

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERIA ESCUELA DE COMPUTACIÓN	
CICLO: 02 2020	GUIA DE LABORATORIO #13 Nombre de la Practica: Introducción a React y Angular Lugar de Ejecución: Centro de cómputo Tiempo Estimado: 2 horas con 30 minutos MATERIA: Lenguajes Interpretado en el Cliente	

I. OBJETIVOS

Que el estudiante:

- Diseñe aplicaciones web utilizando funciones de React.
- Diseñe aplicaciones con navegación.
- Aprender hacer interfaz para que el usuario pueda interactuar con la aplicación.
- Aprender a crear componente personalizado.
- Aprender hacer interfaz para que el usuario pueda interactuar con la aplicación.
- Hacer uso de los componentes en el diseño de interfaz react.

II. INTRODUCCION TEORICA

Qué es React

React es una librería Javascript focalizada en el desarrollo de interfaces de usuario. Así se define la propia librería y evidentemente, esa es su principal área de trabajo. Sin embargo, lo cierto es que en React encontramos un excelente aliado para hacer todo tipo de aplicaciones web, [SPA \(Single Page Application\)](#) o incluso aplicaciones para móviles. Para ello, alrededor de React existe un completo ecosistema de módulos, herramientas y componentes capaces de ayudar al desarrollador a cubrir objetivos avanzados con relativamente poco esfuerzo.

React Component

Un **componente en React** simplemente es una pieza de UI (user interface), que tiene una funcionalidad independiente definida, ya sea este una **barra de búsqueda** es un componente, pues tiene una función independiente, una **botón** llegaría también a ser un componente pues también cumple una función, básicamente, casi todo puede llegar a ser un componente, siempre y cuando sea lógico su agrupamiento. En react, un componente se define usando clases, la estructura básica de un componente es:

```
import React, {Component} from 'react'

class MiPrimerComponente extends Component {
  render() {
    return(
      <h1>Soy un componente...!</h1>
    )
  }
}

export default MiPrimerComponente
```

Para interactuar con los componentes debemos trabajar con los siguientes dos elementos:

Props: Es el apocope de propiedades (del inglés properties).

```
// src/components/Mensaje.js
import React, {Component} from 'react'

class Mensaje extends Component {
  render () {
    const styles = {
      color: this.props.color,
      fontSize: this.props.size
    }
    return (
      <h3 style={styles}>{this.props.contenido}</h3>
    )
  }
}
export default Mensaje
```

State: El State es el estado de un componente.

En resumen los Props son usados para pasar datos de componentes padres a sub componentes, son inmutables y no deberían cambiar y state es usado para datos cambiantes.

```
import React, {Component} from 'react'
import {Button, Loader} from 'semantic-ui-react'
class Cargando extends Component {
  constructor (args) {
    super(args)
    this.state = {
      loading: false
    }
    this.changeLoading = this.changeLoading.bind(this)
  }
  changeLoading () {
    this.setState((prevState, props) {
      return {loading: !prevState.loading}
    })
  }
  render () {
    let contenido
    if (this.state.loading) {
      contenido = <Loader inline active />
    } else {
      contenido = <span>No esta cargando</span>
    }
    return (
      <div>
        {contenido}
      </div>
    )
  }
}
```

Qué es Angular CLI

Dentro del ecosistema de Angular encontramos una herramienta fundamental llamada "Angular CLI" (Command Line Interface). Es un producto que en el momento de escribir este artículo todavía se encuentra en fase beta, pero que ya resulta fundamental para el trabajo con el framework.

los directorios y archivos generados al crear un nuevo proyecto son necesarios para que estas herramientas funcionen. Entre otras cosas tendremos:

- Un servidor para servir el proyecto por HTTP
- Un sistema de live-reload, para que cuando cambiamos archivos de la aplicación se refresque el navegador
- Herramientas para testing
- Herramientas para despliegue del proyecto
- Etc.

Un *Component* controla una zona de espacio de la pantalla que podríamos denominar vista. Un componente es una clase estándar de ES6 decorada con `@Component`.

El componente define propiedades y métodos que están disponibles en su template, pero eso no te da licencia para meter ahí todo lo que te parezca. Es importante seguir una aproximación de diseño S.O.L.I.D., y extraer toda la lógica en servicios para que el controlador solo se encargue de gestionar 1 única cosa: la vista.

El siguiente ejemplo, donde definimos un componente *TodoList*, que se encargará de controlar un listado de tareas para hacer ("to do" tasks).

```
// app/todos/todos-list/todo-list.component.ts
import { OnInit } from '@angular/core';
import { Todo } from '../shared/todo.model';
import { TodoService } from '../shared/todo.service';

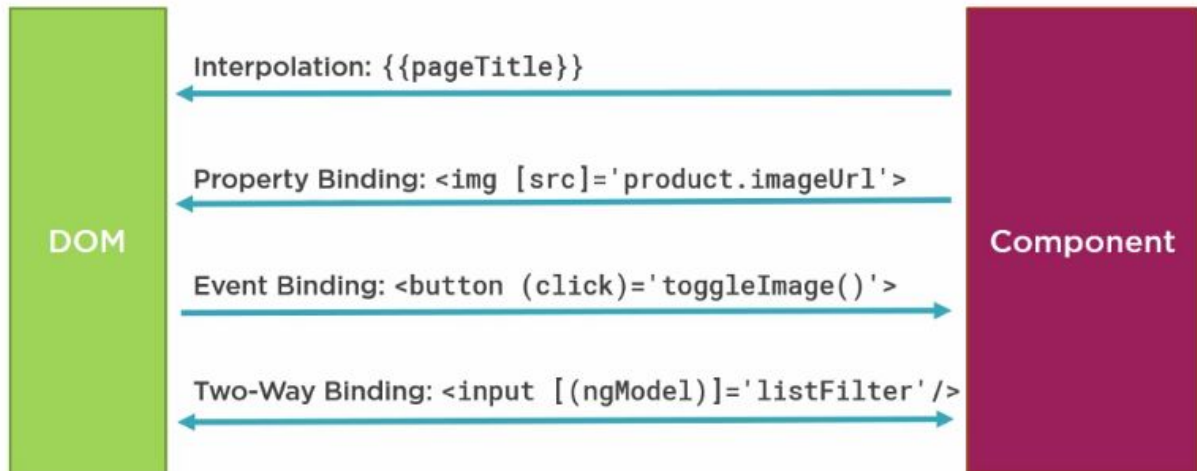
export class TodoListComponent implements OnInit {
  todos: Todo[];
  selectedTodo: Todo;
  constructor(private service: TodoService) {}
  ngOnInit() {
    this.todos = this.service.getTodos();
  }
  selectTodo(todo: Todo) { this.selectedTodo = todo; }
}
```

Como ves, el componente contiene un listado de tareas, una tarea seleccionada, el método para seleccionar la tarea y una llamada a `ngOnInit` para inicializar -gracias al servicio inyectado *TodoService*- el listado de tareas; es de mencionar que el método `ngOnInit` que usa nuestro componente porque implementa la interfaz `OnInit`, se llama cuando se crea el componente y forma parte de las llamadas del ciclo de vida de los componentes.

