TAREA 2.1: PLANTILLAS DE DESARROLLO

1. Portada

- **Título de la tarea**: Tarea 2.1: Plantillas de Desarrollo
- **Asignatura**: DESARROLLO WEB AVANZADO
- **Nombre del estudiante**: Reyes Ariel y Villarreal Anthony
- **Fecha de entrega**: 14-1-2025
- **Nombre del profesor o docente**: DORIS KARINA CHICAIZA ANGAMARCA
- **Universidad:** Universidad de las Fuerzas Armadas ESPE

2. Objetivo General

El objetivo principal de esta tarea es desarrollar una serie de componentes dentro de un proyecto en Angular para crear una estructura visual dinámica, permitiendo interactuar con diversas plantillas y componentes. Este proyecto busca implementar prácticas comunes de desarrollo web avanzado, utilizando tecnologías como Angular, CSS y HTML para la creación de una interfaz que sea funcional y atractiva.

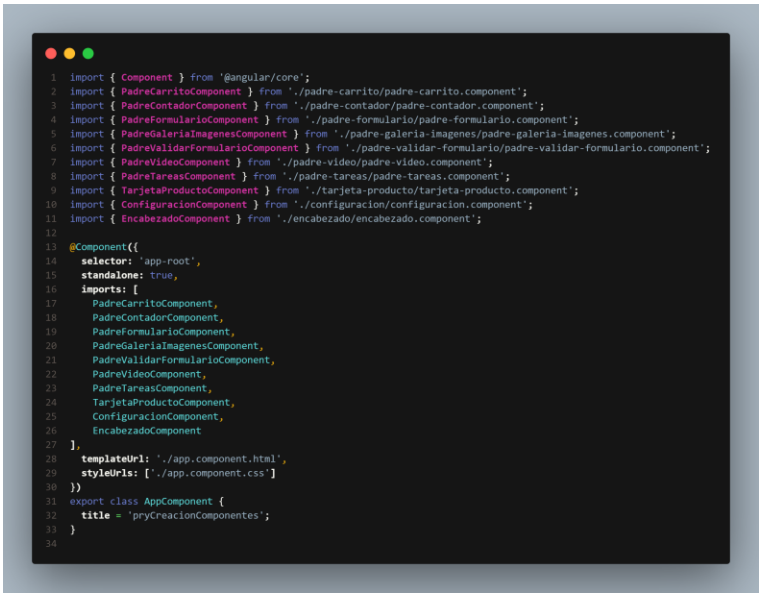
3. Desarrollo

- **Conceptos base**: En esta tarea, el enfoque se centra en el desarrollo de aplicaciones utilizando Angular, un framework para la creación de aplicaciones web dinámicas y eficientes. Los componentes en Angular permiten crear interfaces de usuario modulares y reutilizables, lo cual facilita tanto el mantenimiento como la escalabilidad del proyecto. Durante esta tarea, se abordan aspectos clave como la creación de componentes hijos y padres, la comunicación entre ellos mediante eventos y bindings, y el diseño responsivo para garantizar la accesibilidad desde dispositivos móviles.

  La importancia de este proyecto radica en aplicar conceptos fundamentales de desarrollo web avanzado, como la reutilización de componentes y la interacción de estos con el estado y la vista, lo cual es crucial para el desarrollo de aplicaciones web modernas y mantenibles.

- **Herramientas de desarrollo**:
  - **Angular:** Framework utilizado para crear componentes interactivos y dinámicos en la aplicación.
  - **HTML/CSS:** Lenguajes de marcado y estilo para estructurar y diseñar la interfaz de usuario.
  - **JavaScript/TypeScript:** Lenguaje de programación utilizado para definir la lógica de negocio y la interacción de los componentes en Angular.
  - **Visual Studio Code:** Entorno de desarrollo utilizado para escribir y gestionar el código fuente del proyecto.
  - **GitHub:** Plataforma utilizada para el control de versiones y colaboración entre los miembros del equipo.

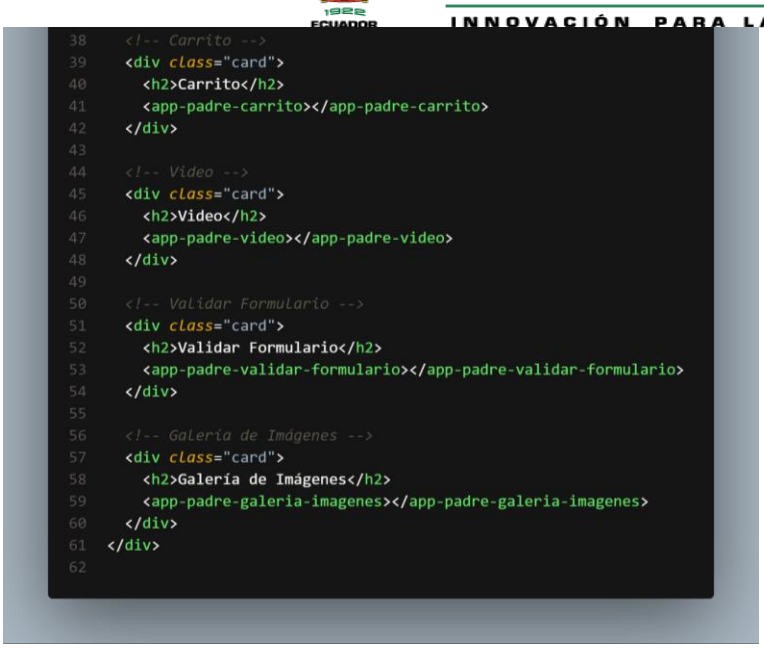
- **Cuerpo o Desarrollo**:
  - **APP.COMPONENT**
    - **Contenido:** El AppComponent actúa como el contenedor principal de la aplicación Angular. Es el punto de entrada donde se reúnen todos los componentes hijos y se estructuran en una interfaz visual cohesiva.
    - **TS (TypeScript)**
      - El archivo app.component.ts define la clase AppComponent, que contiene lo siguiente:
      - **Imports de componentes:**
        - Aquí se importan todos los componentes padres (por ejemplo, PadreCarritoComponent, PadreFormularioComponent, etc.) y otros elementos, como EncabezadoComponent, para utilizarlos en el componente principal.
        - Esto permite incorporar funcionalidades de diferentes secciones de la aplicación.
      - **Decorador @Component:**
        - Define el selector del componente (app-root), que se utiliza en el HTML para incluir este componente.
        - Lista todos los componentes importados en la propiedad imports para declararlos como dependencias que pueden ser utilizadas en la plantilla HTML.
        - Asocia el archivo HTML (app.component.html) y CSS (app.component.css) al componente.
      - **Propiedad title:**
        - Almacena el título de la aplicación ('pryCreacionComponentes'), que podría ser utilizada para personalizar dinámicamente algún elemento visual.



```
import { Component } from '@angular/core';
import { PadreCarritoComponent } from './padre-carrito/padre-carrito.component';
import { PadreContadorComponent } from './padre-contador/padre-contador.component';
import { PadreFormularioComponent } from './padre-formulario/padre-formulario.component';
import { PadreGaleriaImagenesComponent } from './padre-galeria-imagenes/padre-galeria-imagenes.component';
import { PadreValidarFormularioComponent } from './padre-validar-formulario/padre-validar-formulario.component';
import { PadreVideoComponent } from './padre-video/padre-video.component';
import { PadreTareasComponent } from './padre-tareas/padre-tareas.component';
import { TarjetaProductoComponent } from './tarjeta-producto/tarjeta-producto.component';
import { ConfiguracionComponent } from './configuracion/configuracion.component';
import { EncabezadoComponent } from './encabezado/encabezado.component';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [
    PadreCarritoComponent,
    PadreContadorComponent,
    PadreFormularioComponent,
    PadreGaleriaImagenesComponent,
    PadreValidarFormularioComponent,
    PadreVideoComponent,
    PadreTareasComponent,
    TarjetaProductoComponent,
    ConfiguracionComponent,
    EncabezadoComponent
],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'pryCreacionComponentes';
}
```

