Abril - 2024

Version 0.1

ANGULAR V.16

con TypeScript + node.js

Andres Felipe Cárdenas Alarcon
Sergio Arley Puerto Moreno
Facultad de Ingeniería de Sistemas

TABLA DE CONTENDIÓ

Lista de imágenes

Lista de tablas

1. Requisitos previos del lector

Para empezar a aprender de Angular, es importante disponer de algunos conocimientos y manejo de algunas de las herramientas básicas que se trabajarán y se usarán a lo largo de este material. En la siguiente lista se presentan los requisitos previos con los que se debe estar familiarizado.

- **Conocimientos básicos de JavaScript:** Angular se basa bajo el lenguaje de programación interpretado JavaScript, por lo hay que tener un cierto conocimiento del lenguaje.
- **Familiaridad con la línea de comandos/terminal:** A lo largo del proceso de instalación y cuando se utiliza Angular, la creación de componentes y otros archivos, se trabajan bajo la línea de comandos (en Windows) o el terminal (en macOS y Linux). La familiaridad con los comandos básicos y la navegación es esencial para empezar a utilizar Angular.
- **Familiaridad con TypeScript (opcional):** Aunque no es estrictamente necesario, tener un conocimiento básico de TypeScript puede ser útil cuando se trabaja con Angular, ya que el framework está construido sobre TypeScript. Para tener más información se puede remitir a la documentación oficial.
- **Node.js y npm instalados**: Angular requiere que tanto Node.js como el Gestor de Paquetes de Node (npm) que estén instalados en su sistema. Si aún no los tiene instalados, a continuación, se presentan los pasos para esta y todas las herramientas que se usan a lo largo de este material.
- **Conocimientos de HTML/CSS:** A lo largo de este material, se trabajará con plantillas HTML y los estilos que provee Bootstrap 5. La familiaridad que el lector tiene con respecto a los conocimientos básicos de las etiquetas HTML y su significado, así como algunos de los conceptos de CSS.
  g

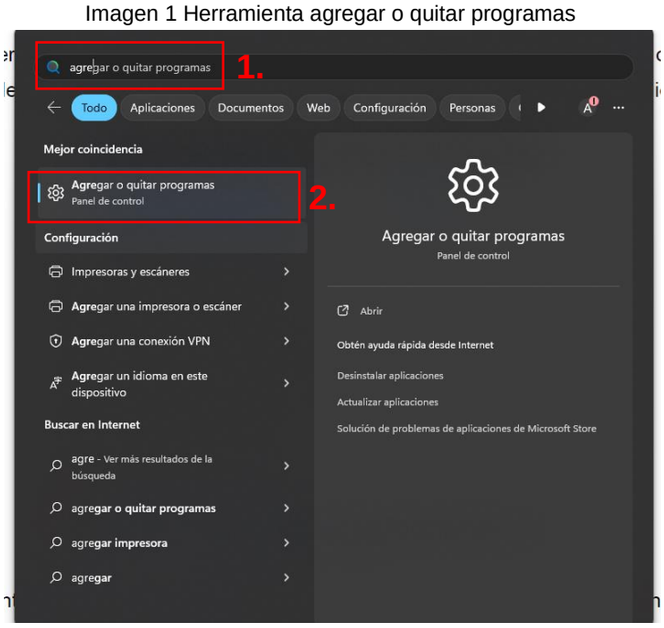
2. Instalación de herramientas

Si ha instalado versiones inferiores a la 16.17.5 LTS de Node JS, el siguiente apartado puede ser muy útil pues le permitirá remover las versiones antiguas e instalar una versión más actual de Node JS.

2.1. Remover versiones anteriores Node JS

Para remover instalaciones previas de Node JS se utilizará la herramienta agregar o quitar programas de Windows. La Imagen 1 presenta los pasos a seguir para lanzar la aplicación.

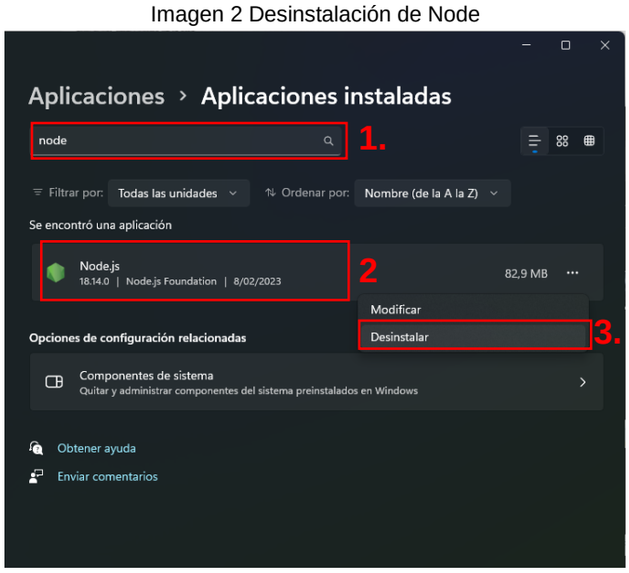
Imagen 1 Herramienta agregar o quitar programas



Fuente Autor.

La herramienta para eliminar programas de Windows presenta una caja de texto en donde se puede escribir el nombre del programa que se desea eliminar, con esto se filtran las demás aplicaciones y se previenen errores. La Imagen 2 presenta la interfaz de la herramienta para eliminar programas.

Imagen 2 Desinstalación de Node

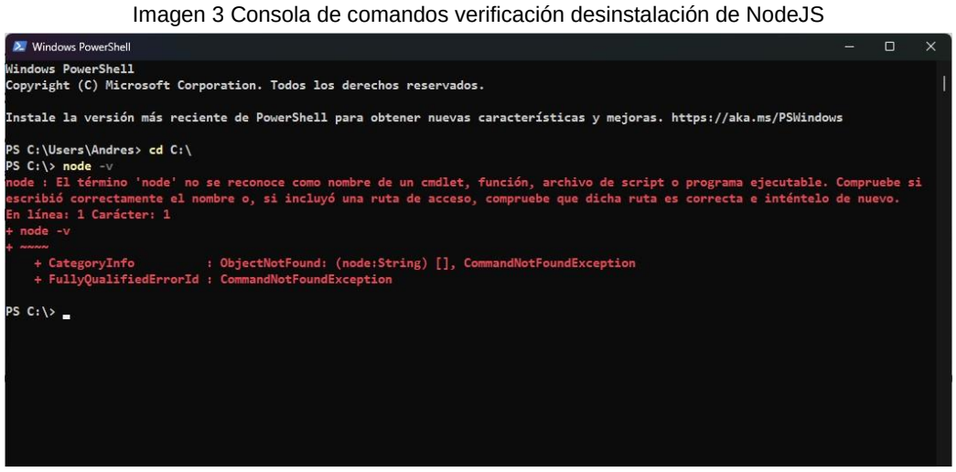


Fuente Autor.

Finalizado el proceso de desinstalación del Node JS, es posible verificar que efectivamente ha sido desinstalado, para esto, se debe abrir una consola de comandos de Windows (CMD). La Imagen 3 presenta la verificación de la desinstalación de Node JS. El comando para verificar la existencia de Node JS en el equipo es:

```
node -v
```

Imagen 3 Consola de comandos verificación desinstalación de NodeJS

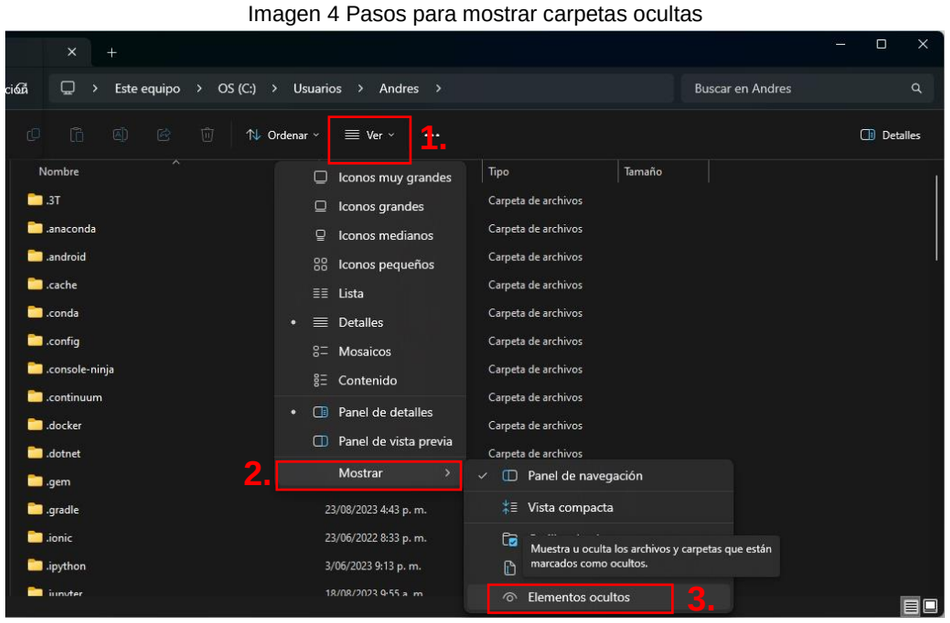


Fuente Autor.

Uno de los problemas frecuentes al desinstalar aplicaciones es la persistencia de archivos adicionales. La información adicional que no es eliminada generalmente corresponde a archivos de configuración o caché, los cuales son almacenados para mejorar la experiencia del usuario ante una posible nueva instalación. Sin embargo, es recomendable eliminar todo rastro de la versión anterior de Node JS y de posibles Frameworks adicionales que hayan sido instalados. La Imagen 5 presenta las carpetas que deben ser eliminadas manualmente para finalizar el proceso de desinstalación de Node JS.

Nota: Pero antes de todo debemos mostrar los elementos y capetas ocultas como se muestra en la Imagen 4.

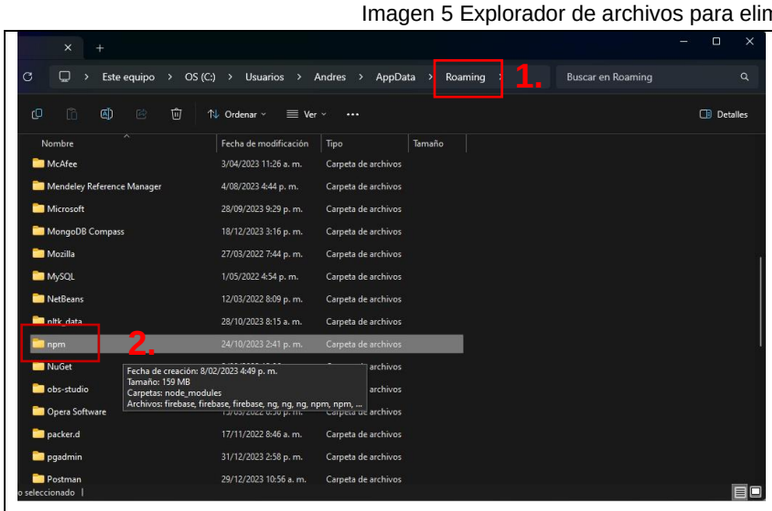
Imagen 4 Pasos para mostrar carpetas ocultas



Fuente Autor.

Esta son las carpetas que debemos eliminar de manera manual.

Imagen 5 Explorador de archivos para eliminar carpetas NPM



Fuente Autor.

2.2. Instalación de Node JS

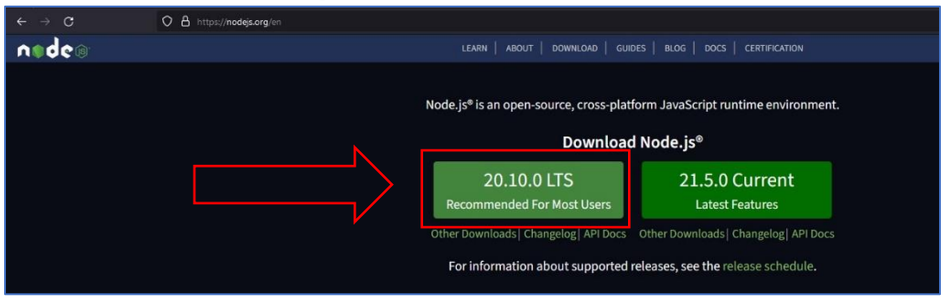
Requisitos para NodeJs: Node.js no requiere una configuración de hardware sofisticada para funcionar; la mayoría de los ordenadores de esta época deberían manejar Node.js de forma eficiente. Incluso los ordenadores más miniaturizados como el BeagleBone o el Arduino YÚN pueden ejecutar Node.js.

Node es ambiente para ejecutar código en JavaScript. Si una aplicación necesita o utiliza código en JavaScript es susceptible de utilizar Node para ejecutar código en un servidor a través de Ajax. Básicamente funciona con una arquitectura basada en eventos[9].

La potencia de NodeJS se basa en la capacidad para gestionar la memoria, un sistema tradicional incrementa el uso de memoria a medida que un usuario se conecta, esto implica que la cantidad de usuarios conectados va a depender de la cantidad de RAM del equipo, por ejemplo, si un servidor tiene 32 gigas en RAM, tiene una alta probabilidad de tener capacidad para administrar máximo 16.000 usuarios, lo cual es un número aceptable. No obstante, ¿Qué pasa si los usuarios superan por mucho ese número? Acá es donde entra Node JS pues no utiliza el concepto de hilos, por el contrario, ejecuta en su motor un nuevo evento cuando se ejecuta un nuevo usuario, lo cual implica una optimización extrema del uso de la memoria, hasta el punto de permitir hasta millones de usuarios. Si es desarrollador de sistemas Web tradicionales, es posible que le interese este artículo: https://kinsta.com/es/blog/nginx-vs-apache/ [9]

Como se presenta en la Imagen 6, se puede descargar de forma gratuita del sitio: https://nodejs.org/es/

Imagen 6 Sitio Web oficial de NODE JS



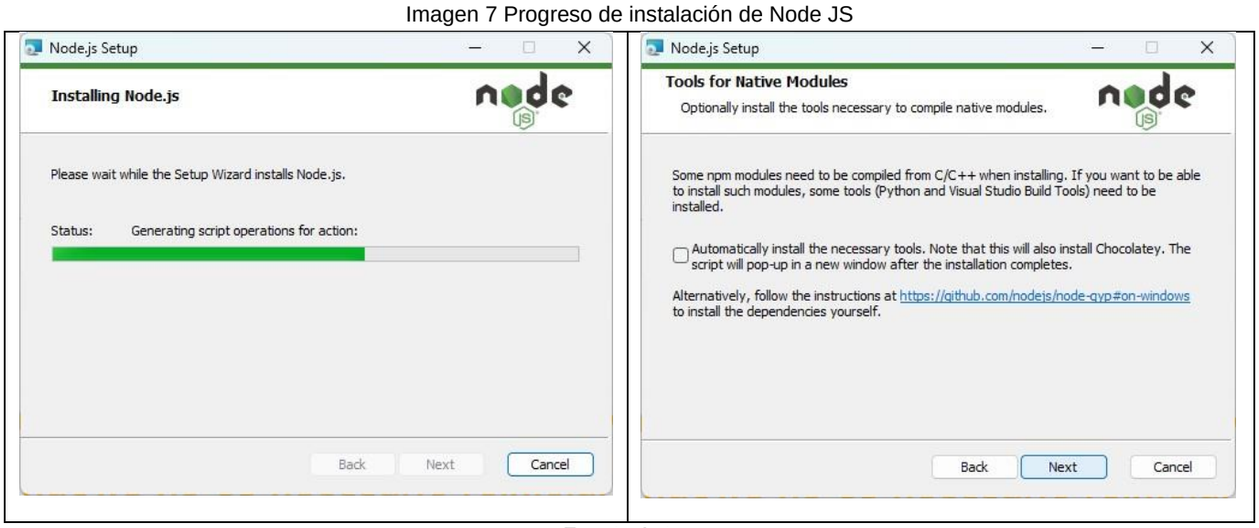
Fuente Autor.

En esta guía se utilizará la versión 20.10.0 LTS para Windows de 64 Bits. Si aún desea utilizar una versión previa de Node JS, asegúrese que sea una versión superior a Node JS 16.0, las versiones anteriores probablemente no funcionarán con este contenido.

¿Por qué usar una versión recomendada en vez de la versión actual? como se puede visualizar en la Imagen 6 la versión 20.10.0 LTS termina con las siglas LTS, esto significa que la versión tiene soporte a largo plazo, garantiza que los errores críticos se solucionaran en un lapso corto, lo que no sucede con la versión actual que no cuenta con el soporte a largo plazo.

Después de descargar Node JS, se debe realizar la instalación. La instalación no requiere conocimientos técnicos, es sencilla y basta con dar clic en siguiente varias veces. Se sugiere no cambiar las opciones por defecto de la instalación. La Imagen 7 presenta el progreso de la instalación de Node JS.

Imagen 7 Progreso de instalación de Node JS



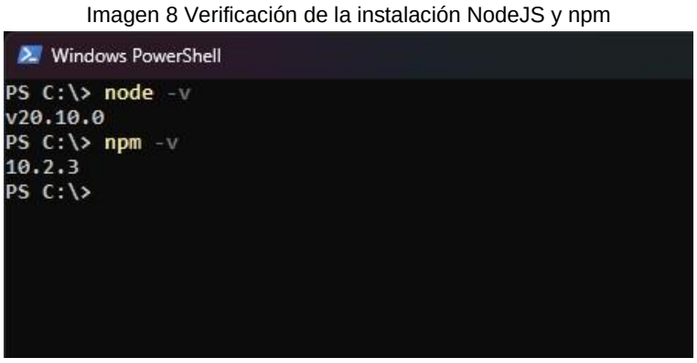
Fuente Autor.

Se recomienda reiniciar el equipo Ya que Windows después de borrar e Instalar NODEJS no actualiza los nuevos cambios.

Para verificar la instalación de Node JS y su gestor de paquetes npm (Node Package Manager) se debe abrir una consola de comando de Windows (CMD) como se aprecia en la Imagen 8 para ejecutar los siguientes comandos:

```
node -v
npm -v
```

Imagen 8 Verificación de la instalación NodeJS y npm



Fuente Autor.

2.3. Instalación de Angular

Requisitos para AngularJs: Para AngularJs es esencial contar con los siguientes requisitos de nuestra maquina:

- Sistema operativo: Windows 10, macOS 10.10 (Yosemite) o posterior, o una distribución reciente de Linux (como Ubuntu 18.04 o posterior)
- Memoria: Al menos 4 GB de RAM
- Espacio de almacenamiento: Al menos 10 GB de espacio libre en disco

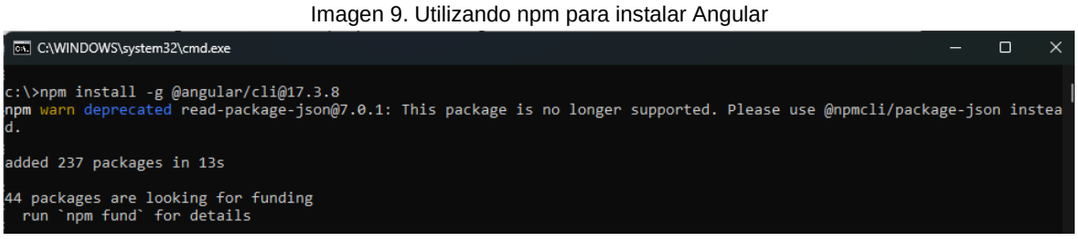
Estos requisitos garantizan que Angular se ejecute sin problemas en nuestro sistema para desarrollar y probar las aplicaciones con eficacia.

Para instalar Angular es necesario tener instalado NodeJS y por ende el gestor de paquetes npm. Angular se puede instalar de forma **global**, esto quiere se podrá utilizar desde cualquier carpeta del disco e instalará la última versión disponible. EL comando para instalar Angular es:

```
npm install -g @angular/cli@17.3.8
```

La Imagen 9 presenta el resultado de la instalación global de angular en el equipo.

Imagen 9. Utilizando npm para instalar Angular



Fuente Autor.

Para verificar la versión de Angular instalada se debe ejecutar el siguiente comando en una consola de Windows:

```
ng version
```

La Imagen 10 presenta el resultado del comando ng version.

Imagen 10 Versión instalada de angular, nodeJs y npm

Fuente Autor.

Si ya tenía instalada una versión de Angular y se desea conservar la versión de Node JS, en posible desinstalar solamente angular de la siguiente manera:

```
npm uninstall -g @angular/cli
npm cache clean
npm cache verify
npm cache clean --force
```

En caso de tener problemas en la instalación de estas dos herramientas en este Link podrá encontrar una guía de instalación más detallada.

2.4. Instalación de TypeScript

Este lenguaje para apoyar el desarrollo sobre JavaScript se puede utilizar de dos formas, por un lado, realizando la instalación de manera global, así cualquier proyecto tanto en angular como en NodeJS lo podrá utilizar. Por otro lado, instalándolo como un paquete adicional en un proyecto particular. En esta guía vamos a instalar TypeScript de manera

global a través de la ejecución del siguiente comando.

```
npm install -g typescript
```

Para verificar la instalación de TypeScript se utiliza el siguiente comando:

```
tsc -v
```

La Imagen 11 presenta el resultado de la instalación global de TypeScript y la versión Instalada

Imagen 11 Instalación de typescript y su versión

Fuente Autor.

2.5. Instalación de Visual Studio Code

Requisitos para Visual Studio Code: VS Code es compatible con las siguientes plataformas:

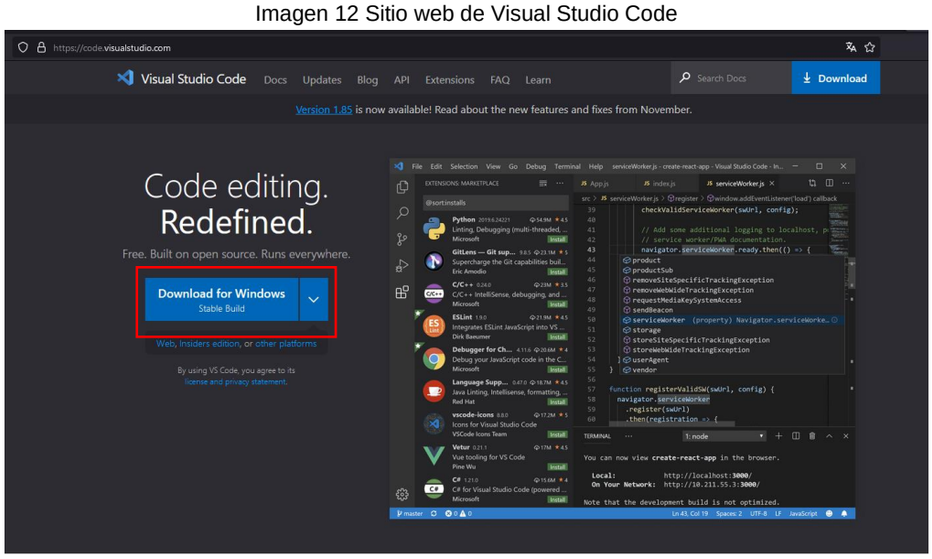
- Windows 10 y 11 (64 bits)
- Versiones de macOS compatibles con actualizaciones de seguridad de Apple. Esta suele ser la última versión y las dos versiones anteriores.
- Linux (Debian): Escritorio Ubuntu 20.04, Debian 10
- Linux (Red Hat): Red Hat Enterprise Linux 8, Fedora 36

Hardware: Visual Studio Code es una descarga pequeña (< 200 MB) y ocupa un espacio en disco menos de 500 MB. VS Code es liviano y debería ejecutarse fácilmente en el hardware actual. Se recomienda:

- Procesador de 1,6 GHz o más rápido
- 1 GB de RAM

La descarga del editor seleccionado se realiza del sitio web https://code.visualstudio.com en la opción “***Download***” en la parte superior derecha. En la Imagen 12 se puede apreciar el sitio web del editor de código seleccionado para el Frontend.

Imagen 12 Sitio web de Visual Studio Code



Fuente Autor.