Los "Data Bindings" o enlace de datos es una técnica general que une una fuente de datos entre el origen de datos y la vista, y se encarga de sincronizarlos. Esto provoca que cada cambio de datos se refleje automáticamente en los elementos que están sincronizados.

Los databinding son la sincronización automática de los datos entre los componentes del modelo y la vista. La vista es una proyección del modelo en todo momento. Cuando el modelo cambia, la vista refleja el cambio, y viceversa. Por eso, debes conocer cómo tratar los datos y su enlace con las vistas.

Es decir nos interesa transmitir las propiedades de los Component al DOM (la vista) para por ejemplo cambiar el color del fondo de la página de blanco a negro. Y también necesitamos que los eventos de la vista sean comunicados al Component, por ejemplo, para implementar un botón con el cual mostrar / ocultar las imágenes de la lista de productos.



III. MATERIALES Y EQUIPO

Para la realización de la guía de práctica se requerirá lo siguiente:

No.	Requerimiento	Cantidad
1	Guía de práctica #13: REACT y Angular	1
2	Computadora con editor HTML y navegadores instalados	1
3	Memoria USB o disco flexible	1

IV. PROCEDIMIENTO

REACT

Ejercicio 1: Realizar aplicación web con react usando componentes

1. Verificar que tenga instalado node versión 6 o superior, con el siguiente comando
node -v

2. Verificar que tenga instalado npm versión 3 o superior con el siguiente comando:
npm -v

NOTA: si no está instalado proceda a instalar el node 6 desde la página oficial.

<https://nodejs.org/en/>

3. Deberemos instalar create-react-app, lo haremos a través de la consola, escribiendo el siguiente comando:

npm install -g create-react-app

1. En consola debemos ubicarnos en el directorio en el cual queremos crear nuestro proyecto react.

2. Para crear nuestro proyecto lo haremos escribiendo en consola:

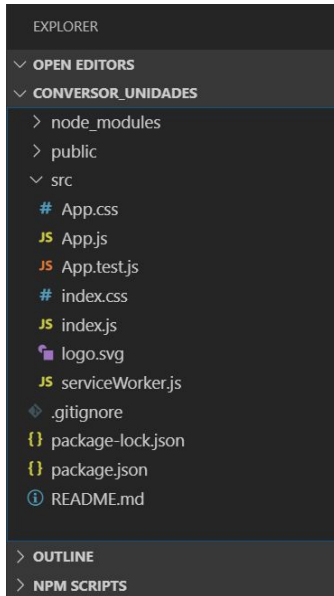
create-react-app conversor_unidades

4. Una vez creado el proyecto inicial, entrar en la carpeta del proyecto con el siguiente comando:

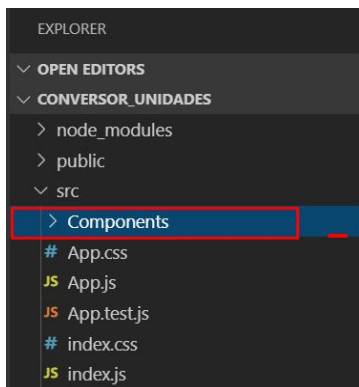
cd conversor

5. Ahora abriremos el nuestro proyecto con visual studio code.

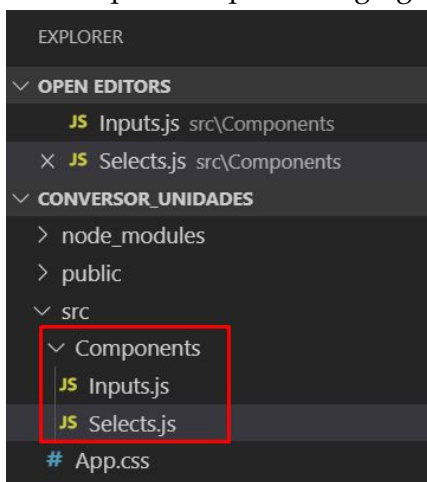
6. Una vez abierto visual studio code, encontrará un listado de archivos similares a este.



- Definiremos nuestro primer componente, agregando una nueva carpeta de nombre Components en la carpeta src.



- Dentro de la carpeta Components, agregar el archivo inputs.js y selects.js, dando clic derecho en la carpeta components agregar nuevo archivo.



9. Agregaremos el siguiente código dentro del archivo inputs.js

```
import React,{Component} from 'react';
import PropTypes from 'prop-types';

class Inputs extends Component{
  constructor(props){
    //recibir datos para componentes
    super(props);
    //estado para capturar valores
    this.state ={textovalor:''}
    //haciendo escuchadores de eventos a traves de bind
    this.Cambiar=this.Cambiar.bind(this);
    this.MostrarValor=this.MostrarValor.bind(this);
  }
  //funciones para capturar datos en componente padre
  Cambiar(e){
    this.setState({textovalor: e.target.value});
  }

  //Para mostrarlo en componente padre
  MostrarValor(){
    return this.state.textovalor;
  }
  render(){
    //creacion de etiqueta input generica reutilizable en todo el proyecto
    return(
      <div>
        <label>{this.props.textolabel}</label>
        <input
          type={this.props.tipoInput}
          value={this.state.textovalor}
          onChange={this.Cambiar}
          required={this.props.esrequerido}
        />
      </div>);
  }
}
```

```
//variables para trabajar con datos
Inputs.propTypes={
  textolabel: PropTypes.string,
  tipoInput:PropTypes.string,
  esrequerido:PropTypes.bool
}

export default Inputs;
```

