**FRONTEND**

**TEMA1:**

**¿Qué es Git, GitHub?**

**Git**

Git es un sistema de control de versiones distribuido. Git fue impulsado por Linus Torvalds y el equipo de desarrollo del Kernel de Linux. Ellos estaban usando otro sistema de control de versiones de código abierto, que ya por aquel entonces era distribuido. Todo iba bien hasta que los gestores de aquel sistema de control de versiones lo convirtieron en un software propietario. Lógicamente, no era compatible estar construyendo un sistema de código abierto, tan representativo como el núcleo de Linux, y estar pagando por usar un sistema de control de versiones propietario. Por ello, el mismo equipo de desarrollo del Kernel de Linux se tomó la tarea de construir desde cero un sistema de versionado de software, también distribuido, que aportase lo mejor de los sistemas existentes hasta el momento.

**GitHub**

es un servicio para alojamiento de repositorios de software gestionados por el sistema de control de versiones Git. Por tanto, Git es algo más general que nos sirve para controlar el estado de un desarrollo a lo largo del tiempo, mientras que GitHub es algo más particular: un sitio web que usa Git para ofrecer a la comunidad de desarrolladores repositorios de software. En definitiva, GitHub es un sitio web pensado para hacer posible el compartir el código de una manera más fácil y al mismo tiempo darle popularidad a la herramienta de control de versiones en sí, que es Git.

**Comandos básicos de Git**

|  |  |
| --- | --- |
| **Comando** | **Función** |
| git help | Muestra una lista con los comandos más utilizados en GIT. |
| git init | Podemos ejecutar ese comando para crear localmente un repositorio con GIT y así utilizar todo el funcionamiento que GIT ofrece. |
| git add + path | Agrega al repositorio los archivos que indiquemos. |
| git add -A | Agrega al repositorio TODOS los archivos y carpetas que estén en nuestro proyecto, los cuales GIT no está siguiendo. |
| **git commit -m "mensaje" + archivos** | Hace commit a los archivos que indiquemos, de esta manera quedan guardados nuestras modificaciones. |
| **git commit -am "mensaje"** | Hace commit de los archivos que han sido modificados y GIT los está siguiendo. |
| **git checkout -b NombreDeBranch** | Crea un nuevo branch y automáticamente GIT se cambia al branch creado, clonando el branch desde donde ejecutamos el comando. |
| **git branch** | Nos muestra una lista de los branches que existen en nuestro repositorio. |
| **git checkout NombreDeBranch** | Sirve para moverse entre branches, en este caso vamos al branch que indicamos en el comando. |
| **git merge NombreDeBranch** | Hace un merge entre dos branches, en este caso la dirección del merge sería entre el branch que indiquemos en el comando, y el branch donde estemos ubicados. |
| **git status** | Nos indica el estado del repositorio, por ejemplo, cuales están modificados, cuales no están siendo seguidos por GIT, entre otras características. |
| **git clone URL/name.git NombreProyecto** | Clona un proyecto de git en la carpeta NombreProyecto. |
| **git push origin NombreDeBranch** | Luego de que hicimos un git commit, si estamos trabajando remotamente, este comando va a subir los archivos al repositorio remoto, específicamente al branch que indiquemos. |
| **git pull origin NombreDeBranch** | Hace una actualización en nuestro branch local, desde un branch remoto que indicamos en el comando. |

**Comandos basicos para repositorios remotos**

|  |  |
| --- | --- |
| **Comando** | **Función** |
| git remote | Mostrar lista de repositorios remotos |
| git remote add [nombre] [url] | Añadir repositorios remotos |
| git fetch [remote-name] | Recibir repositorios remotos |
| git push [nombre-remoto][nombre-rama] | Recibir repositorios remotos |
| git remote show [nombre] | Inspeccionar repositorios remotos |
| git remote rename | Eliminar repositorios remotos |

**Push**  
El comando git push se usa para cargar contenido del repositorio local en un repositorio remoto, Empujar es cómo transferir confirmaciones desde su repositorio local a un repositorio remoto.