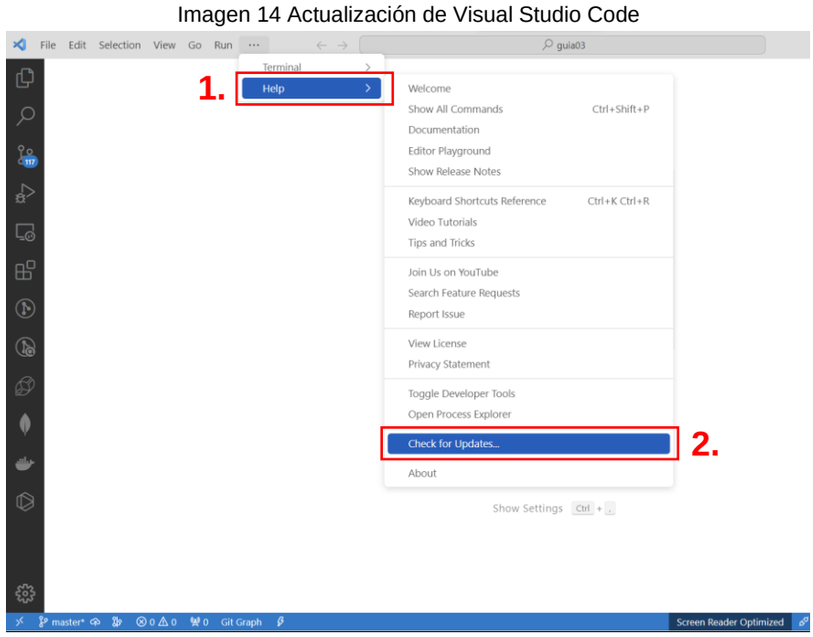
El proceso de instalación es simple, como todo instalador de Windows con varios clics en siguiente, siguiente y finalizar es suficiente. La Imagen 13 presenta la instalación exitosa

Imagen 13 Instalación correcta de Visual Studio Code

Fuente Autor.

Si ya tiene instalado el editor Visual Studio Code, no es necesario desinstalarlo, es suficiente con hacer clic en el menú ayuda (Help) y seleccionar la opción buscar actualizaciones (Check for Updates). La Imagen 14 presenta la forma de realizar la actualización en Visual Studio Code y las extensiones más útiles al momento de desarrollar en Angular.

Imagen 14 Actualización de Visual Studio Code

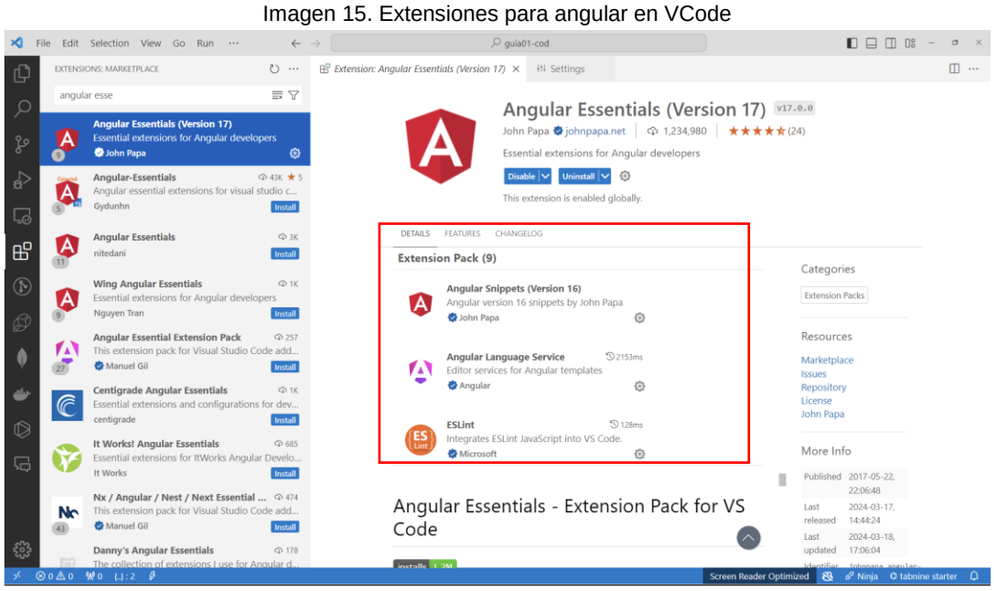


Fuente Autor.

2.6. Extensiones Angular VCode

Como se puede observar en la Imagen 15 Se recomienda instalar el paquete de extensiones llamado "Angular Essentials (Version 17)". Este conjunto de extensiones ha sido diseñado específicamente para potenciar el entorno de desarrollo AngularJS en la version 17.

Imagen 15. Extensiones para angular en VCode



Fuente: Autor

2.7. Extensión Auto Import

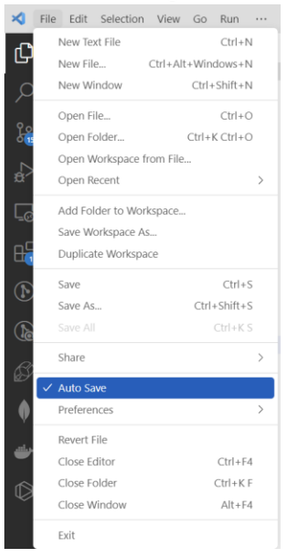
Encuentra, analiza y proporciona automáticamente acciones de código y finalización de código para todas las importaciones disponibles. Funciona con Typecript y TSX.

Imagen 16. Extensión para importación de files

Fuente: Autor

	Para optimizar la experiencia de desarrollo en Visual Studio Code, se recomienda activar la función de auto guardado (Auto Save) como se muestra en la Imagen 17.

Imagen 17 Recomendación del Auto guardado activo para Visual Studio Code

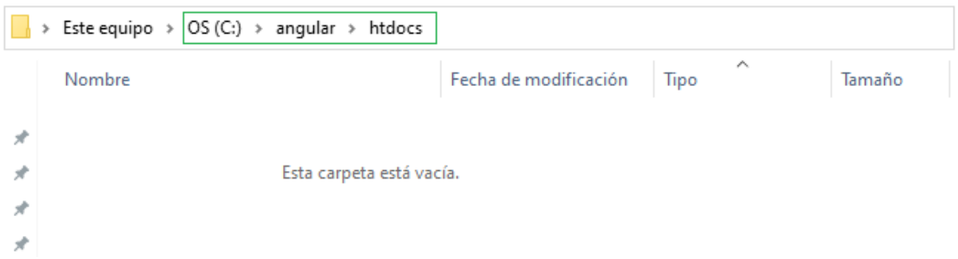


Fuente: Autor

2.8. Estructura de carpetas sugerida

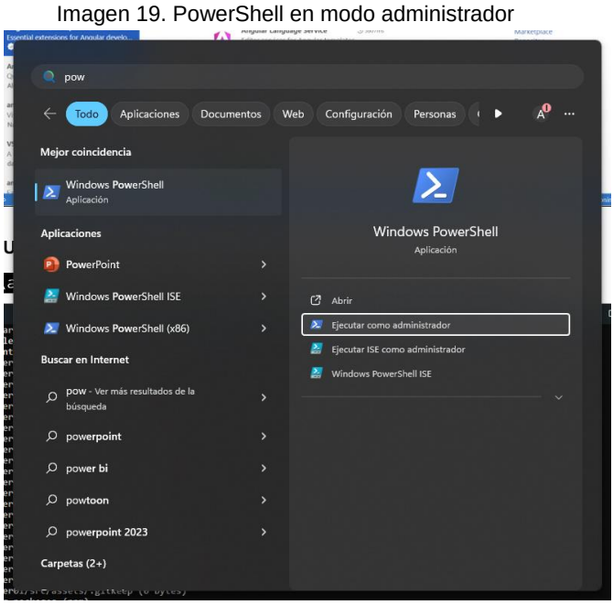
La organización de las carpetas para administrar los proyectos de Angular queda a criterio del desarrollador, sin embargo, a continuación, se propone una estructura de carpetas que permita organizar los proyectos de forma similar a como lo hacen otros lenguajes de programación Web. La Imagen 18 presenta la estructura de carpetas sugerida. La creación de estas carpetas se realiza de manera manual en el explorador de Windows o por la línea de comandos en una terminal.

Imagen 18. Estructura de carpetas



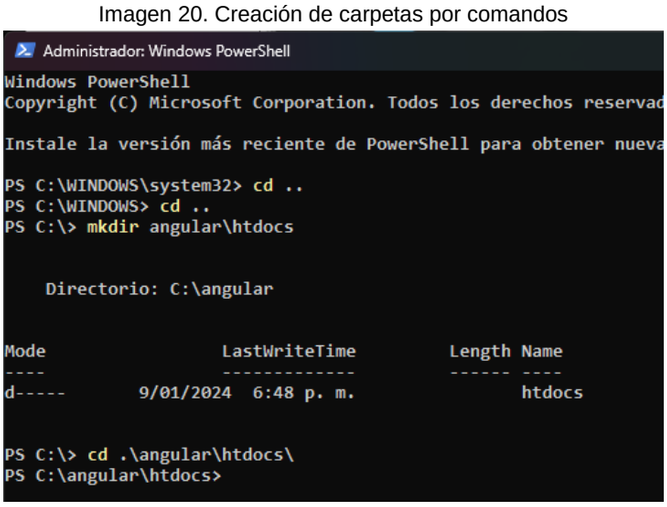
Fuente: Autor

Imagen 19. PowerShell en modo administrador



Fuente: Autor

Imagen 20. Creación de carpetas por comandos



Fuente: Autor

3. Presentación

Angular es un marco de trabajo basado en JavaScript y soportado por Google. Su función es crear aplicaciones del tipo SPA (*Single Page Aplicación*) eficientes y sofisticadas. La estrategia de Angular es administrar el software desarrollado a partir de módulos y componentes, por lo tanto, comprender la estructura de una página en forma de componentes es la clave para entender este marco de trabajo.

Las razones por la cual Angular eleva la productividad de los desarrolladores están soportadas en:

1. AngularCLI, es un conjunto de herramientas en modo consola que permite crear cualquier elemento necesario en Angular.

2. El uso de plantillas, en Angular reutilizar plantillas html5 es una tarea que simplifica el desarrollo Web.

3. Escalabilidad, Angular está diseñado para que sus aplicaciones sean escalables de un proyecto pequeño a uno de nivel empresarial.

4. Desarrollar por componentes independientes hacen del proceso una tarea más eficiente.

Para complementar el proceso de desarrollo, el Frontend requiere de su contra parte de un Backend, por lo tanto, es importante introducir el entorno de desarrollo de JavaScript más popular: NodeJS.

NodeJS es un entorno de ejecución de JavaScript con una estructura basada en eventos para aplicaciones escalables. NodeJS se destaca por la comunicación con el protocolo HTTP con una baja latencia entre sus comunicaciones, lo cual lo convierte en un recurso atractivo para librerías o Frameworks web.

Como motor de persistencia, en estas guías de conocimiento se utilizará la base de datos PostgreSQL. Es el más grande sistema de bases de datos relacionales de tipo open source en la actualidad. Desde su lanzamiento, PostgreSQL ha demostrado ser confiable, asegurando la integridad en lo datos, la extensibilidad y su adaptabilidad a volúmenes de información altos con tolerancia a fallos.

3.1.1. Características de la máquina utilizada en el desarrollo

Ítem	Descripción
Procesador	11th Gen Intel(R) Core (TM) i5-1135G7 @ 2.40GHz
Memoria física total	12 gigas
Espacio disco disponible	480 gigas SSD
Adaptador de pantalla	Intel(R) Iris(R) Xe Graphics
Sistema operativo	Windows 11 version 22H2

Fuente: Autor

4. Repositorios Git con el "*Scaffolding*" de la guía 01

En los siguientes repositorios encontrara la estructura de carpetas, archivos e imágenes que se necesitaran para desarrollar el proyecto.

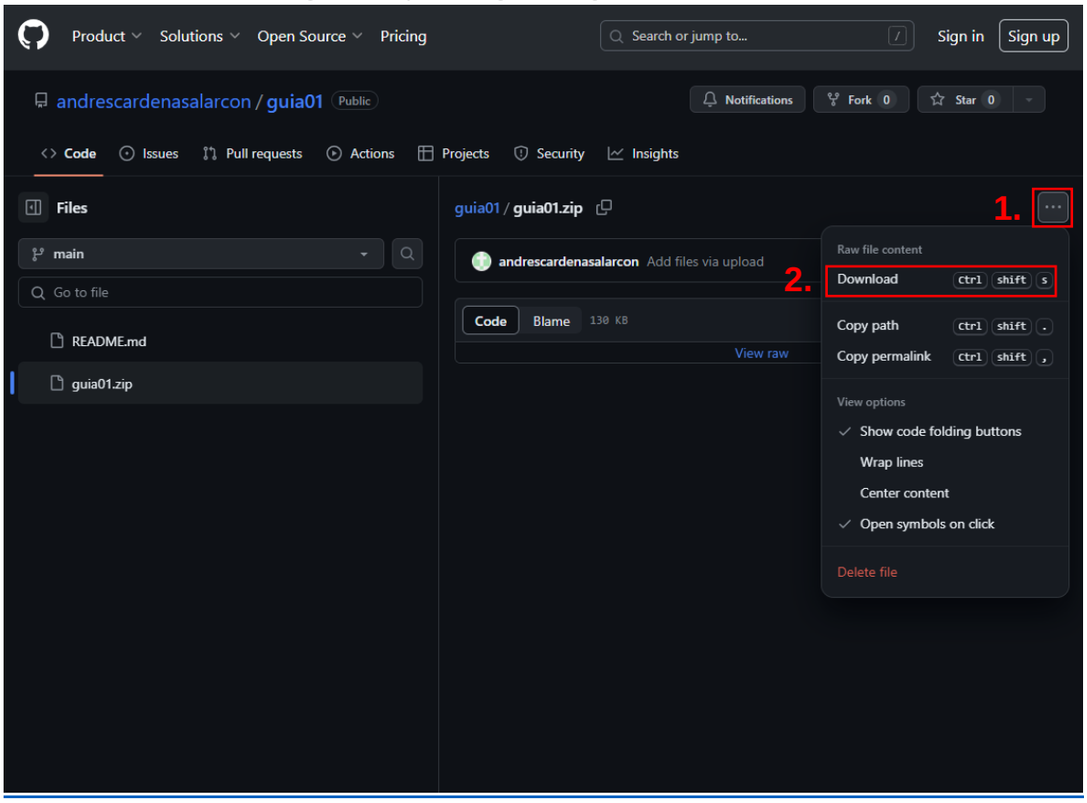
https://github.com/andrescardenasalarcon/guia01/blob/main/guia01.zip
https://github.com/andrescardenasalarcon/guia02/blob/main/guia02.zip
https://github.com/andrescardenasalarcon/guia03/blob/main/guia03.zip

El siguiente proceso aplica para cualquier repositorio, accediendo al enlace de la guía a trabajar.

Pasos:

1. Descargar el proyecto a trabajar como se muestra en la Imagen 21.

Imagen 21 Repositorio git de las guías



Fuente: Autor

2. Descomprimir el archivo zip de la guía, accedemos a la carpeta que abriremos en nuestro editor de código VCode.
3. En la terminal de nuestro proyecto en VCode ejecutaremos el comando "npm i", esto para descargar las librerías node_modules como se muestra en la Imagen 22.

Imagen 22 Instalación de node_modules

Fuente: Autor

4. Empezar a aprender de AngularJs desde el punto [**5.3**] ✌ .

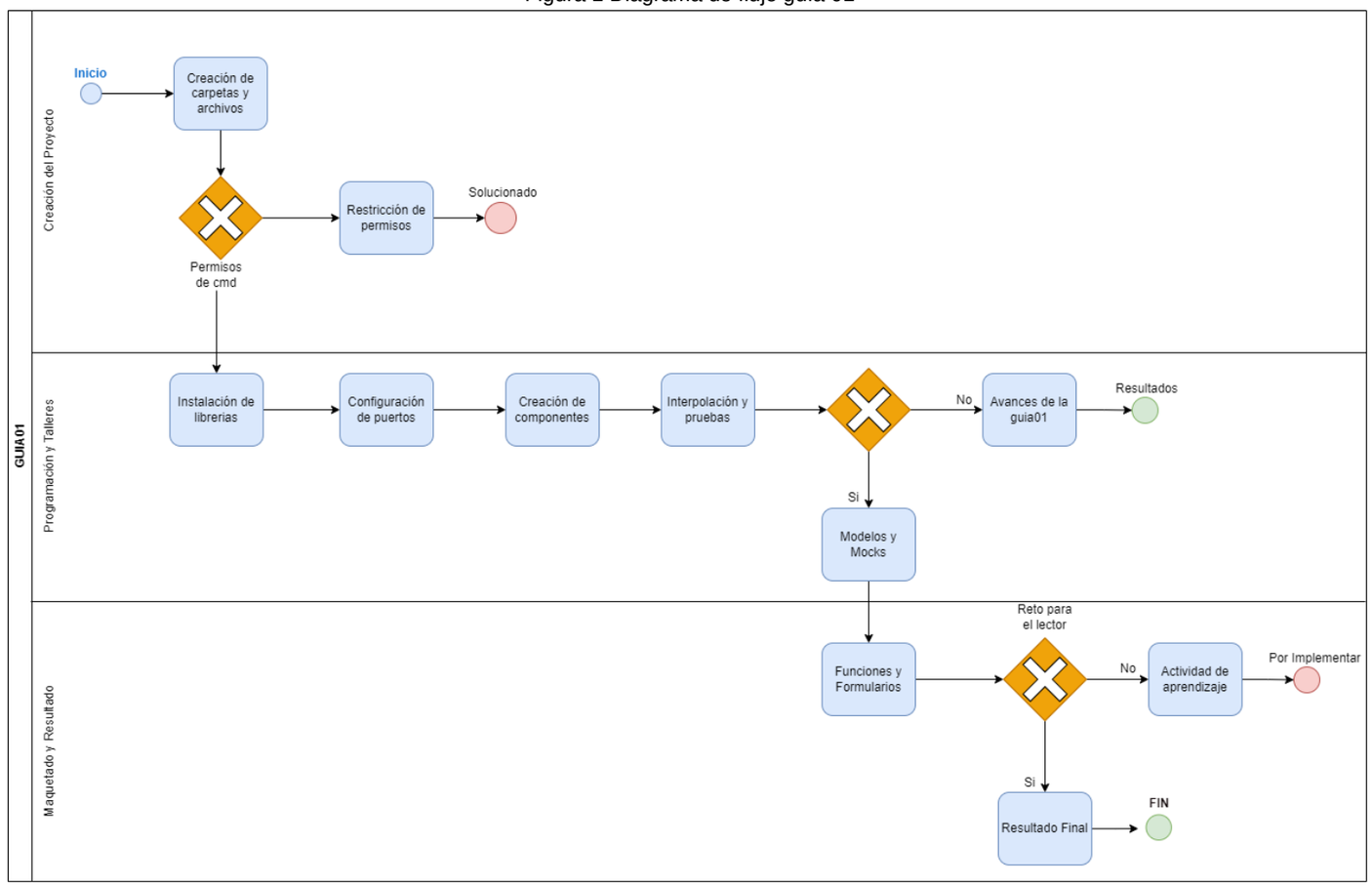
5. Guía 01

Requisitos y temáticas para la Guía 01	
Problemática Ejemplo	Una empresa fabricante de computadoras ha expresado su interés en desarrollar una aplicación web que le permita crear y configurar computadoras, con distintos periféricos. La aplicación debe permitir la configuración de componentes, incluyendo un monitor con especificaciones de marca y tamaño, así como un ratón y un teclado, ambos con especificaciones idénticas de dispositivo de entrada y marca.
Entradas	En la construcción de los talleres para la parte de Frontend se tiene propuestos una cantidad de talleres que serán acumulativos estos contendrán los contenidos temáticos para el área de Frontend en Angular.
Proceso	• Taller instalación y prerrequisitos es un proyecto generado por la CLI de Angular el cual solo es ejecutado. • Taller de estructura proyecto generado por la CLI de Angular contendrá el uso del componente principal la creación de estilos y maquetado HTML. • Taller múltiples componentes proyecto generado por la CLI de Angular contendrá la creación de componentes que son cargados en el componente principal. • Taller instalación de librerías proyecto generado por la CLI de angular en este taller se hace uso de Bootstrap y PopperJs creando dos componentes que usaran maquetado de este framework de diseño. • Taller uso de iconos y asignación de puerto proyecto generado por la CLI de angular en este taller se hará uso de la librería de iconos FontAwesome y se hace el cambio de puerto en el despliegue de los proyectos. • Taller de CRUD uso de modelos y pruebas de front, en este proyecto se genera el primer CRUD con su respectivo modelo y se hacen las pruebas de Frontend con un mock. • Taller para la gestión de formularios proporcionada por NgForm y así gestionar la entrada de datos y garantizar la validez de la información ingresada.
Resultados	• Imágenes adicionadas a guías de conocimiento

Fuente: Autor

En la Figura 1 se muestra el diagrama de flujo con todos los pasos y las temáticas que se van a abordar a lo largo de esta guía.

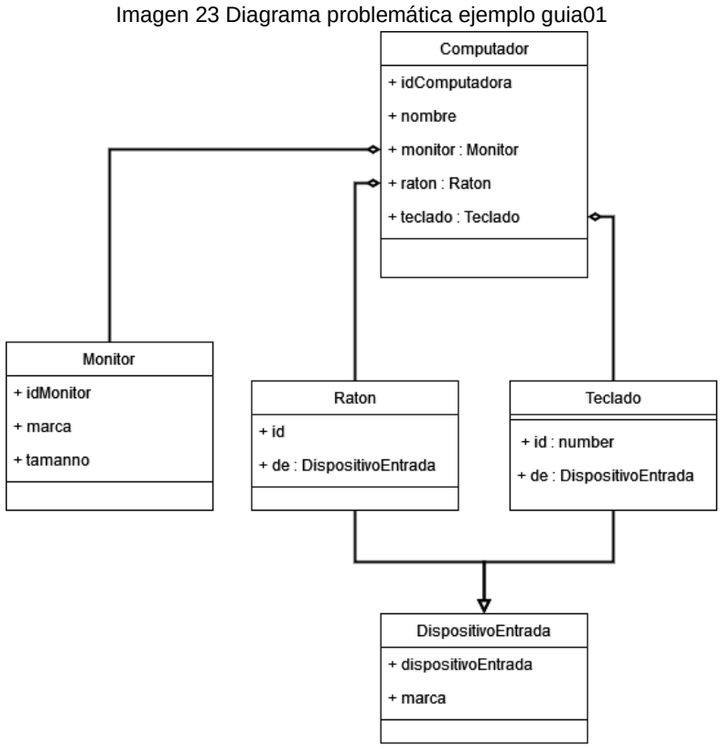
Figura 1 Diagrama de flujo guía 01



Fuente: Autor

En la Imagen 23 se ve un diagrama UML con las clases que se van a trabajar a manera de ejemplo para esta guía.

Imagen 23 Diagrama problemática ejemplo guia01



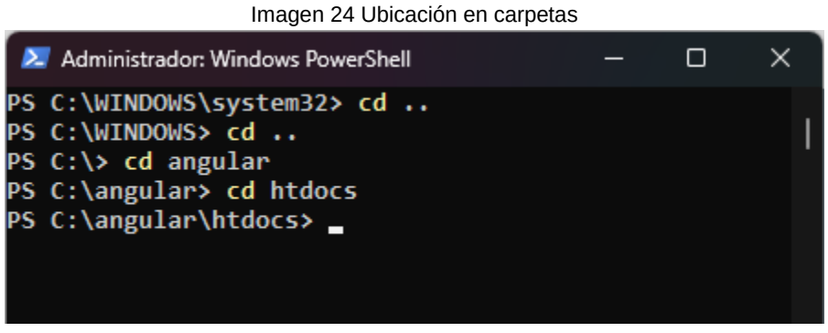
Fuente: Autor

	Se **recomienda** descargar y hacer uso del repositorio de GitHub, para el desarrollo de esta guía, ya que ahí se encontrará código y material ya preestablecido que ayudará en el proceso de aprendizaje e implementación de los temas propuestos. **3. <u>Repositorios Git con el "*Scaffolding*" de la guía 01</u>**

5.1. Ubicación del proyecto para la "Guia01"

Para ubicar un proyecto en Angular se debe abrir una consola de Windows y cambiar de directorio hasta la estructura sugerida en el punto anterior. La Imagen 24 presenta los pasos para cambiar de carpeta.

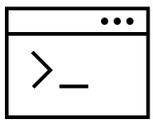
Imagen 24 Ubicación en carpetas



Fuente: Autor

5.2. Creación de un proyecto nuevo

Para crear un proyecto desde cero, mediante el CLI de angular, se sugiere ejecutar el siguiente comando para crear futuros proyectos.