- **HTML**
  - El archivo app.component.html organiza la interfaz gráfica de la aplicación. Está estructurado en tarjetas (card) para mostrar cada funcionalidad o sección.
  - **Diseño general:**
    - Utiliza un contenedor ( ) con estilo en CSS para centrar y organizar visualmente las tarjetas.
    - Cada tarjeta está dedicada a un componente padre, con un encabezado (<h2>) que indica la funcionalidad que representa.
  - **Componentes hijos incluidos:**
    - Cada componente padre está referenciado dentro de su respectiva tarjeta utilizando su selector (por ejemplo, <app-tarjeta-producto> o <app-padre-tareas>).
    - Esto muestra el contenido y la funcionalidad de cada componente dentro de su tarjeta.

```html
<div class="container">
  <!-- Tarjeta de Producto -->
  <div class="card">
    <h2>Tarjeta de Producto</h2>
    <app-tarjeta-producto></app-tarjeta-producto>
  </div>

  <!-- Encabezado -->
  <div class="card">
    <h2>Encabezado</h2>
    <app-encabezado></app-encabezado>
  </div>

  <!-- Tareas -->
  <div class="card">
    <h2>Tareas</h2>
    <app-padre-tareas></app-padre-tareas>
  </div>

  <!-- Configuración -->
  <div class="card">
    <h2>Configuración</h2>
    <app-configuracion></app-configuracion>
  </div>

  <!-- Formulario -->
  <div class="card">
    <h2>Formulario</h2>
    <app-padre-formulario></app-padre-formulario>
  </div>

  <!-- Contador -->
  <div class="card">
    <h2>Contador</h2>
    <app-padre-contador></app-padre-contador>
  </div>
```



```
<!-- Carrito -->
<div class="card">
  <h2>Carrito</h2>
  <app-padre-carrito></app-padre-carrito>
</div>

<!-- Video -->
<div class="card">
  <h2>Video</h2>
  <app-padre-video></app-padre-video>
</div>

<!-- Validar Formulario -->
<div class="card">
  <h2>Validar Formulario</h2>
  <app-padre-validar-formulario></app-padre-validar-formulario>
</div>

<!-- Galería de Imágenes -->
<div class="card">
  <h2>Galería de Imágenes</h2>
  <app-padre-galeria-imagenes></app-padre-galeria-imagenes>
</div>
</div>
```

- **CSS**
  - El archivo app.component.css aplica estilos al contenedor y a las tarjetas individuales para crear un diseño atractivo y consistente.
  - **Estilo del contenedor:**
    - Usa flexbox para organizar las tarjetas en filas y columnas, adaptándose al tamaño de la pantalla.
    - Proporciona espacio entre las tarjetas (gap: 16px) y un fondo claro.
  - **Estilo de las tarjetas:**
    - Cada tarjeta tiene:
      - Un fondo blanco con bordes redondeados.
      - Una sombra ligera que crea un efecto elevado.
      - Transiciones suaves al interactuar con el mouse (hover).
      - Textos centrados y colores temáticos.
    - Se adapta a diferentes tamaños de pantalla para mejorar la experiencia del usuario.
  - **Botones:**
    - Botones dentro de las tarjetas tienen un diseño uniforme con colores temáticos (#0078d7 para fondo y blanco para texto).
    - Incluyen un efecto de cambio de color al pasar el mouse (hover).

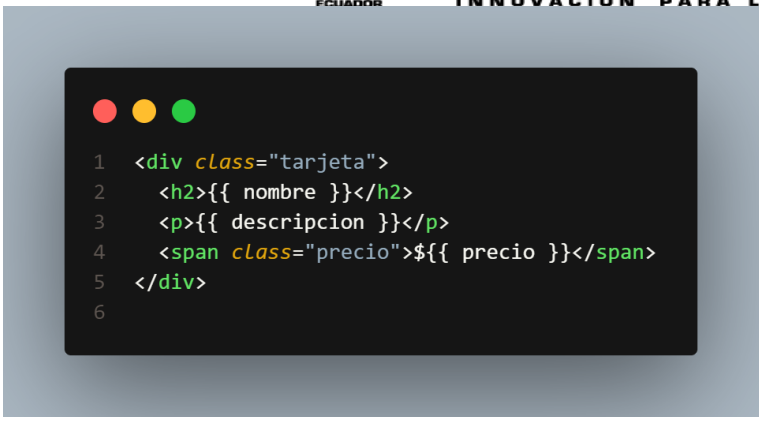
```css
/* General Container */
.container {
  display: flex;
  flex-wrap: wrap;
  gap: 16px;
  justify-content: center;
  padding: 16px;
  background-color: #f5f5f5;
}

/* Individual Card */
.card {
  background-color: #ffffff;
  border: 2px solid #e0e0e0;
  border-radius: 8px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  padding: 16px;
  width: 300px; /* Fixed width to keep content constrained */
  max-width: 100%; /* Prevent overflow */
  text-align: center; /* Center align text and inline content */
  font-family: 'Arial', sans-serif;
  transition: transform 0.2s ease, box-shadow 0.2s ease;
  overflow: hidden; /* Ensure content doesn't overflow the card */
  display: flex;
  flex-direction: column; /* Stack elements vertically */
  justify-content: center; /* Vertically center content */
  align-items: center; /* Horizontally center content */
  box-sizing: border-box; /* Include padding and border in size */
}

.card h2 {
  color: #0078d7;
  font-size: 18px;
  margin-bottom: 8px;
  text-transform: uppercase;
  word-wrap: break-word; /* Break long words */
}

.card p {
  font-size: 14px;
  color: #333333;
  margin: 8px 0;
  word-wrap: break-word; /* Ensure text doesn't overflow */
}

.card img {
  max-width: 100%; /* Prevent image from exceeding card width */
  height: auto; /* Maintain aspect ratio */
  margin-bottom: 8px;
  border-radius: 4px; /* Add a slight curve to images */
}

.card:hover {
  transform: translateY(-4px);
  box-shadow: 0 8px 12px rgba(0, 0, 0, 0.2);
}

/* Buttons Styling (if any inside components) */
button {
  background-color: #0078d7;
  color: #ffffff;
  border: none;
  border-radius: 4px;
  padding: 8px 12px;
  cursor: pointer;
  font-size: 14px;
  margin-top: 8px;
  width: 100%; /* Ensure button fits within card */
  max-width: 80%; /* Add some margin on the sides */
  text-align: center;
  box-sizing: border-box;
}

button:hover {
  background-color: #005bb5;
}
```

- **Flujo del Componente**
  - Entrada de datos: Cada tarjeta representa un componente funcional que puede recibir y mostrar datos. Por ejemplo, PadreValidarFormularioComponent muestra la validación del formulario.
  - Interactividad: Los usuarios pueden interactuar con elementos dentro de las tarjetas, como botones, formularios o imágenes, dependiendo de la funcionalidad del componente.
  - Reutilización: La estructura modular permite añadir, modificar o eliminar componentes sin afectar el diseño general.

- **Creación de un Componente de Tarjeta de Producto.**
  - **Contenido**: Crear un componente "Tarjeta-Producto" que muestre una tarjeta de producto con su nombre, precio y descripción.

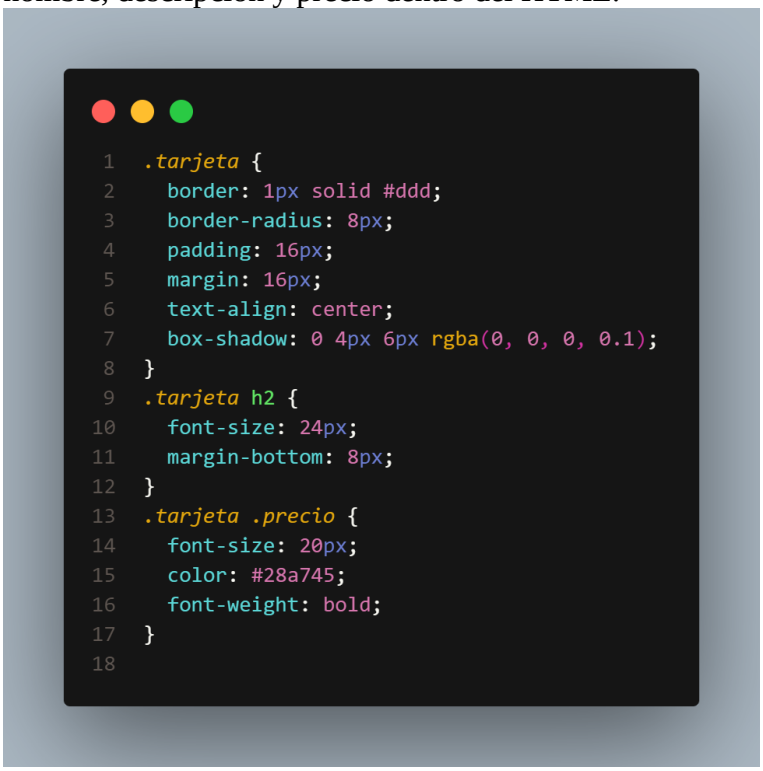
```typescript
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-tarjeta-producto',
  standalone: true,
  templateUrl: './tarjeta-producto.component.html',
  styleUrls: ['./tarjeta-producto.component.css']
})
export class TarjetaProductoComponent {
  @Input() nombre: string = 'Celular IPHONE 14 PLUS';
  @Input() precio: number = 500;
  @Input() descripcion: string = 'El mejor celular del mercado';
}
```

