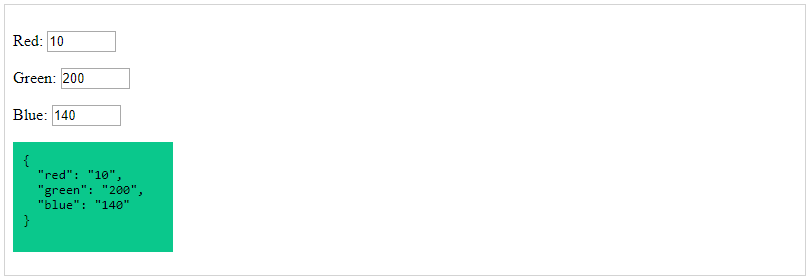
Utilice el enlace de propiedad y el decodificador @Input para pasar datos a un componente, y @Output y EventEmitter para pasar los datos y vincularlos a un evento.

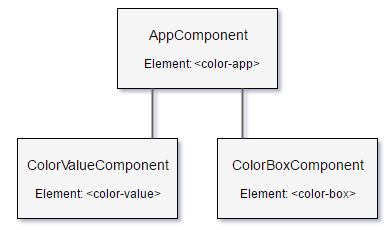
**La aplicación**

Esta es la aplicación que estamos construyendo. Permite cambiar el color de la caja estableciendo los valores rojo, verde y azul en los campos de entrada. Es una aplicación que funciona, así que adelante y pruébalo.



**Los componentes**

La aplicación consta de tres tipos de componentes; *ColorValueComponent* le permite al usuario ingresar el valor de color en un campo de entrada, *ColorBoxComponent* muestra el cuadro de color (junto con los valores rojo, verde y azul), y *AppComponent* actúa como un contenedor para los otros componentes.



Vamos a sumergirnos y ver cómo estos componentes se comunican entre sí.

**Inicialización de cadenas por única vez con @Input**

Utilizamos una inicialización de cadena de una sola vez para pasar una constante de cadena a un componente secundario, pero solo una vez durante el inicio. Aquí pasamos los valores iniciales a las propiedades *label* e *initValue* del elemento personalizado.

<color-value

label="Blue"

initValue="140"

(colorValue)="color.blue=$event">

</color-value>

Esta sintaxis solo permite pasar una constante de cadena y no se realiza detección de cambios.

Para recibir este valor, el componente secundario incluye un decorador @Input en una propiedad, como se muestra en el ejemplo a continuación. Aquí, las propiedades *label* e *initValue* están asociadas a los nombres de las variables coincidentes de los decoradores @Input.

export class ColorValueComponent implements OnInit {

@Input() label: string;

@Input() initValue: string = "0";

...

**El código de la aplicación**

Ahora volvamos a nuestra aplicación. Utilizamos la inicialización de cadenas por única vez en el componente padre para establecer la propiedad *label* en el elemento *<color-value>*. Esta propiedad coincide con la variable de la propiedad *label* en el componente secundario que se ha decorado con *@Input*.

Hacemos lo mismo para *initValue* pero esta vez hacemos coincidir la propiedad *initValue* en el componente secundario.

**app.component.ts**

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

**import** { RedGreenBlue } **from** './red-green-blue';

@Component({

selector: 'color-app',

template: `

<color-value label="Red" initValue="10" (colorValue)="color.red=$event"></color-value>

<color-value label="Green" initValue="200" (colorValue)="color.green=$event"></color-value>

<color-value label="Blue" initValue="140" (colorValue)="color.blue=$event"></color-value>

<color-box [rgb]="color"></color-box>`,

})

**Export** **class** AppComponent {

color: RedGreenBlue = **new** RedGreenBlue();

}

**color-value.component.ts**

**import** { Component, Input, Output, EventEmitter, OnInit } **from** '@angular/core';

@Component({

selector: 'color-value',

template: `

<p>

{{label}}:

<input #color

type="number" min="0" max="255" step="10"

value="{{initValue}}"

(input)="sendColor(color.value)"/>

</p>`,

})

**export** **class** ColorValueComponent **implements** OnInit {

@Input() label: **string**;

@Input() initValue = '0';

@Output() colorValue: EventEmitter<**string**> = **new** EventEmitter();

ngOnInit() {

**this**.sendColor(**this**.initValue);

}

sendColor(color: **string**): **void** {

**this**.colorValue.emit(color);

}

}