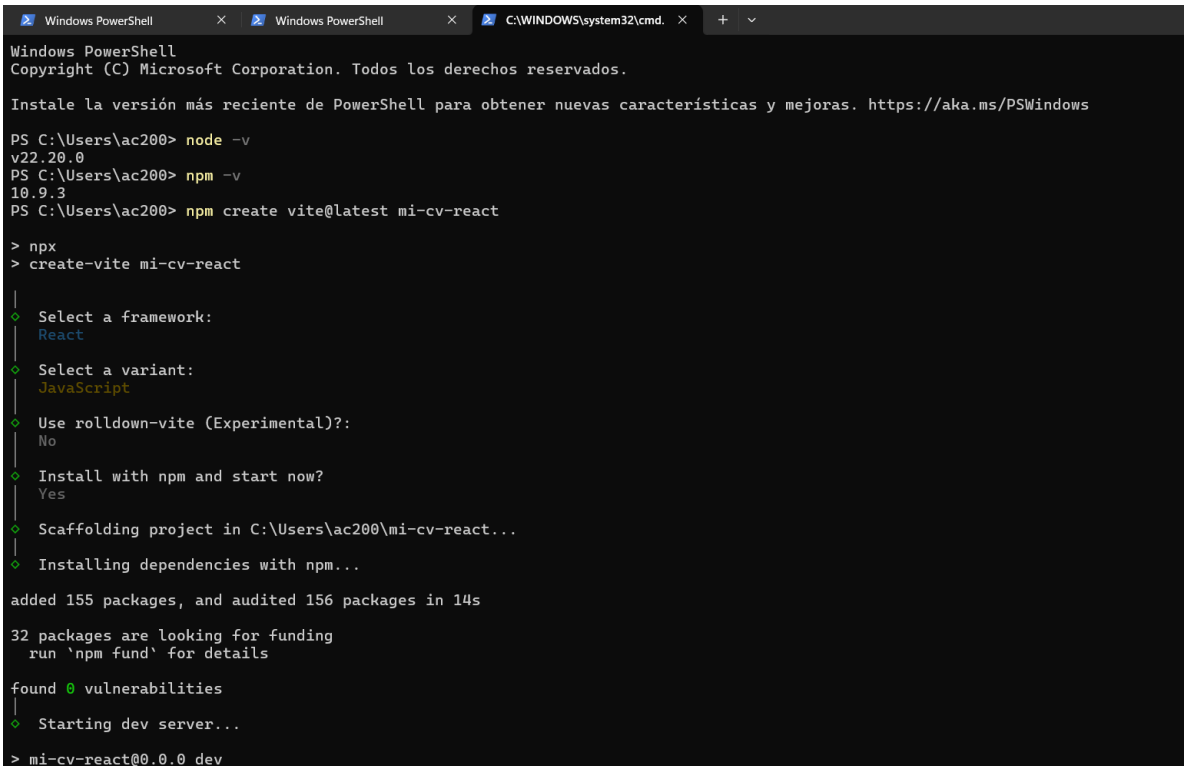
En caso de iniciar desde cero, crea una nueva carpeta manualmente con los siguientes comandos:

mkdir mi-cv-react
cd mi-cv-react

2. Inicializar el proyecto con npm

Ejecuta el siguiente comando para generar el archivo **package.json**, que contendrá la configuración base del proyecto:

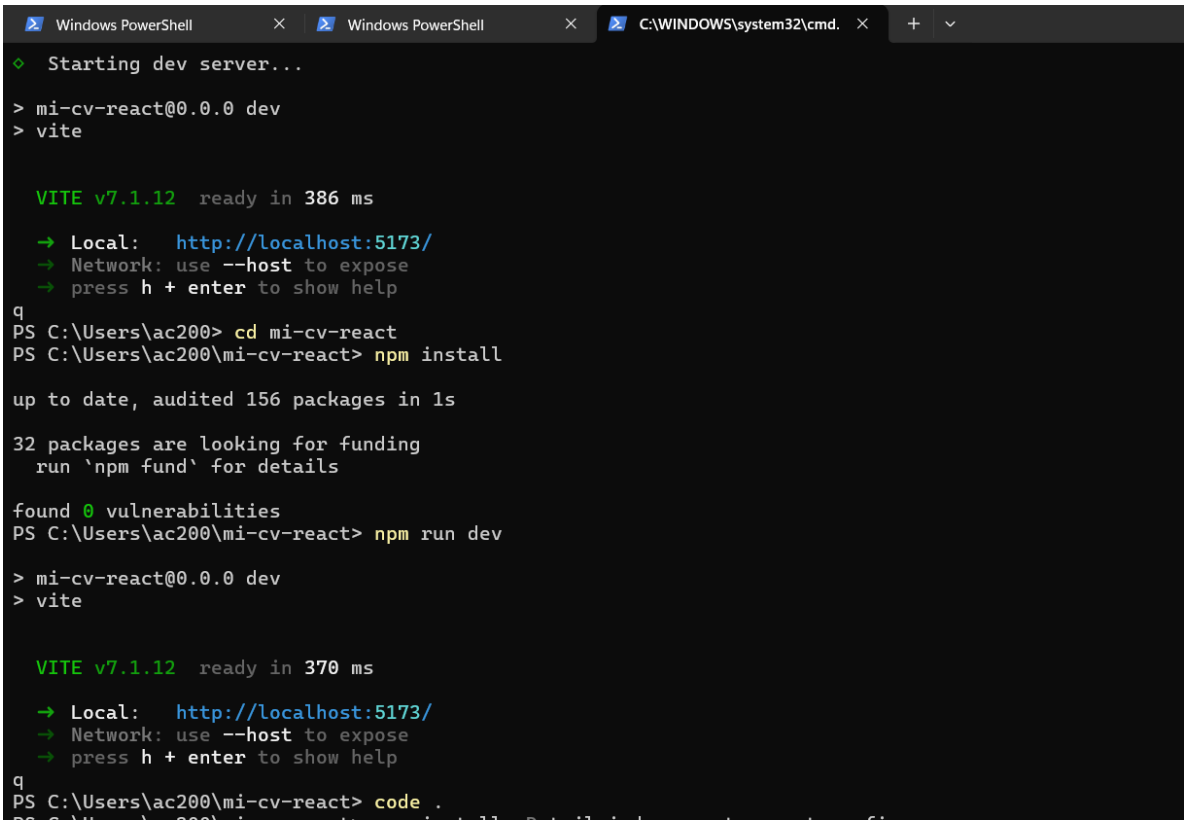
npm init –y



- En mi caso solo verifique y puse el comando **npm create vite@lastest mi-cv-react** y elegí las opciones para lo cual lo necesitaba, luego de esto entre en el mismo con **cd mi-cv-react** para instalar las dependencias de npm con el comando **npm install**
- Luego intente correr mi proyecto en un localHost con el comando **npm run dev**, esto me arrojo el enlace donde se encontraba mi app react
- Utilice el comando **code .** Para abrirlo en mi Visual Studio
- Abre el archivo **package.json** y dentro de la sección **"scripts"**, agrega lo siguiente:

  "scripts": {
  "dev": "vite",
  "build": "vite build",
  "preview": "vite preview"
  }

  Esto permitirá ejecutar, compilar y previsualizar el proyecto de forma sencilla.



3. Verificar dependencias instaladas

Si deseas confirmar que las dependencias se instalaron correctamente, ejecuta el siguiente comando:

npm list --depth=0

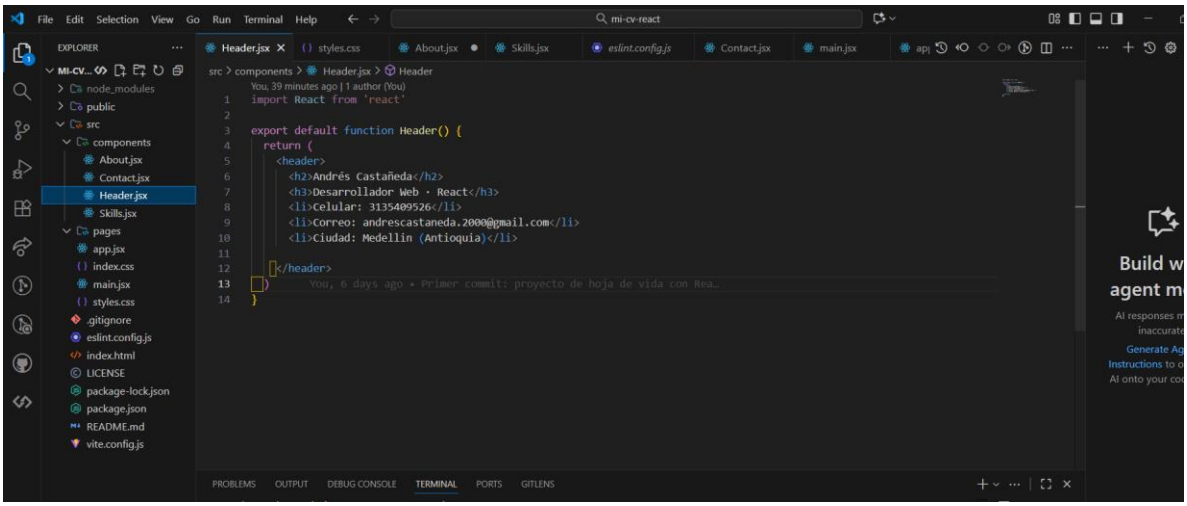
Esto mostrará una lista con los paquetes principales instalados.