@Input(): Se utilizan decoradores @Input para pasar propiedades del componente padre al componente hijo. En este caso, el componente hijo Tarjeta-Producto recibe tres propiedades: nombre, precio, y descripción.
nombre, precio, descripción: Son las propiedades que se mostrarán en la tarjeta de producto, con valores predeterminados asignados.
templateUrl y styleUrls: Estos atributos especifican la ubicación de los archivos HTML y CSS asociados al componente.



```html
<div class="tarjeta">
  <h2>{{ nombre }}</h2>
  <p>{{ descripcion }}</p>
  <span class="precio">${{ precio }}</span>
</div>
```

Se utiliza la interpolación de Angular {{}} para insertar las propiedades de nombre, descripción y precio dentro del HTML.



```css
.tarjeta {
  border: 1px solid #ddd;
  border-radius: 8px;
  padding: 16px;
  margin: 16px;
  text-align: center;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}
.tarjeta h2 {
  font-size: 24px;
  margin-bottom: 8px;
}
.tarjeta .precio {
  font-size: 20px;
  color: #28a745;
  font-weight: bold;
}
```

Estilos de la tarjeta: Se le da un borde con un box-shadow para crear un efecto de profundidad y un borde redondeado (border-radius).
Estilo del precio: El precio se destaca con un color verde y un tamaño de fuente más grande, para darle mayor énfasis.

- **Apoyo visual**: La tarjeta tiene un diseño limpio con un borde, un título (nombre), una descripción y un precio. El precio se muestra en verde para destacarse del resto del contenido.

- **Diseño de un Componente de Encabezado.**
  - **Contenido**: Crear un componente de encabezado (Encabezado) que incluya un título y un menú de navegación con enlaces a redes sociales.

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

@Component({
  selector: 'app-encabezado',
  standalone: true,
  templateUrl: './encabezado.component.html',
  styleUrls: ['./encabezado.component.css'],
})
export class EncabezadoComponent {}

```

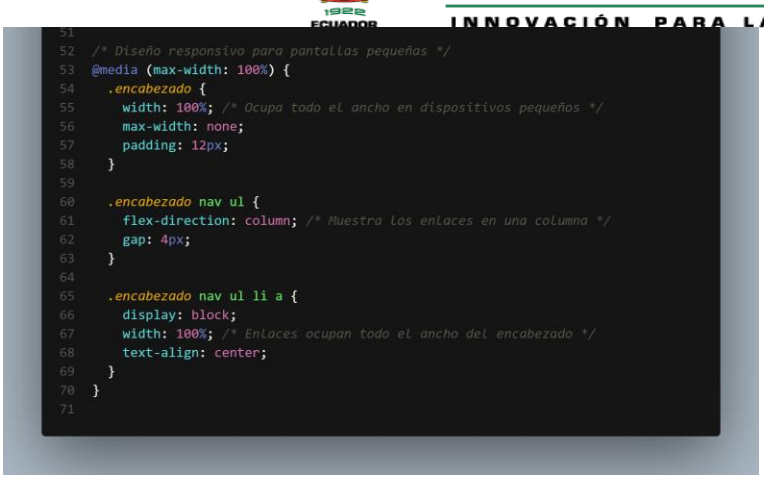
Este componente es más sencillo y solo contiene un título de la aplicación y un menú de navegación, sin recibir datos a través de @Input().

```html
<header class="encabezado">
  <h1>Mi Aplicación Angular</h1>
  <nav>
    <ul>
      <li><a href="https://www.facebook.com" target="_blank" rel="noopener noreferrer">Facebook</a></li>
      <li><a href="https://www.instagram.com" target="_blank" rel="noopener noreferrer">Instagram</a></li>
      <li><a href="https://www.twitter.com" target="_blank" rel="noopener noreferrer">X</a></li>
    </ul>
  </nav>
</header>
```

<header>: El componente utiliza una etiqueta de encabezado HTML para envolver el contenido principal.

Menú de navegación: Dentro del encabezado, se encuentra un menú de navegación con enlaces a redes sociales, como Facebook, Instagram y Twitter. Los enlaces están diseñados para abrirse en nuevas pestañas (target="_blank").

```css
/* Encabezado principal */
.encabezado {
  background-color: #007bff;
  color: white;
  padding: 16px;
  border-radius: 8px; /* Coincide con las tarjetas */
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1); /* Estilo similar a las tarjetas */
  width: 200; /* Ajuste al ancho de la tarjeta */
  max-width: 100%; /* Responsivo, no supera el contenedor */
  text-align: center; /* Centrado del contenido */
  font-family: 'Arial', sans-serif;
  margin: 0 auto; /* Centra horizontalmente dentro de la página */
  position: relative; /* Se puede ajustar para más personalización */
}

/* Titulo del encabezado */
.encabezado h1 {
  font-size: 18px;
  margin: 0;
  font-weight: bold;
  text-transform: uppercase; /* Resalta el titulo */
  color: #ffffff;
}

/* Navegación */
.encabezado nav ul {
  list-style: none;
  display: flex;
  gap: 8px; /* Espaciado reducido para adaptarse al tamaño de la tarjeta */
  margin: 8px 0 0; /* Espaciado entre el titulo y los enlaces */
  padding: 0;
  justify-content: center; /* Centra los enlaces dentro del encabezado */
}

/* Estilos de los enlaces */
.encabezado nav ul li a {
  color: white;
  text-decoration: none;
  font-size: 14px;
  font-weight: 500;
  padding: 4px 8px; /* Botones más compactos */
  border-radius: 4px; /* Coincide con el diseño de la tarjeta */
  transition: background-color 0.3s, color 0.3s;
}

/* Hover en los enlaces */
.encabezado nav ul li a:hover {
  background-color: #0056b3;
  color: #f9f9f9;
}
```

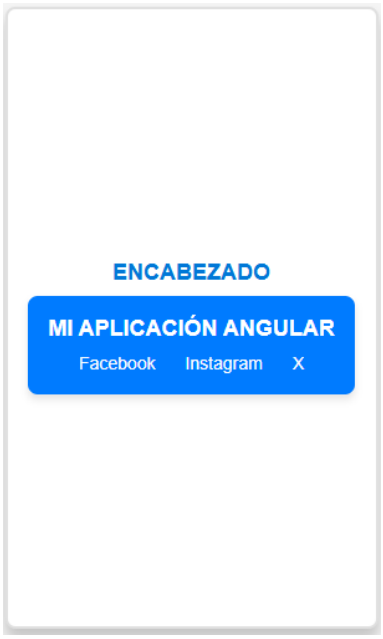


```
/* Diseño responsivo para pantallas pequeñas */
@media (max-width: 100%) {
  .encabezado {
    width: 100%; /* Ocupa todo el ancho en dispositivos pequeños */
    max-width: none;
    padding: 12px;
  }

  .encabezado nav ul {
    flex-direction: column; /* Muestra los enlaces en una columna */
    gap: 4px;
  }

  .encabezado nav ul li a {
    display: block;
    width: 100%; /* Enlaces ocupan todo el ancho del encabezado */
    text-align: center;
  }
}
```

Estilo del encabezado: Se define un fondo azul con un borde redondeado y sombras. Se utiliza flexbox para alinear los elementos de navegación de forma horizontal, y en pantallas pequeñas, se reorganiza en una columna.

Interactividad: Los enlaces del menú tienen efectos hover para mejorar la experiencia del usuario.

- **Apoyo visual**: El encabezado tiene un fondo azul con el nombre de la aplicación en blanco y un menú de enlaces que se ajustan bien al diseño responsivo. En dispositivos más pequeños, los enlaces se apilan verticalmente.



- **Desarrollar un componente padre que envíe una lista de tareas a un componente hijo encargado de mostrar cada tarea en un formato de lista.**

- **Contenido**: En este ejercicio, crearemos un componente padre que tiene una lista de tareas (un array de cadenas) y pasaremos esta lista al componente hijo mediante el uso de @Input(). El componente hijo se encargará de mostrar las tareas en una lista ordenada.
- **PADRE**: El componente padre tiene una lista de tareas y las pasa al componente hijo.