10. Agregaremos el siguiente código dentro del archivo select.js

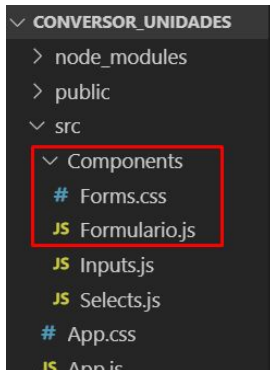
```
import React,{Component} from 'react';
import PropTypes from 'prop-types';
//estructura de datos a recibir para opciones
export const opciones = {
  texto: PropTypes.string,
  valor: PropTypes.string };
class Selects extends Component{
  constructor(props){
    //recibir datos para componentes
    super(props);
    //estado para capturar valor del input
    this.state ={ValorSelect:''}
    this.Cambiar=this.Cambiar.bind(this);
    this.MostrarValor=this.MostrarValor.bind(this);
  }
  //funciones para capturar datos del componente padre
  Cambiar(e){
    this.setState({ValorSelect: e.target.value});
  }
  //Para mostrar datos en componente padre
  MostrarValor(){
    return this.state.ValorSelect;
  }
  render(){
    return(
      <div>
        <label>{this.props.textolabel}</label>
        <select value={this.state.ValorSelect} onChange={this.Cambiar}>
```

```

        <option value="" disabled>seleccione</option>
        {this.props.Options.map((opciones) =>
            <option value={opciones.valor}>{opciones.texto}</option>
        )}
    </select>
</div>);
}
}
Selects.prototype={
    textlabel: PropTypes.string,
    Options:PropTypes.arrayOf(opciones)
}
export default Selects;

```

11. Ya hemos creado los componentes que podemos reutilizar para nuestros formulario, en este ejemplo haremos un formulario para conversión de longitudes, así que crearemos el componente padre de nombre formulario.js y el archivo para darle estilo al formulario Forms.css dentro de la carpeta components:



12. Dentro del archivo Forms.css, escribir el siguiente código:

```

/* Fonts*/
@import 'https://fonts.googleapis.com/css?family=Lato:100,300,400,700,900';
@import
'https://fonts.googleapis.com/css?family=Playfair+Display:400,700,900';

.form{
    width:340px;
    height:440px;
    background:#e6e6e6;
    border-radius:8px;
    box-shadow:0 0 40px -10px #000;
}

```



```
margin:calc(50vh - 220px) auto;
padding:20px 30px;
max-width:calc(100vw - 40px);
box-sizing:border-box;
font-family: 'Lato', sans-serif;
position:relative}

h2{
  font-family: 'Playfair Display';
  font-weight: 600;
  font-size: 36px;
  color:#78788c;
  border-bottom:3px solid #78788c
}

label {
  display: block;
  text-transform: uppercase;
  font-size: 11px;
  color: hsla(0, 0, 0, .6);
  margin-bottom: 5px;
  font-weight: bold;
  font-family: Inconsolata;
}

input,select{
  width:100%;
  padding:10px;
  box-sizing:border-box;
  background:none;
  outline:none;
  resize:none;
  border:0;
  font-family:'Montserrat',sans-serif;
  transition:all .3s;
  border-bottom:2px solid #bebed2}

input:focus{
```

```
border-bottom:2px solid #78788c}

button{
  float:right;
  padding:8px 12px;
  margin:30px 0 0;
  font-family:'Montserrat',sans-serif;
  border:2px solid #78788c;
  background:0;
  color:#5a5a6e;
  cursor:pointer;
  transition:all .3s}

button:hover{
  background:#78788c;
  color:#fff}

.resultado{
  display: block;
  color:#47BFAA;
  text-transform: uppercase;
  margin-bottom: 5px;
  font-size: 12px;
  font-weight: 900;
}
```

13. Dentro del archivo Formulario.js deberemos ir creando el siguiente código:

- Creando el formulario inicializando y estructura.

```
import React,{Component} from 'react';
import './Forms.css';

class App extends Component{
  render(){
    return(
      <div className="form">
        </div>
    );
  }
}
```

```
}  
  
export default App;
```

- Ya tenemos lista nuestra estructura y ahora agregaremos los componentes primero el componente para capturar el valor de conversión, agregar dentro del archivo Formulario.js el código resaltado en letras rojas:

```
import React,{Component} from 'react';  
import Inputs from './Inputs';  
import './Forms.css';  
class App extends Component{  
  
  render(){  
  
    return(  
      <div className="form">  
        <h2>Conversor de Longitud</h2>  
        <Inputs  
          tipoInput="text"  
          textolabel= "Valor:"  
          ref="numero"  
          esrequerido={true}  
        />  
  
        </div>  
      );  
    }  
  }  
  
  export default App;
```

- Ahora agregaremos las opciones de conversión proporcionada por la página que se mostrarán en el select, agregar dentro del archivo Formulario.js el código resaltado en letras rojas:

```
import React,{Component} from 'react';  
import Inputs from './Inputs';  
import Selects from './Selects';  
import './Forms.css';  
class App extends Component{  
  render(){  
    // valores para el arreglo de objetos a nuestra lista  
    const medidas=[
```

```

        {valor:'cm',texto:'Centimetro'},
        {valor:'m',texto:'Metro'},
        {valor:'mm',texto:'Milimetro'},
        {valor:'km',texto:'Kilometro'}
    ];

    return(
    <div className="form">
    <h2>Conversor de Longitud</h2>
    <Inputs
    tipoInput="text"
    textolabel= "Valor:"
    ref="numero"
    esrequerido={true}
    />

    <Selects textlabel= "Longitud Origen:"
    Options={medidas} ref="uni1"/>
    <Selects textlabel= "Longitud Destino:"
    Options={medidas} ref="uni2"/>
    </div>
    );
    }
}

export default App;

```

14. Ya tenemos listo nuestro código que muestra los componentes que nosotros hemos creado, ahora debemos crear eventos de clic para que realice la conversión para ello realizaremos los siguientes cambios:
- Crear el botón para llamar la función conversión y agregar label donde mostraremos el resultado dentro de nuestra estructura,agregar dentro del archivo Formulario.js el código resaltado en letras rojas:

```

import React,{Component} from 'react';
import Inputs from './Inputs';
import Selects from './Selects';
import './Forms.css';
class App extends Component{

    render(){
    //valores para el arreglo de objetos a nuestra lista
    const medidas=[
    {valor:'cm',texto:'Centimetro'},
    {valor:'m',texto:'Metro'},