Uso de git push:

|  |  |
| --- | --- |
| **Comando** | **Función** |
| git push <remote> <branch> | Esto crea una sucursal local en el repositorio de destino. |
| git push <remote> --force | Igual que el comando anterior, pero forzar el empuje incluso si resulta en una fusión que no avanza rápidamente. |
| git push <remote> --all | Empuje todas sus sucursales locales al control remoto especificado. |
| git push <remote> --tags | as etiquetas no se insertan automáticamente cuando empuja una rama o usa la opción --todos. La bandera --tags envía todas sus etiquetas locales al repositorio remoto. |

**¿Qué es Nodejs?**

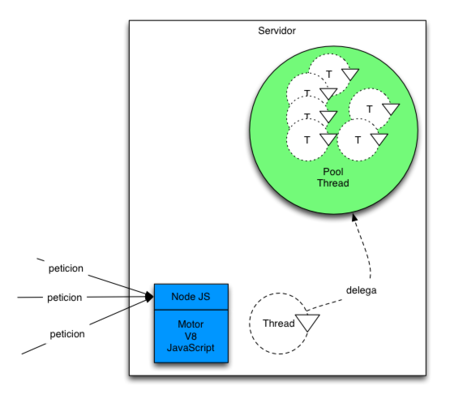
Node.js es un entorno JavaScript de lado de servidor que utiliza un modelo asíncrono y dirigido por eventos.

Es una Máquina Virtual tremendamente rápida y de gran calidad escrita por gente como Lars Bak, uno de los mejores ingenieros del mundo especializados en VMs (Virtual Machines). No olvidemos que V8 es actualizado constantemente y es uno de los intérpretes más rápidos que puedan existir en la actualidad para cualquier lenguaje dinámico. Además, las capacidades de Node.js para I/O (Entrada/Salida) son realmente ligeras y potentes, dando al desarrollador la posibilidad de utilizar a tope la I/O del sistema.

Node soporta protocolos TCP, DNS y HTTP. Y fue creado por Ryan Dahl en 2009 y su evolución está apadrinada por la empresa Joyent que además tiene contratado a Dahl.

Uno de los puntos fuertes de Node.js es su capacidad de mantener muchas conexiones abiertas y esperando.

En la imagen podemos observar como se comporta Nodejs dentro de nuestro sistema.



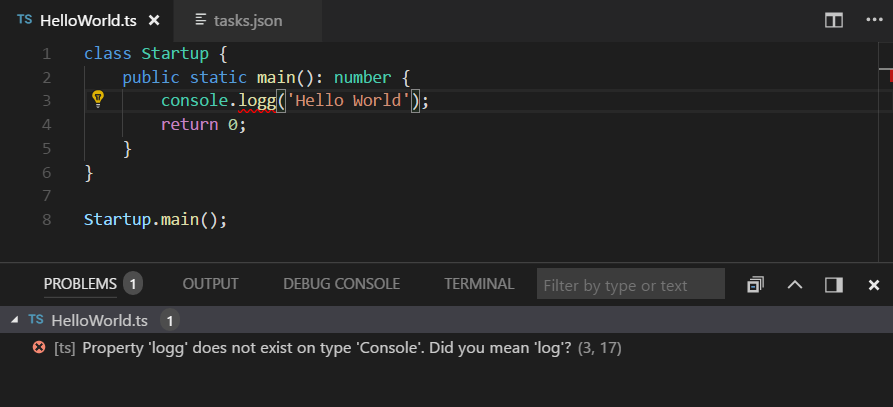
**¿Qué es TypeScript?**

TypeScript es un lenguaje de programación de código abierto desarrollado por Microsoft, el cual cuenta con herramientas de programación orientada a objetos, muy favorable si se tienen proyectos grandes. Anders Hejlsberg, arquitecto principal del desarrollo del lenguaje de programación C#, es el principal participante en el desarrollo de este lenguaje.

