

Informe de Laboratorio 09

Tema: Angular

Nota

Estudiante	Escuela	Asignatura
Luis Guillermo Luque Condori, Fernando Miguel Garambel Marín, William Herderson Choquehuanca Berna, Jeans Anthony Ajra Huacso lluquecon@unsa.edu.pe fgarambel@unsa.edu.pe jajra@unsa.edu.pe wchoquehuancab@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Laboratorio de P.Web Semestre: III

Laboratorio	Tema	Duración
09	Angular	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 18 de junio de 2024	Al 22 de junio de 2024

1. REPOSITORIO GitHub

- <https://github.com/nuevo637/Lab-09-Angular>

2. Videos

- <https://flip.com/s/cK4x1VbL49nw>
- <https://flip.com/s/ch62CS-ZBK3q>
- <https://flip.com/s/qoMBkftQRS6v>
- <https://flip.com/s/FZFjYhEbH6P2>

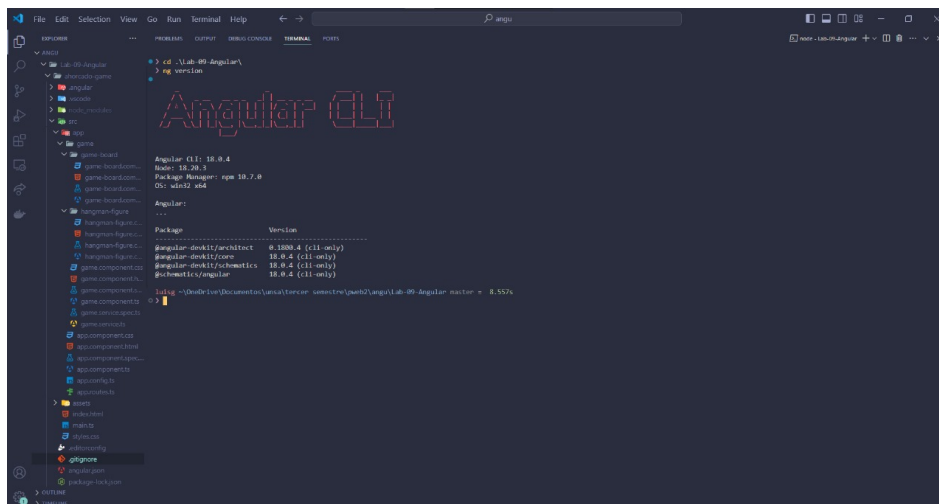
3. Introducción Angular

- Primero tenemos que tener una version de TypeScript de la 17 en adelante para poder instalar Angular.
- Luego usamos los siguientes comandos para instalarlo y verificar la instalación.

Listing 1: Comandos para instalar Angular

```
$ npm install -g @angular/cli
$ ng version
```

- si nos sale la siguiente imagen angular estará instalado



- para crear un proyecto en angular usamos la siguiente linea de comandos.

Listing 2: Crear Proyecto en Angular

```
$ ng new ahorcado-game
```

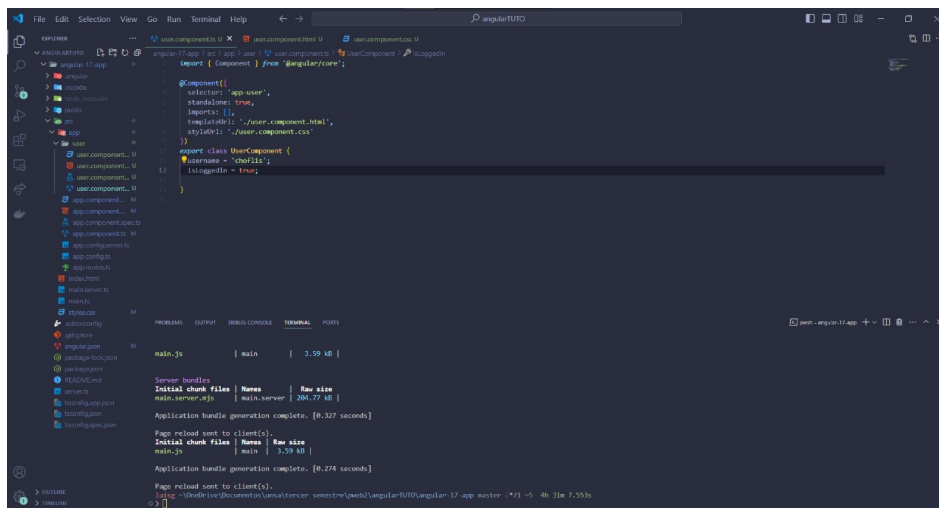
- Luego observaremos un archivo padre que seria nuestra app donde tendremos estilos que son genericos, html y un archivo de componentes que nos ayudaran a juntar mas componentes después.

- Los componentes nos ayudan a dar valores y lógica a nuestra página web.
- Ahora para crear un componente usaremos la siguiente línea de comandos.

Listing 3: Crear Componentes

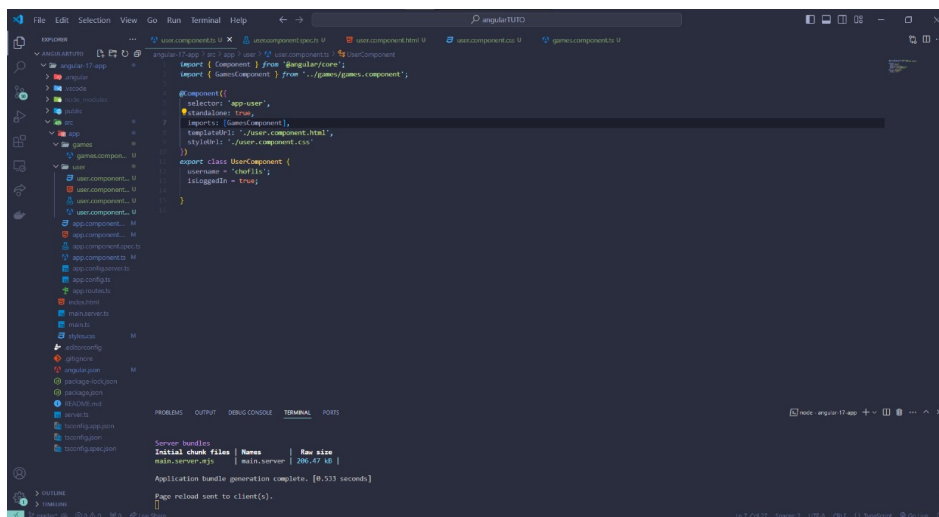
```
$ ng generate component game-board
$ ng generate component hangman-figure
```

- Esto nos ayuda para poder trabajar por partes y no tenerlo todo en un solo código además al momento de cambiar estilos es más fácil.
- En nuestro caso usamos 2 componentes uno para la lógica y otro para crear el muñeco de ahorcado.



The screenshot shows a VS Code editor with a file explorer on the left displaying a project structure with various components. The main editor shows the code for a component, and the terminal at the bottom displays the output of the Angular CLI command, including the generation of the component and the bundling of the application.

- Luego probaremos el servidor que crea angular, para que funcione un componente hijo en un componente padre este tiene que ser importado en el componente padre.



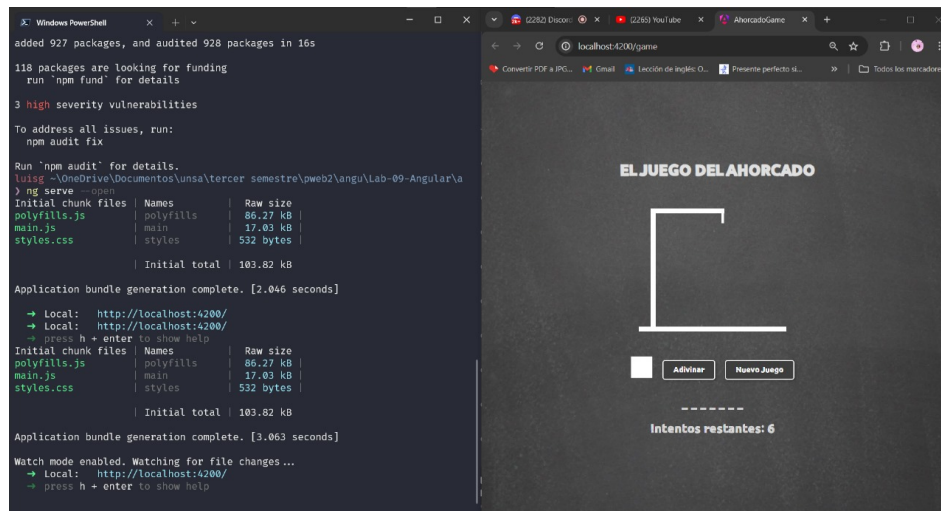
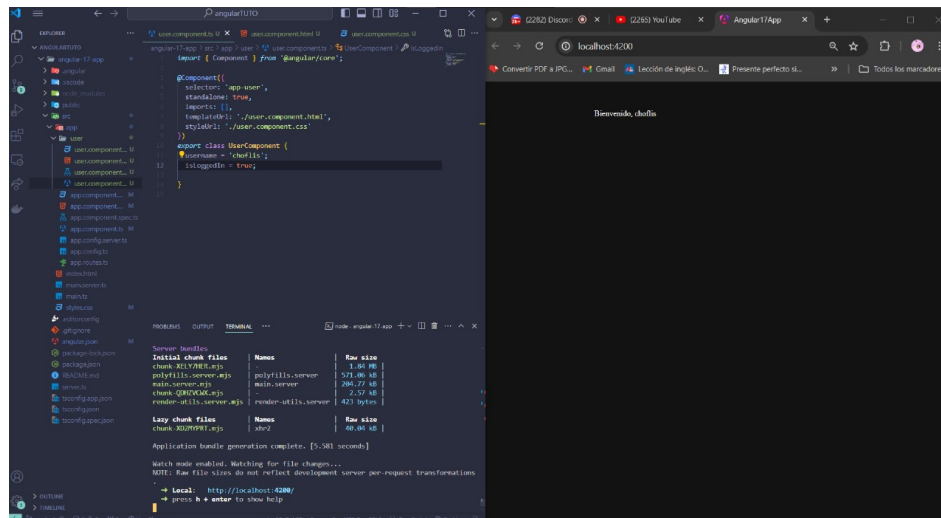
The screenshot shows the VS Code editor with the component code updated to import the 'GameComponent' from the 'game-board' module. The terminal output shows the successful generation of the component and the bundling of the application.

- ahora el comando para correr el servidor.

Listing 4: Comandos del servidor

```
$ ng serve
$ ng serve --open
```

- Ahora podremos ver las modificaciones que hicimos



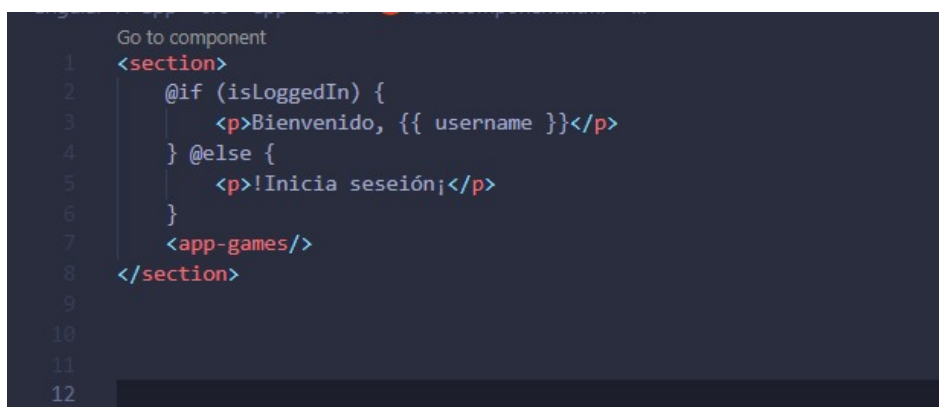
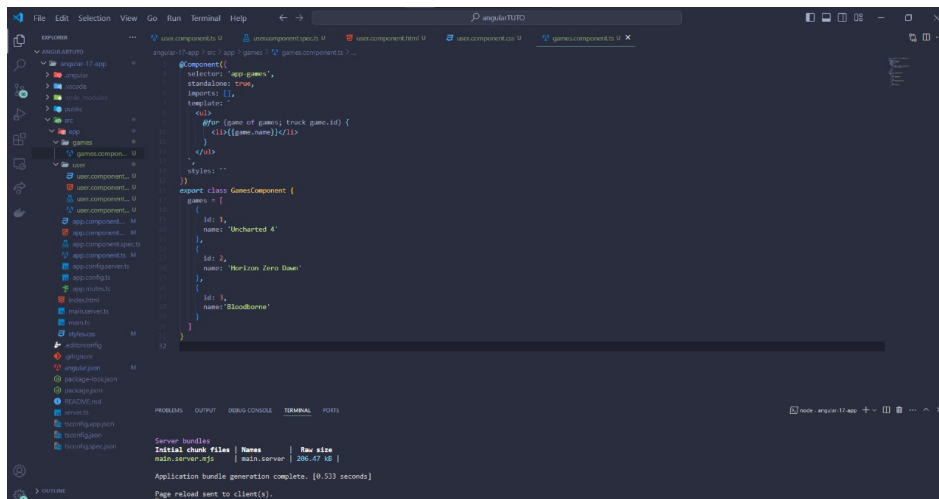
- Inicialmente nos sale una página que nos da la bienvenida, pero como hicimos modificaciones nos aparece las páginas creadas.
- Podemos usar los componentes que necesitamos, en caso no requieramos de uno, usamos la siguiente linea de comandos.

```
Creates a new, generic component definition in the given project.

Arguments:
  name The name of the component. [string]

Options:
  --help Shows a help message for this command in the console. [boolean] [default: true]
  -i, --interactive Enable interactive input prompts. [boolean] [default: false]
  -f, --dry-run Run through and reports activity without writing out results. [boolean] [default: false]
  --defaults Disable interactive input prompts for options with a default. [boolean] [default: false]
  --force Force overwriting of existing files. [boolean] [default: false]
  -t, --change-detection The change detection strategy to use in the new component. [string] [choices: "Default", "OnPush"] [default: "Default"]
  -b, --display-block Specifies if the style will contain ':host { display: block; }'. [boolean] [default: false]
  --export The declaring NgModule exports this component. [boolean] [default: false]
  --flat Create the new files at the top level of the current project. [boolean] [default: false]
  -s, --inline-style Include styles inline in the component.ts file. Only CSS styles can be included inline. By default, an external styles file is created and referenced in the component.ts file. [boolean] [default: false]
  -t, --inline-template Include template inline in the component.ts file. By default, an external template file is created and referenced in the component.ts file. [boolean] [default: false]