En una carpeta de ejemplo como "**c:\angular\htdocs**" se debe ejecutar el comando:

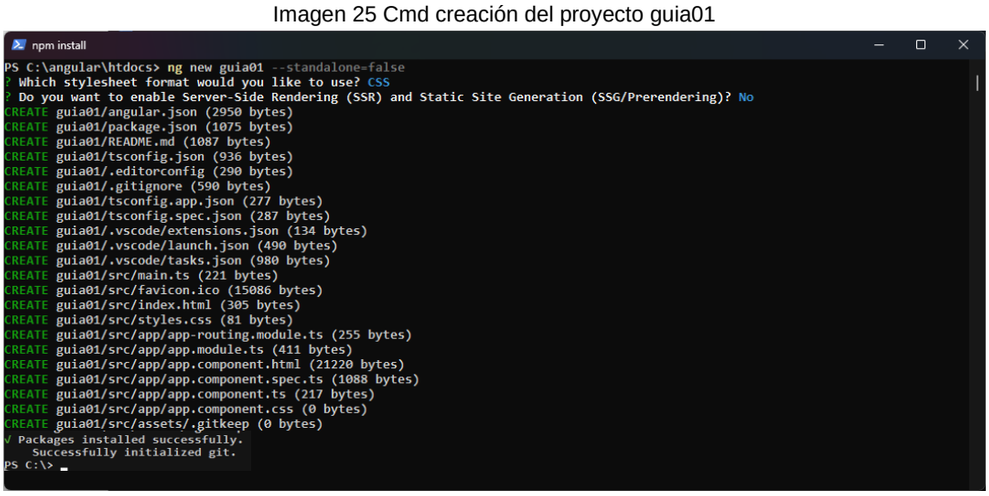
```
C:\angular\htdocs> ng new guia01 --standalone=false
```

Este es el nuevo comando (**en las últimas versiones de angular v17**) para poder crear un proyecto, esto por la estructura que se maneja en este material, se le adiciona el parámetro *--stanalone=false* indica que se debe crear un componente independiente (stanalone component) en lugar de un componente regular.

El comando creará la estructura básica de los archivos, conocida como "**Scaffolding**". La Imagen 25 presenta la ejecución del comando para crear un proyecto en Angular. Al ejecutar el comando *ANGULAR CLI* le hará dos preguntas:

*1) ¿Qué tipo de hoja de estilo se desea utilizar?, se sugiere **CSS**,*
*2) ¿Quiere habilitar el Server-Side Rendering (SSR) y los Sitios de Generación estática (SSG/Prerendering), se sugiere **NO***
con [Enter] iniciará el proceso de creación del proyecto. El proceso consiste en la creación de una estructura básica de proyecto la cual es descargada de internet.

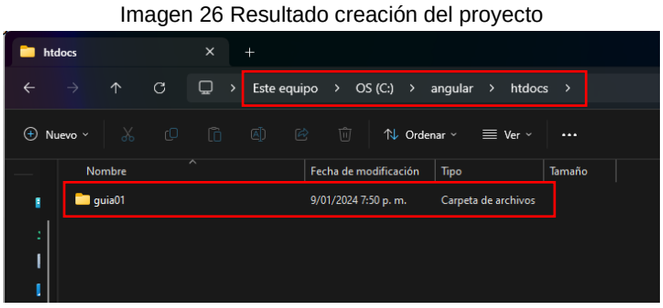
Imagen 25 Cmd creación del proyecto guia01



Fuente: Autor

En este caso la creación del proyecto fue exitosa y se generó una advertencia de inicialización de Git, puesto que en este computador está instalado **(No es necesario tener GIT instalado para crear un proyecto)**. En el explorador de archivos de Windows se puede observar la carpeta que se generó al crear el proyecto como se presenta en la Imagen 26.

Imagen 26 Resultado creación del proyecto

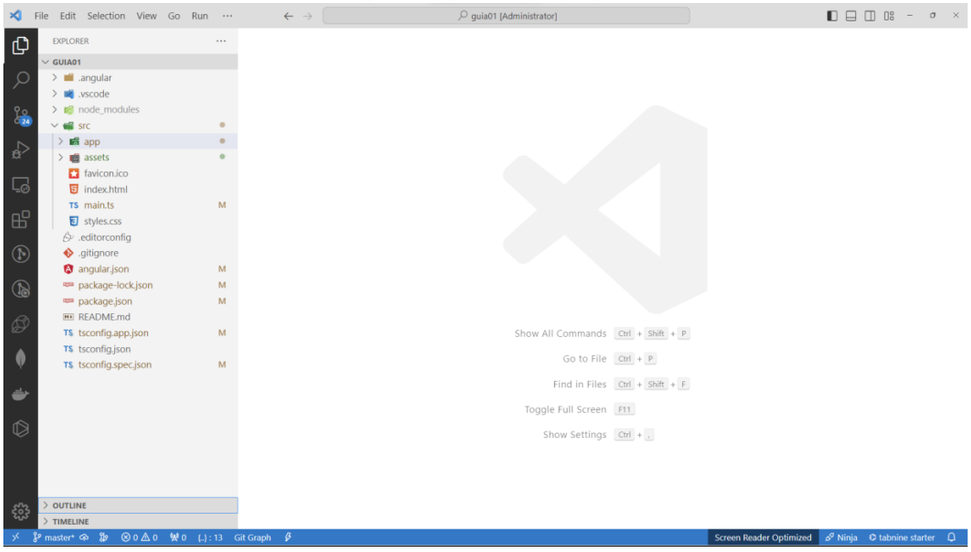


Fuente: Autor

5.3. Apertura del proyecto en VisualCode

El editor de código seleccionado no maneja el concepto de proyecto, en su defecto utiliza el concepto de carpeta. Esto quiere decir que se pueden ejecutar varias instancias de Visual Studio con diferentes carpetas abiertas. Con Visual Studio Code abierto, se debe seleccionar el menú "***archivo***" o "***File***" y luego la opción "***abrir carpeta***" o "***Open Folder***", se debe buscar la carpeta creada en el paso anterior. La Imagen 27 presenta el proyecto cargado.

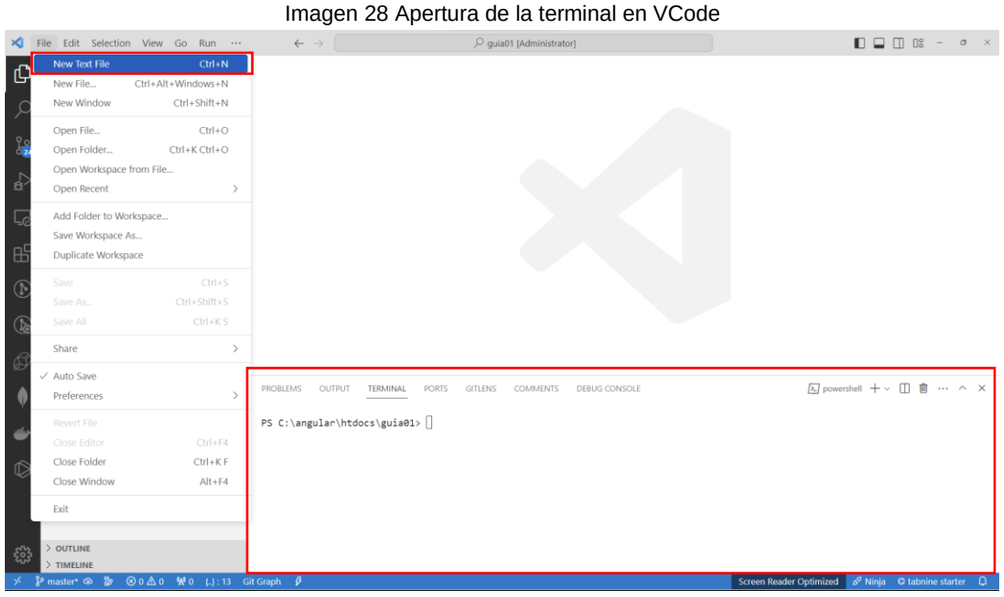
Imagen 27 Apertura del proyecto



Fuente: Autor

El siguiente paso es ejecutar el proyecto desde el editor, con esto se busca probar el editor de código y ver el resultado de correr el primer proyecto creado con Angular CLI. Para ejecutar el proyecto se procede abrir una terminal desde el editor. En el menú superior se selecciona la opción "***Terminal***" y luego "***New Terminal***". La Imagen 28 presenta el editor con la consola o terminal lista para recibir comandos.

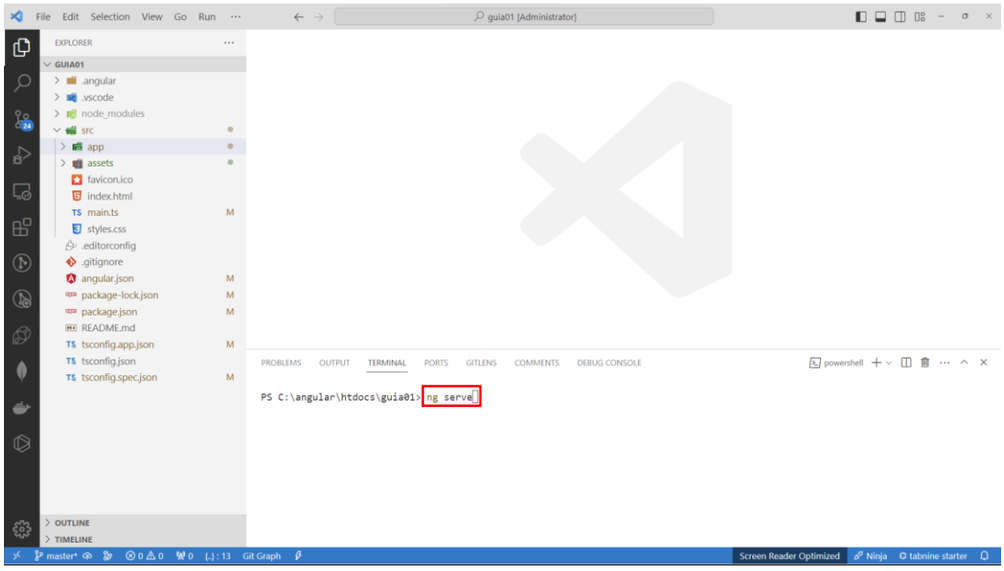
Imagen 28 Apertura de la terminal en VCode



Fuente: Autor

Para iniciar el proyecto creado con Angular CLI, se debe ejecutar desde la terminal de Visual Estudio Code, el comando "***ng serve***", la Imagen 29 presenta la distribución normal del editor de código seleccionado.

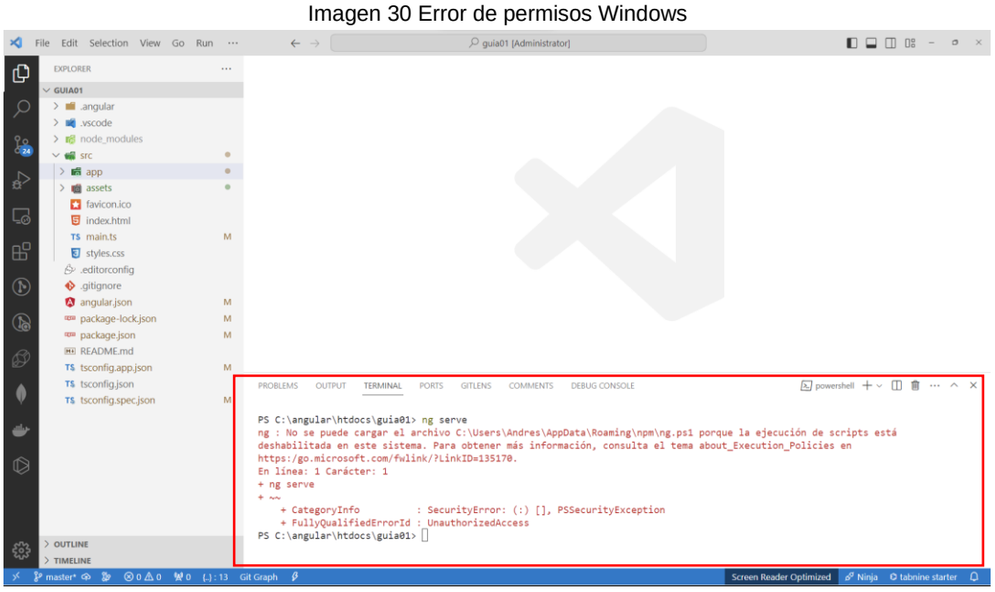
Imagen 29 Comando para ejecutar un proyecto Angular



Fuente: Autor

Al intentar ejecutar el proyecto desde el editor de código, puede que se presente un error. El error se da porque Windows no permite la ejecución de Scripts, esto se debe a las políticas de seguridad de Windows. Dado que los scripts que se van a utilizar serán creados por el desarrollador, es factible eliminar las restricciones para trabajar sin problemas con Visual Studio Code. La Imagen 30 presenta el error que se puede generar.

Imagen 30 Error de permisos Windows

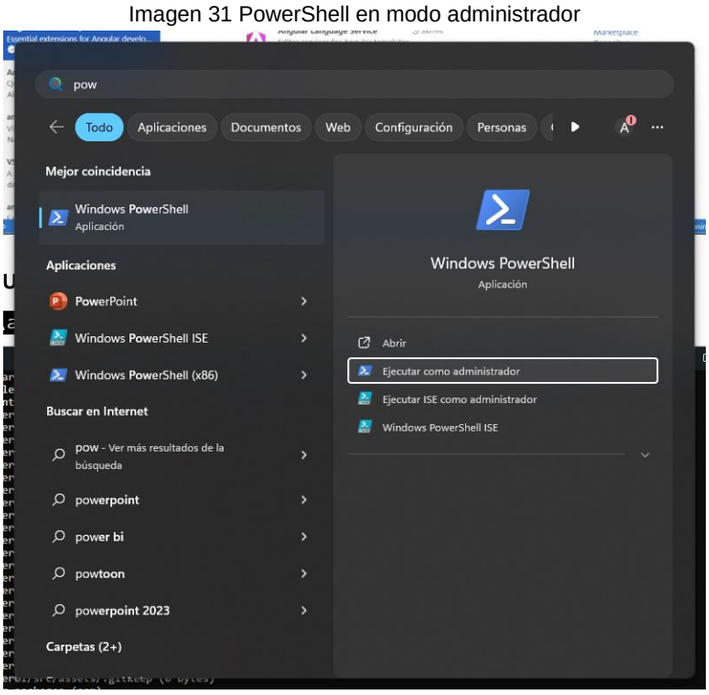


Fuente: Autor

5.3.1. Eliminación de restricciones scripts de la terminal

Para eliminar el error en la ejecución de Scripts se debe abrir una consola de PowerShell en modo Administrador. En la Barra de Windows se debe escribir "***PowerShell***" y aparecerá una opción como la presentada en la Imagen 31.

Imagen 31 PowerShell en modo administrador



Fuente: Autor

En la consola por defecto "Azul" de PowerShell se deben ejecutar los comandos para verificar las políticas en cuanto a ejecución de Scripts:

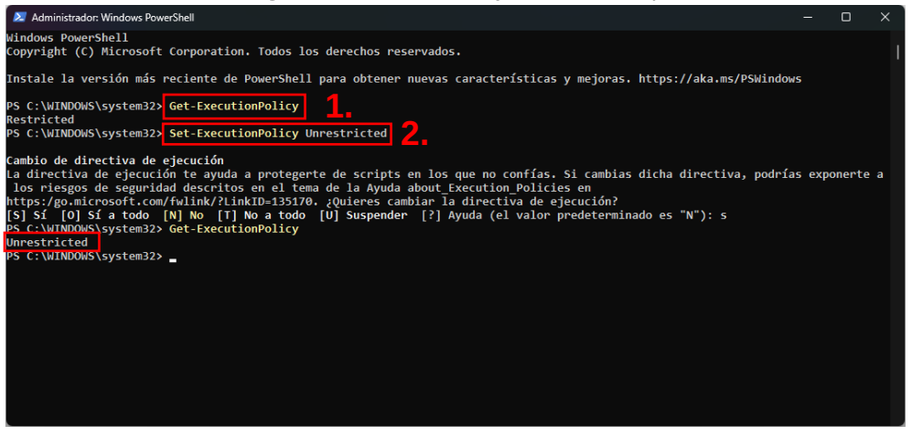
```
Get-ExecutionPolicy
```

El comando anterior se ejecuta para consultar el estado de las políticas de ejecución de Scripts.

```
Set-ExecutionPolicy Unrestricted
```

El comando anterior se ejecuta para eliminar las restricciones de ejecución de Scripts. La Imagen 32 presenta la ejecución de los comandos anteriores.

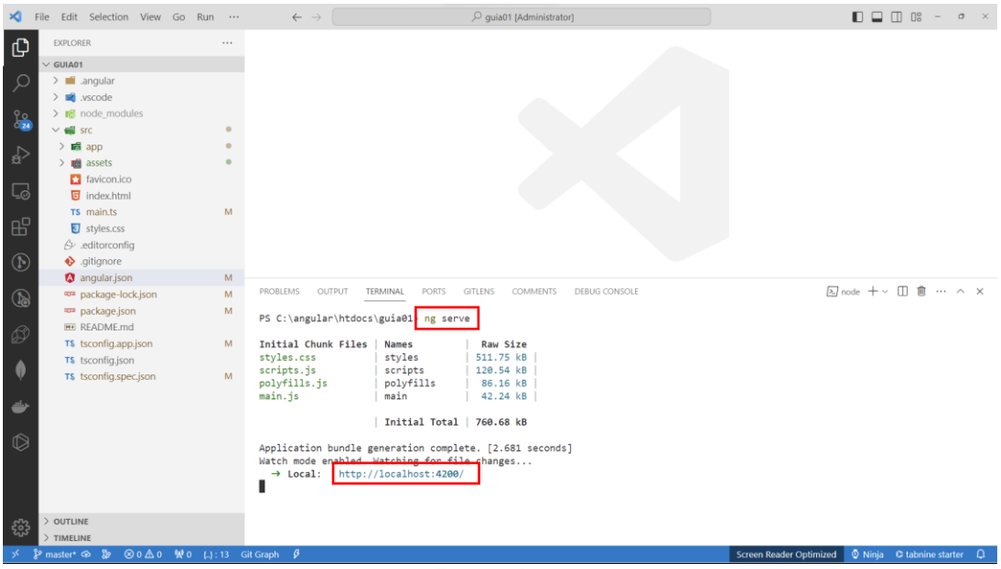
Imagen 32 Restricción de ejecución de Scripts



Fuente: Autor

Una vez eliminadas las restricciones para la ejecución de Scripts se procede a ejecutar nuevamente el comando "***ng serve***" en la terminal del Visual Studio Code. Al ejecutar el comando, Angular preguntará si se desean compartir datos de forma anónima con Google. Si la data que se está trabajando en el proyecto no es privada, se recomienda compartir los datos. El comando "***ng serve***" realizará un proceso de transpiración del código generado en el proyecto. La Imagen 33 presenta el resultado al ejecutar el servidor y habilitar el proyecto.

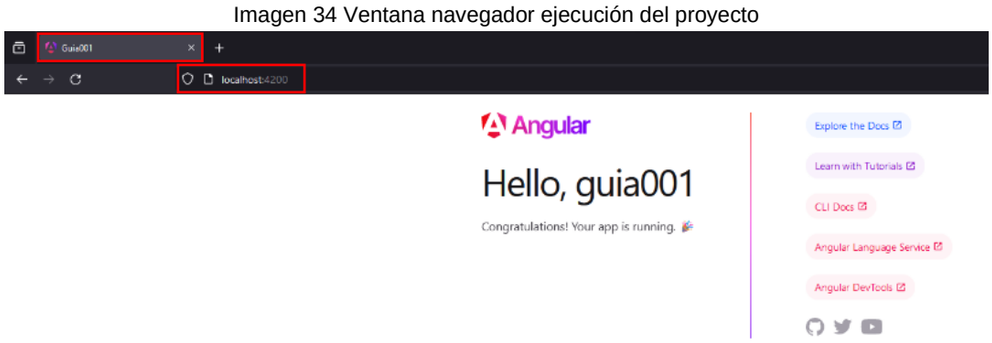
Imagen 33 Transpiración y generación de un proyecto



Fuente: Autor

El resultado de la compilación no arroja errores y permite ver que el proyecto puede ser verificado en un navegador Web en la URL http://localhost:4200. La Imagen 34 presenta el proyecto ejecutándose correctamente el navegador Firefox.

Imagen 34 Ventana navegador ejecución del proyecto

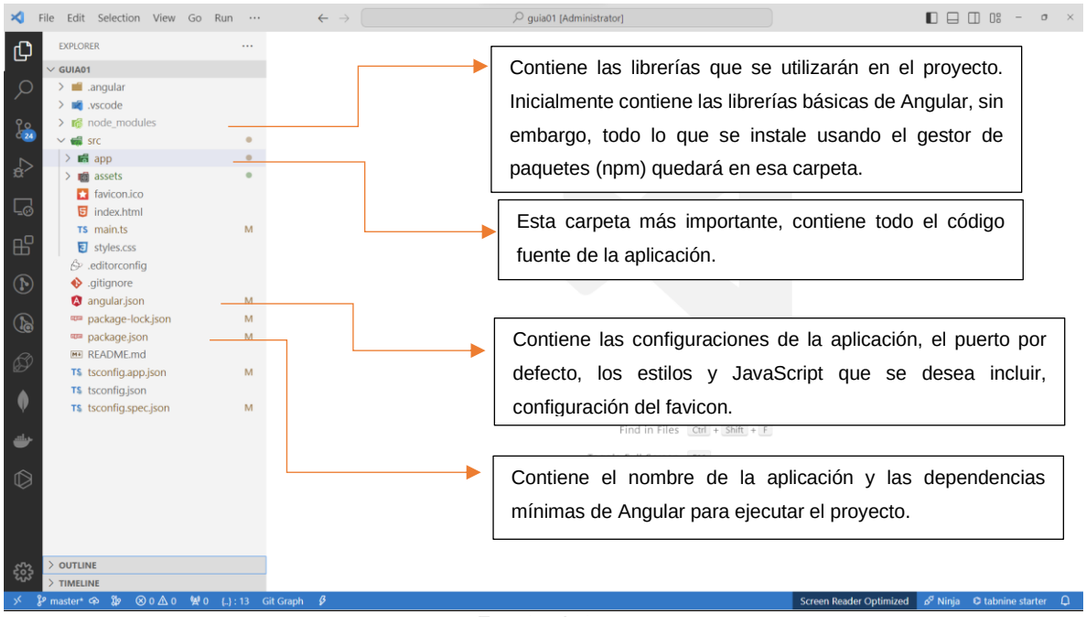


Fuente: Autor

5.3.2. Estructura y descripción de las principales carpetas y archivos de Angular

Si había otro proyecto abierto en el editor de código, sencillamente Visual Studio Code abrirá la nueva carpeta seleccionada y cerrará la anterior. La Imagen 35 describe las principales carpetas y archivos de un proyecto en Angular.

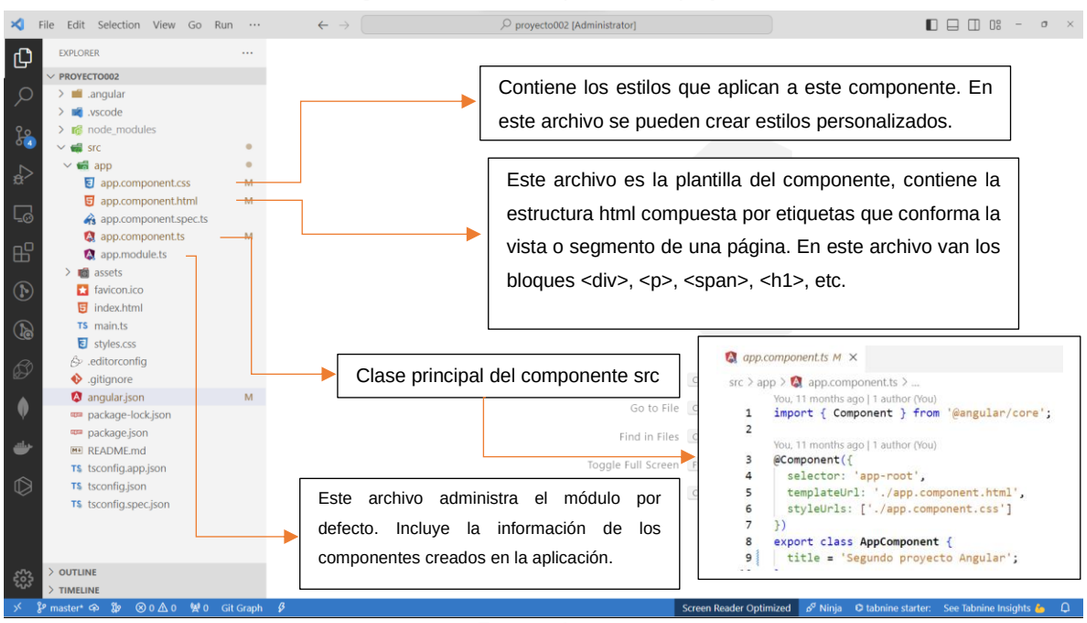
Imagen 35 Estructura general del proyecto



Fuente: Autor

Toda aplicación en Angular tiene un componente por defecto. Este componente se encuentra en la carpeta "**src\app**" y contiene 5 archivos. La Imagen 36 presenta los archivos que tienen un el componente por defecto.