TypeScript convierte su código en Javascript común. Es llamado también Superset de Javascript, lo que significa que, si el navegador está basado en Javascript, este nunca llegará a saber que el código original fue realizado con TypeScript y ejecutará el Javascript como lenguaje original.

Actualmente Angular 2, uno de los frameworks más famosos de JavaScript, está siendo desarrollando en TypeScript, para lo cual conocer este lenguaje será fundamental para entender y darle un mejor uso a la nueva versión de Angular.

En este ejemplo se puede observar código hecho en Type Script.



**¿Qué es Angular?**

Angular.js es un proyecto de código abierto, realizado en JavaScript que contiene un conjunto de librerias útiles para el desarrollo de aplicaciones web y propone una seria de patrones de diseño para llevarlas a cabo. En pocas palabras, es lo que se conoce como un framework estructural orientado a la lógica para el desarrollo, en este caso sobre el lenguaje JavaScript con programación del lado del cliente.

Este framework se utiliza enfocado en el desarrollo del Web Front End, especialmente de Single Page Applications (SPA).

**SPA**

SPA son aplicaciones web creadas sobre una sola página que muta mediante interacciones JavaScript.

Podemos decir también que una Single Page Application o SPA, es un sitio web que recarga y muestra su contenido en respuesta a acciones propias de la navegación (enviar un formulario, clickar en un enlace, acceder a una sección interna…) sin tener que realizar peticiones al servidor para volver a cargar más código HTML. Las aplicaciones web básicas lo hacían por completo, AJAX lo intentaba resolver parcialmente y ahora SPA intenta reducirlo al mínimo necesario.

Características de un SPA

* Un punto de entrada único, un que se va transformando y adaptando mediante acciones.
* Como en el caso de una aplicación de escritorio, nos mantenemos en un “marco único” y fijo, mientras que “vistas dinámicas” van ofreciéndonos las distintas posibilidades del uso y navegación.
* Es posible que la dirección URL sufra cambios en base a las actividades de uso de la plataforma y vaya modificándose, aunque ese “marco único” se mantenga fijado. Esto es un tanto reduccionista (existen SPA que no transforman sus direcciones), pero es útil para comprender su mecánica.
* Las peticiones cliente — servidor tienden a ser más laxas y más livianas que en el caso de una plataforma web al uso. Datos, solo datos. Y además muchos procesos quedan del lado del navegador web del cliente a partir de diversas herramientas (como LocalStorage, por ejemplo). Datos y preferentemente en formato JSON.

Ejemplos de aplicaciones que usan SPA:

Gmail Netflix



Trello GitHub

**CLI Angular**

Angular tiene una herramienta muy interesante que por medios de comando nos permite crear varias partes de nuestro proyecto. Es a través de la herramienta angular cli (command line interface).

Los requerimientos de la herramienta son tener instalado node.js y npm

Instalación

Para instalar angular cli vamos a correr el siguiente comando en nuestra terminal de node:

npm install -g angular-cli

Para crear un nuevo proyecto a través de esta herramienta y escribiendo un solo comando, lo hacemos así.

ng new nombre-del-proyect

Para correr la aplicación vamos al directorio de nuestra aplicación, por ejemplo, así.

cd mi-proyecto-angular

Para conectarnos al servidor local escribimos el comando.

ng serve

**Comandos ng Angular**

Para verificar la instalación de angular.

ng –version

Para ver la instrucciones de los comandos ng.

ng help

Para crear un nuevo proyecto a través de esta herramienta y escribiendo un solo comando, lo hacemos así.

ng new nombre-del-proyect

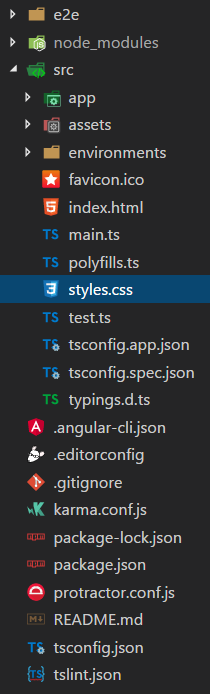
Para conectarnos al servidor local escribimos el comando.

ng serve

**Estilos Angular**

Para agregar estilos a angular debemos hacer lo siguiente:

Después de la creación del proyecto, veremos la siguiente estructura en nuestro editor de texto favorito.