```ts
import { Component } from '@angular/core';
import { ListaTareasComponent } from '../lista-tareas/lista-tareas.component';

@Component({
  selector: 'app-padre-tareas',
  standalone: true,
  imports: [ListaTareasComponent],
  templateUrl: './padre-tareas.component.html',
  styleUrls: ['./padre-tareas.component.css']
})
export class PadreTareasComponent {
  tareas: string[] = ['Estudiar Angular', 'Terminar proyecto', 'Leer documentación'];
}
```

```html
<app-lista-tareas [tareas]="tareas"></app-lista-tareas>
```

- **HIJO**: El componente hijo recibe la lista de tareas a través de la propiedad @Input() y las muestra en una lista HTML.

```ts
import { Component, Input } from '@angular/core';
import { CommonModule } from '@angular/common';
@Component({
  selector: 'app-lista-tareas',
  standalone: true,
  imports: [CommonModule], // Importar CommonModule
  templateUrl: './lista-tareas.component.html',
  styleUrls: ['./lista-tareas.component.css']
})
export class ListaTareasComponent {
  @Input() tareas: string[] = [];
}
```

```html
<ul>
  <li *ngFor="let tarea of tareas">{{ tarea }}</li>
</ul>
```

```css
ul {
  list-style-type: none;
  padding: 0;
  margin: 0;
}

li {
  padding: 10px;
  margin: 5px 0;
  border: 1px solid #ddd;
  border-radius: 5px;
  background-color: #f9f9f9;
  transition: background-color 0.3s ease;
}

li:hover {
  background-color: #e0e0e0;
}

li:nth-child(odd) {
  background-color: #f3f3f3;
}
```

- **Apoyo visual**:
  - **Componente Padre:** Tiene un array de tareas y las pasa al componente hijo mediante la vinculación de datos [tareas]="tareas".
  - **Componente Hijo:** El hijo recibe los datos a través de @Input() y los muestra en una lista utilizando la directiva *ngFor.

- **Crea un componente padre que pase datos de configuración (por ejemplo, idioma o tema) a un componente hijo encargado de renderizar contenido.**
  - **Contenido**: En este ejercicio, el componente padre gestiona una configuración (como idioma y tema) y pasa estos datos al componente hijo. El hijo renderiza contenido basado en la configuración y ofrece opciones para cambiar el idioma y el tema.
  - **PADRE**: El componente padre contiene una configuración y la pasa al hijo, permitiendo también cambiar la configuración mediante un método.

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ContenidoComponent } from '../contenido/contenido.component';

@Component({
  selector: 'app-configuracion',
  standalone: true,
  imports: [CommonModule, ContenidoComponent],
  templateUrl: './configuracion.component.html',
  styleUrls: ['./configuracion.component.css'],
})
export class ConfiguracionComponent {
  config = { idioma: 'es', tema: 'oscuro' };

  actualizarConfig(nuevaConfig: { idioma: string; tema: string }) {
    this.config = nuevaConfig; // Actualiza la configuración en el padre
  }
}
```

```
<app-contenido
  [config]="config"
  (configChange)="actualizarConfig($event)"
></app-contenido>
```

```
.container {
  padding: 20px;
  background-color: #f4f4f4;
  border: 1px solid #ddd;
  border-radius: 5px;
  max-width: 400px;
  margin: auto;
}

h4 {
  margin-bottom: 15px;
}

p {
  font-size: 14px;
  color: #555;
}
```

- **HIJO**: El componente hijo recibe la configuración a través de @Input() y emite eventos cuando el usuario cambia la configuración (idioma o tema).

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

@Component({
  selector: 'app-contenido',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './contenido.component.html',
  styleUrls: ['./contenido.component.css'],
})
export class ContenidoComponent {
  @Input() config: { idioma: string; tema: string } = { idioma: 'es', tema: 'claro' };
  @Output() configChange = new EventEmitter<{ idioma: string; tema: string }>();

  cambiarIdioma() {
    this.config.idioma = this.config.idioma === 'es' ? 'en' : 'es';
    this.configChange.emit({ ...this.config });
  }

  cambiarTema() {
    this.config.tema = this.config.tema === 'claro' ? 'oscuro' : 'claro';
    this.configChange.emit({ ...this.config });
  }
}
```

```html
<div [ngClass]="config.tema">
  <h2>{{ config.idioma === 'es' ? 'Configuración' : 'Settings' }}</h2>
  <p>{{ config.idioma === 'es' ? 'Idioma actual:' : 'Current Language:' }} {{ config.idioma }}</p>
  <p>{{ config.idioma === 'es' ? 'Tema actual:' : 'Current Theme:' }} {{ config.tema }}</p>

  <button (click)="cambiarIdioma()">
    {{ config.idioma === 'es' ? 'Cambiar a Inglés' : 'Switch to Spanish' }}
  </button>
  <button (click)="cambiarTema()">
    {{ config.idioma === 'es' ? 'Cambiar a Oscuro' : 'Switch to Dark' }}
  </button>
</div>
```

```css
/* Estilos para los temas */
.claro {
  background-color: #f9f9f9;
  color: #333;
  padding: 20px;
  border-radius: 10px;
  text-align: center;
}
```

```css
.oscuro {
  background-color: #333;
  color: #f9f9f9;
  padding: 20px;
  border-radius: 10px;
  text-align: center;
}

button {
  margin: 10px;
  padding: 10px 20px;
  font-size: 16px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

button:hover {
  opacity: 0.8;
}

button:nth-child(1) {
  background-color: #007bff;
  color: #fff;
}

button:nth-child(2) {
  background-color: #28a745;
  color: #fff;
}
```

- **Apoyo visual**:
  - **Componente Padre:** Administra el estado de la configuración (idioma y tema) y lo pasa al componente hijo.
  - **Componente Hijo:** Muestra la configuración actual y permite cambiarla. También emite eventos para que el componente padre actualice la configuración.

- **Implementa un componente hijo con un formulario que emita un evento al padre cuando se envíen los datos.**
  - **Contenido**: En este ejercicio, se creó un formulario en el componente hijo que emite un evento al componente padre cuando se envían los datos. El componente padre recibirá estos datos y los mostrará en su plantilla.
  - **HIJO**:
    - Utilizamos @Output para emitir un evento con los datos introducidos en el formulario.
    - El formulario tiene dos campos: nombre y email.
    - El formulario del componente hijo permite ingresar los datos.
    - Al enviar el formulario, el evento enviarFormulario es emitido con los valores de nombre y email.

```ts
import { Component, EventEmitter, Output } from '@angular/core';
import { FormsModule } from '@angular/forms'; // Importa FormsModule
@Component({
  selector: 'app-hijo-formulario',
  standalone: true,
  imports: [FormsModule], // Incluye FormsModule aquí
  templateUrl: './hijo-formulario.component.html',
  styleUrls: ['./hijo-formulario.component.css'],
})
export class HijoFormularioComponent {
  // Evento para emitir datos al componente padre
  @Output() enviarFormulario = new EventEmitter<{ nombre: string; email: string }>();

  // Variables locales para el formulario
  nombre: string = '';
  email: string = '';

  // Método que se ejecuta al enviar el formulario
  enviarDatos() {
    this.enviarFormulario.emit({ nombre: this.nombre, email: this.email });
  }
}

```

```html
<div class="formulario">
  <h3>Formulario</h3>
  <form (ngSubmit)="enviarDatos()">
    <label for="nombre">Nombre:</label>
    <input id="nombre" [(ngModel)]="nombre" name="nombre" type="text" required />

    <label for="email">Email:</label>
    <input id="email" [(ngModel)]="email" name="email" type="email" required />

    <button type="submit">Enviar</button>
  </form>
</div>

```

```css
/* Formulario */
.formulario {
  padding: 16px;
  border: 2px solid #e0e0e0;
  border-radius: 8px;
  background-color: #ffffff;
  max-width: 300px;
  width: 80%; /* Asegura que ocupe todo el ancho disponible dentro de la tarjeta */
  margin: 0 auto; /* Centra el formulario */
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1); /* Sombra sutil */
  font-family: 'Arial', sans-serif;
}

