

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2025/05/03</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 1</p>

## INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	Programacion Web 2				
TÍTULO DE LA PRÁCTICA:	Laboratorio 9 - Angular				
NÚMERO DE PRÁCTICA:	9	AÑO LECTIVO:	2025	NRO. SEMESTRE:	1
FECHA DE PRESENTACIÓN	29/06/2025	Repositorio	<a href="https://github.com/JesusFSP/Curso-PWeb2.git">https://github.com/JesusFSP/Curso-PWeb2.git</a>		
INTEGRANTE (s): Silva Pino Jesus Francisco				NOTA:	
DOCENTE(s): CARLO JOSE LUIS CORRALES DELGADO					

SOLUCIÓN Y RESULTADOS
<p>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</p> <h2>Implementación del Juego del Ahorcado en Angular</h2> <h3>1. Servicio del Juego (game.service.ts)</h3> <pre>import { Injectable } from '@angular/core';  @Injectable({ providedIn: 'root' }) export class GameService {   private words = ['ANGULAR', 'TYPESCRIPT', 'COMPONENT', 'SERVICE']; // Palabras a adivinar   selectedWord = '';</pre>

	<p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y</b>  <b>SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2025/05/03</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 2</p>

```

guessedLetters: string[] = [];
attemptsLeft = 6;

constructor() {
  this.newGame();
}

newGame() {
  this.selectedWord = this.words[Math.floor(Math.random() * this.words.length)];
  this.guessedLetters = [];
  this.attemptsLeft = 6;
}

guessLetter(letter: string) {
  if (!this.guessedLetters.includes(letter) && this.attemptsLeft > 0 && !this.isGameWon()) {
    this.guessedLetters.push(letter);
    if (!this.selectedWord.includes(letter)) {
      this.attemptsLeft--;
    }
  }
}

getWordDisplay() {
  return this.selectedWord
    .split("")
    .map(char => this.guessedLetters.includes(char) ? char : '_')
    .join(' ');
}

isGameWon(): boolean {
  return this.selectedWord.split("").every(char => this.guessedLetters.includes(char));
}

```

	<p align="center"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y</b>  <b>SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p align="center"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2025/05/03</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p align="right"><b>Página:</b> 3</p>

```
}
```

```
isGameLost(): boolean {
  return this.attemptsLeft <= 0;
}
}
```



- **Lógica clave:**

- newGame(): Selecciona una palabra aleatoria y reinicia el estado del juego.
- guessLetter(): Valida y procesa la letra ingresada, actualizando los intentos y las letras adivinadas.
- getWordDisplay(): Genera la representación de la palabra con guiones bajos para las letras no adivinadas (ej: \_ N G U \_ \_ R).
- isGameWon() / isGameLost(): Métodos booleanos que determinan el estado final del juego.

## 2. Componente Principal (game-board.component.ts)

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { GameService } from '../game.service';
import { HangmanDisplayComponent } from '../hangman-display/hangman-display.component';
import { KeyboardComponent } from '../keyboard/keyboard.component';
import { GameStatusComponent } from '../game-status/game-status.component';
```

```
@Component({
  selector: 'app-game-board',
  standalone: true,
  imports: [CommonModule, HangmanDisplayComponent, KeyboardComponent,
GameStatusComponent],
  template: `
    <h1>Juego del Ahorcado</h1>
    <app-hangman-display [attemptsLeft]="gameService.attemptsLeft"></app-hangman-display>
```

	<p align="center"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y</b>  <b>SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p align="center"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2025/05/03</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p align="right"><b>Página:</b> 4</p>

```

<h2>{{ gameService.getWordDisplay() }}</h2>
<app-game-status></app-game-status>
<app-keyboard
  (letterSelected)="gameService.guessLetter($event)"
  [disabledLetters]="gameService.guessedLetters">
</app-keyboard>
,
})
export class GameBoardComponent {
  constructor(public gameService: GameService) {}
}

```

- **Flujo de datos:**

- Pasa attemptsLeft del servicio al componente HangmanDisplay usando property binding ([]).
- Escucha el evento (letterSelected) del KeyboardComponent para enviar la letra seleccionada al servicio.
- Envía la lista de guessedLetters al KeyboardComponent para deshabilitar las teclas correspondientes.

### 3. Componente del Teclado (keyboard.component.ts)

```

import { Component, Input, Output, EventEmitter } from '@angular/core';
import { CommonModule } from '@angular/common';

```

```

@Component({
  selector: 'app-keyboard',
  standalone: true,
  imports: [CommonModule],
  template: `
    <div class="keyboard-container">
      <button
        *ngFor="let letter of letters"

```

```

(click)="selectLetter(letter)"
[disabled]="disabledLetters.includes(letter)">
{{ letter }}
</button>
</div>
,
})
export class KeyboardComponent {
  @Input() disabledLetters: string[] = [];
  @Output() letterSelected = new EventEmitter<string>();

  letters = 'ABCDEFGHIIJKLMNOPQRSTUVWXYZ'.split('');

  selectLetter(letter: string) {
    this.letterSelected.emit(letter);
  }
}

```

- **Funcionamiento:**

- Renderiza un botón para cada letra del abecedario usando la directiva \*ngFor.
- Emite la letra seleccionada al componente padre (GameBoardComponent) a través del @Output() letterSelected.
- Recibe las letras usadas a través del @Input() disabledLetters y deshabilita los botones correspondientes con [disabled].



## 4. Componente del Ahorcado (hangman-display.component.ts)