```

```

        {valor:'mm',texto:'Milimetro'},
        {valor:'km',texto:'Kilometro'}
    ];

    return(
        <div className="form">
        <h2>Conversor de Longitud</h2>
        <Inputs
            tipoInput="text"
            textolabel= "Valor:"
            ref="numero"
            esrequerido={true}
        />

        <Selects textlabel= "Longitud Origen:" Options={medidas} ref="uni1"/>
        <Selects textlabel= "Longitud Destino:" Options={medidas} ref="uni2"/>
        <label className="resultado">{this.state.resultado}</label>
        <button onClick={this.conversion}>Conversion</button>
        </div>
    );
}

}

export default App;

```

- Crear eventos de escucha y función conversión, agregar dentro del archivo Formulario.js el código resaltado en letras rojas:

```

import React,{Component} from 'react';
import Inputs from './Inputs';
import Selects from './Selects';
import './Forms.css';
class App extends Component{
    constructor(){
        super();
        this.conversion=this.conversion.bind(this);
        this.state={ resultado:'' }
    }

    render(){
        // valores para el arreglo de objetos a nuestra lista
        const medidas=[
            {valor:'cm',texto:'Centimetro'},
            {valor:'m',texto:'Metro'},

```

```

        {valor:'mm',texto:'Milimetro'},
        {valor:'km',texto:'Kilometro'}
    ];

    return(
        <div className="form">
        <h2>Conversor de Longitud</h2>
        <Inputs
        tipoInput="text"
        textolabel= "Valor:"
        ref="numero"
        esrequerido={true}
        />

        <Selects textlabel= "Longitud Origen:" Options={medidas} ref="uni1"/>
        <Selects textlabel= "Longitud Destino:" Options={medidas} ref="uni2"/>
        <label className="resultado">{this.state.resultado}</label>
        <button onClick={this.conversion}>Conversion</button>
        </div>
    );
}

conversion(e){
    var u1,u2,v,conversion;
    v=this.refs.numero.MostrarValor();
    u1=this.refs.uni1.MostrarValor();
    u2=this.refs.uni2.MostrarValor();

    switch(u1){
    case "cm":
    switch(u2){
    case "m":
    conversion= parseFloat(v)/100;
    break;
    case "mm":
    conversion= parseFloat(v)*10;
    break;
    case "km":
    conversion= parseFloat(v)/100000;
    break;
    default:
    this.setState({
    resultado:"Su conversion es a la misma unidad de origen"
    })
    break;
    }
    break;
}
}

```

```

case "m":
switch(u2){
case "cm":
conversion= parseFloat(v)*100;
break;
case "mm":
conversion= parseFloat(v)*1000;
break;
case "km":
conversion= parseFloat(v)/1000;
break;
default:
this.setState({
    resultado:"Su conversion es a la misma unidad de origen"
})
break;
}
break;
case "mm":
switch(u2){
case "cm":
conversion= parseFloat(v)/10;
break;
case "m":
conversion= parseFloat(v)/1000;
break;
case "km":
conversion= parseFloat(v)/0.000001;
break;
default:
this.setState({
    resultado:"Su conversion es a la misma unidad de origen"
})
break;
}
break;
case "km":
switch(u2){
case "cm":
conversion= parseFloat(v)*100000
break;
case "mm":
conversion= parseFloat(v)*1000000
break;
case "m":
conversion= parseFloat(v)*1000
break;

```

```

        default:
        this.setState({
        resultado:"Su conversion es a la misma unidad de origen"
        })
        break;
        }
        break;
        default:
        this.setState({
        resultado:"Seleccione unidades de longitud"
        })
        break;
        }
        if(conversion!==undefined){
        this.setState({
        resultado:"Su conversion de "+u1+" a "+u2+" es:"+conversion +" "+u2
        })
        }
    }
}

export default App;

```

15. Para ver los cambios realizados en nuestro proyecto react, debemos de ir a modificar el archivo App.js, el cual debe quedar de la siguiente forma:

```

import React,{Component} from 'react';
import './App.css';
import Formulario from './Components/Formulario';

class App extends Component{

  render(){
    return(
      <div className="App">

        <Formulario></Formulario>
      </div>
    );
  }
}

```



```
export default App;
```

16. Probar pagina desde consola digite el siguiente comando:

`npm start`

17. Nuestro navegador deberá mostrar el siguiente diseño:

Ejercicio 2: Realizar aplicación web con react usando JSON

1. Crear nuevo proyecto de nombre productos:

`create-react-app productos`

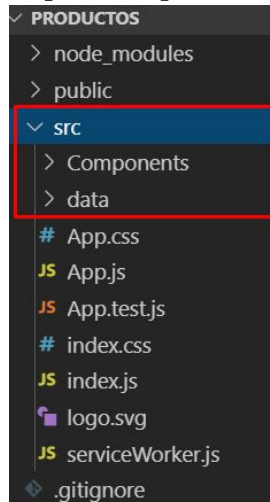
2. Una vez creado el proyecto inicial, entrar en la carpeta del proyecto con el siguiente comando:

`cd productos`

3. Para este ejemplo usaremos bootstrap así que lo agregaremos a nuestro proyecto con el siguiente comando:

`npm install --save bootstrap`

4. Para este ejemplo crearemos las siguientes carpetas: **Components** y **data**, dentro de nuestra carpeta src que se encuentra nuestro proyecto:



5. Dentro de nuestra carpeta **data**, agregaremos el archivo json, proporcionado en los recursos entregados.

6. Dentro de la carpeta components, agregaremos dos archivos:

- card.js que deberá presentar la siguiente estructura

```
import React, {Component} from 'react';
import PropTypes from 'prop-types';
class Card extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div class="card col-sm-4">
        <h5 class="card-header"> {this.props.MarcaProducto} </h5>
        <div class="card-body">
          <h5 class="card-title">{this.props.NombreProducto} </h5>
          <p class="card-text">{this.props.PresentacionProducto}</p>
        </div>
      </div>
    );
  }
}
Card.propTypes = {
  MarcaProducto: PropTypes.string,
  NombreProducto: PropTypes.string,
  PresentacionProducto: PropTypes.string
}
export default Card;
```

- Productos.js que deberá presentar la siguiente estructura

```
import React, {Component} from 'react';
//agregamos referencia al archivo json
import ListaProductos from '../data/productos.json';
import Card from './Card';
class Productos extends Component {
  render() {
    //Recorrido del json para mostrar los datos en componente Card
    return (
```

```

        <div className="row">
          {ListaProductos.map((ProductosDetalle,index) => (
            <Card
              MarcaProducto= {ProductosDetalle.value.marca}
              NombreProducto={ProductosDetalle.value.nombre}
              PresentacionProducto={ProductosDetalle.value.presentacion}
            />
          ))}
        </div>
      );
    }
  }
  export default Productos;

```

7. para poder utilizar bootstrap, deberemos realizar los siguientes cambios en nuestro archivo App.js.

```

import React,{Component} from 'react';
import './App.css';
//Agregando componente Productos
import Productos from './Components/Productos';
//Agregando bootstrap
import 'bootstrap/dist/css/bootstrap.min.css';
class App extends Component {
  render(){
    return (
      <div className="container">
        <div class="jumbotron jumbotron-fluid">
          <div class="container">
            <h1 class="display-4">Productos</h1>
            <p class="lead">Estos productos son leídos desde un archivo json local.</p>
          </div>
        </div>
        <Productos></Productos>
      </div>
    );
  }
}
export default App;

```

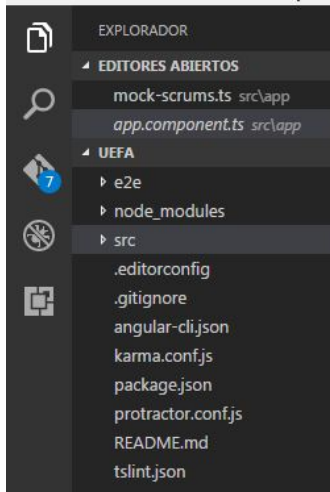
8. Ya con estos cambios estamos listos para probar la aplicación y deberemos ver lo siguiente:

<h2>Productos</h2> <p>Estos productos son leídos desde un archivo json local.</p>		
<p>SCHNEIDER</p> <p>Cerveza Botella Schneider x 1 Un</p> <p>1.0 un</p>	<p>SIN MARCA</p> <p>REJILLA COMPACTA COL</p> <p>1.0 ud</p>	<p>SIN MARCA</p> <p>REJ.SECAVAJ.43X55CM</p> <p>1.0 ud</p>
<p>SIN MARCA</p> <p>REJILLA AUTO 50X50CM</p> <p>1.0 ud</p>	<p>SIN MARCA</p> <p>LECHE L.V.ENTERA</p> <p>1.0 lt</p>	<p>SIN MARCA</p> <p>Trapo Piso Blanco Consorcio x 1 Un</p>

ANGULAR

Ejercicio 1: Realizar aplicación web con angular CLI

1. Verificar que tenga instalado node versión 6 o superior, con el siguiente comando
`node -v`
2. Verificar que tenga instalado npm versión 3 o superior con el siguiente comando:
`npm -v`
NOTA: si no está instalado proceda a instalar el node 6 desde la página oficial.
<https://nodejs.org/en/>
3. Deberemos instalar angular-cli, lo haremos a través de la consola, escribiendo el siguiente comando:
`npm install -g angular-cli`
1. En consola debemos ubicarnos en el directorio en el cual queremos crear nuestro proyecto angular.
2. Para crear nuestro proyecto lo haremos escribiendo en consola:
`ng new uefa`
4. Una vez creado el proyecto inicial, entrar en la carpeta del proyecto con el siguiente comando:
`cd uefa`
5. Ahora abriremos el nuestro proyecto con visual studio code, digitando la siguiente instrucción en la consola:
`Code .`
6. Una vez abierto visual studio code, encontrara un listado de archivos similares a este.



- Definiremos nuestra primera clase, dentro de la carpeta `src` y dentro de la carpeta `app`, crear el archivo **scrum.ts**, y escribir el siguiente código:

```
export class Scrum{
  id:number;
  name:string;
  players:string[];
}
```

- Siempre dentro de la carpeta `app`, crearemos el archivo **mock-scrums.ts**, donde tendremos un arreglo de equipos de la Liga de Campeones de la UEFA. Escriba el siguiente código:

```
import {Scrum} from './scrum';

export const SCRUMS: Scrum[]=[
  {id:1, name:'Real Madrid',players:['James Rodríguez','Gareth Bale','Cristiano Ronaldo']},
  {id:2, name:'Barcelona',players:['Lionel Messi','Neymar','Luis Suarez']},
  {id:3, name:'Mónaco',players:['Marcos Lopes','Hélder Costa']},
  {id:4, name:'Sevilla',players:['Paulo Henrique','Samir Nasri']},
  {id:5, name:'Arsenal',players:['Hiroshi']},
  {id:6, name:'Juventus',players:['Gonzalo Higuaín','Paulo Dybala']},
  {id:7, name:'Bayer',players:['Javier Hernández','Hakan']},
  {id:8, name:'Dortmund',players:['Marco Reus','Pierre Emerick','Mario Götze']},
  {id:9, name:'Leicester City',players:['Jamie Vardy','Riyad Mahrez']},
  {id:10,name:'Napoli',players:['Arkadiusz Milik','Marek Hamsík']}
];
```

- Dentro de la carpeta `app`, crearemos el servicio de angular y obtendremos los datos a través del archivo **scrum.service.ts**, crearlo y escribir el siguiente código dentro de él:

```
import{Injectable} from '@angular/core';
import {Scrum} from './scrum';
import{SCRUMS} from './mock-scrums';

@Injectable()
```

```
export class ScrumService{
  getScrums():Promise<Scrum[]>{
    return Promise.resolve(SCRUMS);
  }

  getScrumsSlowly(): Promise<Scrum[]> {
    return new Promise(resolve => {
      // Simulate server latency with 2 second delay
      setTimeout(() => resolve(this.getScrums()), 2000);
    });
  }
}
```

10. Ahora que ya está todo listo para montar la lista de equipos, empezaremos a crear la vista de detalle de cada equipo, creando el archivo **scrum-detail.component.ts** digite el siguiente código:

```
import { Component, Input } from '@angular/core';
import { Scrum } from './scrum';

@Component({
  selector: 'my-scrum-detail',
  template: `
<div *ngIf="scrum">
  <h2>{{scrum.name}} Detalles:</h2>
  <div>
    <label>id:</label>{{scrum.id}}
  </div>
  <div>
    <label>nombre:</label>
    <input [(ngModel)]="scrum.name" placeholder="name">
  </div>
  <div>
    <label>Jugadores:</label>
    <ul>
      <li *ngFor="let item of scrum.players">
        <span>{{item}}</span>
      </li>
    </ul>
  </div>
</div>`
})
export class ScrumDetailComponent {
  @Input()
  scrum:Scrum;
}
```

11. Para generar la lista dinámicamente a partir de nuestro servicio debemos consumir nuestro archivo mock para ello abriremos el archivo **app.component.ts**, digite el siguiente código:

```
import { Component, OnInit } from '@angular/core';
import { Scrum } from './scrum';
import { ScrumService } from './scrum.service';
```

```
@Component({
  selector: 'app-root',
  template: `
<div class="container">
  <div class="jumbotron">
    <div class="media">
      <div class="media-left">
        <a href="#">

src="https://1.bp.blogspot.com/-EhgIDI-lkvc/VeQKGu13JOI/AAAAAAAAALK/stFLi1Yh81I/s320/Champions%2BLeague.png">
        </a>
      </div>
      <div class="media-body">
        <h1>{{title}}</h1>
      </div>
    </div>
    <div class="row">
      <div class="col-sm-4">
        <h2>Equipos</h2>
        <ul class="scrums">
          <li *ngFor="let scrum of scrums"
            [class.selected]="scrum === selectedScrum"
            (click)="onSelect(scrum)">
            <span class="badge">{{scrum.id}}</span> {{scrum.name}}
          </li>
        </ul>
      </div>
      <div class="col-sm-8">
        <my-scrum-detail [scrum]="selectedScrum"></my-scrum-detail>
      </div>
    </div>
  </div>
  `
  styleUrls: ['./app.component.css'],
  providers:[ScrumService]
})

export class AppComponent implements OnInit{
  title = 'Liga de Campeones de la UEFA';
  scrums : Scrum[];
  selectedScrum:Scrum;
```

```

    constructor(private scrumservice:ScrumService){ }

    getScrums():void{
        this.scrumservice.getScrums().then(scrums => this.scrums = scrums);
    }

    ngOnInit():void{
        this.getScrums();
    }

    onSelect(scrum: Scrum): void {
        this.selectedScrum = scrum;
    }

}

```

12. Ahora debemos modificar el archivo **app.module.ts** el contiene todos nuestros modulos y debe verse de la siguiente forma:

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';

import { AppComponent } from './app.component';
import { ScrumDetailComponent } from './scrum-detail.component';

@NgModule({
  declarations: [
    AppComponent,
    ScrumDetailComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

13. Ahora para que funcione nos falta el archivo index.html (en la raíz del proyecto), escribir el siguiente código:

```

<!doctype html>
<html>

```

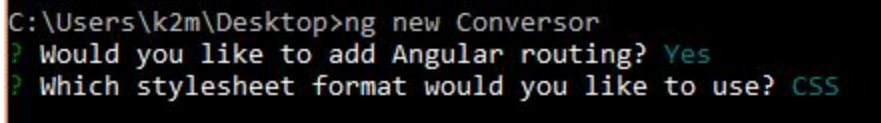


```
<head>
  <meta charset="utf-8">
  <title>Uefa</title>
  <base href="/">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/css/bootstrap.min.css"
integrity="sha384-rwvOIResjU2yc3z8GV/NPeZWA56rSmLldC3R/AZzGRnGxQQKnKkoFVhFQh
NUwEyJ" crossorigin="anonymous">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

14. Para que nuestra pagina se vea con un buen diseño agregaremos en código css dentro del archivo app.component.css en el cual digitaremos el siguiente código:
15. Probar pagina desde consola digite el siguiente comando:
ng server

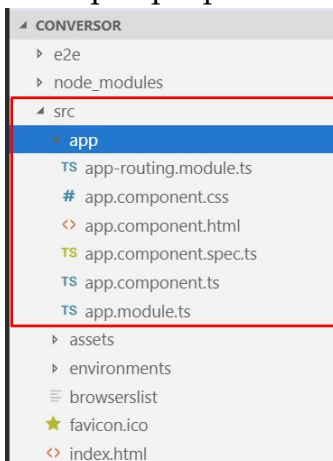
Ejercicio 2: Conversor de longitud

1. Cree un proyecto angular llamado "Conversor".

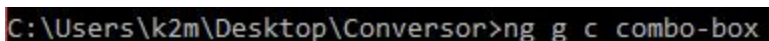


```
C:\Users\k2m\Desktop>ng new Conversor
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
```

2. Abra la carpeta del proyecto "Conversor" dentro de visual code.
3. Verifique que presente la siguiente estructura:

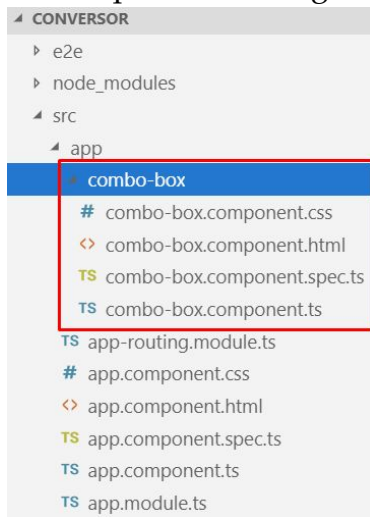


4. Desde la terminal procederemos a crear un nuevo componente con el siguiente comando:



```
C:\Users\k2m\Desktop\Conversor>ng g c combo-box
```

5. Deberá presentar la siguiente estructura:



6. Ahora abrimos el documento “combo-box.component.html” y lo modificaremos con el siguiente código:

```
<div class="header">
  <h1>Conversion de Longitud</h1>
  <p>Ingrese los datos solicitados para realizar la conversion</p>
</div>
<label>Valor:</label>
<input type="text" [(ngModel)]="valorcm">
<label>Longitud:</label>
<select [(ngModel)]="opcionSeleccionado" (change)="capturar()">
  <option value="Selecciona">Selecciona</option>
  <option *ngFor="let u of unidades" value="{{u}}" >
    {{u}}
  </option>
</select>
<label [hidden]="opcionSeleccionado == 'Selecciona'">Su valor en {{ opcionSeleccionado }} es:
{{valorconversion}}</label>
```

7. Ahora debemos moveremos al archivo combo-box.component.ts y debe quedar de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
//Agregar las siguientes modulos
import {BrowserModule}from '@angular/platform-browser'
import {FormsModule} from '@angular/forms'

@Component({
  selector: 'app-combo-box',
  templateUrl: './combo-box.component.html',
  styleUrls: ['./combo-box.component.css']
})
export class ComboBoxComponent implements OnInit {
  //crear las siguientes variables
```

```

unidades;
opcionSeleccionado: string ;
valorcm:number;
valorconversion:number;

constructor() {

}

ngOnInit() {
  //inicializar las variables
  this.unidades = ["Pulgada","Metro","Kilometro"];
  this.opcionSeleccionado="Selecciona";
  this.valorcm=0;
  this.valorconversion=0;
}
//funcion que realiza los calculos
capturar() {
  switch(this.opcionSeleccionado){
    case'Pulgada':
      this.valorconversion=this.valorcm* 0.39370;
      break;
    case'Metro':
      this.valorconversion=this.valorcm/100;
      break;
    case'Kilometro':
      this.valorconversion=this.valorcm/100000;
      break;
  }
}
}
export class AppComponent {
}

