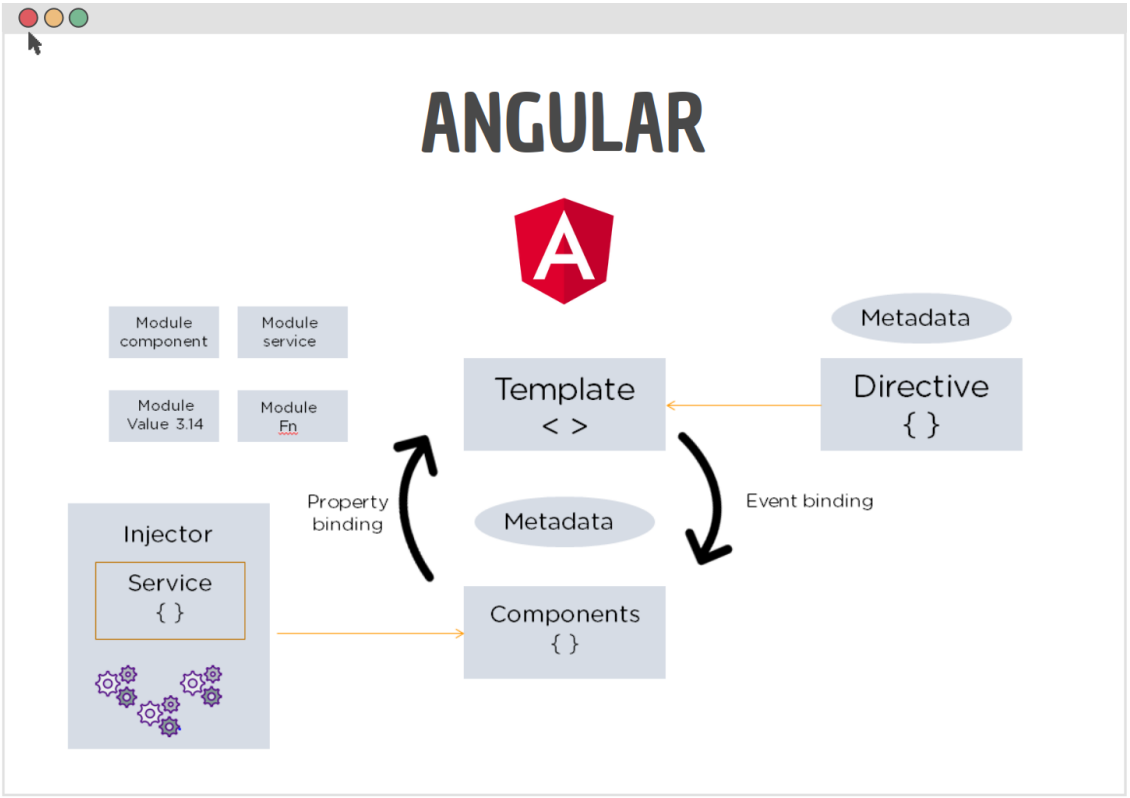
PROGRAMACIÓN WEB 2



Profesor(a):
Carlo Jose Luis Corrales Delgado

Estudiantes:
Armando Steven Cuno Cahuari
Victor Narciso Mamani Anahua
Jorge Luis Mamani Huarsaya

Repositorio GitHub:
https://github.com/jorghee/angular-hangman-s-game

21 de junio, 2024

Lo primero que hacemos es modificar la plantilla HTML del componente raiz de nuestra aplicación *src/app/app.component.html*. Por el momento simplemente queremos renderizar la plantilla HTML de los componentes hijos que vamos a definir: `game` y `hangman-drawing`.

```
<app-game></app-game>
```

Este es el bloque en el cual Angular inyectará el componente `game`. Luego de hacer esto, nosotros debemos de comunicar a Angular de la existencia de este nuevo component y por lo tanto nos vemos en la necesidad de importarlo.

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { GameComponent } from './game/game.component';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, GameComponent],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'hangmans-game';
}
```

El componente `game`

Como hemos visto anteriormente, este es el componente que usa el componente root de nuestra aplicación. Este componente se encargará de representar la dinámica del juego. **No hemos optado por crear un servicio** ya que el juego es muy simple y con el uso de este componente como base es más que suficiente.

La clase `GameComponent` debe de almacenar varios estados del juego, como son: una lista de las palabras a adivinar, la palabra aleatoriamente elegida, la palabra escondida, las letras adivinadas y por último el cantidad de vida. Por ello definimos estos estados como propiedades de la clase `GameComponent`.

```
import { Component } from '@angular/core';
import { HangmanDrawingComponent } from
↪  '../hangman-drawing/hangman-drawing.component'

@Component({
  selector: 'app-game',
  standalone: true,
  imports: [HangmanDrawingComponent],
  templateUrl: './game.component.html',
  styleUrl: './game.component.css'
})
export class GameComponent {
  words: string[] = ['angular', 'typescript', 'javascript', 'developer'];
  selectedWord: string = '';
  displayWord: string[] = [];
  guessedLetters: string[] = [];
```

```
  lives: number = 6;

  //...
}
```

Ahora necesitaremos una forma de iniciar el juego. Debería de ser un método que se pueda llamar con un evento en el HTML, esto permitirá al usuario volver a jugar el juego una vez que gane o pierde. Además debe de ejecutarse al momento de crear la clase y se debe de encargar de inicializar todas las propiedades definidas anteriormente.