Imagen 36 Estructura especifica del proyecto

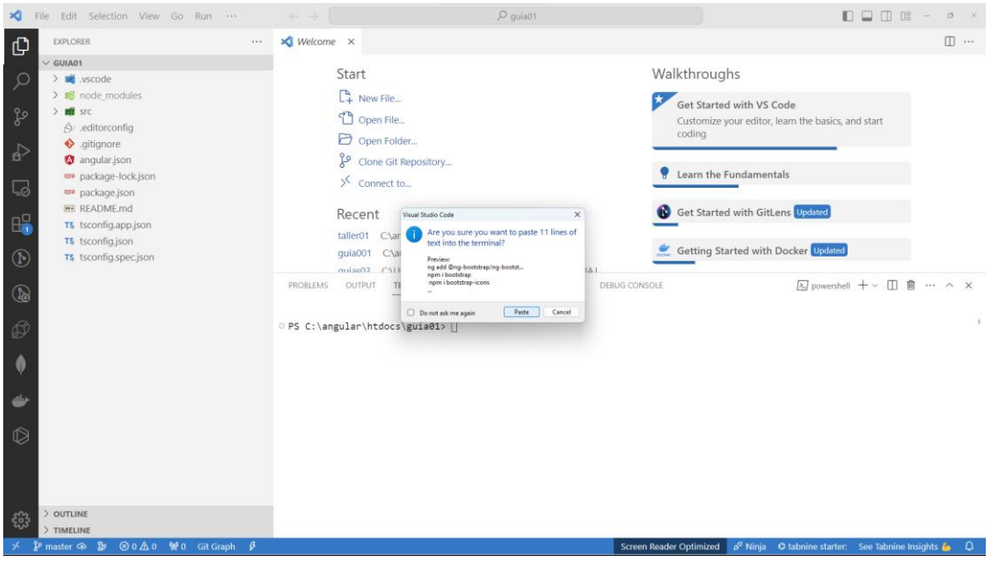


Fuente: Autor

5.4. Instalación de librerías

La siguiente Imagen 37 presenta el momento en donde se ejecutan todas las librerías necesarias para estas guías.

Imagen 37 Selección e instalación de librerías



Fuente: Autor

`ng add @ng-bootstrap/ng-bootstrap`

El comando ng add @ng-bootstrap/ng-bootstrap se utiliza para agregar la biblioteca ng-bootstrap a un proyecto de Angular. ng-bootstrap es una biblioteca que proporciona componentes de interfaz de usuario basados en Bootstrap para su uso en aplicaciones de Angular.

Imagen 38 Instalación de ng-boostrap

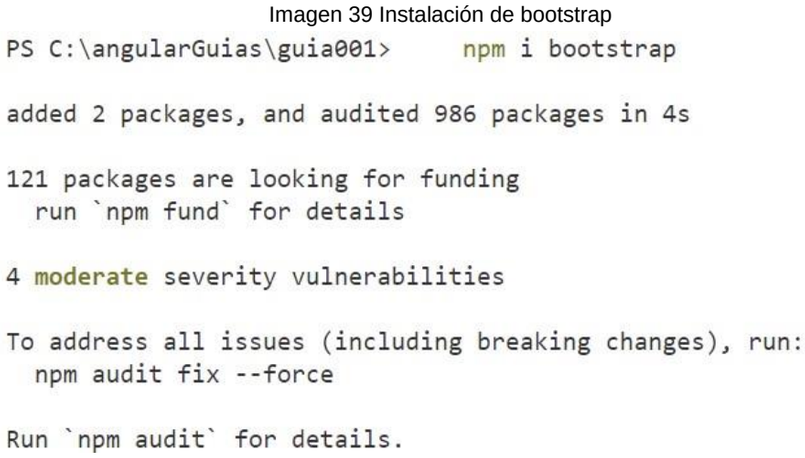
```
PS C:\angularGuias\guia001> ng add @ng-bootstrap/ng-bootstrap
i Using package manager: npm
✓ Found compatible package version: @ng-bootstrap/ng-bootstrap@16.0.0.
✓ Package information loaded.

The package @ng-bootstrap/ng-bootstrap@16.0.0 will be installed and executed.
Would you like to proceed? Yes 1.
✓ Packages successfully installed.
UPDATE package.json (1178 bytes)
✓ Packages installed successfully.
UPDATE angular.json (2835 bytes)
UPDATE src/main.ts (301 bytes)
UPDATE tsconfig.app.json (310 bytes)
UPDATE tsconfig.spec.json (315 bytes)
PS C:\angularGuias\guia001>
```

Fuente: Autor

`npm install bootstrap`

El comando npm install bootstrap se utiliza para instalar la biblioteca Bootstrap en un proyecto de Node.js. Bootstrap es un framework de front-end que proporciona componentes y estilos predefinidos para facilitar el desarrollo web.

Imagen 39 Instalación de bootstrap

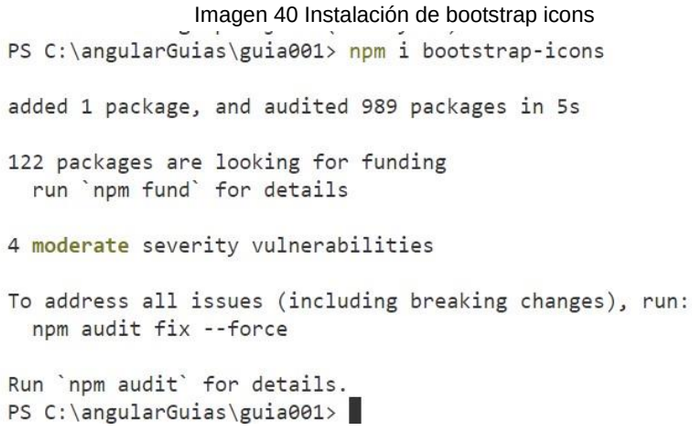


Fuente: Autor

`npm install bootstrap-icons`

El comando npm i bootstrap-icons se utiliza para instalar la biblioteca de iconos Bootstrap Icons en un proyecto. Bootstrap Icons es una biblioteca de iconos SVG de código abierto y oficial para Bootstrap, que cuenta con más de 2,000 iconos disponibles.

Imagen 40 Instalación de bootstrap icons

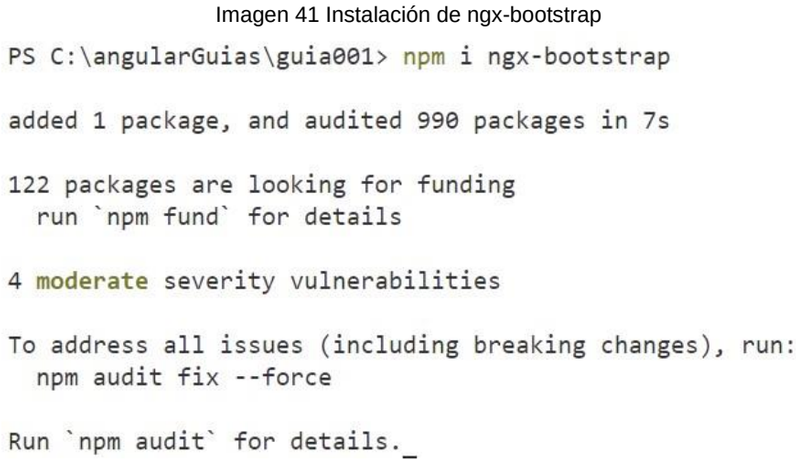


Fuente: Autor

`npm install ngx-bootstrap`

El comando npm i ngx-bootstrap se utiliza para instalar la biblioteca ngx-bootstrap en un proyecto de Angular. ngx-bootstrap es una biblioteca que proporciona componentes de Bootstrap predefinidos y optimizados para su uso en aplicaciones Angular.

Imagen 41 Instalación de ngx-bootstrap

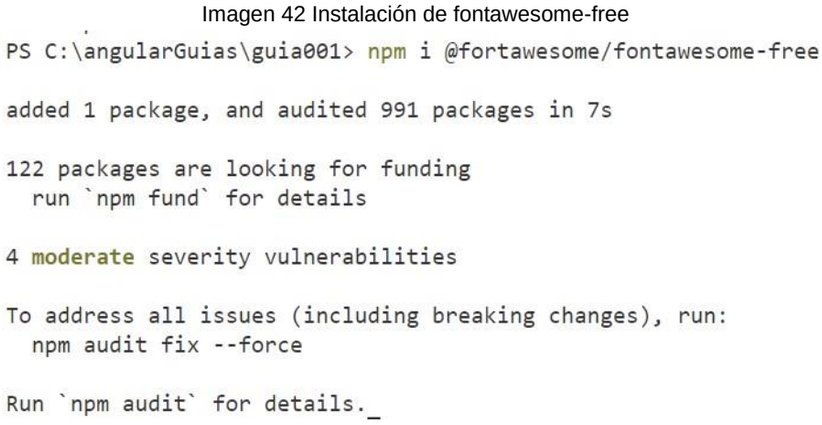


Fuente: Autor

`npm install @fortawesome/fontawesome-free`

El comando npm i @fortawesome/fontawesome-free se utiliza para instalar la biblioteca FontAwesome en un proyecto. FontAwesome es una biblioteca de iconos que proporciona una amplia gama de iconos escalables que se pueden utilizar en aplicaciones web.

Imagen 42 Instalación de fontawesome-free

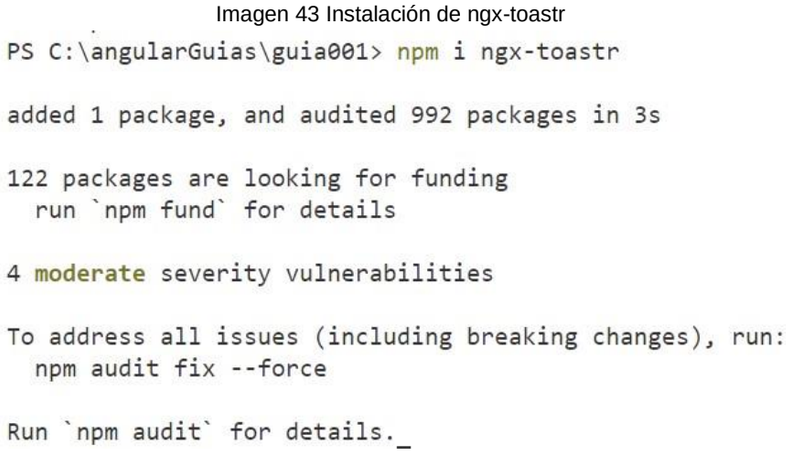


Fuente: Autor

`npm install ngx-toastr`

El comando "***ngx-toastr***" es una librería con componentes que se utiliza para el manejo de las alertas.

Imagen 43 Instalación de ngx-toastr

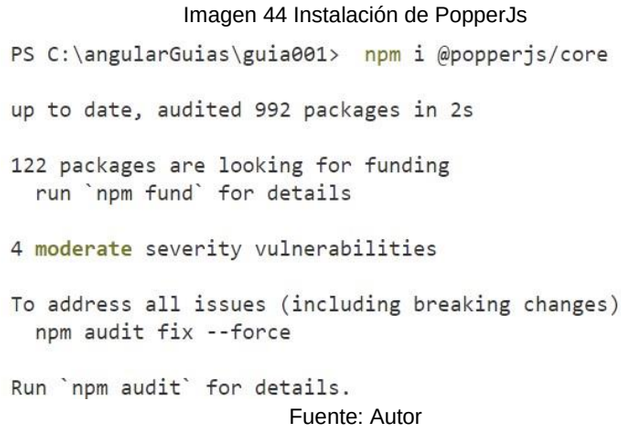


Fuente: Autor

`npm install @popperjs/core`

La "***popperJS***" es una librería que permite crear etiquetas o tooltips sobre un elemento con solo escribir una línea de código. El uso de esta librería potencia el uso de algunas opciones

Imagen 44 Instalación de PopperJs

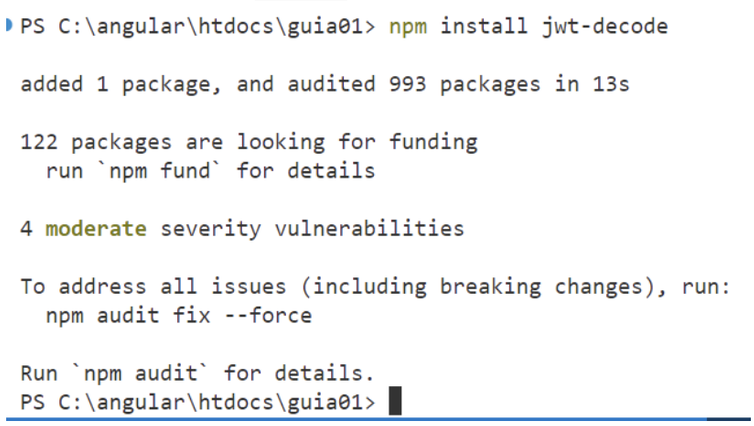


Fuente: Autor

`npm install jwt-decode`

El comando npm i jwt-decode se utiliza para instalar la biblioteca `jwt-decode` en un proyecto. Esta biblioteca permite decodificar tokens JWT (JSON Web Tokens) en aplicaciones JavaScript.

Imagen 45 Instalación de jwt-decode



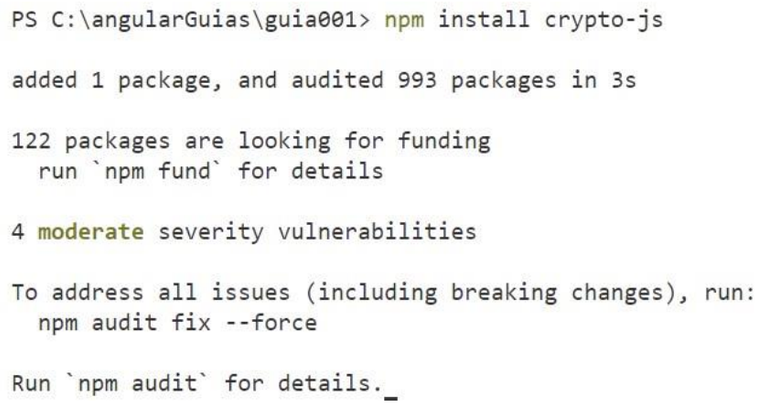
Fuente: Autor

`npm install crypto-js`

`npm install --save-dev @types/crypto-js`

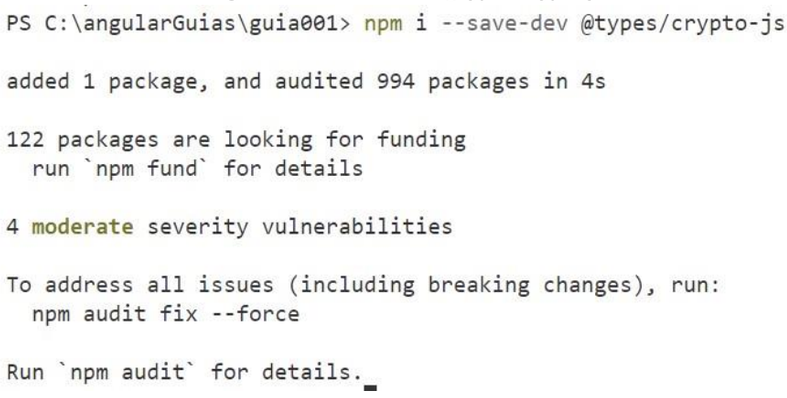
La librería crypto-js es una biblioteca de JavaScript que proporciona implementaciones de algoritmos criptográficos estándar y seguros.

Imagen 46 Instalación de crypto-js



Fuente: Autor

Imagen 47 Instalación de @types/crypto-js



Fuente: Autor

El comodín “***install***” de los comandos anteriores se puede reemplazar por la letra “***i***”:

```
npm i bootstrap
```

De manera automática, en el archivo “***package.json***” aparecerán las referencias de las librerías y framework instalados con los comandos anteriores. La Imagen 48 presenta parte del contenido del “***package.json***”

Imagen 48 Contenido del archivo package.json

package.json M

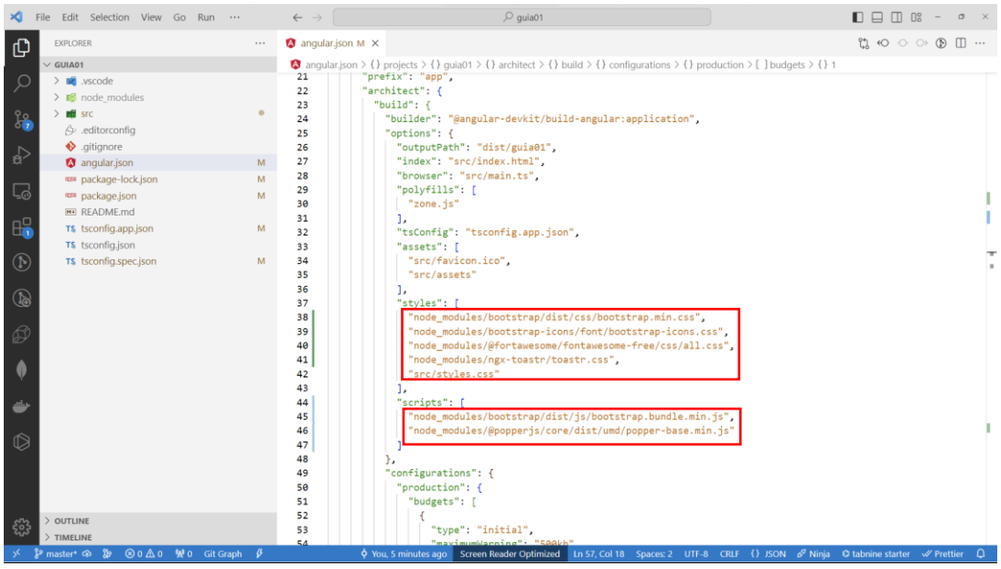
package.json > ...

```
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^17.0.0",
    "@angular/common": "^17.0.0",
    "@angular/compiler": "^17.0.0",
    "@angular/core": "^17.0.0",
    "@angular/forms": "^17.0.0",
    "@angular/platform-browser": "^17.0.0",
    "@angular/platform-browser-dynamic": "^17.0.0",
    "@angular/router": "^17.0.0",
    "@fortawesome/fontawesome-free": "^6.5.1",
    "@ng-bootstrap/ng-bootstrap": "^16.0.0",
    "@popperjs/core": "^2.11.8",
    "bootstrap": "^5.3.2",
    "bootstrap-icons": "^1.11.3",
    "crypto-js": "^4.2.0",
    "jwt-decode": "^4.0.0",
    "ngx-bootstrap": "^12.0.0",
    "ngx-toastr": "^18.0.0",
    "rxjs": "~7.8.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.14.2"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "^17.0.9",
    "@angular/cli": "^17.0.9",
    "@angular/compiler-cli": "^17.0.0",
    "@angular/localize": "^17.0.0",
    "@types/crypto-js": "^4.2.1",
    "@types/jasmine": "~5.1.0",
    "jasmine-core": "~5.1.0",
    "karma": "~6.4.0",
    "karma-chrome-launcher": "~3.2.0",
    "karma-coverage": "~2.2.0",
    "karma-jasmine": "~5.1.0",
    "karma-jasmine-html-reporter": "~2.1.0",
    "typescript": "~5.2.2"
  }
}

```

Fuente: Autor

Un proceso que **NO** es automático es la inclusión de los archivos fuentes o librerias en el proyecto de desarrollo. Se debe editar el archivo “angular.json” e incluir las instrucciones presentadas en la Imagen 49.

Imagen 49 Inclusión de estilos y JavaScript en angular.json



Fuente: Autor

Para efectos prácticos se comparte el código para copiarlo y pegarlo en el archivo “***angular.json***”

```
"build": {
          "builder": "@angular-devkit/build-angular:application",
          "options": {
            "outputPath": "dist/guia001",
            "index": "src/index.html",
            "browser": "src/main.ts",
            "polyfills": ["zone.js"],
            "tsConfig": "tsconfig.app.json",
            "assets": ["src/favicon.ico", "src/assets"],
            "styles": [
              "node_modules/bootstrap/dist/css/bootstrap.min.css",
              "node_modules/bootstrap-icons/font/bootstrap-icons.css",
              "node_modules/@fortawesome/fontawesome-free/css/all.css",
              "node_modules/ngx-toastr/toastr.css",
              "src/styles.css"
            ],
            "scripts": [
              "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js",
             "node_modules/@popperjs/core/dist/umd/popper-base.min.js"
            ]
          },
```

Tabla 1 Inclusión de estilos y JavaScript en el archivo angular.json

5.5. Configuración de puertos

En el mismo archivo “**angular.json**” se puede configurar el puerto por defecto en donde se ejecutará el proyecto al ser visualizado en un navegador. Basta con agregar la propiedad ***“port”*** y asignar un valor válido de puerto. La Imagen 50 presenta la inclusión de la propiedad que permite definir el puerto en el cual se podrá visualizar el proyecto.

Imagen 50 Configuración del puerto en el archivo angular.json

Fuente: Autor

En la Tabla 2 se encuentra el bloque de código que se debe agregar para configurar los puertos en modo producción y desarrollo.

```
"serve": {
          "builder": "@angular-devkit/build-angular:dev-server",
          "configurations": {
            "production": {
              "buildTarget": "guia01:build:production",
              "port": 8090
            },
            "development": {
              "buildTarget": guia01:build:development",
              "port": 8091
            }
          },
          "defaultConfiguration": "development"
        },
```

Tabla 2 Configuración de puertos en angular.json

5.6. Inclusión de paquetes en el módulo principal

Para utilizar las funcionalidades de algunos paquetes instalados se necesitará importar los paquetes en el módulo principal "***src\app\app.module.ts***"

Imagen 51 Inclusión de paquetes en em modulo app.module.ts

Fuente: Autor

En el módulo principal "***src\app\app.module.ts***" se incluirán los módulos y componentes que harán parte de la funcionalidad para de esta guía, Tabla 3 presenta la inclusión de los módulos instalados.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

import { NgbModule } from '@ng-bootstrap/ng-bootstrap';
import {FormsModule} from '@angular/forms';
import {ModalModule} from 'ngx-bootstrap/modal';
import {ToastrModule} from 'ngx-toastr';
import {BrowserAnimationsModule} from '@angular/platform-browser/animations';

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,

    NgbModule,
    FormsModule,
    ModalModule.forRoot(),
    ToastrModule.forRoot(),
    BrowserAnimationsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Tabla 3 Inclusión de paquetes en el módulo principal app.module.ts

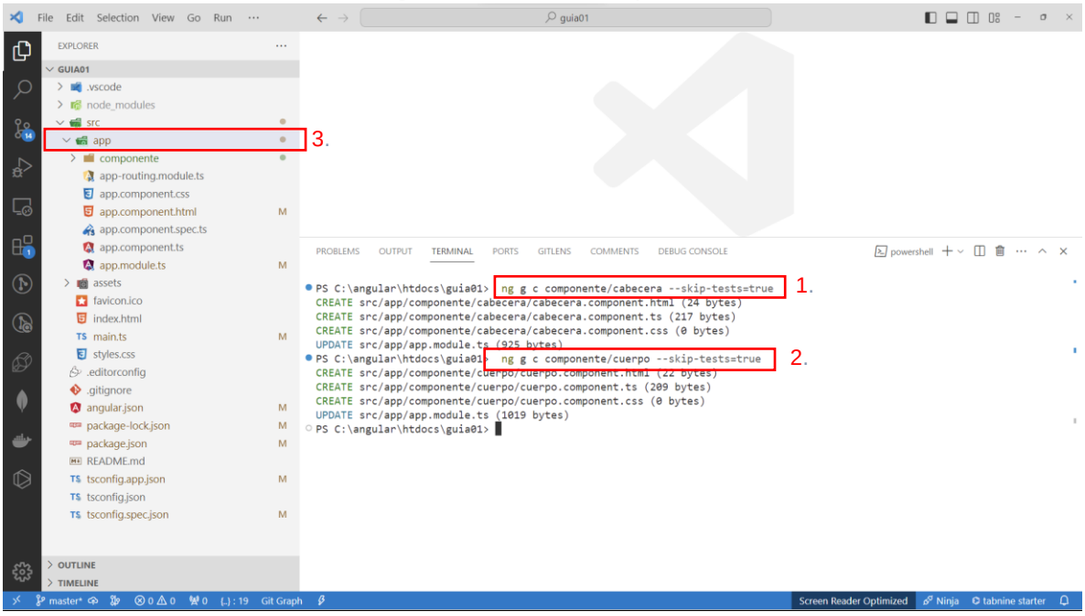
5.7. Creación de componentes

Es recomendable empezar a trabajar los proyectos con el editor Visual Studio Code (VSC), de esta manera se podrá utilizar la terminar para ejecutar los comandos. Para este taller, se van a crear dos componentes, para esto ejecutamos las siguientes instrucciones en la terminal de VSC:

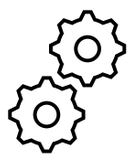
`ng g c componente/cabecera --skip-tests=true`
 	
`ng g c componente/cuerpo --skip-tests=true`
	ng → ng-model
	g → generate
	c → component
	***--skip-test → para no generar un archivo de pruebas unitarias**.*

La Imagen 52 presenta la carpeta de proyecto cargada en VSC y la ejecución de las instrucciones en la terminal.