```

- Crearemos nuestras reglas CSS para nuestra agregando el siguiente código en el archivo combo-box.component.css

```

.button {
  background-color: #4CAF50; /* Green */
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
}

input,select {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
}

```

```

    box-sizing: border-box;
    font-size: 2vw;
  }
  label{
    font-size: 2vw;
  }
  .header {
    padding: 40px;
    text-align: center;
    background: #1abc9c;
    color: white;
    font-size: 30px;
  }

```

9. Para poder trabajar con la captura de datos desde el formulario debemos realizar los siguientes cambios en el archivo en app.module.ts

- Primero agregar el modulo de formulario de angular

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ComboBoxComponent } from './combo-box/combo-box.component';
//agregar modulo de formulario
import { FormsModule } from '@angular/forms';

```

- Finalmente importar el modulo al proyecto

```

1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { ComboBoxComponent } from './combo-box/combo-box.component';
7 //agregar modulo de formulario
8 import { FormsModule } from '@angular/forms';
9
10 @NgModule({
11   imports: [
12     BrowserModule,
13     AppRoutingModule,
14     FormsModule
15   ],
16   declarations: [
17     AppComponent,
18     ComboBoxComponent
19   ],
20   providers: [],
21   bootstrap: [AppComponent]
22 })
23 export class AppModule {}

```

10. Ahora solo debemos agregar nuestro componente a la aplicación, y eso lo haremos en el archivo app.component.html, donde agregaremos el siguiente código:

```
<app-combo-box></app-combo-box>
```

11. Proceda a levantar los servicios de angular y deberá presentarse la siguiente pantalla:

Conversion de Longitud

Ingrese los datos solicitados para realizar la conversion

Valor:

Longitud:

Pulgada

Su valor en Pulgada es: 0.3937

Ejercicio 3: Validación de formularios

1. Cree un nuevo proyecto angular llamado "Alumnos"
2. Procederemos a crear nuevo componente llamado "Registro".
3. Ahora abrimos el documento "Registro.component.html" y lo modificaremos con el siguiente código:

```
<div class="header">
  <h1>Datos de Alumno</h1>
  <p>Ingrese los datos de los alumnos</p>
</div>
<label>Nombre:</label>
<input type="text" [(ngModel)]="nombre">
<label>Edad:</label>
<input type="text" [(ngModel)]="edad">
<button class="button" (click)="ingresar()">Ingresar</button>
<br/><br/>
<table [hidden]="contador == 0">
  <thead>
    <tr>
      <th>Nombre</th>
      <th>Edad</th>
      <th>Mayor de Edad</th>
    </tr>
  </thead>

  <tbody>
    <tr *ngFor="let talumno of registro">
      <td>{{talumno.nombre}}</td>
      <td>{{talumno.edad}}</td>
      <td>{{talumno.mayor}}</td>
    </tr>
  </tbody>
</table>
```

4. Ahora debemos moveremos al archivo Registro.component.ts y debe quedar de la

siguiente manera:

```
import { Component, OnInit } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser'
@Component({
  selector: 'app-registro',
  templateUrl: './registro.component.html',
  styleUrls: ['./registro.component.css']
})
export class RegistroComponent implements OnInit {
  registro=[];
  alumno:any;
  nombre:string;
  mayor:string;
  edad:number;
  contador:number;
  constructor() { }

  ngOnInit() {
    this.edad=0;
    this.nombre="";
    this.contador=0;
  }

  ingresar(){
    if(this.edad>0 && this.edad<18){
      this.mayor='No'
    }else{
      this.mayor='Si'
    }
    this.alumno={"nombre":this.nombre,"edad":this.edad,"mayor":this.mayor};
    this.registro.push(this.alumno);
    this.contador++;
  }
}
```

5. Crearemos nuestras reglas CSS para nuestra agregando el siguiente código en el archivo combo-box.component.css

```
.button {
  background-color: #4CAF50; /* Green */
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
}

input,select {
  width: 100%;
```

```
padding: 12px 20px;
margin: 8px 0;
box-sizing: border-box;
font-size: 2vw;
}
label{
  font-size: 2vw;
}
.header {
padding: 40px;
text-align: center;
background: #1abc9c;
color: white;
font-size: 30px;
}
```

6. Para poder trabajar con la captura de datos desde el formulario debemos realizar los siguientes cambios en el archivo en app.module.ts

- Primero agregar el modulo de formulario de angular

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { RegistroComponent } from './registro/registro.component';
import { FormsModule } from '@angular/forms';
@NgModule({
  declarations: [
    AppComponent,
    RegistroComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule, FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- Finalmente importar el modulo al proyecto

```
9  declarations: [
10  AppComponent,
11  RegistroComponent
12  ],
13  imports: [
14  BrowserModule,
15  AppRoutingModule, FormsModule
16  ],
17  providers: [],
18  bootstrap: [AppComponent]
19  })
20  export class AppModule { }
21
```

7. Ahora solo debemos agregar nuestro componente a la aplicación, y eso lo haremos en el archivo app.component.html, donde agregaremos el siguiente código:

```
<app-registro></app-registro>
```

8. Proceda a levantar los servicios de angular y deberá presentarse la siguiente

pantalla:

Datos de Alumno

Ingrese los datos de los alumnos

Nombre:

Pedro Picapiedra

Edad:

15

Ingresar

Nombre	Edad	Mayor de Edad
Karens Medrano	50	Si
Rafael Torres	90	Si
Pedro Picapiedra	15	No

Ejercicio 4: Validación de formularios

12. Cree un proyecto angular llamado “Formulario”.
13. Desde la terminal procederemos a crear un nuevo componente de nombre Cliente:
ng generate component Cliente
14. Desde terminal procedemos a crear una nueva clase con nombre Cliente:
ng generate class Cliente
15. Dentro del proyecto instale bootstrap 4 con el siguiente comando:
npm install bootstrap jquery --save
16. Debemos configurar Bootstrap y JQuery en el archivo *angular.json*:

```
"styles": [
  "src/styles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [
  "node_modules/jquery/dist/jquery.min.js",
  "node_modules/bootstrap/dist/js/bootstrap.min.js"
]
```

17. Agregamos el selector app-cliente en el archivo app.component.html:

```
<div class="container">
  <div class="row">
    <div class="col-sm-4">
      <app-clientes></app-clientes>
    </div>
  </div>
</div>
```

18. Debemos importar FormsModule en el archivo AppModule.ts


```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ClientesComponent } from './clientes/clientes.component';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    AppComponent,
    ClientesComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
})
```

19. Modificar el archivo Cliente.ts:

```
export class Cliente {
  constructor(
    public id: number,
    public nombre: string,
    public nacionalidad: string,
    public edad?: number
  ) {}
}
```

20. Modificar el archivo Cliente.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { Cliente } from '../cliente'

@Component({
  selector: 'app-clientes',
  templateUrl: './clientes.component.html',
  styleUrls: ['./clientes.component.css']
})
export class ClientesComponent implements OnInit {
  nacionalidad = ['', 'El Salvador', 'Mexico', 'Rusia', 'Mongolia'];
  cliente = new Cliente(1, '', '', 23);
  enviar=false;
  constructor() { }
  ngOnInit() {
  }
  onSubmit(){
    this.enviar=true;
  }
}
```

21. Modificar el template del componente cliente:

```
<div>
  <h2>Formulario Cliente</h2>
  <form>

    <div class="form-group">
      <label for="nombre">Nombre</label>
      <input type="text" class="form-control" id="nombre">
    </div>

    <div class="form-group">
      <label for="edad">Edad</label>
      <input type="text" class="form-control" id="edad">
    </div>

    <div class="form-group">
      <label for="nacionalidades">Nacionalidad</label>
      <select class="form-control" id="nacionalidades" required>
        <option *ngFor="let nacion of nacionalidad" [value]="nacion">{{nacion}}</option>
      </select>
    </div>

    <button type="submit" class="btn btn-success">Submit</button>
  </form>
</div>
```

22. Para usar ngModel en directivas de two-way realizamos el siguiente cambio:

```
<div>
  <h2>Formulario Cliente</h2>
  <form #clienteform="ngForm">

    <div class="form-group">
      <label for="nombre">Nombre</label>
      <input type="text" class="form-control" id="nombre" required [(ngModel)]="cliente.nombre"
name="nombre" >
    </div>

    <div class="form-group">
      <label for="edad">Edad</label>
      <input type="text" class="form-control" id="edad" [(ngModel)]="cliente.edad" name="edad">
    </div>

    <div class="form-group">
      <label for="nacionalidades">Nacionalidad</label>
      <select class="form-control" id="nacionalidades"
        required [(ngModel)]="cliente.nacionalidades" name="nacionalidades" >
        <option *ngFor="let nacion of nacionalidad" [value]="nacion">{{nacion}}</option>
      </select>
    </div>

    <button type="submit" class="btn btn-success"
[disabled]="!clienteform.form.valid">Submit</button>
  </form>
</div>
<div>{{cliente|json}}</div>
```

23. Realice las pruebas respectivas en la aplicación.

24. Anote

los

resultados: _____

25. Nuestra aplicación ya funciona correctamente para ello modificaremos primero el archivo cliente.component.css con el siguiente código:

```
.ng-valid[required], .ng-valid.required {
  border-left: 5px solid rgba(32, 77, 32, 0.623);
}

.ng-invalid:not(form) {
  border-left: 5px solid rgb(148, 27, 27);
}
```

26. Modificaremos el archivo cliente.component.html con el siguiente código:

```
<div [hidden]="enviar" >
  <h2>Formulario Cliente</h2>
  <form (ngSubmit)="onSubmit()" #clienteform="ngForm">

    <div class="form-group">
      <label for="nombre">Nombre</label>
      <input type="text" class="form-control" id="nombre" required [(ngModel)]="cliente.nombre"
name="nombre" #name="ngModel">
      <div [hidden]="name.valid || name.pristine">
        Nombre es requerido
      </div>
    </div>

    <div class="form-group">
      <label for="edad">Edad</label>
      <input type="text" class="form-control" id="edad" [(ngModel)]="cliente.edad" name="edad">
    </div>

    <div class="form-group">
      <label for="nacionalidades">Nacionalidad</label>
      <select class="form-control" id="nacionalidades"
        required [(ngModel)]="cliente.nacionalidades" name="nacionalidades"
#nacionalidades="ngModel">
        <option *ngFor="let nacion of nacionalidad" [value]="nacion">{{nacion}}</option>
      </select>
      <div [hidden]="nacionalidades.valid || nacionalidades.pristine" class="alert alert-danger">
        Nacionalidad es requerida
      </div>
    </div>

    <button type="submit" class="btn btn-success"
[disabled]="!clienteform.form.valid">Submit</button>
  </form>
</div>

<div [hidden]="!enviar">
  <h2>Enviar Cliente</h2>
  <hr>
  <div><span class="badge badge-info">Id</span> -> {{cliente.id}}</div>
  <div><span class="badge badge-info">Nombre</span> -> {{cliente.nombre}}</div>
```

```
<div><span class="badge badge-info">Edad</span> -> {{cliente.edad}}</div>
<div><span class="badge badge-info">Nacionalidad</span> -> {{cliente.nacionalidades}}</div>
<br>
<button class="btn btn-primary" (click)="enviar=false">Edit</button>
</div>
```

V. DISCUSIÓN DE RESULTADOS

1. Tomando como base el ejemplo 2, realice nuevamente el mismo ejemplo utilizando React Bootstrap, puede apoyarse en la siguiente documentación para la implementación de este módulo.
<https://react-bootstrap.github.io/>

VII. BIBLIOGRAFÍA

- Flanagan, David. JavaScript La Guía Definitiva. 1ra Edición. Editorial ANAYA Multimedia. 2007. Madrid, España.
- Tom Negrito / Dori Smith. JavaScript y AJAX para diseño web. Editorial Pearson Prentice Hall. Madrid, España. 2007.