  -m, --module The declaring NgModule. [string] [default: false]
  -p, --prefix The prefix to apply to the generated component selector. [string] [default: false]
  --project The name of the project. [string] [default: false]
  --selector The HTML selector to use for this component. [string] [default: false]
  --skip-import Do not import this component into the outgoing NgModule. [boolean] [default: false]
  --skip-selector Specifies if the component should have a selector or not. [boolean] [default: false]
  --skip-tests Do not create "spec.ts" test files for the new component. [boolean] [default: false]
  --standalone Whether the generated component is standalone. [boolean] [default: true]
  --style The file extension or preprocessor to use for style files, or 'none' to skip generating the style file. [string] [choices: "css", "scss", "sass", "less", "none"] [default: "css"]
  --type Adds a developer-defined type to the filename, in the format "name.type.ts". [string] [default: "Component"]
  -v, --view-encapsulation The view encapsulation strategy to use in the new component. [string] [choices: "Emulated", "None", "ShadowDom"] [default: "Emulated"]
```

- Ahora para las funciones de for o de if en angular tiene una sintaxis interesante ya que funcionan de la siguiente manera "@for." "@if".



- Algo interesante de la ultima imagen es que para que un componente creado este en otro componente en el html tenemos que poner una referencia a ese componente en este caso la referencia es app-games.
- Por último podemos hacer que ocurran acciones como cuando damos un click o cosas asi, para esto necesitamos usar (click) dentro de nuestro código.

```
angular-17-app / src / app / user / user.component.html / ...
Go to component
1 <section>
2   @if (isLoggedIn) {
3     <p>Bienvenido, {{ username }}</p>
4     
7   } @else {
8     <p>!Inicia sesión</p>
9   }
10   <app-games/>
11 </section>
12
13
14
15
```