4. Registrar cambios en Git y GitHub

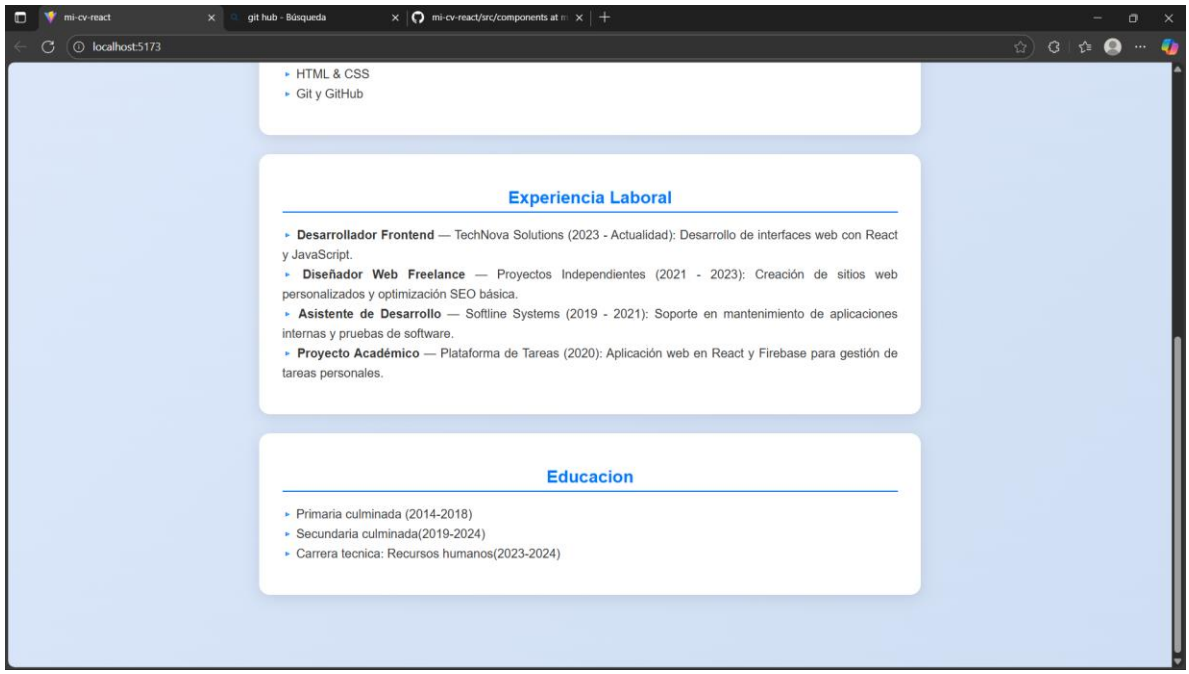
Finalmente, guarda los avances realizados en el proyecto y súbelos al repositorio remoto con los siguientes comandos:

git add . Sube a una sala de espera los archvivos a los que les hicite cambios
git commit -m "feat: proyecto configurado y ejecutado correctamente con npm"
git push origin main y este comando los sube al git hub, antes de ejecutar este comando debes tener un repositorio conectado con este proyecto, el comando para conectar estos dos (repositorio y proyecto) es **git remote add origin** y seguido va la ruta o URL de tu repositorio