También crearemos dos componentes llamados card-one y card-two de la siguiente forma: ng generate component card-one y ng generate component card-two.

Luego, en styles.css (que es el archivo de los estilos globales de la aplicación), escribiremos lo siguiente:

body {

color: #11bbbb;

display: flex;

font-family: 'Kirang Haerang', cursive;

font-size: 80px;

text-align: center;

}

. card {

border: 10px solid #11bbbb;

box-shadow: 0 10px 20px rgba(0,0,0,0.19),

0 6px 6px rgba(0,0,0,0.23);

padding: 10px 30px;

}

Y en el componente global (que en este caso es AppComponent) importamos los componentes app-card-one y app-card-two, así:

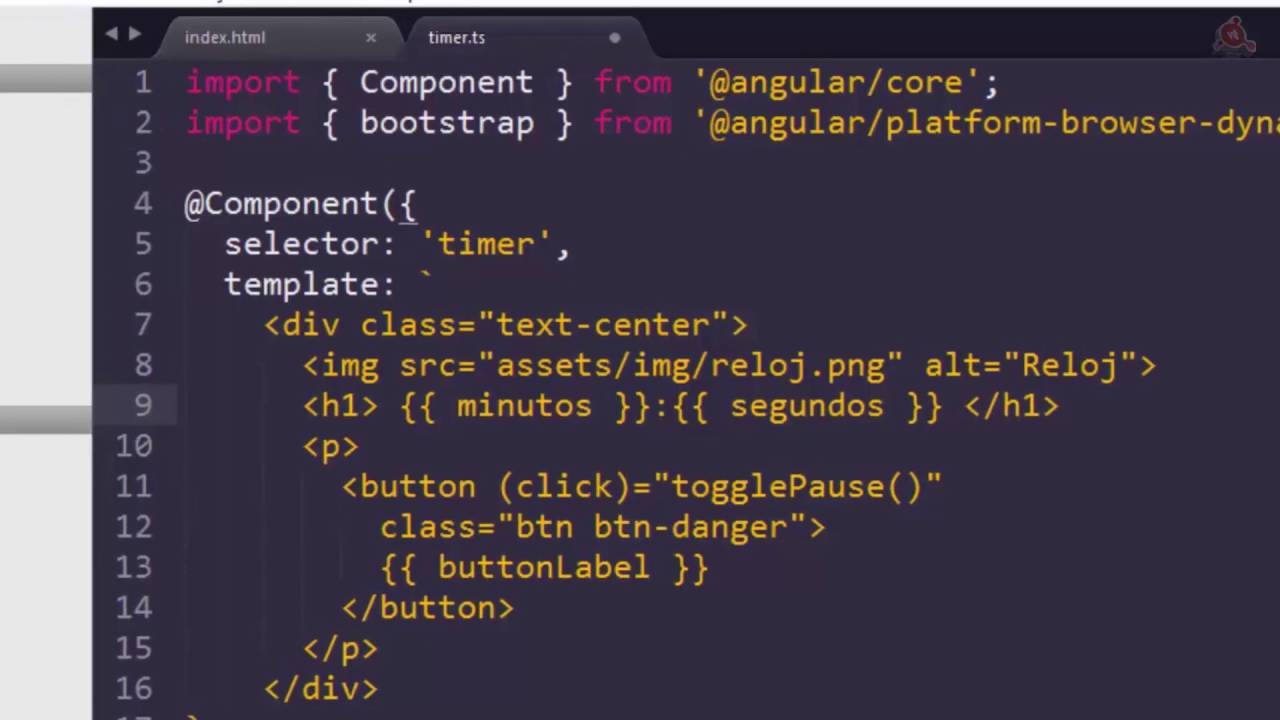
import { Component } from '@angular/core';@Component({  
 selector: 'app-root',  
 template: `  
 <div class="card">  
 <app-card-one></app-card-one>  
 <app-card-two></app-card-two>  
 </div>  
 `,  
})export class AppComponent {   
 constructor() { }  
}

**Componentes Angular**

Para la creación de un nuevo componente en angular solo hay que ejecutar el comando.

ng g c “nombre del componente”

Así se vería un componente en angular.