/* Etiquetas */
label {
  display: block;
  margin: 8% 0 4%;
  font-weight: bold;
  font-size: 14px;
  color: #333;
}

/* Campos de entrada */
input {
  width: 100%;
  padding: 10px;
  margin-bottom: 12px;
  border: 1px solid #ccc;
  border-radius: 4px;
  font-size: 14px;
  box-sizing: border-box; /* Asegura que el padding se incluya en el ancho total */
}
```

```css
/* Botón de envío */
button {
  background-color: #007bff;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-size: 16px;
  width: 100%; /* Asegura que el botón ocupe todo el ancho disponible */
  transition: background-color 0.3s;
}

button:hover {
  background-color: #0056b3;
}

/* Espaciado adicional entre campos y el botón */
input, button {
  margin-top: 8px;
}
```

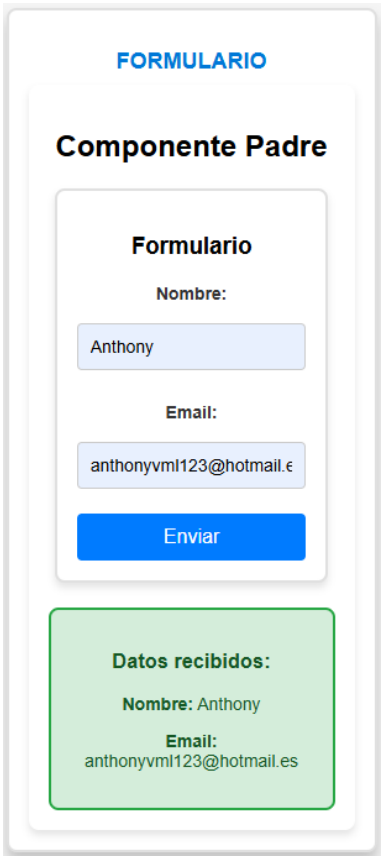
- **PADRE:**
  - Recibe el evento enviarFormulario desde el componente hijo.
  - Al recibir los datos, los muestra en su plantilla.
  - El componente padre recibe esos datos y los muestra en su sección correspondiente.

```typescript
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HijoFormularioComponent } from '../hijo-formulario/hijo-formulario.component';

@Component({
  selector: 'app-padre-formulario',
  standalone: true,
  imports: [CommonModule, HijoFormularioComponent],
  templateUrl: './padre-formulario.component.html',
  styleUrls: ['./padre-formulario.component.css'],
})
export class PadreFormularioComponent {
  datosRecibidos: { nombre: string; email: string } | null = null;

  // Método que se ejecuta al recibir datos del hijo
  manejarFormulario(datos: { nombre: string; email: string }) {
    this.datosRecibidos = datos;
  }
}
```

```html
<div class="padre">
  <h2>Componente Padre</h2>
  <app-hijo-formulario (enviarFormulario)="manejarFormulario($event)"></app-hijo-formulario>

  <div *ngIf="datosRecibidos" class="resultado">
    <h3>Datos recibidos:</h3>
    <p><strong>Nombre:</strong> {{ datosRecibidos.nombre }}</p>
    <p><strong>Email:</strong> {{ datosRecibidos.email }}</p>
  </div>
</div>
```

```css
/* Contenedor principal del padre */
.padre {
  padding: 16px;
  border-radius: 8px; /* Bordes redondeados para un diseño más suave */
  background-color: #ffffff; /* Fondo blanco para separar el contenido del fondo */
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1); /* Sombra suave para darle profundidad */
  max-width: 350px; /* Limita el ancho para que no se expanda demasiado */
  margin: 0 auto; /* Centra el contenedor */
}

/* Estilos del resultado */
.resultado {
  margin-top: 20px;
  border: 2px solid #28a745; /* Un borde más grueso para destacar */
  padding: 16px;
  border-radius: 8px; /* Bordes redondeados */
  background-color: #d4edda; /* Fondo verde claro */
  font-size: 14px; /* Ajuste de fuente para mayor legibilidad */
  color: #155724; /* Color del texto para combinar con el fondo */
  box-sizing: border-box; /* Asegura que el padding no afecte el tamaño total */
}
```

- **Apoyo visual**:



- **Diseña un contador en un componente hijo que informe al padre cada vez que se incremente o disminuya el valor.**

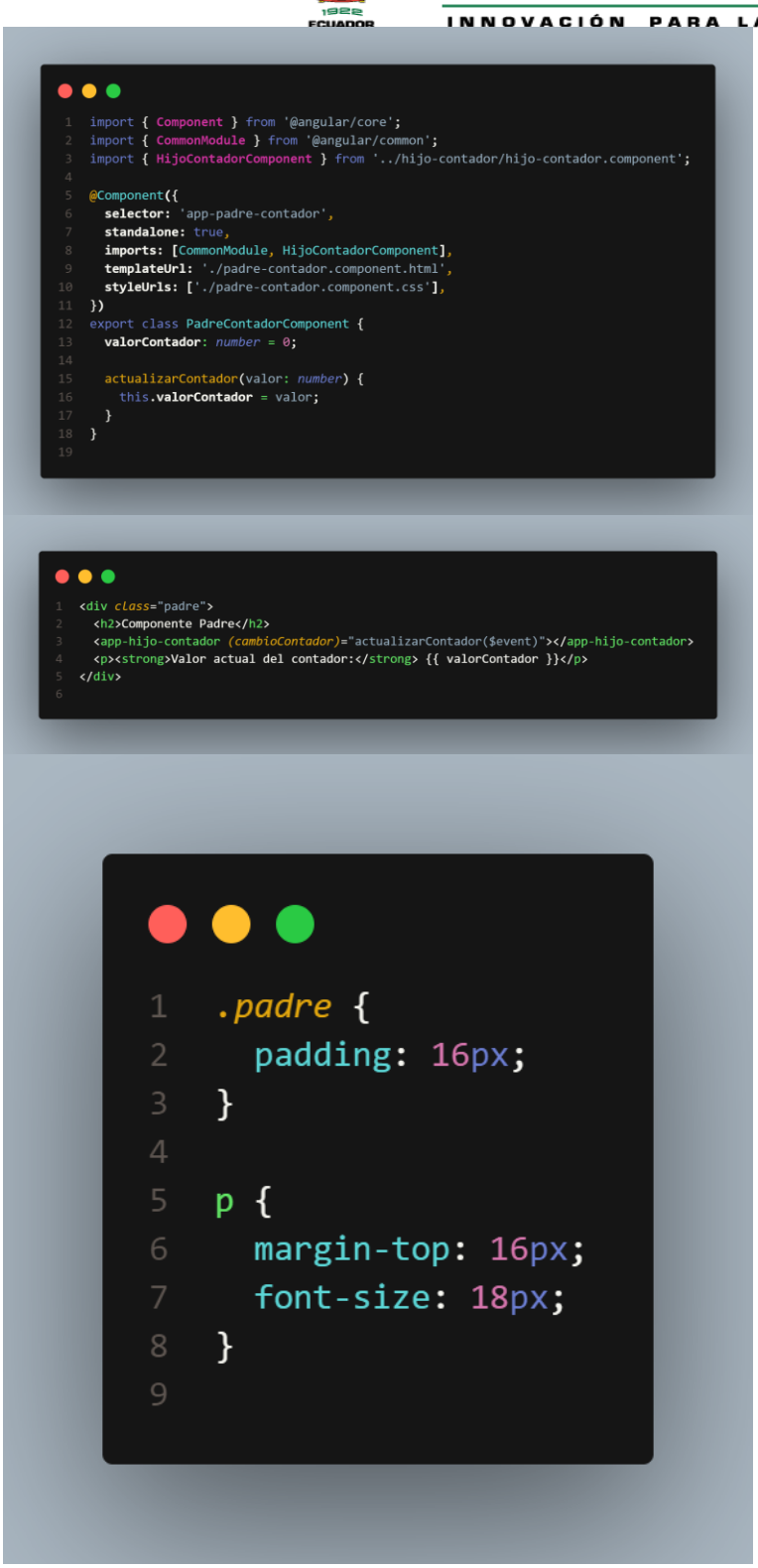
- **Contenido**: En este ejercicio, vamos a crear un contador en el componente hijo que emite eventos al padre cada vez que el valor se incrementa o decrementa.
- **HIJO**:
- Tiene un contador que se incrementa y decrementa mediante botones.
- Cada vez que el valor cambia, emite un evento con el nuevo valor al componente padre.
- El componente hijo contiene botones para incrementar y decrementar el contador.