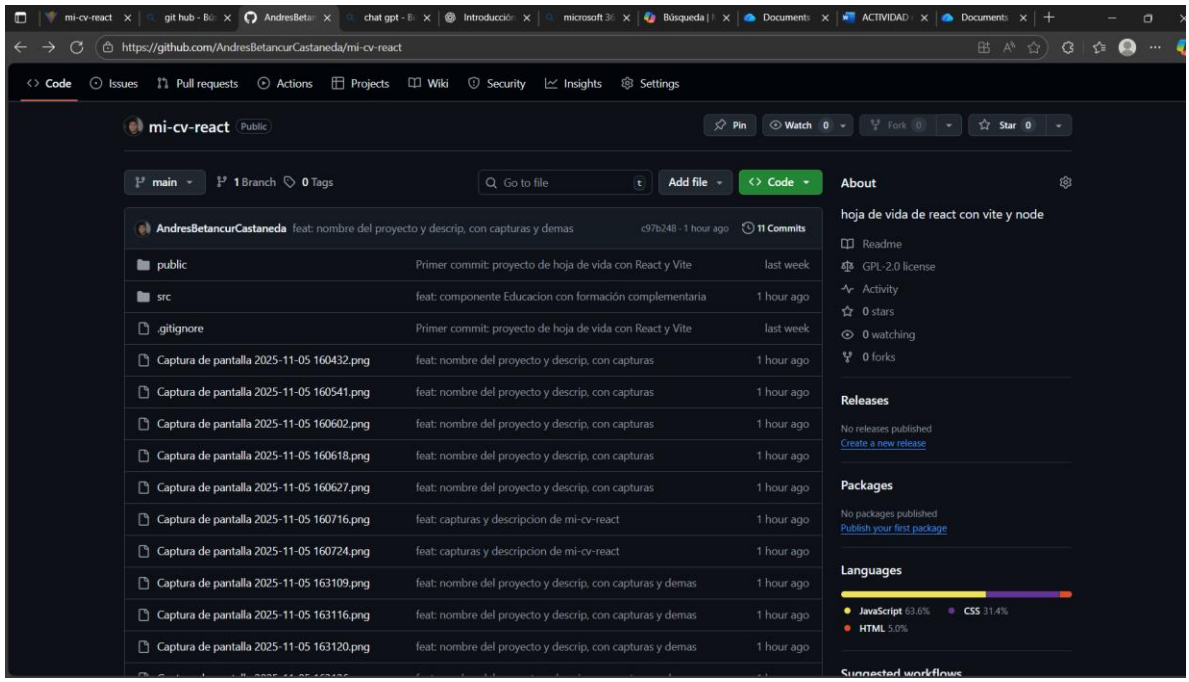
VISTA DE MI VISUAL STUDIO



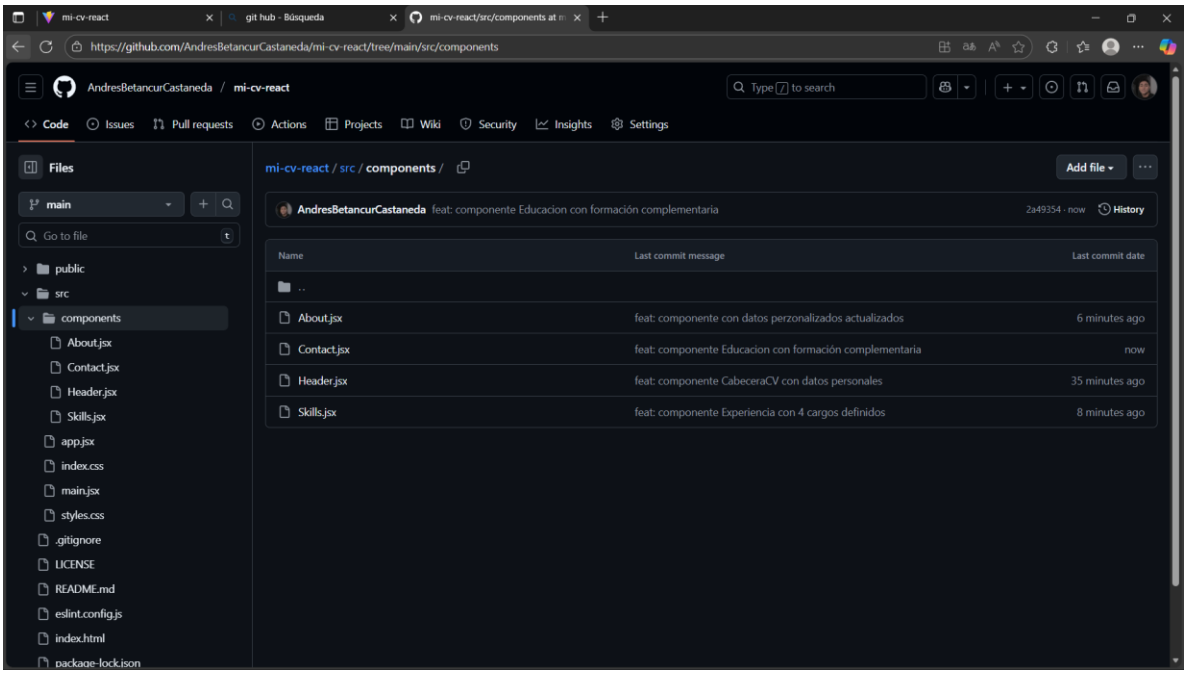
VISTA DE MI NAVEGADOR



VISTA DESDE MI REPOSITORIO



VISTA DE COMMITS REALIZADOS



Resultados Esperados

Con estas actividades espero haber afianzado mis conocimientos sobre el desarrollo web usando **React, Vite y npm**, además de mejorar mi manejo de herramientas básicas como **Git y GitHub**.

Durante el proceso aprendí a configurar correctamente mi entorno de trabajo, comprendiendo cómo se crean los proyectos desde cero con **Vite**, y cómo manejar la estructura de archivos que este genera. Esto me permitió entender mejor el flujo de

trabajo en un proyecto real, además de ver la importancia de mantener un código limpio y organizado.

Otro resultado esperado fue poder **crear un CV modular en React**, donde cada parte de la hoja de vida estaba separada en componentes. Gracias a esto, pude practicar la programación modular y entender cómo se relacionan los diferentes elementos dentro de una aplicación. También reforcé el uso de funciones, importaciones, props y la forma correcta de estructurar un proyecto para hacerlo más fácil de mantener.

Aprendí a **usar Git y GitHub** como herramientas de control de versiones, realizando commits y subiendo mi proyecto al repositorio remoto. Esto me ayudó a comprender cómo se guarda y comparte el código, y la importancia de tener un historial claro de los cambios realizados.

Finalmente, con **npm** logré entender cómo se instalan y actualizan las dependencias del proyecto, además de conocer el papel del archivo `package.json`. Este conocimiento me ayudará en el futuro a trabajar con nuevas librerías o frameworks sin dificultad.

En general, el resultado esperado fue lograr manejar de forma más completa las herramientas necesarias para construir un proyecto con React, desde su instalación hasta su despliegue. Siento que logré avanzar mucho y ahora tengo una mejor base para seguir practicando y aprendiendo más sobre desarrollo web.

Conclusión

Después de realizar todas las actividades, puedo decir que aprendí muchísimo sobre el desarrollo front-end y sobre cómo usar herramientas modernas como **Vite**, **React** y **npm** de manera correcta y práctica. No solo entendí la teoría, sino que también pude aplicarla en ejercicios reales, lo que me ayudó a afianzar lo aprendido de una forma más completa.

Durante la configuración del entorno con **Vite**, me di cuenta de lo útil que es trabajar con tecnologías que facilitan y agilizan el desarrollo. Esta herramienta me permitió crear proyectos más ligeros, con una estructura ordenada y un mejor rendimiento.

En la parte de la **creación de mi hoja de vida modular en React**, logré aplicar los conceptos de la programación basada en componentes, comprendiendo cómo dividir una aplicación en partes pequeñas que se pueden reutilizar. Esto me ayudó bastante a mantener el código limpio y organizado. Además, el uso de **Git y GitHub** fue fundamental para aprender a documentar los avances y tener un control más profesional del trabajo, algo muy importante cuando se trabaja en equipo o se lleva un registro personal de los proyectos.

Por otra parte, el aprendizaje sobre **npm** me permitió entender mejor cómo funcionan las dependencias en los proyectos de JavaScript. Con la práctica, pude ver cómo se instalan, actualizan y controlan los paquetes, y cómo mantener estable el entorno de desarrollo.

En conclusión, todo este proceso me ayudó a fortalecer tanto mis conocimientos técnicos como mi manera de trabajar. Me siento más seguro al usar herramientas modernas y con una mejor base para seguir aprendiendo y creando proyectos por mi cuenta. Estas actividades fueron una gran oportunidad para crecer como programador y para entender cómo se aplican realmente las tecnologías que se usan hoy en el desarrollo web.