```
constructor() {
  this.startNewGame();
}

startNewGame() {
  this.lives = 6;
  this.guessedLetters = [];
  this.selectedWord = this.words[Math.floor(Math.random() * this.words.length)];
  this.displayWord = Array(this.selectedWord.length).fill('_');
}
```

Tener en cuenta que **mostrar la palabra significa mostrar la palabra pero oculta**, por ello es que rellenamos el arreglo, de la misma longitud de la palabra escogida aleatoriamente, con el caracter _.

Hasta el momento ya tenemos inicializado el juego, ahora necesitamos implementar la lógica del juego que se encargue de hacer las verificaciones. Para ello vamos a crear un método que realizará las siguientes instrucciones:

- Primeramente el método debe de recibir una letra, y debe comprobar si la letra ya fue adivinada anteriormente, por precaución tambien podría verificar que el nivel de vida sea mayor que 0. Si sucede cualquiera de estos casos, el método no debe de hacer nada.
- Si no pasa que la letra ya fue adivinada y que el nivel de vida no es menor igual que 0, entonces debemos agregar dicha letra al arreglo de letras adivinadas.
- Ahora necesitamos verificar si la letra coincide con alguna letras que componen la palabra adivinar. Si este es el caso, entonces necesitamos cambiar el caracter de ocultamiento por las letras coincidentes.
- Si sucede que la letra no coincide con ninguna letra que compone la palabra a adivinar, entonces restamos el nivel de vida.

```
guessLetter(letter: string) {
  if (this.guessedLetters.includes(letter) || this.lives <= 0) return;

  this.guessedLetters.push(letter);

  if (this.selectedWord.includes(letter)) {
    for (let i = 0; i < this.selectedWord.length; i++) {
      if (this.selectedWord[i] === letter) {
        this.displayWord[i] = letter;
      }
    }
  } else {
```

```
    this.lives--;
  }
}
```

Finalmente debemos implementar dos métodos que se encarguen de verificar si se ha perdido el juego o si se ha ganado. Esto debe de hacerse cada vez que se ingrese una letra.

```
isGameOver(): boolean {
  return this.lives <= 0 || !this.displayWord.includes('_');
}

isGameWon(): boolean {
  return !this.displayWord.includes('_');
}
```

Uso de la nueva sintaxis de las directivas en Angular

Ahora que ya hemos completado la lógica del juego, necesitamos definir la plantilla HTML básica que se encargue de mostrar la ejecución del juego.

Para ello vamos a utilizar la funcionalidad potente de Angular. Primeramente podemos colocar un titulo descriptivo del juego, luego estará la representación grafica del juego y por esta necesida se ha creado un nuevo componente que se encargará de generar la representación grafica.

Sin embargo aqui es **importante que este nuevo componente hijo deba recibir la cantidad de vida restante**. Por ello es que estamos obligados a utilizar el concepto de **Paso de información de hijo a padre y viceversa**.

```
<div>
  <h1>Juego del Ahorcado</h1>
  <app-hangman-drawing [lives]="lives"></app-hangman-drawing>
  <p>{{ displayWord.join(' ') }}</p>

  @if (isGameOver()) {
    @if (isGameWon()) {
      <div>
        <p>¡Ganaste!</p>
      </div>
    } @else {
      <div>
        <p>Perdiste. La palabra era: {{ selectedWord }}</p>
        <button (click)="startNewGame()">Jugar de nuevo</button>
      </div>
    }
  } @else {
    <div>
      <p>Vidas restantes: {{ lives }}</p>
      <div>
        @for (letter of 'abcdefghijklmnopqrstuvwxyz'; track letter) {
          <button (click)="guessLetter(letter)">{{ letter }}</button>
        }
      </div>
    </div>
```

```
  }
</div>
```

Como observamos, estamos haciendo uso de los métodos definidos en la clase `GameComponent` y en especial estamos asociando el método `guessLetter(letter)` al evento de cliquear en la etiqueta `button` generada 27 veces.

Estos botones estan simulando como si fuera un teclado en el cual precionamos para introducir algún caracter. Entonces con esto logramos que esta letra que presionemos sea evaluada en el evento.

El componente `hangman-drawing`

Esta clase grafica el juego y cambia dinamicamente. Como mencionamos anteriormente, este componente hijo debe de recibir la cantidad de vida y debe de generar la representación correcta en función a la cantidad de vida.

Para solucionar este problema, hemos decidido crear estilos deferentes y únicos para cada nivel de vida. Por lo tanto al cambiar el nivel de vida, tambien cambian los estilos de la representación.

```
import { Component, Input } from '@angular/core';
import { GameComponent } from '../game/game.component'
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-hangman-drawing',
  standalone: true,
  imports: [GameComponent, CommonModule],
  templateUrl: './hangman-drawing.component.html',
  styleUrl: './hangman-drawing.component.css'
})
export class HangmanDrawingComponent {
  @Input() lives: number = 0;

  getDrawingSteps() {
    const steps = [
      'head',
      'body',
      'left-arm',
      'right-arm',
      'left-leg',
      'right-leg'
    ];
    return steps.slice(0, 6 - this.lives);
  }
}
```

Como observamos lo que retorna es una lista de nombres o partes que estan en función a la disminución del nivel de vida. Para entenderlo mejor debemos irnos a la plantilla HTML de este componente. Aqui definimos la estructura directiva `@for` que generará etiquetas `<div>` y cada una tendra una clase CSS especifica.

```
<div class="hangman">
  @for (step of getDrawingSteps(); track step) {
```

```
    <div [ngClass]="step"></div>
  }
</div>
```

Esta es la lógica del juego del ahorcado, hemos visto cómo es que simplifica la nueva sintaxis de directivas. Para culminar con el proyecto solo falta definir las clases en CSS.

Ejecución del juego del ahorcado

```
$ npm install
$ ng serve
```

Nos dirigimos a nuestro navegador e ingresamos la URL. Luego podemos jugar el juego cuantas veces querramos.