```ts
import { Component, EventEmitter, Output } from '@angular/core';

@Component({
  selector: 'app-hijo-contador',
  standalone: true,
  templateUrl: './hijo-contador.component.html',
  styleUrls: ['./hijo-contador.component.css'],
})
export class HijoContadorComponent {
  contador: number = 0;

  // Eventos para informar al padre
  @Output() cambioContador = new EventEmitter<number>();

  incrementar() {
    this.contador++;
    this.cambioContador.emit(this.contador);
  }

  decrementar() {
    this.contador--;
    this.cambioContador.emit(this.contador);
  }
}

```

```html
<div class="contador">
  <h3>Contador: {{ contador }}</h3>
  <button (click)="incrementar()">Incrementar</button>
  <button (click)="decrementar()">Decrementar</button>
</div>

```

```css
.contador {
  text-align: center;
}

button {
  margin: 4px;
  padding: 8px 16px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

button:first-child {
  background-color: #28a745;
  color: white;
}

button:last-child {
  background-color: #dc3545;
  color: white;
}
```

- **PADRE**:
- Recibe el evento cambioContador desde el componente hijo.
- Muestra el valor actualizado del contador en su plantilla.
- El componente padre muestra el valor actualizado del contador.
- html



```ts
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HijoContadorComponent } from '../hijo-contador/hijo-contador.component';

@Component({
  selector: 'app-padre-contador',
  standalone: true,
  imports: [CommonModule, HijoContadorComponent],
  templateUrl: './padre-contador.component.html',
  styleUrls: ['./padre-contador.component.css'],
})
export class PadreContadorComponent {
  valorContador: number = 0;

  actualizarContador(valor: number) {
    this.valorContador = valor;
  }
}
```

```html
<div class="padre">
  <h2>Componente Padre</h2>
  <app-hijo-contador (cambioContador)="actualizarContador($event)"></app-hijo-contador>
  <p><strong>Valor actual del contador:</strong> {{ valorContador }}</p>
</div>
```

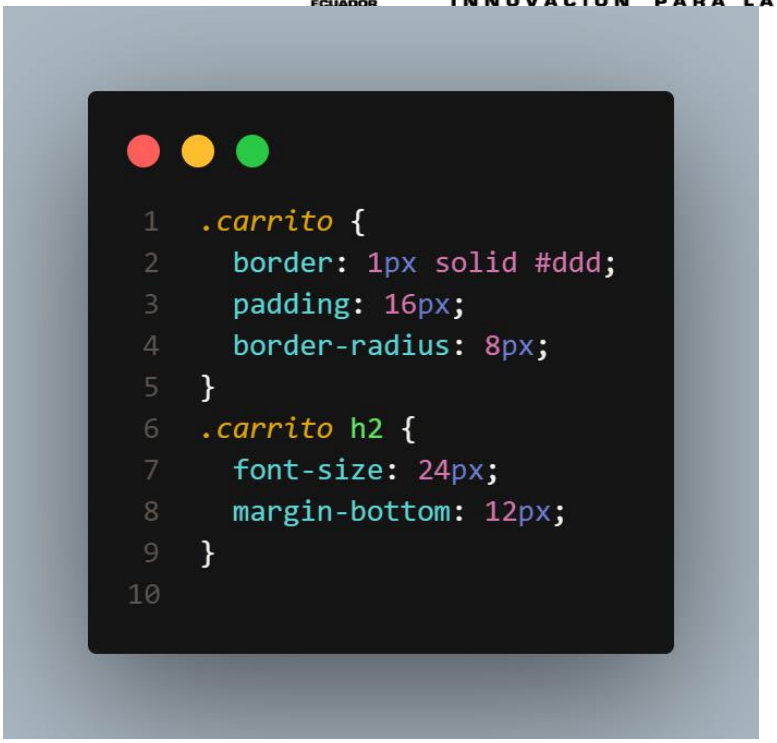
```css
.padre {
  padding: 16px;
}

p {
  margin-top: 16px;
  font-size: 18px;
}
```

- **Apoyo visual**:

- **Crear un componente hijo con un método para calcular un valor y llamar a este método desde el template del componente padre.**
  - **Contenido**: Este ejercicio consiste en implementar una funcionalidad en un componente hijo para realizar cálculos (como el total de un carrito de compras) y luego llamar a este cálculo desde el componente padre.
  - **HIJO**:
    - El componente hijo tiene un arreglo items, donde se almacenan los productos del carrito, y un método calcularTotal(), que calcula el total sumando los precios de todos los productos en el carrito.
    - **Archivo TS:** Aquí, el método calcularTotal se encarga de sumar los precios de los productos usando el método reduce de JavaScript.
    - **Archivo HTML:** El HTML muestra una lista de productos, usando *ngFor para iterar sobre los productos.

```ts
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
@Component({
  selector: 'app-hijo-carrito',
  standalone: true,
  imports: [CommonModule], // Importa el módulo necesario para usar *ngFor
  templateUrl: './hijo-carrito.component.html',
  styleUrls: ['./hijo-carrito.component.css'],
})
export class HijoCarritoComponent {
  items = [
    { nombre: 'Producto 1', precio: 10 },
    { nombre: 'Producto 2', precio: 20 },
    { nombre: 'Producto 3', precio: 15 },
  ];

  calcularTotal(): number {
    return this.items.reduce((total, item) => total + item.precio, 0);
  }
}

```

```html
<div class="carrito">
  <h2>Carrito de Compras</h2>
  <ul>
    <li *ngFor="let item of items">
      {{ item.nombre }} - ${{ item.precio }}
    </li>
  </ul>
</div>

```



```css
.carrito {
  border: 1px solid #ddd;
  padding: 16px;
  border-radius: 8px;
}
.carrito h2 {
  font-size: 24px;
  margin-bottom: 12px;
}
```

- **PADRE**:
  - El componente padre usa @ViewChild para acceder al método calcularTotal del hijo. Al hacer clic en el botón "Calcular Total", el componente padre llama al método del hijo y muestra el total calculado.
  - **Archivo TS:** Aquí, el componente padre tiene una variable total que se actualiza con el valor calculado por el hijo.
  - **Archivo HTML:** En el HTML del padre se incluye un botón que, al hacer clic, llama a obtenerTotal(), que a su vez invoca calcularTotal() del hijo.

```ts
import { Component, ViewChild } from '@angular/core';
import { HijoCarritoComponent } from '../hijo-carrito/hijo-carrito.component';

@Component({
  selector: 'app-padre-carrito',
  standalone: true,
  templateUrl: './padre-carrito.component.html',
  styleUrls: ['./padre-carrito.component.css'],
  imports: [HijoCarritoComponent],
})
export class PadreCarritoComponent {
  @ViewChild(HijoCarritoComponent) hijoCarrito!: HijoCarritoComponent;

  total: number = 0;

  obtenerTotal() {
    this.total = this.hijoCarrito.calcularTotal();
  }
}

```

```html
<div class="padre-carrito">
  <app-hijo-carrito></app-hijo-carrito>
  <button (click)="obtenerTotal()">Calcular Total</button>
  <p>Total: ${{ total }}</p>
</div>

```

```css
.padre-carrito {
  padding: 16px;
}
button {
  margin-top: 16px;
  padding: 8px 16px;
  font-size: 16px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
button:hover {
  background-color: #0056b3;
}
```

- **Apoyo visual**:
  - Diagrama de interacción entre componentes:
    - El componente padre incluye al componente hijo dentro de su plantilla.
    - El padre llama a un método del hijo mediante @ViewChild.
    - El hijo ejecuta la lógica de cálculo y devuelve el resultado al padre.

- **Diseñar un componente hijo que permita reproducir/pausar un video, controlado desde un botón en el template del componente padre.**
  - **Contenido**: Este ejercicio muestra cómo interactuar con un componente hijo que controla la reproducción de un video, y cómo el componente padre puede controlar ese video mediante botones.
  - **HIJO**:
    - El hijo contiene un elemento video que tiene métodos reproducir y pausar. Estos métodos interactúan directamente con el DOM del video para controlar su estado.
    - **Archivo TS:** El componente usa @ViewChild para acceder al elemento <video> en el DOM, y proporciona dos métodos: reproducir() y pausar(), que controlan el video.
    - **Archivo HTML:** El HTML incluye un elemento de video que se puede controlar desde el padre.

```ts
import { Component, ViewChild, ElementRef } from '@angular/core';

@Component({
  selector: 'app-hijo-video',
  standalone: true,
  templateUrl: './hijo-video.component.html',
  styleUrls: ['./hijo-video.component.css'],
})
export class HijoVideoComponent {
  @ViewChild('videoPlayer') videoPlayer!: ElementRef<HTMLVideoElement>;

  reproducir() {
    this.videoPlayer.nativeElement.play();
  }

  pausar() {
    this.videoPlayer.nativeElement.pause();
  }
}

```

```html
<div class="video-container">
  <video #videoPlayer width="320" height="240" controls>
    <source src="https://www.w3schools.com/html/mov_bbb.mp4" type="video/mp4" />
    Tu navegador no soporta el elemento de video.
  </video>
</div>

```

```css
.video-container {
  text-align: center;
  margin-top: 2px;
}

```

- **PADRE**:
  - El componente padre también usa @ViewChild para acceder al hijo, y controla la reproducción y pausa del video desde botones.

- **Archivo TS:** El componente padre tiene dos métodos, reproducirVideo y pausarVideo, que llaman a los métodos correspondientes del hijo.
- **Archivo HTML:** Los botones del padre llaman a estos métodos para reproducir o pausar el video.

```ts
import { Component, ViewChild } from '@angular/core';
import { HijoVideoComponent } from '../hijo-video/hijo-video.component';

@Component({
  selector: 'app-padre-video',
  standalone: true,
  templateUrl: './padre-video.component.html',
  styleUrls: ['./padre-video.component.css'],
  imports: [HijoVideoComponent],
})
export class PadreVideoComponent {
  @ViewChild(HijoVideoComponent) hijoVideo!: HijoVideoComponent;

  reproducirVideo() {
    this.hijoVideo.reproducir();
  }

  pausarVideo() {
    this.hijoVideo.pausar();
  }
}
```

```html
<div class="padre-video">
  <app-hijo-video></app-hijo-video>
  <button (click)="reproducirVideo()">Reproducir</button>
  <button (click)="pausarVideo()">Pausar</button>
</div>
```

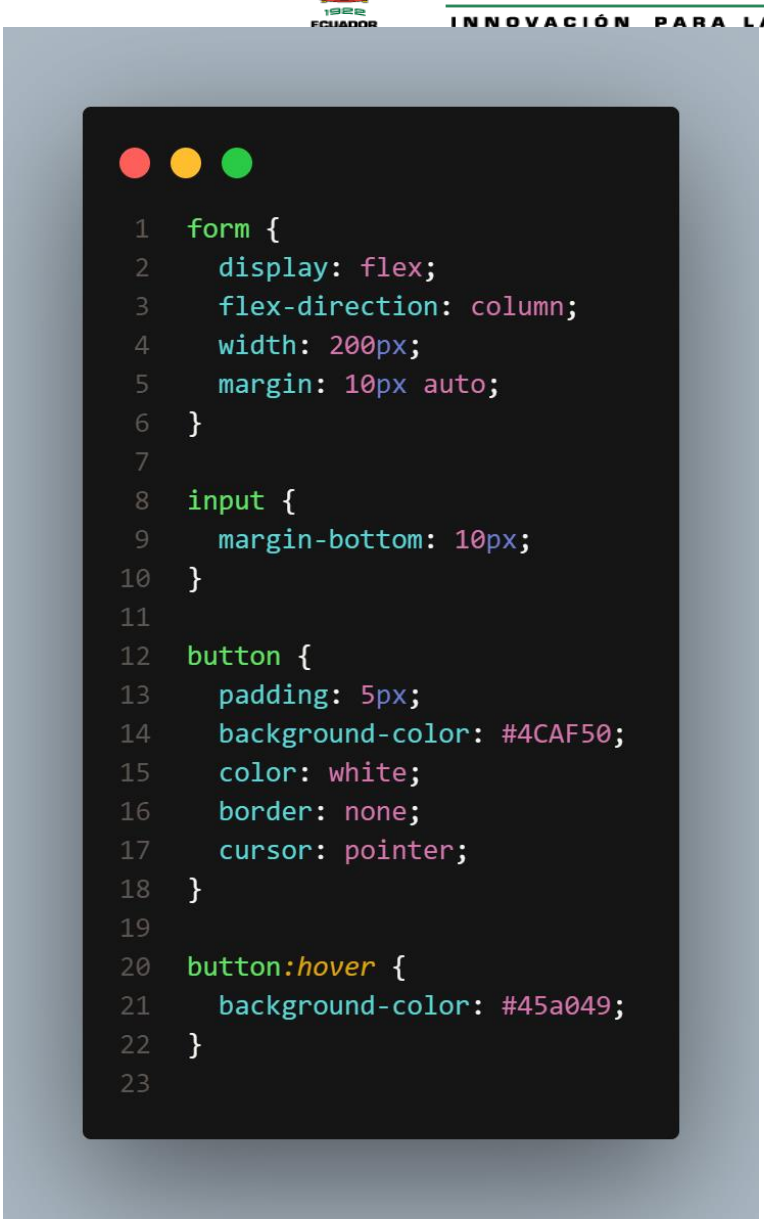
```css
/* Contenedor del video padre */
.padre-video {
  text-align: center;
  padding: 16px;
  border-radius: 8px; /* Bordes redondeados */
  background-color: #ffffff; /* Fondo blanco para distinguirlo del fondo general */
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1); /* Sombra para darle profundidad */
  max-width: 100%; /* Limita el ancho máximo */
  margin: center; /* Centrado automático */
}

/* Estilos para los botones */
button {
  margin: 8px;
  padding: 8px 16px;
  font-size: 16px;
  background-color: #28a745;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.3s ease, transform 0.3s ease;
}

/* Efecto hover para el primer botón */
button:hover {
  background-color: #218838;
  transform: translateY(-2px); /* Efecto de elevación al pasar el ratón */
}

/* Estilo para el segundo botón */
button:nth-child(2) {
  background-color: #dc3545; /* Fondo rojo para el segundo botón */
}

/* Efecto hover para el segundo botón */
button:nth-child(2):hover {
  background-color: #c82333;
  transform: translateY(-2px); /* Efecto de elevación al pasar el ratón */
}
```

- **Apoyo visual**:
  - Diagrama de interacción entre componentes:
    - El componente padre incluye el componente hijo.
    - El padre puede controlar el video utilizando los botones para reproducir o pausar.
    - El hijo recibe las solicitudes del padre y realiza las acciones sobre el video.

- **Llamar a Métodos de Validación de un Formulario desde el Componente Padre**
  - **Contenido**: En este ejercicio, se crea un componente hijo encargado de validar un formulario. Desde el componente padre, se llama al método de validación del hijo y se maneja la respuesta para mostrar un mensaje de validación.
  - **HIJO**:
    - Contiene un formulario con dos campos (dato_1 y dato_2).
    - Define un método onSubmit() que valida si ambos campos están llenos y emite un evento (formularioValido) al componente padre.
    - Implementa el formulario con campos enlazados mediante [(ngModel)].
    - Se aplica un diseño básico al formulario con estilos atractivos para los botones y campos.

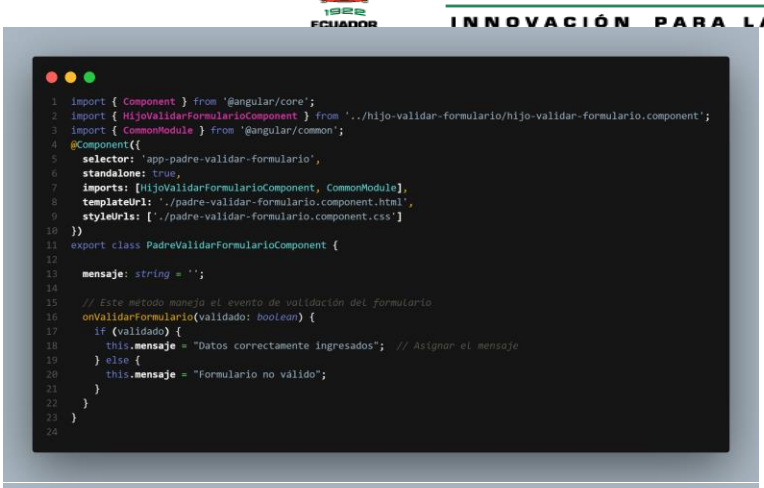
```ts
import { Component, EventEmitter, Output } from '@angular/core';
import { FormsModule } from '@angular/forms';  // Importa FormsModule
@Component({
  selector: 'app-hijo-validar-formulario',
  standalone: true,
  imports: [FormsModule],  // Importa FormsModule
  templateUrl: './hijo-validar-formulario.component.html',
  styleUrls: ['./hijo-validar-formulario.component.css']
})
export class HijoValidarFormularioComponent {