```

import { Component, Input } from '@angular/core';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-hangman-display',

```

	<p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y</b>  <b>SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2025/05/03</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 6</p>

```
standalone: true,
imports: [CommonModule],
template: `
<svg viewBox="0 0 200 250" class="hangman-svg">
  <!-- Patíbulo -->
  <line x1="20" y1="230" x2="100" y2="230" /> <!-- Base -->
  <line x1="60" y1="230" x2="60" y2="20" /> <!-- Poste -->
  <line x1="60" y1="20" x2="150" y2="20" /> <!-- Viga -->
  <line x1="150" y1="20" x2="150" y2="50" /> <!-- Cuerda -->

  <!-- Cuerpo del ahorcado -->
  <circle *ngIf="attemptsLeft < 6" cx="150" cy="70" r="20"/> <!-- Cabeza -->
  <line *ngIf="attemptsLeft < 5" x1="150" y1="90" x2="150" y2="150"/> <!-- Torso -->
  <line *ngIf="attemptsLeft < 4" x1="150" y1="110" x2="120" y2="140"/> <!-- Brazo Izquierdo -->
  <line *ngIf="attemptsLeft < 3" x1="150" y1="110" x2="180" y2="140"/> <!-- Brazo Derecho -->
  <line *ngIf="attemptsLeft < 2" x1="150" y1="150" x2="120" y2="180"/> <!-- Pierna Izquierda -->
</svg>
`
,
})
export class HangmanDisplayComponent {
  @Input() attemptsLeft!: number;
}
```

- **Visualización:**

- Utiliza un gráfico SVG para dibujar las partes del patíbulo y del ahorcado.
- La directiva \*ngIf se encarga de renderizar cada parte del cuerpo condicionalmente, basándose en el número de attemptsLeft. Cada error revela una nueva parte.

## 5. Componente de Estado (game-status.component.ts)

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

	<p align="center"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y</b>  <b>SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p align="center"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2025/05/03</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p align="right"><b>Página:</b> 7</p>

```
import { CommonModule } from '@angular/common';
import { GameService } from '../game.service';

@Component({
  selector: 'app-game-status',
  standalone: true,
  imports: [CommonModule],
  template: `
    <div *ngIf="gameService.isGameLost() || gameService.isGameWon()" class="status-
container">
      <div *ngIf="gameService.isGameWon()" class="win-message">¡Felicidades, ganaste!</div>
      <div *ngIf="gameService.isGameLost()" class="lose-message">Perdiste. La palabra era:
{{ gameService.selectedWord }}</div>
      <button (click)="gameService.newGame()">Jugar de Nuevo</button>
    </div>
  `
})
export class GameStatusComponent {
  constructor(public gameService: GameService) {}
}
```

- **Mensajes contextuales:**

- Muestra el mensaje de victoria o derrota inyectando el GameService y llamando a sus métodos isGameWon() y isGameLost() dentro de directivas \*ngIf.
- El botón "Jugar de Nuevo" invoca directamente el método newGame() del servicio para reiniciar la partida.

## Diagrama de Flujo de Datos

```
sequenceDiagram
    participant Usuario
    participant KeyboardComponent
    participant GameBoardComponent
    participant GameService
```

participant HangmanDisplay

participant GameStatus

Usuario->>KeyboardComponent: Clic en letra "A"

KeyboardComponent->>GameBoardComponent: emite letterSelected("A")

GameBoardComponent->>GameService: llama a guessLetter("A")

GameService->>GameService: Actualiza estado (guessedLetters, attemptsLeft)

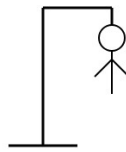
GameService-->>GameBoardComponent: Estado actualizado disponible

GameBoardComponent-->>HangmanDisplay: [attemptsLeft] se actualiza

GameBoardComponent-->>KeyboardComponent: [disabledLetters] se actualiza

GameBoardComponent-->>GameStatus: El estado se refleja a través del servicio

### Juego del Ahorcado



A N G U L A R

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

¡Felicidades! ¡Ganaste!

Nuevo Juego



	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2025/05/03</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 9</p>

## II. CONCLUSIONES

El código sigue una arquitectura reactiva y modular con una clara separación de responsabilidades:

1. **Servicio Centralizado (GameService):** Actúa como la única fuente de verdad (Single Source of Truth) para el estado del juego.
2. **Componentes "Dumb":** Componentes como Keyboard y HangmanDisplay son principalmente de presentación. Reciben datos a través de @Input y notifican eventos a través de @Output.
3. **Componente "Smart" (GameBoardComponent):** Orquesta la comunicación entre el servicio y los componentes hijos, manteniendo la lógica de la aplicación fuera de la vista.

**Inyección de Dependencias:** Angular gestiona la instancia del GameService, facilitando un código limpio y comprobable.

## RETROALIMENTACIÓN GENERAL

## REFERENCIAS Y BIBLIOGRAFÍA