Imagen 52 Creación de componentes

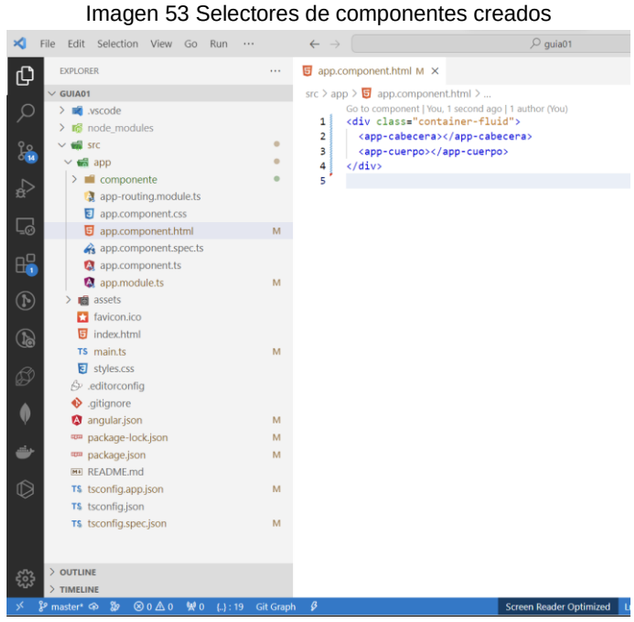


Fuente: Autor

	Al realizar estos comandos en nuestra terminal de manera **automática** en nuestro archivo "***src\app\app.module.ts***" se realizará la declaración e importación de los componentes cabecea y cuerpo.

En el archivo "***src\app\app.component.html***" aparece el código HTML por defecto al crear el componente, todo este código se debe cambiar por el código presentado en la Tabla 4.

Imagen 53 Selectores de componentes creados



Fuente: Autor

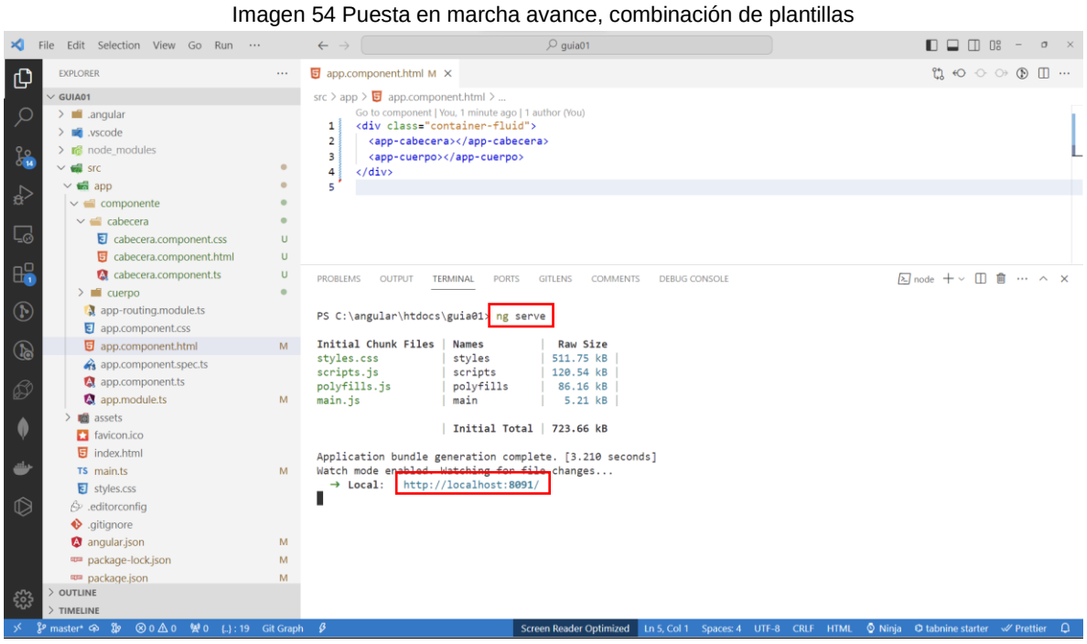
```
<!-- El fragmento de código crea un contenedor con la clase "container-fluid" y
anida dos componentes
personalizados `<app-cabecera></app-cabecera>` y `<app-cuerpo></app-cuerpo>`
dentro del mismo. esta estructura
se usa comúnmente en marcos de desarrollo web como Angular para organizar y
mostrar contenido en un
Página web. -->
<div class="container-fluid">
  <app-cabecera></app-cabecera>
  <app-cuerpo></app-cuerpo>
</div>
```

Tabla 4 Plantilla de componentes creados

5.8. Prueba parcial de la guia01

Una verificación de cada uno de esos archivos permitirá encontrar un texto sencillo como “ componenteX works! ”. Todo el contenido que se coloque en estos archivos se presentará en el navegador de acuerdo con el orden de los selectores que se haya definido en la plantilla principal de la aplicación, es decir en el archivo **“src\app\app.component.html”**.

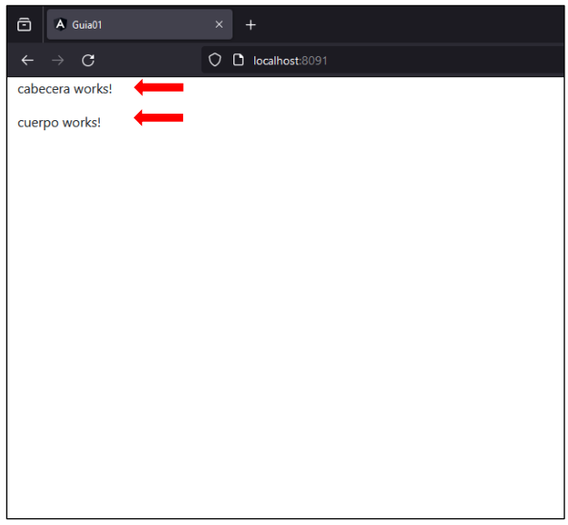
Imagen 54 Puesta en marcha avance, combinación de plantillas



Fuente: Autor

Para verificar el avance del taller, se sugiere al lector que ejecute el proyecto. Desde la terminal de Visual Estudio Code se debe ejecutar el comando “ng serve”. La Imagen 55 presenta el resultado del avance del taller hasta este punto.

Imagen 55 Taller en ejecución, combinando componentes



Fuente: Autor

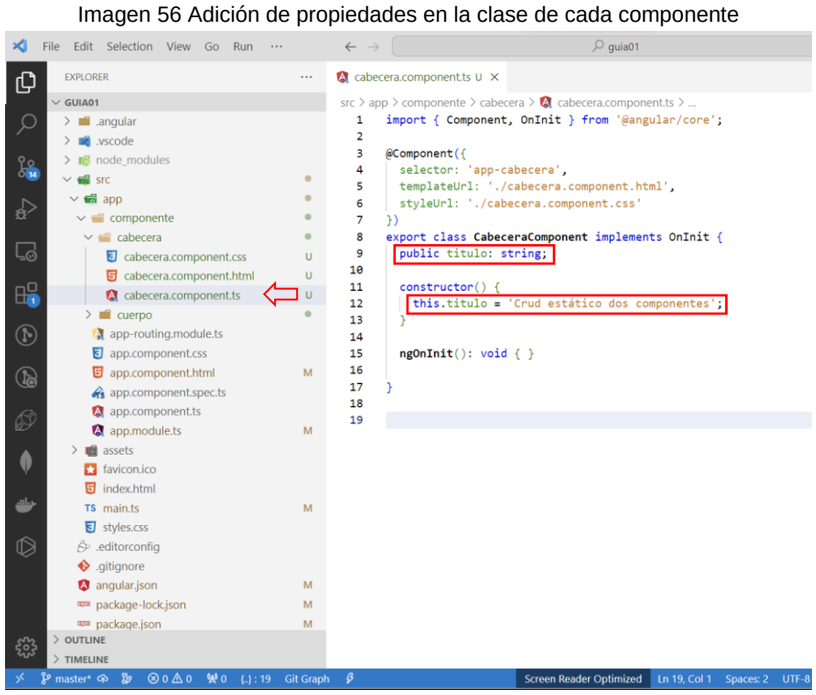
5.8.1. Interpolación en componentes

A continuación, se agregará una propiedad o atributo a la clase principal de cada componente, de esta manera, se puede incluir código en las plantillas de cada componente por medio de la “***Interpolación***”. Para realizar este cambio, se tomará como referencia la clase principal del primer componente, es decir “***src\app\componente\cabecera\cabecera.component.ts***”.

5.8.1.1 Componente cabecera

En la clase del componente “Cabecera” se adicionará una propiedad con su respectiva inicialización en el constructor. La Imagen 56 presenta el contenido de la clase y los ajustes sugeridos.

Imagen 56 Adición de propiedades en la clase de cada componente



Fuente: Autor

A manera de ejemplo se presenta el código "**app/componente/cabecera/cabecera.component.ts**" de la clase cabecera en la Tabla 5.

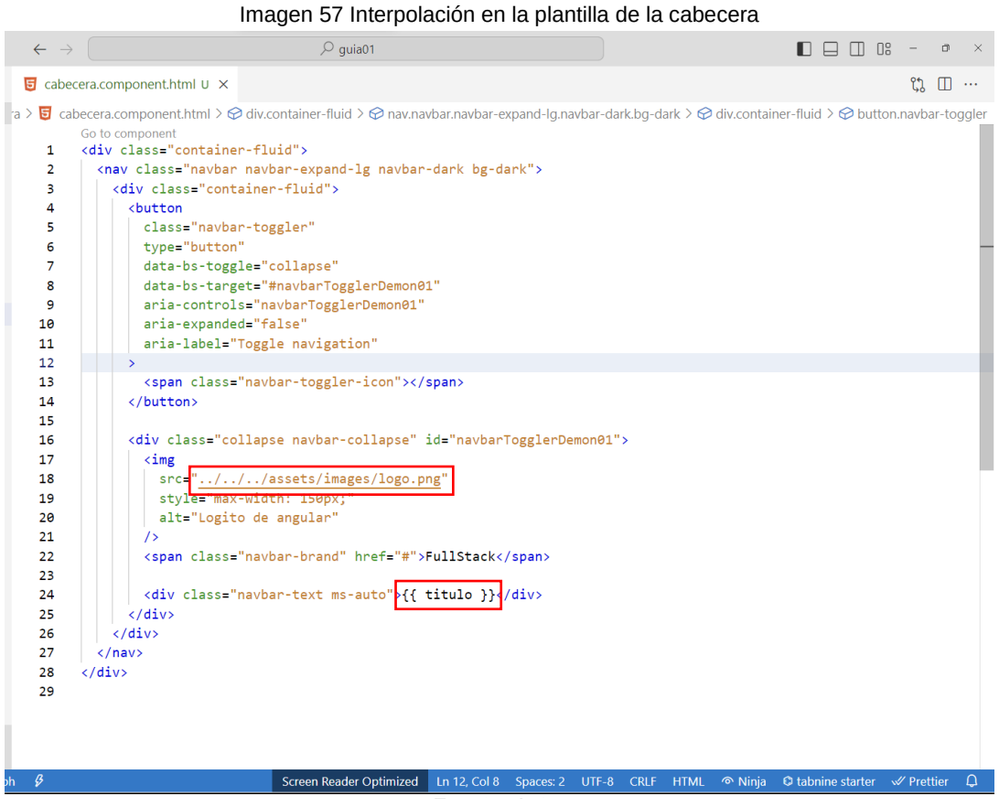
```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-cabecera',
  templateUrl: './cabecera.component.html',
  styleUrl: './cabecera.component.css'
})
export class CabeceraComponent implements OnInit {

  public titulo: string; // Declara una propiedad pública de tipo string

  constructor() {
    this.titulo = 'Crud estático dos componentes'; //Valor inicial del
titulo
  }

  ngOnInit(): void { }
}
```

Tabla 5 Propiedades para la cabecera

La interpolación en Angular es la inyección de una cadena o valor en una plantilla, se conoce también como "***string interpolation***". Para realizar la interpolación solo basta con usar llaves dobles de apertura y de cierre "***{{ }}***" en la plantilla del componente deseado.

Para la plantilla de la componente cabecera se utilizará un maquetado básico con base en la librería Bootstrap. La Imagen 57 presenta el maquetado sugerido para este componente.

Imagen 57 Interpolación en la plantilla de la cabecera



Fuente: Autor

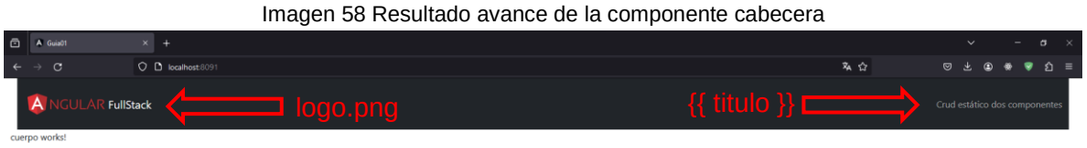
A manera de ejemplo se presenta el código "**app/componente/cabecera/cabecera.component.html**" de la clase cabecera en la Tabla 6.

```
<div class="container-fluid">
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
      <div class="collapse navbar-collapse" id="navbarTogglerDemon01">
        <img
          src="../../../assets/images/logo.png"
          style="max-width: 150px"
          alt="Logito de angular"
        />
        <span class="navbar-brand" href="#">FullStack</span>

        <div class="navbar-text">{{ titulo }}</div>
      </div>

      <button
        class="navbar-toggler ms-auto"
        type="button"
        data-bs-toggle="collapse"
        data-bs-target="#navbarTogglerDemon01"
        aria-controls="navbarTogglerDemon01"
        aria-expanded="false"
        aria-label="Toggle navigation"
      >
        <span class="navbar-toggler-icon"></span>
      </button>
    </div>
  </nav>
</div>
```

Tabla 6 Interpolación y estilos de la cabecera

Resultado después de agregar los estilos de bootstrap y el llamado del título como interpolación desde la plantilla HTML.

Imagen 58 Resultado avance de la componente cabecera



Fuente: Autor

	**Dobles llaves {{ x }}: El valor entre ellas es llamado expresión. Se pueden crean expresiones simples o complejas, en este ejercicio se utilizó el valor de una propiedad como ejemplo, sin embargo, pueden ser valores calculados o métodos**

5.9. Modelos

Un modelo es la representación de un objeto por medio de atributos o propiedades. Para crear un modelo se puede utilizar los comandos de "***Angular CLI***" o se pueden crear de maneral manual. A continuación, se presenta el comando a ejecutar desde la terminal de Visual Estudio Code.

¡¡¡Bonus!!! En siguiente blog nos explican más a fondo que es una clase en typescript para angular "***Clases en Typescript***".

```
ng g class modelos/DispositivoEntrada --type=model --skip-tests=true
```

ng → ng-model
g → generate
class → archivo de tipo clase
--type=model → extensión de tipo modelo al archivo.
--skip-test → para no generar un archivo de pruebas unitarias.

En el archivo "***src\app\modelos\dispositivoEntrada.model.ts***" se definen las propiedades que posteriormente serán inicializadas en el constructor. Imagen 59 presenta la clase "***DispositivoEntrada***" con sus propiedades y la inicialización de variables en el constructor.

Imagen 59 Modelo Dispositivo Entrada

Fuente: Autor

Para efectos prácticos se comparte el código para copiarlo y pegarlo en el archivo **“src\app\modelos\dispositivoEntrada.ts”**

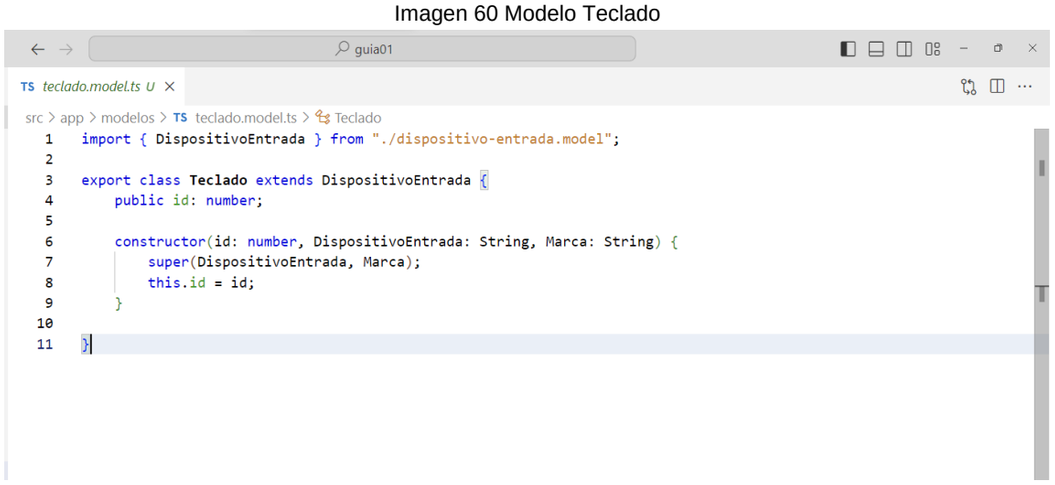
```
/* La clase DispositivoEntrada representa un dispositivo de entrada con
propiedades para el tipo de dispositivo y su marca. */

export class DispositivoEntrada {
    public dispositivoEntrada: String;
    public marca: String;

    constructor(DispositivoEntrada: String, Marca: String) {
        this.dispositivoEntrada = DispositivoEntrada;
        this.marca = Marca;
    }
}
```

Tabla 7 Tabla Dispositivo-Entrada

La Imagen 60 presenta el modelo llamado “***Teclado***” que se utilizará en este taller, el cual estará almacenado en el archivo: “***src\app\modelos\teclado.model.ts***”.

```
ng g class modelos/Teclado --type=model --skip-tests=true
```

Imagen 60 Modelo Teclado



Fuente: Autor

Para efectos prácticos se comparte el código para copiarlo y pegarlo en el archivo “***src\app\modelos\teclado.model.ts***

```
/* La clase `Teclado` extiende la clase `DispositivoEntrada` en
TypeScript y agrega una propiedad `id`. */

import { DispositivoEntrada } from './dispositivo-entrada.model';
export class Teclado extends DispositivoEntrada {
  public id: number;
  constructor(id: number, DispositivoEntrada: String, Marca: String) {
    /*Esto permite que la clase `Teclado` herede propiedades y
comportamiento de la clase `DispositivoEntrada`.*/
    super(DispositivoEntrada, Marca);
    this.id = id;
  }
}
```

Tabla 8 Tabla Teclado

La Imagen 61 presenta el modelo llamado "***Raton***" que se utilizará en este taller, el cual estará almacenado en el archivo: "***src\app\modelos\ raton.model.ts***".

```
ng g class modelos/Raton --type=model --skip-tests=true
```

Imagen 61 Modelo Ratón

Fuente: Autor

Para efectos prácticos se comparte el código para copiarlo y pegarlo en el archivo "***src\app\modelos\ ratón.model.ts***"

```
/* La clase `Raton` extiende `DispositivoEntrada` y agrega una propiedad `id`
para representar un mouse
dispositivo. */

import { DispositivoEntrada } from "./dispositivo-entrada.model";

export class Raton extends DispositivoEntrada {
    public id: number;
    constructor(id: number,DispositivoEntrada: String, Marca: String) {
        super(DispositivoEntrada, Marca);
        this.id = id;
    }
}
```

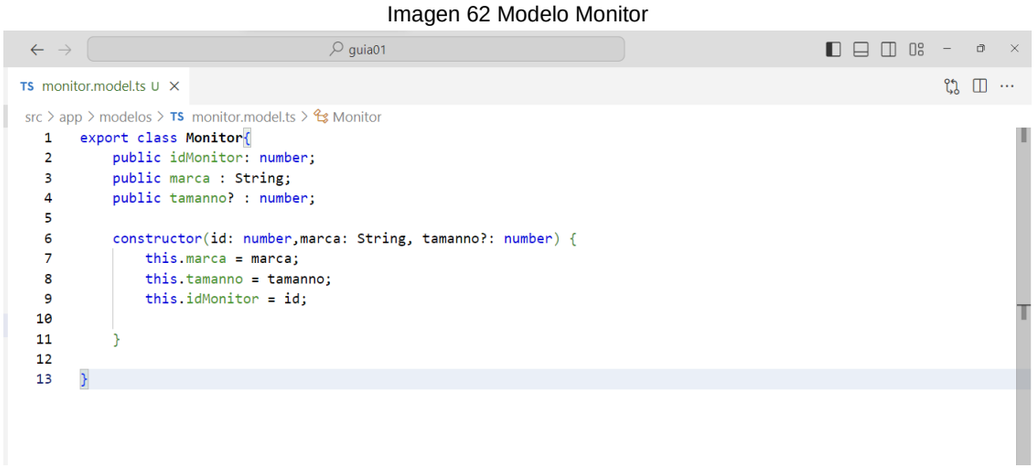
Tabla 9 Modelo Ratón

¡¡¡Bonus!!! En el siguiente blog se explican que son las herencias "extends" para Typescript mediante imágenes, ejemplos y las ventajas de hacer una herencia "***Herencias de clases en TypeScript***".

La Imagen 62 presenta el modelo llamado "***Monitor***" que se utilizará en este taller, el cual estará almacenado en el archivo: "***src\app\modelos\ monitor.model.ts***".

```
ng g class modelos/Monitor --type=model --skip-tests=true
```

Imagen 62 Modelo Monitor



Fuente: Autor

Para efectos prácticos se comparte el código para copiarlo y pegarlo en el archivo "***src\app\modelos\ monitor.model.ts***"

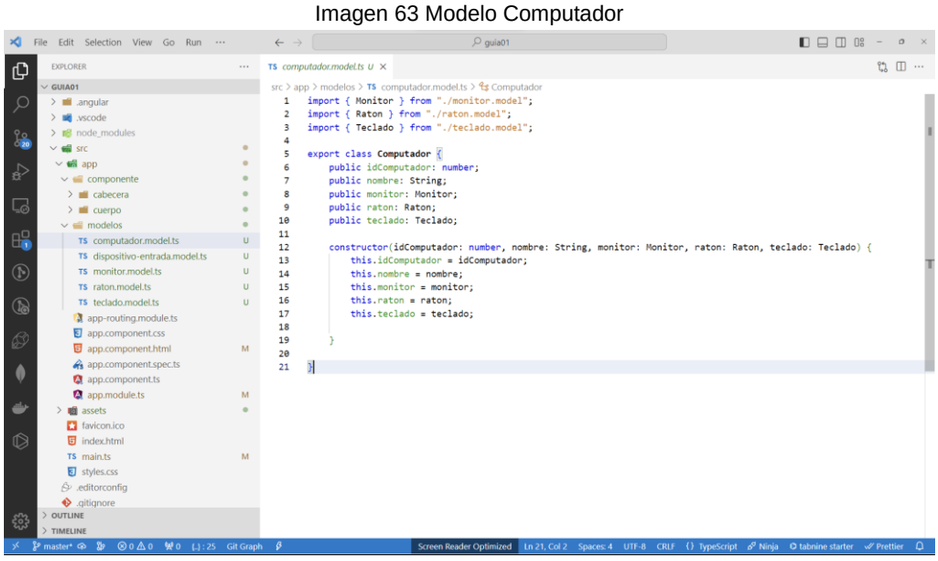
```
/* La clase `Monitor` en TypeScript representa un monitor con propiedades para
ID, marca y opciones tamaño. */
export class Monitor{
    public idMonitor: number;
    public marca : String;
    public tamanno? : number;

    constructor(id: number,marca: String, tamanno?: number) {
        this.marca = marca;
        this.tamanno = tamanno;
        this.idMonitor = id;
    }
}
```

Tabla 10 Modelo Monitor

La Imagen 63 presenta el modelo llamado "***Computador***" que se utilizará en este taller, el cual estará almacenado en el archivo: "***src\app\modelos\computador.model.ts***".

```
ng g class modelos/Computador --type=model --skip-tests=true
```

Imagen 63 Modelo Computador



Fuente: Autor

Para efectos prácticos se comparte el código para copiarlo y pegarlo en el archivo "***src\app\modelos\ computador.model.ts***"

```
/* La clase `Computador` representa una entidad Computador con propiedades como
id, nombre, monitor, Ratón y teclado. */

import { Monitor } from "./monitor.model";
import { Raton } from "./raton.model";
import { Teclado } from "./teclado.model";
export class Computador {
    public idComputador: number;
    public nombre: String;
    public monitor: Monitor;
    public raton: Raton;
    public teclado: Teclado;

    constructor(idComputador: number, nombre: String, monitor: Monitor,
raton: Raton, teclado: Teclado) {
        this.idComputador = idComputador;
        this.nombre = nombre;
        this.monitor = monitor;
        this.raton = raton;
        this.teclado = teclado;
    }
}
```

Tabla 11 Modelo Computador

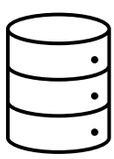
5.10. Mocks

El mock contiene la información para realizar pruebas en el Frontend. La Imagen 64 presenta la ejecución del comando en la terminal, la estructura de datos propuesta y el contenido de archivo mock creado con el siguiente comando:

```
ng g enum mocks/Computadores --type=mock;
ng g enum mocks/Monitores --type=mock;
ng g enum mocks/Ratones --type=mock;
ng g enum mocks/Teclados --type=mock;
```

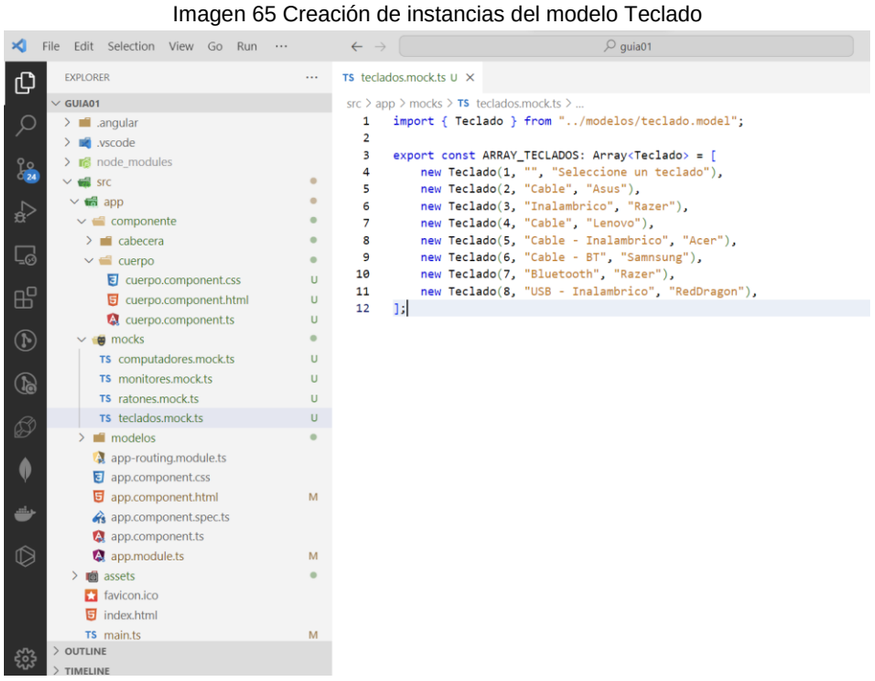
Imagen 64 Carpeta Mocks y archivos

Fuente: Autor

	El mock creado con la “Angular CLI” genera una enum, sin embargo, para los datos de prueba es suficiente con generar un **arreglo** de datos que permita su utilización CRUD en los componentes.

La Imagen 65 presenta el contenido del archivo “**src\app\mocks\teclado.mock.ts**”.

Imagen 65 Creación de instancias del modelo Teclado



Fuente: Autor

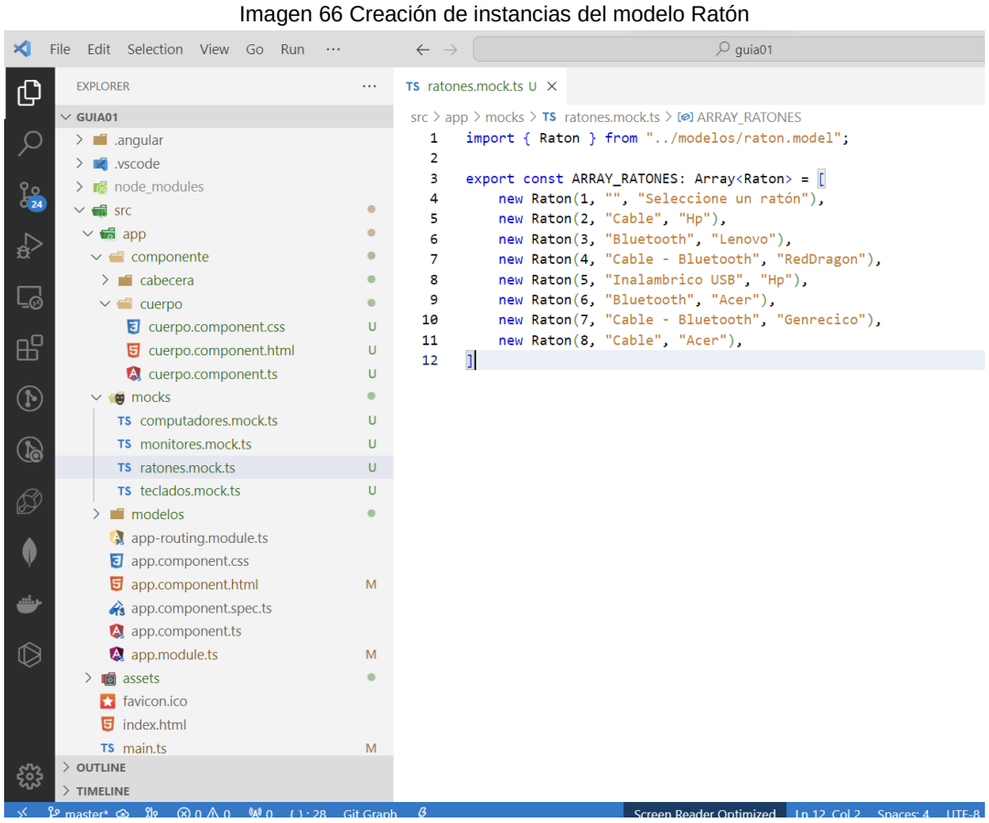
Para efectos prácticos, en la Tabla 12 se comparte el código para copiarlo y pegarlo en el archivo “***src\app\mocks\teclados.mock.ts***”

```
/* Se define una matriz llamada `ARRAY_TECLADOS` que contiene instancias de la
clase `Teclado`. */

import { Teclado } from "../modelos/teclado.model";
export const ARRAY_TECLADOS: Array<Teclado> = [
    new Teclado(1, "", "Seleccione un teclado"),
    new Teclado(2, "Cable", "Asus"),
    new Teclado(3, "Inalambrico", "Razer"),
    new Teclado(4, "Cable", "Lenovo"),
    new Teclado(5, "Cable - Inalambrico", "Acer"),
    new Teclado(6, "Cable - BT", "Samnsung"),
    new Teclado(7, "Bluetooth", "Razer"),
    new Teclado(8, "USB - Inalambrico", "RedDragon"),
];
```

Tabla 12 Mock modelo teclado

La Imagen 66 presenta el contenido del archivo "**src\app\mocks\raton.mock.ts**".

Imagen 66 Creación de instancias del modelo Ratón



Fuente: Autor

Para efectos prácticos, en la Tabla 13 se comparte el código para copiarlo y pegarlo en el archivo "***src\app\mocks\ratones.mock.ts***"

```
/* Este fragmento de código TypeScript define una matriz de objetos `Raton`. */

import { Raton } from "../modelos/raton.model";
export const ARRAY_RATONES: Array<Raton> = [
    new Raton(1, "", "Seleccione un ratón"),
    new Raton(2, "Cable", "Hewlett-Packard"),
    new Raton(3, "Bluetooth", "Lenovo"),
    new Raton(4, "Cable - Bluetooth", "RedDragon"),
    new Raton(5, "Inalambrico USB", "Hewlett-Packard"),
    new Raton(6, "Bluetooth", "Acer"),
    new Raton(7, "Cable - Bluetooth", "Genrecico"),
    new Raton(8, "Cable", "Acer"),
]
```

Tabla 13 Mock modelo ratón

La Imagen 67 presenta el contenido del archivo “**src\app\mocks\monitor.mock.ts**”.

Imagen 67 Creación de instancias del modelo Monitor

Fuente: Autor

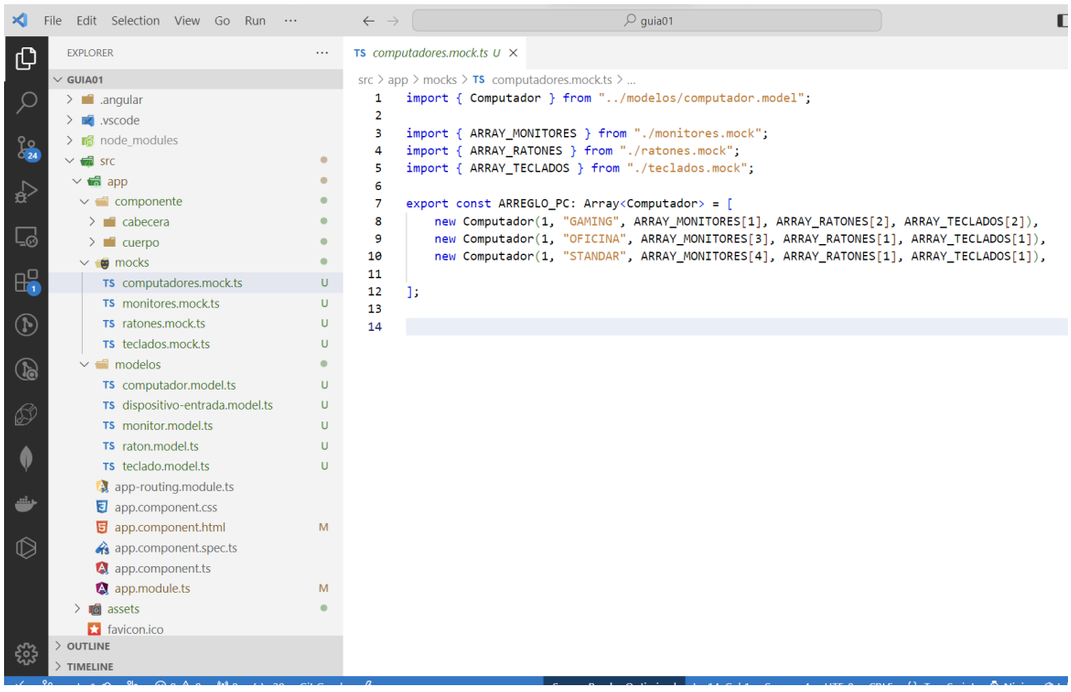
Para efectos prácticos, en la Tabla 14 se comparte el código para copiarlo y pegarlo en el archivo “***src\app\mocks\monitores.mock.ts***”

```
/* La matriz representa una Listado de monitores con sus respectivos detalles
como marca y tamaño. */

import { Monitor } from "../modelos/monitor.model";
export const ARRAY_MONITORES: Array<Monitor> = [
    new Monitor(1,"Seleccione un monitor"),
    new Monitor(2,"ASUS", 34),
    new Monitor(3,"LENOVO", 32),
    new Monitor(4,"AORUS", 24),
    new Monitor(5,"SAMNSUNG", 42)
]
```

Tabla 14 Mock modelo monitor

La Imagen 68 Creación de instancias del modelo Computador presentando el contenido del archivo “**src\app\mocks\computador.mock.ts**”.

Imagen 68 Creación de instancias del modelo Computador



Fuente: Autor

Para efectos prácticos, en la Tabla 15 se comparte el código para copiarlo y pegarlo en el archivo “***src\app\mocks\computador.mock.ts***”

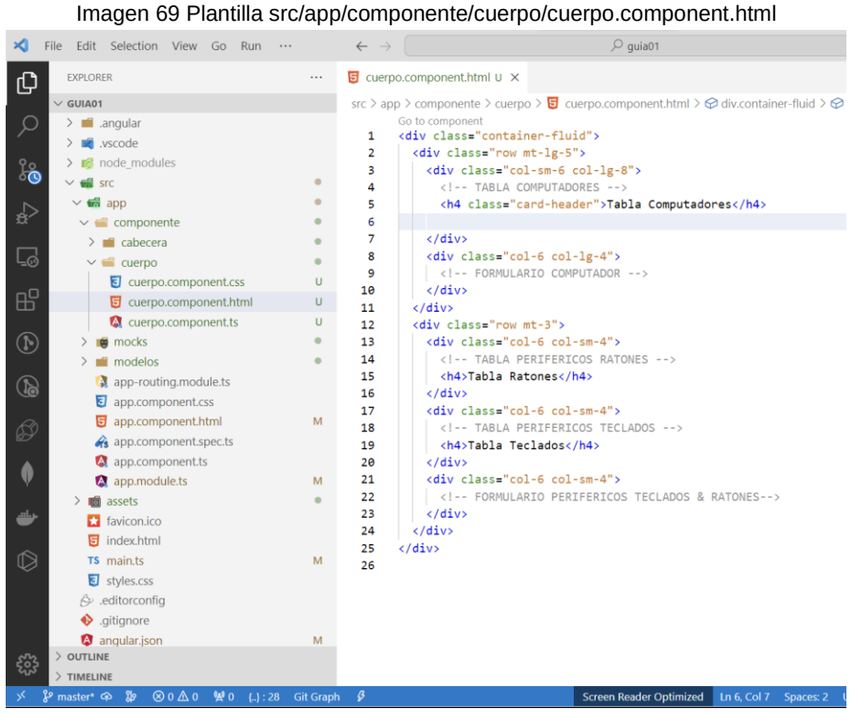
```
/* Este código TypeScript está creando una matriz de objetos `Computador` llamado `ARREGLO_PC`.
*/
import { Computador } from "../modelos/computador.model";

import { ARRAY_MONITORES } from "./monitores.mock";
import { ARRAY_RATONES } from "./ratones.mock";
import { ARRAY_TECLADOS } from "./teclados.mock";

export const ARREGLO_PC: Array<Computador> = [
new Computador(1, "Gaming", ARRAY_MONITORES[1], ARRAY_RATONES[2], ARRAY_TECLADOS[2]),
new Computador(2, "Oficina", ARRAY_MONITORES[3], ARRAY_RATONES[1], ARRAY_TECLADOS[7]),
new Computador(3, "Empresarial", ARRAY_MONITORES[4], ARRAY_RATONES[6], ARRAY_TECLADOS[5]),
new Computador(4, "Básico", ARRAY_MONITORES[2], ARRAY_RATONES[5], ARRAY_TECLADOS[4]),
];
```

Tabla 15 Mock modelo computador

5.11. Componente Cuerpo.html

Para la plantilla del componente cuerpo se utilizará la librería bootstrap. Todo el contenido HTML estará en un bloque “div” utilizando la clase “container-fluid”, en su interior se definirá una serie de bloques que define dos filas que posteriormente tendrá dos y tres columnas para organizar la información. Dado que el contenido de cada columna es un poco más extenso se presentará más adelante. La Imagen 69 presenta el maquetado de la plantilla del componente cuerpo.

Imagen 69 Plantilla src/app/componente/cuerpo/cuerpo.component.html



Fuente: Autor

En la Tabla 16 se presenta a manera de ejemplo el maquetado de la plantilla del componente "**src/app/componente/cuerpo/cuerpo.component.html**"

```
<div class="container-fluid">
  <div class="row mt-lg-5">
    <div class="col-lg-8">
      <!-- TABLA COMPUTADORES -->
      <h4 class="card-header">Tabla Computadores</h4>
    </div>
    <div class="col-lg-4">
      <!-- FORMULARIO COMPUTADOR -->
    </div>
  </div>
  <div class="row mt-3">
    <div class="col-sm-6 col-md-4">
      <!-- TABLA PERIFERICOS RATONES -->
      <h4>Tabla Ratones</h4>
    </div>
    <div class="col-sm-6 col-md-4">
      <!-- TABLA PERIFERICOS TECLADOS -->
      <h4>Tabla Teclados</h4>
    </div>
    <div class="col-lg-4">
      <!-- FORMULARIO PERIFERICOS TECLADOS & RATONES-->
    </div>
  </div>
</div>
```

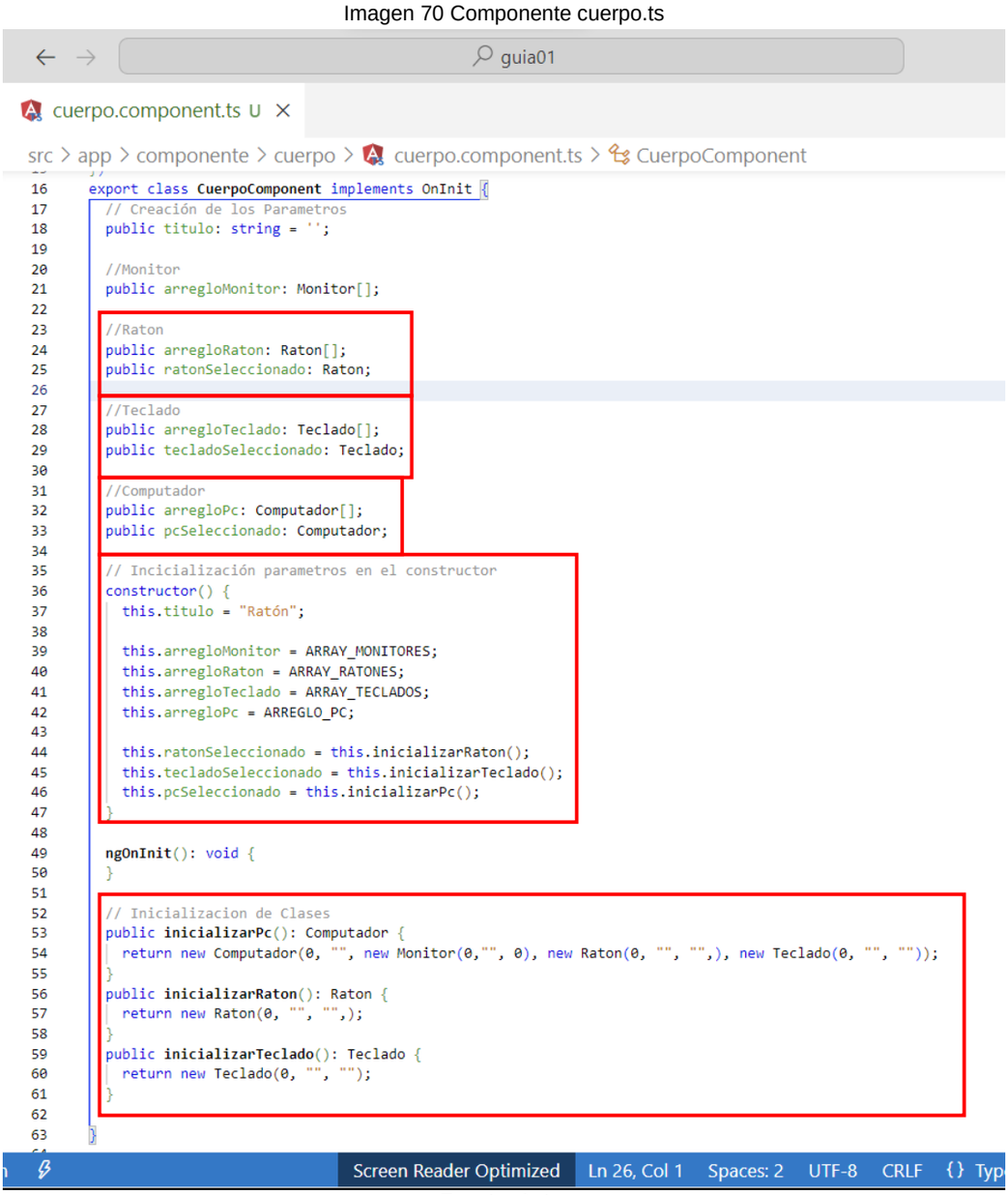
Tabla 16 Maquetado HTML plantilla cuerpo.component.html

5.12. Componente Cuerpo.ts

La clase del componente cuerpo requiere propiedades y métodos que permitan incluir las funcionalidades para el **CRUD** a implementar. La Imagen 70 presenta la inclusión de los modelos y los mocks a utilizar en este taller, así como la definición de las propiedades o atributos de la clase y su inicialización en el constructor.

	Dos métodos claves cuando se trabaja con Angular en el desarrollo de un CRUD son: 1) Método para inicializar nuevamente el objeto que sirve para manipular información. 2) Método de seleccionar el objeto sobre el cual se desean realizar las operaciones. 3) Qué es y cómo funciona un constructor mas a fondo en el siguiente blog "***Creando clases con constructores en Typescript***", aquí explican mas a detalle la composición y funcionamiento de este método.

Imagen 70 Componente cuerpo.ts



Fuente: Autor

En la Tabla 17 se presenta a manera de ejemplo el contenido de la clase del componente cuerpo "**src\app\componente\cuerpo\cuerpo.component.ts**".

```
import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { Monitor } from '../../modelos/monitor.model';
import { Raton } from '../../modelos/raton.model';
import { Teclado } from '../../modelos/teclado.model';
import { Computador } from '../../modelos/computador.model';
import { ARRAY_MONITORES } from '../../mocks/monitores.mock';
import { ARRAY_RATONES } from '../../mocks/ratones.mock';
import { ARRAY_TECLADOS } from '../../mocks/teclados.mock';
import { ARREGLO_PC } from '../../mocks/computadores.mock';
@Component({
  selector: 'app-cuerpo',
  templateUrl: './cuerpo.component.html',
  styleUrl: './cuerpo.component.css'
})
export class CuerpoComponent implements OnInit {
  // Creación de los Parámetros
  public titulo: string = '';

  //Monitor
  public arregloMonitor: Monitor[];
  //Raton
  public arregloRaton: Raton[];
  public ratonSeleccionado: Raton;
  //Teclado
  public arregloTeclado: Teclado[];
  public tecladoSeleccionado: Teclado;
  //Computador
  public arregloPc: Computador[];
  public pcSeleccionado: Computador;

  //Propiedad objeto que recibe Objetos de tipo Raton o Teclado
  public objetoSeleccionado: Raton | Teclado;

  // Propiedad bandera para un combobox
  public esRatonTeclado: boolean;

  // Inicialización parámetros en el constructor
  constructor() {
    this.titulo = "Ratón";

    this.arregloMonitor = ARRAY_MONITORES;
    this.arregloRaton = ARRAY_RATONES;
    this.arregloTeclado = ARRAY_TECLADOS;
    this.arregloPc = ARREGLO_PC;
    this.ratonSeleccionado = this.inicializarRaton();
    this.tecladoSeleccionado = this.inicializarTeclado();
    this.pcSeleccionado = this.inicializarPc();

    this.objetoSeleccionado = this.inicializarRaton();
    this.esRatonTeclado = true;
  }
//*********Inicio de la lógica CRUD para un computador*********
  ngOnInit(): void {
  }
  public seleccionarPc(pc: Computador): void {
  }

  public eliminarPc(del: Computador): void {
  }
//Lógica para el formulario de un Computador
```

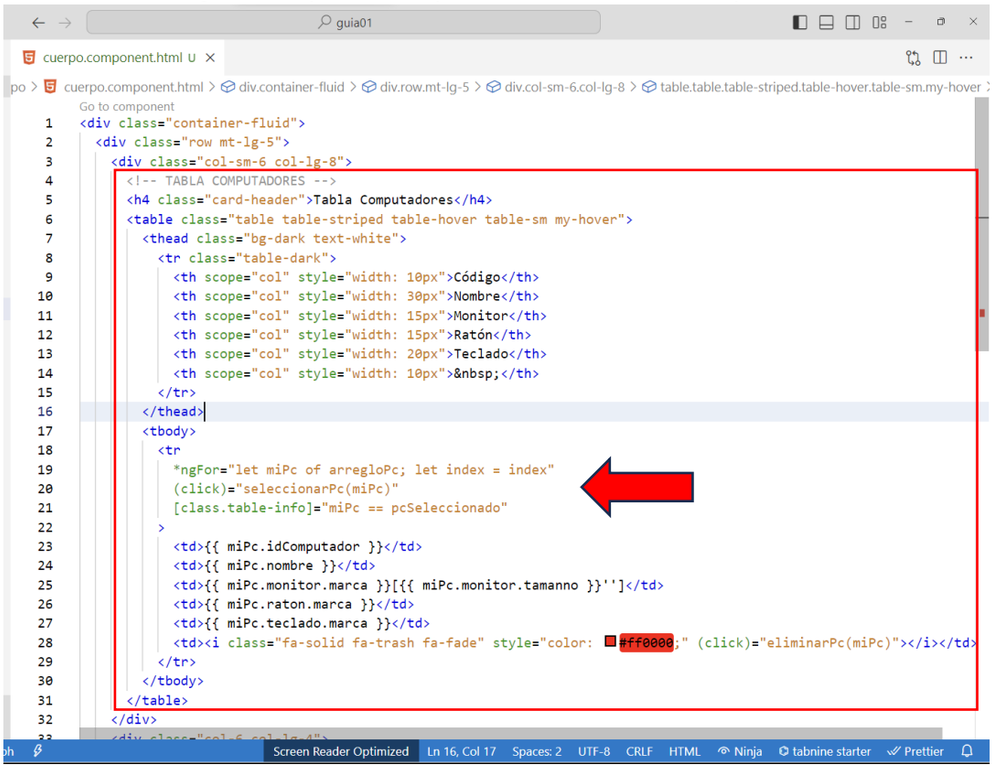
```
//**********Fin de la lógica CRUD para un computador*********

//**********Inicio de la lógica CRUD para los Ratones y Teclado*********
  public seleccionarRaton(ra: Raton): void {
  }

  public seleccionarTeclado(te: Teclado): void {
  }
//**********Fin de la logia CRUD para los periféricos*********

// Inicialización de Clases
  public inicializarPc(): Computador {
    return new Computador(0, "", new Monitor(0,"", 0), new Raton(0, "",
"",), new Teclado(0, "", ""));
  }
  public inicializarRaton(): Raton {
    return new Raton(0, "", "",);
  }
  public inicializarTeclado(): Teclado {
    return new Teclado(0, "", "");
  }
}
```

Tabla 17 Importaciones, propiedades e instancias clase Computador y Periféricos

5.13. Componente Cuerpo.html

La Imagen 71 presenta la inclusión de la tabla que mostrará la información de los **Computadores** en "**cuerpo.componente.html**".

Imagen 71 Maquetado html para la información de los computadores



Fuente: Autor

En la Tabla 17 se presenta el bloque HTML que muestra la tabla de los computadores, esta debe ser añadida **después** del ***comentario*** **"<!----TABLA COMPUTADORES--->"**

```
      <!-- TABLA COMPUTADORES -->
      <h4 class="card-header">Tabla Computadores</h4>
      <table class="table table-striped table-hover table-sm my-hover">
        <thead class="bg-dark text-white">
          <tr class="table-dark">
            <th scope="col" style="width: 10px">Código</th>
            <th scope="col" style="width: 30px">Nombre</th>
            <th scope="col" style="width: 15px">Monitor</th>
            <th scope="col" style="width: 15px">Ratón</th>
            <th scope="col" style="width: 20px">Teclado</th>
            <th scope="col" style="width: 10px"> </th>
          </tr>
        </thead>
        <tbody>
          <tr
            *ngFor="let miPc of arregloPc"
            (click)="seleccionarPc(miPc)"
            [class.table-info]="miPc == pcSeleccionado"
          >
            <td>{{ miPc.idComputador }}</td>
            <td>{{ miPc.nombre }}</td>
            <td>{{ miPc.monitor.marca }}[{{ miPc.monitor.tamanno
}}'']</td>
            <td>{{ miPc.raton.marca }}</td>
            <td>{{ miPc.teclado.marca }}</td>
            <td><i class="fa-solid fa-trash fa-fade" style="color:
#ff0000;" (click)="eliminarPc(miPc)"></i></td>
          </tr>
        </tbody>
      </table>
```

Tabla 18 Maquetado html de la tabla computadores

La Imagen 72 presenta la inclusión de la tabla **Ratón** que mostrará la información en "**cuerpo.componente.html**".

Imagen 72 Maquetado html para la información de los ratones

Fuente: Autor

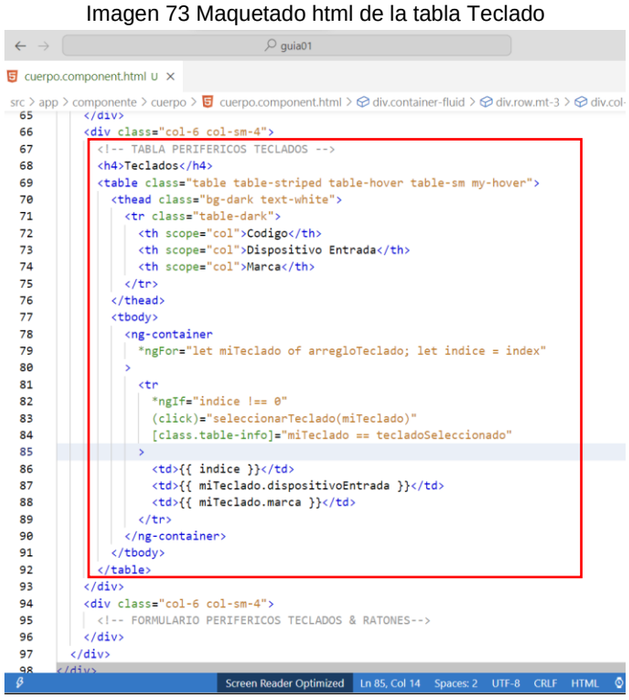
En la Tabla 19 se presenta el bloque HTML que muestra la tabla de los ratones, esta debe ser añadida **después** del ***comentario*** **“<!----TABLA PERIFERICOS RATONES--->”**

```
<!-- TABLA PERIFERICOS RATONES -->
<h4>Tabla Ratones</h4>
<table class="table table-striped table-hover table-sm my-hover">
  <thead class="bg-dark text-white">
    <tr class="table-dark">
      <th scope="col">Código</th>
      <th scope="col">Marca</th>
      <th scope="col">Dispositivo Entrada</th>
    </tr>
  </thead>
  <tbody>
    <ng-container
      *ngFor="let miRaton of arregloRaton;"
    >
      <tr
        *ngIf=" miRaton.id !== 1"
        (click)="seleccionarRaton(miRaton)"
        [class.table-info]="miRaton == ratonSeleccionado"
      >
        <td>{{ miRaton.id }}</td>
        <td>{{ miRaton.marca }}</td>
        <td>{{ miRaton.dispositivoEntrada }}</td>
      </tr>
    </ng-container>
  </tbody>
</table>
```

Tabla 19 Maquetado html de la tabla Ratones

La Imagen 73 presenta la inclusión de la tabla **Teclado** que mostrará la información en “**cuerpo.componente.html**”.

Imagen 73 Maquetado html de la tabla Teclado



Fuente: Autor

En la Tabla 20 se presenta el bloque HTML que muestra la tabla de los teclados, esta debe ser añadida **después** del ***comentario*** **“<!---TABLA PERIFERICOS TECLADOS-->”**

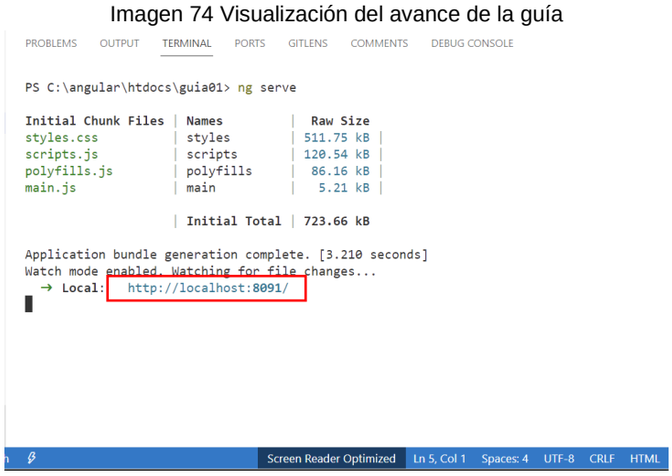
```
<!-- TABLA PERIFERICOS TECLADOS -->
      <h4>Tabla Teclados</h4>
      <table class="table table-striped table-hover table-sm my-hover">
        <thead class="bg-dark text-white">
          <tr class="table-dark">
            <th scope="col">Codigo</th>
            <th scope="col">Marca</th>
            <th scope="col">Dispositivo Entrada</th>
          </tr>
        </thead>
        <tbody>
          <ng-container
            *ngFor="let miTeclado of arregloTeclado;"
          >
            <tr
              *ngIf="miTeclado.id !== 1"
              (click)="seleccionarTeclado(miTeclado)"
              [class.table-info]="miTeclado == tecladoSeleccionado"
            >
              <td>{{ miTeclado.id }}</td>
              <td>{{ miTeclado.marca }}</td>
              <td>{{ miTeclado.dispositivoEntrada }}</td>
            </tr>
          </ng-container>
        </tbody>
      </table>
```

Tabla 20 Maquetado html de la tabla Teclado

*ngFor	La directiva *ngFor utiliza la sintaxis ***let variable of array*** para iterar sobre el array y asignar cada elemento del array a una variable.
*ngIf	La directiva *ngIf utiliza la **sintaxis condición** para evaluar una condición y mostrar u ocultar un elemento HTML.

Para verificar el avance del taller, se sugiere al lector que ejecute el proyecto. Desde la terminal de Visual Estudio Code se debe ejecutar el comando “**ng serve**”.

Imagen 74 Visualización del avance de la guía

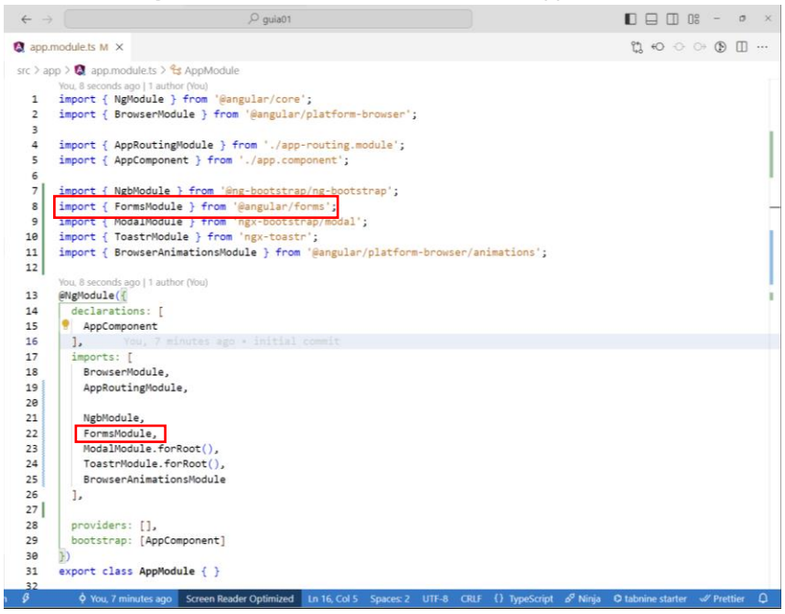


Fuente: Autor

5.14. Módulo @angular/forms

Para la manipulación de formularios en Angular es necesaria la inclusión de la librería "***@angular/forms***", esta librería se debe incluir en el archivo "***src\app\app.module.ts***". La Imagen 75 presenta la inclusión de la librería.

Imagen 75 Inclusión de librerías Forms en app.module.ts

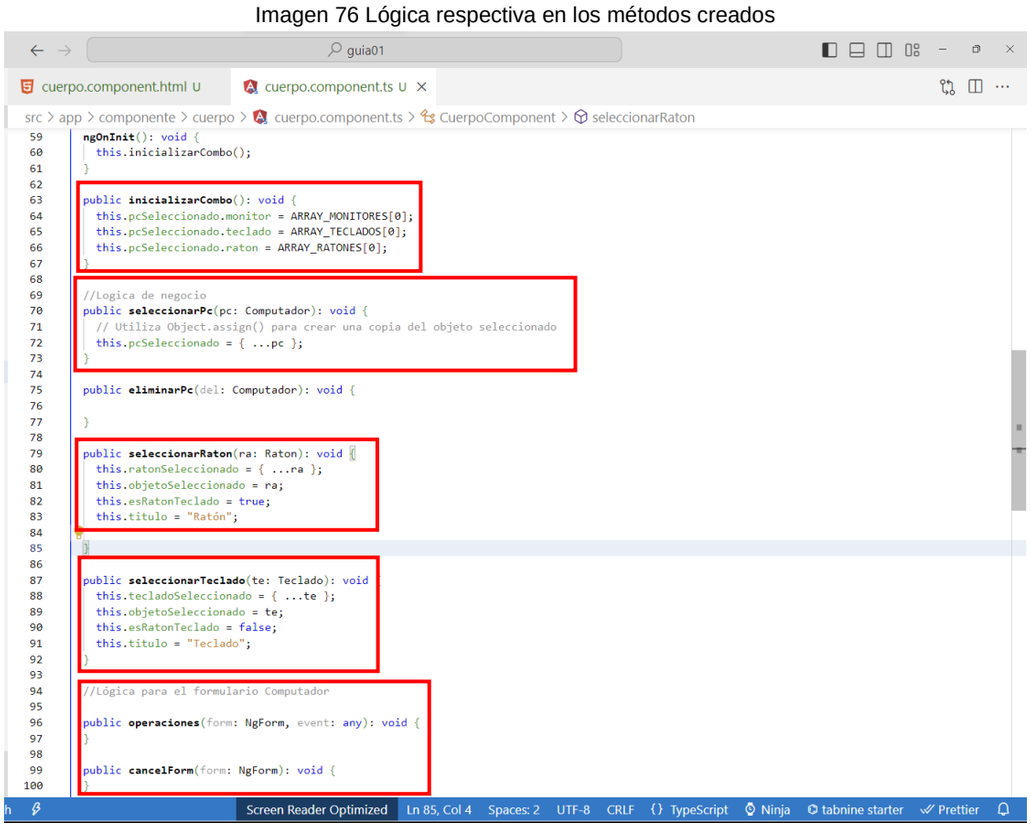


Fuente: Autor

5.15. Funcionalidad de Selección de objetos Componente cuerpo

La clase del componente cuerpo requiere propiedades y métodos que permitan incluir las funcionalidades para el CRUD a implementar. La Imagen 76 presenta la inclusión del modelos y el mock a utilizar en este taller, así como la definición de las propiedades o atributos de la clase y su inicialización en el constructor.

Imagen 76 Lógica respectiva en los métodos creados



Fuente: Autor

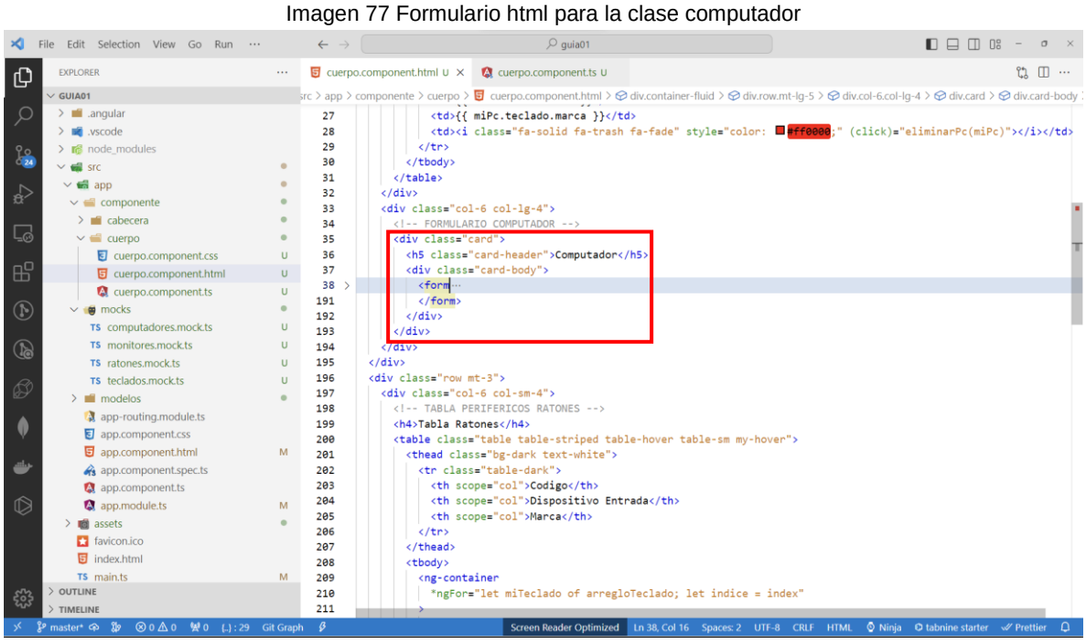
En la Tabla 21 se presenta a manera de ejemplo el contenido de los métodos con el respectivo código en el "**componente.cuerpo.ts"** que debe ser sobrescrito **después** del ***CONTRUCTOR*** entre los comentarios de "`Inicio de la lógica CRUD para un computador`" hasta "`Fin de la lógica CRUD para un computador`".

```
//*********Inicio de la lógica CRUD para un computador*********
/* El método `ngOnInit()` es un gancho (hook) del ciclo de vida de Angular que se
llama después que
Angular ha inicializado todas las propiedades vinculadas a datos de una
directiva. En este caso específico, el
El método `ngOnInit()` está llamando al método `inicializarCombo()` para
inicializar los arrays del monitor,
teclado y mouse en el index 0*/
 ngOnInit(): void {
    this.inicializarCombo();
  }

  public inicializarCombo(): void {
    this.pcSeleccionado.monitor = ARRAY_MONITORES[0];
    this.pcSeleccionado.teclado = ARRAY_TECLADOS[0];
    this.pcSeleccionado.raton = ARRAY_RATONES[0];
  }
/**
El método utiliza `Object.assign()` junto con el operador de extensión
(`{...pc }`) para crear una copia superficial del objeto `pc` y lo asigna.
 */
  public seleccionarPc(pc: Computador): void {
    // Utiliza Object.assign() para crear una copia del objeto seleccionado
    this.pcSeleccionado = { ...pc };
  }

  //Lógica para el eliminar Computador
  public eliminarPc(del: Computador): void {
  }

  //Lógica para el formulario Computador
  public operaciones(form: NgForm, event: any): void {

  }
  //Lógica para el actualizar Computador
  public actualizarPc(form: NgForm): void {

  }
  //Lógica para el limpiar el formulario Computador
  public cancelForm(form: NgForm): void {
  }
  public resetPc(): void {
    this.pcSeleccionado = this.inicializarPc();
  }
//**********Fin de la lógica CRUD para un computador**********
//...
```

Tabla 21 Lógica funcionalidad de selección de objetos en formularios

La Imagen 77 presenta el bloque de código en donde se debe agregar el formulario para la lógica de la clase Computador en "**cuerpo.component.html**".

Imagen 77 Formulario html para la clase computador



Fuente: Autor

En la Tabla 22 se presenta a manera de ejemplo la estructura del formulario con los campos establecidos en el modelo de creación de un computador, **después** del ***comentario*** "`<!-- FORMULARIO COMPUTADOR -->`".

```
      <!-- FORMULARIO COMPUTADOR -->
      <div class="card">
        <h5 class="card-header">Computador</h5>
        <div class="card-body">
          <form
            #formCrearPc="ngForm"
            (submit)="formCrearPc.form.valid && operaciones(formCrearPc,
$event)"
            novalidate
          >
            <div class="row g-3 mt-1">
              <div class="col-3">
                <label for="miNombre" class="form-label">
                  <span class="text-success">Nombre</span>
                </label>
              </div>
              <div class="col-8">
                <input
                  type="text"
                  class="form-control form-control-sm"
                  id="miNombre"
                  name="miNombre"
                  #miNombre="ngModel"
                  [(ngModel)]="pcSeleccionado.nombre"
                  [ngClass]="{
                    'is-invalid': formCrearPc.submitted &&
miNombre.invalid
                  }"
                  required />
```

```
                <div
                  class="invalid-feedback"
                  *ngIf="formCrearPc.submitted && miNombre.invalid"
                >
                  <span *ngIf="miNombre.errors?.['required']">
                    Nombre no registrado!!!
                  </span>
                </div>
              </div>
            </div>
            <div class="row g-3 mt-1">
              <div class="col-3">
                <label for="monitor" class="text-
success">Monitor:</label>
              </div>
              <div class="col-8">
                <select
                  name="monitor"
                  id="monitor"
                  class="form-select form-select-sm"
                  [(ngModel)]="pcSeleccionado.monitor"
                >
                  <option
                    *ngFor="let objMonitor of arregloMonitor"
                    [ngValue]="objMonitor"
                    [disabled]="objMonitor.idMonitor == 1"
                  >
                    {{ objMonitor.marca }} - {{ objMonitor.tamanno }}''
                  </option>
                </select>
              </div>
            </div>
            <div class="row g-3 mt-1">
              <div class="col-3">
                <label for="teclado" class="text-
success">Teclado:</label>
              </div>
              <div class="col-8">
                <select
                  name="teclado"
                  id="teclado"
                  class="form-select form-select-sm"
                  [(ngModel)]="pcSeleccionado.teclado"
                >
                  <option
                    *ngFor="let objTeclado of arregloTeclado"
                    [ngValue]="objTeclado"
                    [disabled]="objTeclado.id == 1"
                  >
                    {{ objTeclado.marca }} - {{
objTeclado.dispositivoEntrada }}
                  </option>
                </select>
              </div>
            </div>
```

```
            <div class="row g-3 mt-1">
              <div class="col-3">
                <label for="raton" class="text-success">Ratón:</label>
              </div>
              <div class="col-8">
                <select
                  name="raton"
                  id="raton"
                  #raton="ngModel"
                  class="form-select form-select-sm"
                  [ariaPlaceholder]="pcSeleccionado"
                  [(ngModel)]="pcSeleccionado.raton"
                >
                  <option
                    *ngFor="let objRaton of arregloRaton"
                    [ngValue]="objRaton"
                    [disabled]="objRaton.id == 1"
                  >
                    {{ objRaton.marca }} - {{ objRaton.dispositivoEntrada
}}
                  </option>
                </select>
              </div>
            </div>

<!-- Bloque con a la directiva *ngIf para mostrar las cajas de crear o de
actualizar dependiendo si el id.Computador es 0 para CREAR y diferente de
para ACTUALIZAR-->
            <div
            *ngIf="
              pcSeleccionado.idComputador == 0;
              then creacionPc;
              else actualizacionPc
            "
          ></div>

            <!-- Muestra el botón de crear una Computador -->
            <ng-template #creacionPc>
              <div class="my-3">
                <div class="d-flex justify-content-center">
                  <div class="d-grid gap-1 col-10">
                    <button
                      type="submit"
                      class="btn btn-primary btn-sm"
                      id="btnCrearPc"
                      name="btnCrearPc"
                    >
                      Crear Computador
                    </button>
                  </div>
                </div>
              </div>
            </ng-template>
```

```
            <!-- Muestra los botones de actualizar o cancelar cuando se
carga un objeto diferente de 0 -->
            <ng-template #actualizacionPc>
              <div class="container overflow-hidden text-center">
                <div class="row gx-5 justify-content-center">
                  <div class="col">
                    <div class="p-3">
                      <button
                        type="submit"
                        class="btn btn-success btn-sm"
                        id="btnActualizarPc"
                        name="btnActualizarPc"
                      >
                        Actualizar
                      </button>
                    </div>
                  </div>
                  <div class="col">
                    <div class="p-3">
                      <button
                        type="button"
                        class="btn btn-secondary btn-sm"
                        (click)="cancelForm(formCrearPc)"
                      >
                        Cancelar
                      </button>
                    </div>
                  </div>
                </div>
              </div>
            </ng-template>
          </form>
        </div>
      </div>
```

Tabla 22 Maquetado HTML formulario computadores

[(ngModel)]	Esta directiva se utiliza para enlazar un elemento HTML con una propiedad de un componente.
[ngValue]	Esta directiva se utiliza para enlazar un objeto con un elemento HTML.
[disabled]	Esta directiva se utiliza para deshabilitar un elemento HTML en función de una condición.

5.15.1. Funcionalidad para eliminar un Computador

La Imagen 78 presenta el bloque de código, con la lógica para eliminar un computador

Imagen 78 Lógica para eliminar un computador

Fuente: Autor

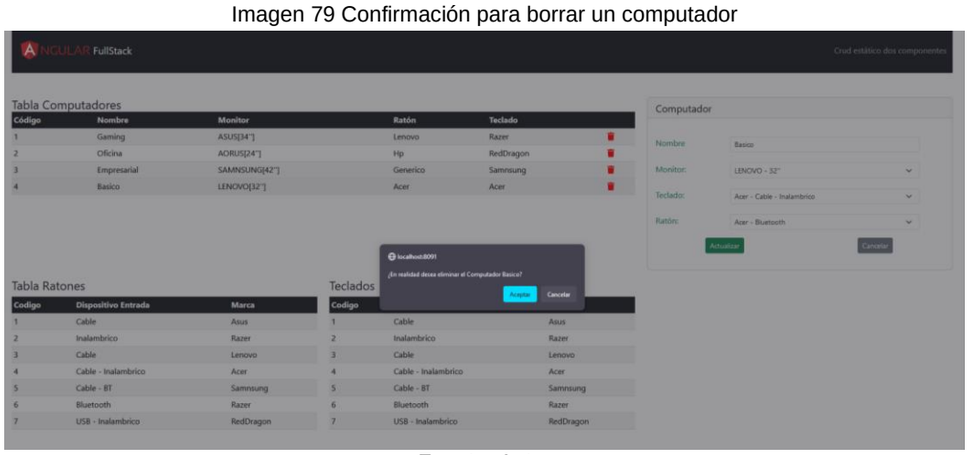
En la Tabla 23 Código para eliminar un computador se presenta a manera de ejemplo la lógica para eliminar un computador. Esta consiste en recorrer el `this.arregloPc` y con el método `filter` se busca y se elimina el objeto "*del*" del computador seleccionado.

```
public eliminarPc(del: Computador): void {
  if (confirm(`¿En realidad desea eliminar el Computador ${del.nombre}?`)) {
    this.arregloPc = this.arregloPc.filter((mipc) => mipc !== del);
  }
}
```

Tabla 23 Código para eliminar un computador

La Imagen 79 presenta a manera de ilustración, un banner con la funcionalidad de borrar.

Imagen 79 Confirmación para borrar un computador

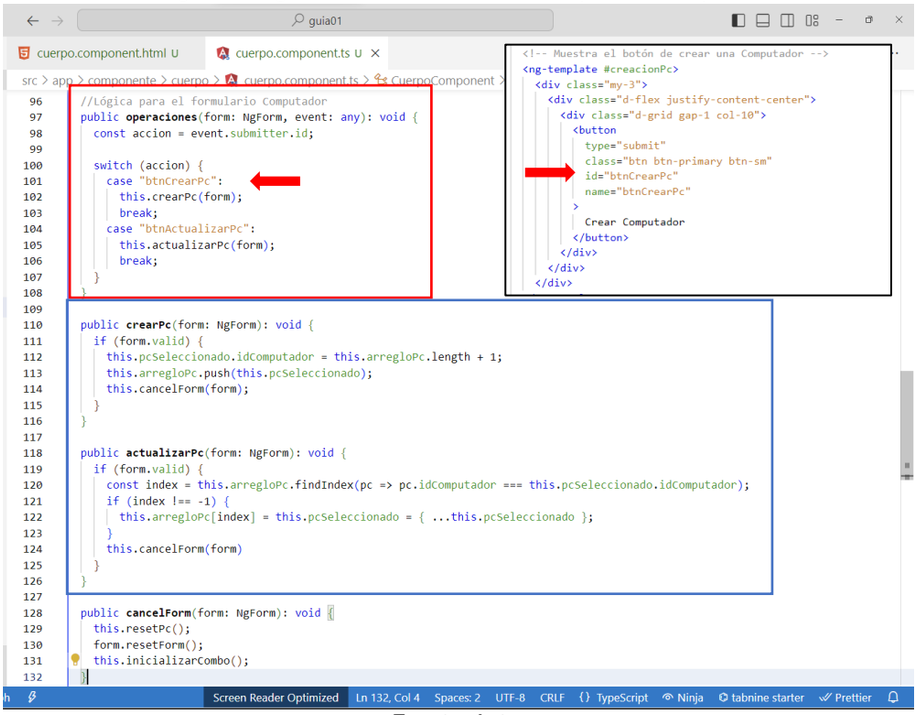


Fuente: Autor

5.15.2. Funcionalidad y lógica para crear y actualizar un Computador

La Imagen 80 presenta a manera de ilustración, el código con la lógica que se debe agregar en los métodos que habían quedado en blanco anteriormente.

Imagen 80 Logia funcionalidad de formulario para un computador



Fuente: Autor

En la Tabla 24 se presenta la lógica que debe ser agregada en el método **operaciones** del archivo **cuerpo.component.ts** **después** del ***comentario*** `//Lógica para el formulario Computador`**;** esto funciona mediante un switch, cuando en el cuerpo.componente.html se presenta un (click) sobre los botones de “Crear Computador” o “Actualizar”, ejecuta el bloque de código al que fue accionado.

```
//Lógica para el formulario Computador
  public operaciones(form: NgForm, event: any): void {
    const accion = event.submitter.id;

    switch (accion) {
      case "btnCrearPc":
        this.crearPc(form);
        break;
      case "btnActualizarPc":
        this.actualizarPc(form);
        break;
    }
  }
```

Tabla 24 Función switch para crear o actualizar un computador

En la Tabla 25 se presentan **nuevas** funcionalidades que se deben **agregar** **después** del método **`operaciones`**, estos métodos funcionan validando los campos del formulario después haciendo un incremento +1 al idComputador en relación con el tamaño del array (arregloPc) que posteriormente se agregará mediante el método *push* del arreglo.

```
  public crearPc(form: NgForm): void {
    if (form.valid) {
      this.pcSeleccionado.idComputador = this.arregloPc.length + 1;
      this.arregloPc.push(this.pcSeleccionado);
      this.cancelForm(form);
    }
  }
```

Tabla 25 Función para crear un computador

En la Tabla 26 se presentan las **nuevas** funciones que deben ser **agregadas después** del método **crearPc,** con la lógica de validación del formulario, y la respectiva actualización del pcSeleccionado; esta consiste buscando la posición donde se encuentra el objeto que posteriormente se asignará al índex del objeto a actualizar.

```
  public actualizarPc(form: NgForm): void {
    if (form.valid) {
      const index = this.arregloPc.findIndex(pc => pc.idComputador ===
this.pcSeleccionado.idComputador);
      if (index !== -1) {
        this.arregloPc[index] = this.pcSeleccionado = { ...this.pcSeleccionado };
      }
      this.cancelForm(form)
    }
  }
  public cancelForm(form: NgForm): void {
    this.resetPc();
    form.resetForm();
    this.inicializarCombo();
  }

  public resetPc(): void {
    this.pcSeleccionado = this.inicializarPc();
  }
//**********Fin de la lógica CRUD para un computador**********
```

Tabla 26 Función para actualizar un computador

En la Imagen 81 se muestra todo el bloque CRUD con las funcionalidades para un Computador.

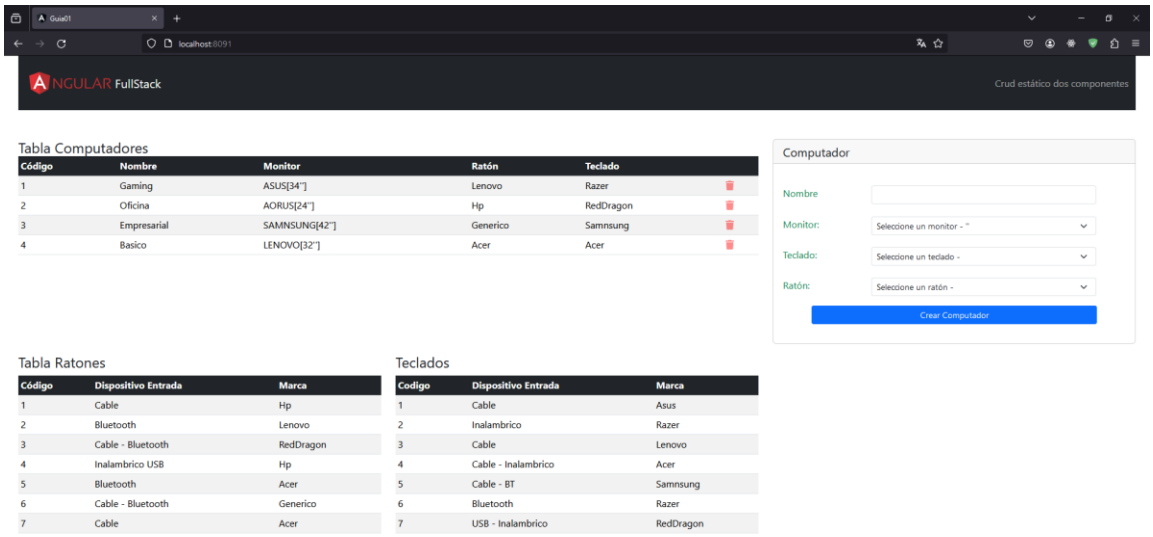
Imagen 81 Bloque CRUD para el modelo computador

```
54  //*********Inicio de la lógica CRUD para un computador*********
55  ngOnInit(): void {
56    this.inicializarCombo();
57  }
58
59  public inicializarCombo(): void {
60    this.pcSeleccionado.monitor = ARRAY_MONITORES[0];
61    this.pcSeleccionado.teclado = ARRAY_TECLADOS[0];
62    this.pcSeleccionado.raton = ARRAY_RATONES[0];
63  }
64
65  //Logica de negocio
66  public seleccionarPc(pc: Computador): void {
67    // Utiliza Object.assign() para crear una copia del objeto seleccionado
68    this.pcSeleccionado = { ...pc };
69  }
70
71  public eliminarPc(del: Computador): void {
72    if (confirm(`¿En realidad desea eliminar el Computador ${del.nombre}?`)) {
73      this.arregloPc = this.arregloPc.filter((mipc) => mipc !== del);
74    }
75  }
76
77  //Lógica para el formulario Computador
78  public operaciones(form: NgForm, event: any): void {
79    const accion = event.submitter.id;
80
81    switch (accion) {
82      case "btnCrearPc":
83        this.crearPc(form);
84        break;
85      case "btnActualizarPc":
86        this.actualizarPc(form);
87        break;
88    }
89  }
```

```
90  public crearPc(form: NgForm): void {
91    if (form.valid) {
92      this.pcSeleccionado.idComputador = this.arregloPc.length + 1;
93      this.arregloPc.push(this.pcSeleccionado);
94      this.cancelForm(form);
95    }
96  }
97
98  public actualizarPc(form: NgForm): void {
99    if (form.valid) {
100     const index = this.arregloPc.findIndex(pc => pc.idComputador === this.pcSeleccionado.idComputador);
101     if (index !== -1) {
102       this.arregloPc[index] = this.pcSeleccionado = { ...this.pcSeleccionado };
103     }
104     this.cancelForm(form)
105   }
106 }
107
108 public cancelForm(form: NgForm): void {
109   this.resetPc();
110   form.resetForm();
111   this.inicializarCombo();
112 }
113
114 public resetPc(): void {
115   this.pcSeleccionado = this.inicializarPc();
116 }
117
118 //**********Fin de la lógica CRUD para un computador**********
```

5.15.3. Revisión avance para el CRUD de un computador

Para este punto, ya están todas las funcionalidades CRUD para el modelo Computador, el lector debe ejecutar en la terminal `ng serve` para ver los avances.



Fuente: Autor

5.15.4. Funcionalidad CRUD para los modelos de Ratón y Teclado

La Imagen 82 presenta, el bloque de métodos necesarios CRUD para los periféricos Ratón y Teclado

Imagen 82 Funcionalidad CRUD periféricos

```
122   //*********Inicio de la lógica CRUD para los Ratones y Teclado*********
123   public seleccionarRaton(ra: Raton): void {···
129   }
130
131   public seleccionarTeclado(te: Teclado): void {···
136   }
137   public operacionesPerifericos(form: NgForm, event: any): void {···
148   }
149   public creacionPerifericos(form: NgForm): void {···
156   }
157   public actualizacionPerifericos(form: NgForm): void {···
163   }
164   public crearRaton(form: NgForm): void {···
173   }
174   public crearTeclado(form: NgForm): void {···
182   }
183   public actualizarRaton(form: NgForm): void {···
192   }
193
194   public actualizarTeclado(form: NgForm): void {···
203   }
204
205   public deletePeriferico(): void {···
215   }
216   public tipoPeriferico(rate: boolean): void {···
220   }
221   public cancelFormPerifericos(form: NgForm): void {···
225   }
226   public resetRaton(): void {···
229   }
230
231   public resetTeclado(): void {···
234   }
235   //*********Fin de la logia CRUD para los periféricos**********
236   // Inicialización de Clases
```

Fuente: Autor

En la Tabla 27 se presenta a manera de ejemplo los métodos CRUD para los periféricos, las siguientes funciones deben ir en bloque **después** del ***comentario***
"//**********Inicio de la lógica CRUD para los Ratones y Teclado"

```
//**********Inicio de la lógica CRUD para los Ratones y Teclado
public seleccionarRaton(ra: Raton): void {
  this.ratonSeleccionado = { ...ra };
  this.objetoSeleccionado = ra;
  this.esRatonTeclado = true;
  this.titulo = "Ratón";
}
public seleccionarTeclado(te: Teclado): void {
  this.tecladoSeleccionado = { ...te };
  this.objetoSeleccionado = te;
  this.esRatonTeclado = false;
  this.titulo = "Teclado";
}
public operacionesPerifericos(form: NgForm, event: any): void {
  const accion = event.submitter.id

  switch (accion) {
    case "crearPeriferico":
      this.creacionPerifericos(form);
      break;
    case "actualizarPeriferico":
      this.actualizacionPerifericos(form);
      break;
  }
}
public creacionPerifericos(form: NgForm): void {
  if (this.esRatonTeclado === true) {
    this.crearRaton(form);
  } else {
    this.crearTeclado(form);

  }
}
public actualizacionPerifericos(form: NgForm): void {
  if (this.esRatonTeclado === true) {
    this.actualizarRaton(form);
  } else {
    this.actualizarTeclado(form);
  }
}
```

*(Esta estos métodos de **`creacionPerifericos`** y **`actualizacionPerifericos`** funcionan con una bandera, boolean y un selectBox en el html, cuando es `true`, ejecuta el código para crear un Ratón y cuando es `false` se ejecuta la creación de un Teclado.)*

```
  public crearRaton(form: NgForm): void {

    if (form.valid) {
      this.ratonSeleccionado.id = this.arregloRaton.length + 1;
      this.arregloRaton.push(this.ratonSeleccionado);
      this.resetRaton();
      form.reset();
      this.inicializarCombo();
    }
  }
  public crearTeclado(form: NgForm): void {
    if (form.valid) {
      this.tecladoSeleccionado.id = this.arregloTeclado.length + 1;
      this.arregloTeclado.push(this.tecladoSeleccionado);
      this.resetTeclado();
      form.reset();
      this.inicializarCombo();
    }
  }

  public actualizarRaton(form: NgForm): void {
    if (form.valid) {
      const index = this.arregloRaton.findIndex(ra => ra.id ===
this.ratonSeleccionado.id);
      if (index !== -1) {
        this.arregloRaton[index] = this.ratonSeleccionado = { ...this.ratonSeleccionado
};
      }
      this.resetRaton();
      form.resetForm();
    }
  }

  public actualizarTeclado(form: NgForm): void {
    if (form.valid) {
      const index = this.arregloTeclado.findIndex(te => te.id ===
this.tecladoSeleccionado.id);
      if (index !== -1) {
        this.arregloTeclado[index] = this.tecladoSeleccionado = {
...this.tecladoSeleccionado };
      }
      this.resetTeclado();
```

```
      form.resetForm();
    }
  }

  public deletePeriferico(): void {
    if (confirm(`¿En realidad desea eliminar el periferico
${this.objetoSeleccionado?.marca}?`)) {
      if (this.objetoSeleccionado instanceof Raton) {
        this.arregloRaton = this.arregloRaton.filter((objRaton) => objRaton !=
this.objetoSeleccionado);
        this.resetRaton();
      } else {
        this.arregloTeclado = this.arregloTeclado.filter((objTeclado) =>
objTeclado != this.objetoSeleccionado);
        this.resetTeclado();
      }
    }
  }

  public tipoPeriferico(rate: boolean): void {
    this.esRatonTeclado = rate;
    this.esRatonTeclado ? this.titulo = "Ratón" : this.titulo = "Teclado";

  }
  public cancelFormPerifericos(form: NgForm): void {
    this.objetoSeleccionado = this.inicializarRaton();
    this.resetRaton();
    this.resetTeclado();
  }
  public resetRaton(): void {
    this.ratonSeleccionado = this.inicializarRaton();
    this.objetoSeleccionado = this.inicializarRaton()
  }

  public resetTeclado(): void {
    this.tecladoSeleccionado = this.inicializarTeclado();
    this.objetoSeleccionado = this.inicializarTeclado();
  }
//*******Fin de la logia CRUD para los periféricos*******
// Inicialización de Clases
```

Tabla 27 Lógica CRUD para los periféricos

La función **`tipoPeriferico()`** recibe la selección de un ComboBox en el formulario HTML de los periféricos.

En la Imagen 83 se presenta el bloque de código en donde se debe **agregar** el formulario para la lógica del formulario Teclado y Ratón.

Imagen 83 Maquetado HTML con el formulario de los Periféricos

Fuente: Autor

La Tabla 28 se presenta a manera de ejemplo la estructura del formulario con los campos establecidos en el modelo de los preferidos Ratón y Teclado, se debe agregar **después** del **comentario** "`<!-- FORMULARIO PERIFERICOS TECLADOS & RATONES-->`".

```
      <!-- FORMULARIO PERIFERICOS TECLADOS & RATONES-->
     <div class="card">
       <h5 class="card-header">Periferico {{titulo}}</h5>
       <div class="card-body">
         <div class="d-flex justify-content-center align-items-center">
           <!-- Input-Radio selección de objeto Ratón ó Teclado -->
         <label>
           <input type="radio"  name="tipoDispositivo"
(click)="tipoPeriferico(true)" checked [disabled]=" (ratonSeleccionado.id ||
tecladoSeleccionado.id) != 0 " /> Ratón
           <input type="radio" name="tipoDispositivo"
(click)="tipoPeriferico(false)" [disabled]=" (ratonSeleccionado.id ||
tecladoSeleccionado.id) != 0 " /> Teclado
         </label>
        </div>
         <form
           #formCrearPerifericos="ngForm"
           (ngSubmit)="operacionesPerifericos(formCrearPerifericos, $event)"
           novalidate
         >
           <div class="row g-3 mt-1">
```

```
              <div class="col-3">
                <label for="marca" class="text-success">Marca:</label>
              </div>
              <div class="col-8">
                <input
                  type="text"
                  class="form-control form-control-sm"
                  id="marca"
                  name="marca"
                  #marca="ngModel"
                  [(ngModel)]="ratonSeleccionado.marca"
                  [(ngModel)]="tecladoSeleccionado.marca"
                  [ngClass]="{
                    'is-invalid': formCrearPerifericos.submitted && marca.invalid
}"
                  required
                />
                <div
                  class="invalid-feedback"
                  *ngIf="formCrearPerifericos.submitted && marca.invalid"
                >
                  <span *ngIf="marca.errors?.['required']">
                    Marca no registrado!!!
                  </span>
                </div>
              </div>
            </div>
            <div class="row g-3 mt-1">
              <div class="col-3">
                <label for="dispositivoEntrada" class="text-success"
                  >Dispositivo Entrada:</label
                >
              </div>
              <div class="col-8">
                <input
                  type="text"
                  class="form-control form-control-sm"
                  placeholder="Alambrico - Inalambrico "
                  id="dispositivoEntrada"
                  name="dispositivoEntrada"
                  #dispositivoEntrada="ngModel"
                  [(ngModel)]="ratonSeleccionado.dispositivoEntrada"
                  [(ngModel)]="tecladoSeleccionado.dispositivoEntrada"
                  [ngClass]="{
                    'is-invalid':
                      formCrearPerifericos.submitted && dispositivoEntrada.invalid
                  }"
                  Required />
```

```
              <div
                  class="invalid-feedback"
                  *ngIf="
                    formCrearPerifericos.submitted && dispositivoEntrada.invalid
                  "
                >
                  <span *ngIf="dispositivoEntrada.errors?.['required']">
                    Dispositivo Entrada no registrado!!!
                  </span>
                </div>
              </div>
            </div>

<!—Si el id del objeto seleccionado es 0 muestra el tempate #crearPeriferico de lo
contrario el #updatePeriferico -->
            <div
              class="d-grid py-4"
              *ngIf="objetoSeleccionado.id === 0; then crearPeriferico; else
updatePeriferico"
            ></div>

            <ng-template #crearPeriferico>
              <div class="my-3">
                <div class="d-flex justify-content-center">
                  <div class="d-grid gap-1 col-10">
                    <button id="crearPeriferico" type="submit" class="btn btn-primary
btn-sm">
                      Crear {{titulo}}
                    </button>
                  </div>
                </div>
              </div>
            </ng-template>
            <ng-template #updatePeriferico>
              <div class="container overflow-hidden text-center">
                <div class="row gx-5 justify-content-center">
                  <div class="col">
                    <div class="p-3">
                      <button id="actualizarPeriferico" type="submit" class="btn
btn-success btn-sm">
                        Actualizar
                      </button>
                    </div>
                  </div>
```

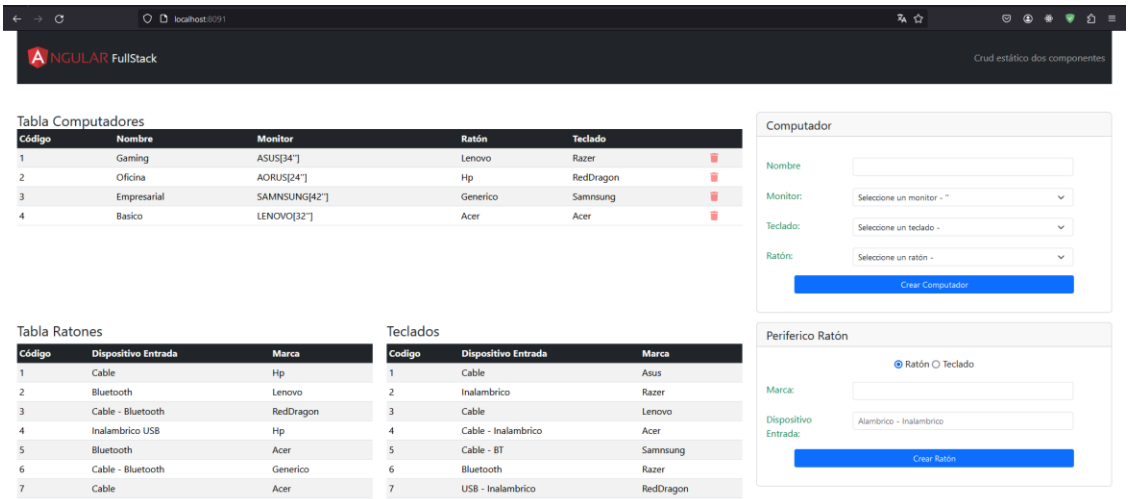
```
                  <div class="col">
                    <div class="p-3">
                      <button
                        type="button"
                        class="btn btn-danger btn-sm"
                        (click)="deletePeriferico()"
                      >
                      <i class="fa-solid fa-trash fa-beat" style="color:
#fff;"></i>
                      </button>
                    </div>
                  </div>
                  <div class="col">
                    <div class="p-3">
                      <button
                        type="button"
                        class="btn btn-secondary btn-sm"
                        (click)="cancelFormPerifericos(formCrearPerifericos)"
                >
                        Cancelar
                      </button>
                    </div>
                  </div>
                </div>
              </div>
            </ng-template>
          </form>
        </div>
      </div>
```

Tabla 28 Formulario HTML para los periféricos

5.16. Prueba final de la guía01

Para verificar el funcionamiento del taller de la Imagen 84, se sugiere al lector que ejecute el proyecto. Desde la terminal de Visual Estudio Code se debe ejecutar el comando "***ng serve***".

Imagen 84 Resultado final de la guia01



Fuente: Autor

5.16.1. Resultado

En el siguiente apartado podrá encontrar el resultado final, de cómo se debe ver y funcionar nuestra página web con los temas vistos para este material.

Click guia01

6. Reto para el lector

El próximo objetivo una vez terminado todo lo propuesto para esta guia01, como ingeniero debe agregar otra fila para darle toda la funcionalidad CRUD a la clase **Monitor**.

Enlace de los recursos propuestos

Titulo	Enlace
Cómo Instalar Angular en Windows, macOS y Linux	https://kinsta.com/es/base-de-conocimiento/instalar-angular/#requisitos-previos-de-angular
Clases en Typescript	https://gustavodohara.com/blogangular/clases-en-typescript/
Herencia de clases en Typescript	https://gustavodohara.com/blogangular/herencia-de-clases-en-typescript/
Creando clases con constructores en Typescript	https://gustavodohara.com/blogangular/creando-clases-con-constructores-en-typescript/