  @Output() formularioValido = new EventEmitter<boolean>();
  dato_1: string = '';
  dato_2: string = '';

  // Método para manejar el envío del formulario
  onSubmit(event: Event) {
    event.preventDefault();

    // Validamos si ambos campos están llenos
    if (this.dato_1 && this.dato_2) {
      this.formularioValido.emit(true);  // Emitir validación exitosa
    } else {
      this.formularioValido.emit(false);  // Emitir validación fallida
    }
  }
}

```

```html
<div class="formulario">
  <form (submit)="onSubmit($event)">
    <label for="dato_1">Dato 1:</label>
    <input type="text" id="dato_1" [(ngModel)]="dato_1" name="dato_1" required />

    <label for="dato_2">Dato 2:</label>
    <input type="dato_2" id="dato_2" [(ngModel)]="dato_2" name="dato_2" required />

    <button type="submit">Validar</button>
  </form>
</div>

```



```css
form {
  display: flex;
  flex-direction: column;
  width: 200px;
  margin: 10px auto;
}

input {
  margin-bottom: 10px;
}

button {
  padding: 5px;
  background-color: #4CAF50;
  color: white;
  border: none;
  cursor: pointer;
}

button:hover {
  background-color: #45a049;
}
```

- **PADRE**:
  - Escucha el evento del hijo para determinar si los datos son válidos.
  - Muestra un mensaje según la validación.
  - Incluye al componente hijo y muestra el mensaje de validación.

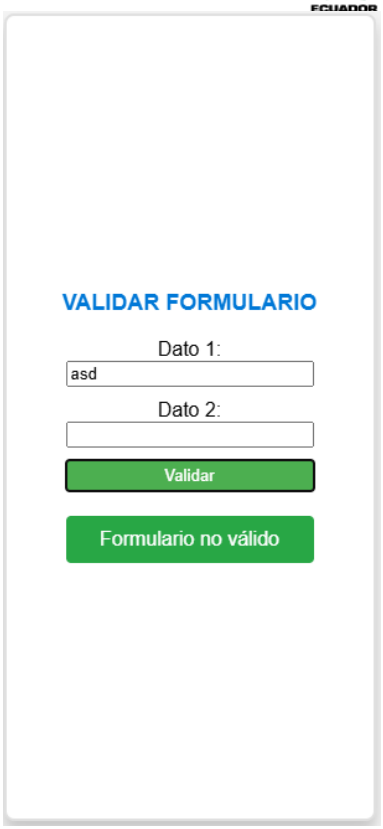


```ts
import { Component } from '@angular/core';
import { HijoValidarFormularioComponent } from '../hijo-validar-formulario/hijo-validar-formulario.component';
import { CommonModule } from '@angular/common';
@Component({
  selector: 'app-padre-validar-formulario',
  standalone: true,
  imports: [HijoValidarFormularioComponent, CommonModule],
  templateUrl: './padre-validar-formulario.component.html',
  styleUrls: ['./padre-validar-formulario.component.css']
})
export class PadreValidarFormularioComponent {

  mensaje: string = '';

  // Este método maneja el evento de validación del formulario
  onValidarFormulario(validado: boolean) {
    if (validado) {
      this.mensaje = "Datos correctamente ingresados";  // Asignar el mensaje
    } else {
      this.mensaje = "Formulario no válido";
    }
  }
}
```

```html
<!-- Componente Hijo -->
<app-hijo-validar-formulario (formularioValido)="onValidarFormulario($event)"></app-hijo-validar-formulario>

<!-- Mensaje de validación -->
<div *ngIf="mensaje" class="mensaje">
  {{ mensaje }}
</div>
```

```css
.mensaje {
  padding: 10px;
  margin-top: 20px;
  font-size: 16px;
  color: #fff;
  background-color: #28a745; /* Verde de éxito */
  border-radius: 4px;
  text-align: center;
}

.mensaje.error {
  background-color: #dc3545; /* Rojo de error */
}
```

- **Apoyo visual**:
  - Formulario del Hijo y Mensaje del Padre.



- **Manipulación de una Galería de Imágenes desde el Componente Padre**
  - **Contenido**: Se crea un componente hijo que muestra una galería de imágenes con botones para avanzar y retroceder. Los métodos del hijo son llamados desde el componente padre.
  - **HIJO**:
    - Contiene una lista de imágenes y métodos para cambiar de imagen.
    - Muestra la imagen actual y los botones de control.
    - Se aplican estilos para centrar y embellecer la galería.

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

@Component({
  selector: 'app-hijo-galeria-imagenes',
  standalone: true,
  templateUrl: './hijo-galeria-imagenes.component.html',
  styleUrls: ['./hijo-galeria-imagenes.component.css']
})
export class HijoGaleriaImagenesComponent {
  imagenes: string[] = [
    'https://i.pinimg.com/236x/f8/ff/f0/f8fff0d5493da02431a35fbbe45dba4c.jpg',
    'https://i.pinimg.com/originals/63/58/39/6358396d6debfcc3927e46e8cfd7bb88.jpg',
    'https://i.pinimg.com/564x/1d/51/3a/1d513a3806a9228cd543a6afcf0b4e48.jpg'
  ];
  indiceActual: number = 0;

  avanzar() {
    if (this.indiceActual < this.imagenes.length - 1) {
      this.indiceActual++;
    }
  }

  retroceder() {
    if (this.indiceActual > 0) {
      this.indiceActual--;
    }
  }
}

```

```html
<div class="galeria">
  <img [src]="imagenes[indiceActual]" alt="Imagen actual" />
  <div class="botones">
    <button (click)="retroceder()">← Retroceder</button>
    <button (click)="avanzar()">Avanzar →</button>
  </div>
</div>

```

```css
.galeria {
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 20px;
}

.galeria img {
  width: 200px;
  max-width: 600px;
  height: auto;
  border-radius: 8px;
}

.botones button {
  padding: 10px 20px;
  background-color: #007bff;
  color: white;
  border: none;
  cursor: pointer;
  margin: 5px;
  border-radius: 5px;
}

.botones button:hover {
  background-color: #0056b3;
}
```

- **PADRE**:
  - Llama a los métodos del hijo.

```typescript
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HijoGaleriaImagenesComponent } from '../hijo-galeria-imagenes/hijo-galeria-imagenes.component';

@Component({
  selector: 'app-padre-galeria-imagenes',
  standalone: true,
  imports: [CommonModule, HijoGaleriaImagenesComponent],
  templateUrl: './padre-galeria-imagenes.component.html',
  styleUrls: ['./padre-galeria-imagenes.component.css']
})
export class PadreGaleriaImagenesComponent {

}
```

```html
<h2>Componente Padre - Galería de Imágenes</h2>
<app-hijo-galeria-imagenes></app-hijo-galeria-imagenes>
```

```css
h2 {
  color: #007bff;
}

button {
  background-color: #007bff;
  color: white;
  padding: 10px;
  border: none;
  cursor: pointer;
  border-radius: 5px;
}

button:hover {
  background-color: #0056b3;
}
```

- **Apoyo visual**:

4. Conclusión

La tarea permitió aplicar de manera práctica los conceptos aprendidos sobre Angular y desarrollo web avanzado. Se lograron crear componentes modulares que interactúan entre sí, y se implementó un diseño responsivo para asegurar la accesibilidad desde diferentes dispositivos. Además, se manejó de manera adecuada la validación de formularios, lo que mejora la experiencia del usuario al interactuar con la aplicación. Este tipo de desarrollo es fundamental para la creación de aplicaciones web modernas y eficientes.

5. Referencias

- Home • Angular. (n.d.). Retrieved January 13, 2025, from https://angular.dev/
- <form>: The Form element - HTML: HyperText Markup Language | MDN. (n.d.). Retrieved January 13, 2025, from https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form
- AngularJS Tutorial. (n.d.). Retrieved January 13, 2025, from https://www.w3schools.com/angular/default.asp
- https://github.com/ANTHONYNESTORVILLARREALMACIAS/Desarrollo-Web-Avanzado-AnthonyV-ArielR.git