4. Ahorcado-Game

- A continuación se presenta la parte más importante del código del juego Ahorcado-game.
- Primero se maneja la lógica central del juego del ahorcado. Esto incluye seleccionar una palabra aleatoria, llevar un registro de los intentos restantes, actualizar la palabra oculta según las letras adivinadas, y emitir eventos cuando cambian los intentos. Además, proporciona métodos para iniciar un nuevo juego y verificar si el juego ha terminado.

```
1 import { Injectable } from '@angular/core';
2 import { Subject } from 'rxjs';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class GameService {
8   palabras: string[] = [
9     'angular',
10    'typescript',
11    'componente',
12    'servicio',
13    'reactivo',
14    'eula',
15    'framework',
16    'biblioteca',
17    'módulo',
18    'dependencia',
19    'paquete',
20    'node.js',
21    'express',
22    'mongodb',
23    'sql',
24    'api',
25    'rest',
26    'graphql',
27    'json',
28    'ajax',
29    'http',
30    'genshin',
31    'frontend',
32    'backend',
33    'base de datos',
34    'honkai',
35    'consulta',
36    'wuthering',
37    'pruebas',
38    'waves'
39  ];
40
41   palabraActual: string = '';
42   palabraOculta: string[] = [];
43   intentos: number = 6;
44
45   // Subject para emitir cambios en los intentos
46   intentosCambio = new Subject<number>();
47
48   constructor() {
49     this.nuevaPalabra();
50   }
51
52   nuevaPalabra() {
53     this.palabraActual = this.palabras[Math.floor(Math.random() * this.palabras.length)];
54     this.palabraOculta = Array(this.palabraActual.length).fill('_');
55     this.intentos = 6;
56     this.intentosCambio.next(this.intentos);
57   }
58
59   adivinar(letra: string) {
60     let acierto = false;
61     for (let i = 0; i < this.palabraActual.length; i++) {
62       if (this.palabraActual[i] === letra) {
63         this.palabraOculta[i] = letra;
64         acierto = true;
65       }
66     }
67     if (!acierto) {
68       this.intentos--;
69       this.intentosCambio.next(this.intentos);
70     }
71   }
72
73   juegoTerminado(): boolean {
74     return this.intentos <= 0 || this.palabraOculta.join('') === this.palabraActual;
75   }
76 }
```

- El siguiente código se encarga de mostrar las partes del cuerpo del muñeco del ahorcado en función de los intentos restantes. Se suscribe a los cambios en el número de intentos a través del GameService y actualiza la visibilidad de las partes del cuerpo en consecuencia. Utiliza clases CSS (hidden y visible) para controlar la visibilidad de los elementos en el DOM y ajusta el color de fondo para asegurarse de que las partes visibles tengan el color correcto.

```

1  import { Component, OnInit } from '@angular/core';
2  import { GameService } from '../game.service';
3
4  @Component({
5    selector: 'hangman-figure',
6    standalone: true,
7    imports: [],
8    templateUrl: './hangman-figure.component.html',
9    styleUrls: ['./hangman-figure.component.css']
10 })
11
12 export class HangmanFigureComponent implements OnInit {
13   partesCuerpo = ['head', 'body', 'left-arm', 'right-arm', 'left-leg', 'right-leg'];
14
15   constructor(public gameService: GameService) {}
16
17   ngOnInit() {
18     this.gameService.intentosCambio.subscribe((intentosRestantes) => {
19       this.actualizarCuerpo(intentosRestantes);
20     });
21   }
22
23   actualizarCuerpo(intentosRestantes: number) {
24     const partesAVisible = this.partesCuerpo.slice(0, 6 - intentosRestantes);
25     this.partesCuerpo.forEach(parte => {
26       const elemento = document.querySelector(`.body-parts .${parte}`);
27       if (elemento) {
28         if (partesAVisible.includes(parte)) {
29           elemento.classList.remove('hidden');
30           elemento.classList.add('visible');
31           (elemento as HTMLElement).style.backgroundColor = 'ffffff';
32         } else {
33           elemento.classList.remove('visible');
34           elemento.classList.add('hidden');
35           (elemento as HTMLElement).style.backgroundColor = 'ffffff00';
36         }
37       }
38     });
39   }
40 }

```

- Por ultimo en cuanto al gameboard, el siguiente codigo es responsable de manejar la interfaz del juego del ahorcado. Proporciona un cuadro de entrada para que el usuario introduzca letras y botones para adivinar letras o iniciar un nuevo juego. Utiliza el GameService para manejar la lógica del juego, como verificar las letras adivinadas y generar nuevas palabras.

```
1 import { Component } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { FormsModule } from '@angular/forms';
4 import { GameService } from '../game.service';
5
6 @Component({
7   selector: 'app-game-board',
8   standalone: true,
9   imports: [CommonModule, FormsModule],
10  templateUrl: './game-board.component.html',
11  styleUrls: ['./game-board.component.css']
12 })
13 export class GameBoardComponent {
14   letra: string = '';
15
16   constructor(public gameService: GameService) {}
17
18   adivinar(letra: string) {
19     if (letra.length === 1) {
20       this.gameService.adivinar(letra);
21     }
22     this.letra = '';
23   }
24
25   nuevaPalabra() {
26     this.gameService.nuevaPalabra();
27   }
28 }
29
```


5. Rúbricas

5.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

6. Referencias

- Sobre Angular
- <https://docs.angular.lat/docs>
- <https://v2.angular.io/docs/ts/latest/guide/>
- <https://angular.dev/>
- <https://angular.dev/tutorials/learn-angular>