**Directivas Angular**

El concepto de directivas tiene qué ver con la modificación directa del DOM de nuestra página html siendo una de las principales características de Angular.

Existen 3 tipos de directivas:

* **Componentes**: Corresponden a las etiquetas HTML que se usan para gestionar el componente, donde ubicamos estas etiquetas es donde se renderizará el html como tal, por ejemplo en el ejercicio anterior el componente “footer” hace uso de la directiva

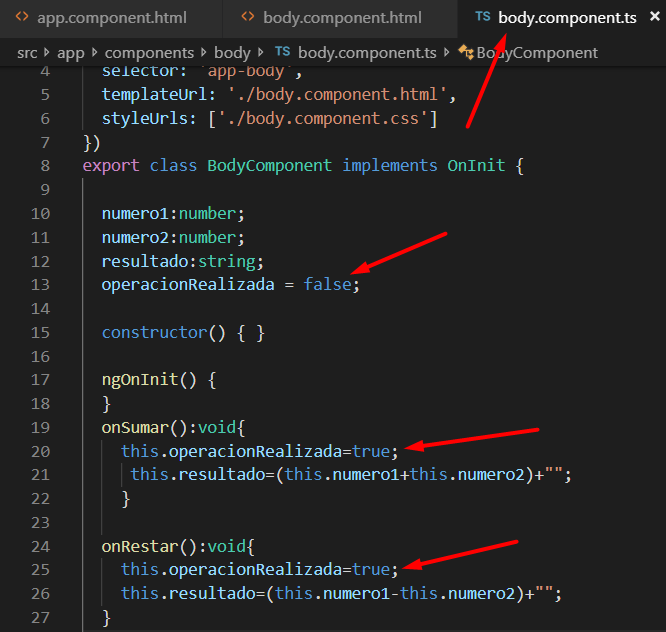
**<app-footer></app-footer>**

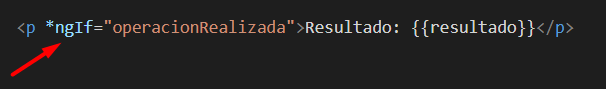
* **De atributos:** Estas directivas modifican el comportamiento de un elemento HTML como si se hiciera uso de un atributo propio de HTML, se identifican por tener el prefijo **ng**
* **Estructurales:** Estas directivas modifican el layout del elemento HTML en el que se asocian, permiten agregar, eliminar u ocultar elementos en el DOM, se identifican por tener un **\*** antes del prefijo **ng**

## Directiva ngIf.

Este tipo de directiva corresponde al grupo de las estructurales y permite hacer uso de la estructura if desde el HTML de nuestra aplicación, permitiendo que un elemento pueda ser mostrado o no dependiendo de la función que queramos darle.

Ejemplo:

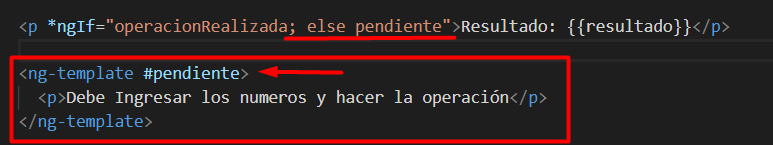




## Directiva ng template.

Este corresponde básicamente a la opción de else asociada a un if, nos permitirá definir lo que queremos mostrar en caso de qué la propiedad del ngIf sea falsa.

Ejemplo:



## Directiva ngFor.

Esta es una directiva estructural y es una de las más usadas en Angular, permite realizar iteraciones y presentar listas de elementos en pantalla, ngFor corresponde a un ciclo en el qué podemos iterar un arreglo desde la vista HTML.

Ejemplo:

