

# C4PI DOCUMENTACION

Jorge Vicente Ramiro

DAW2

## Tabla de contenido

Introducción .....	1
Descripción del proyecto.....	1
Breve introducción a la clasificación de la personalidad en colores.....	1
¿Que es C4PI?.....	2
¿ Por que hacer este proyecto ?.....	2
Planteamiento.....	2
Ciclo de vida .....	3
Análisis.....	3
Tecnologías aplicadas en el proyecto.....	3
Pagina web SPA.....	3
Desarrollo .....	5
Backend .....	5
Template driven forms.....	10
Pantalla de inicio .....	10
Pantalla de registro .....	15
Pantalla principal.....	17
Seguridad.....	43
Sanitización de las interpolaciones de datos con Angular .....	43
Bindeo de datos.....	43
Hasheo de contraseñas .....	43
Fichero de environment.....	44
Unsubscribe.....	45
Bibliografía .....	46

## Introducción

### Descripción del proyecto

Breve introducción a la clasificación de la personalidad en colores.

La clasificación de la personalidad en diferentes colores procede en origen de la teoría de los humores de Hipócrates. Mas adelante seria desarrollado por el psicólogo Carl Jung ya en el

siglo XX. Esta teoría clasifica a las personas en 4 grandes tipos de personalidad. Rojo, amarillo, azul y verde. Según esta clasificación las personas con una personalidad de color rojo serán extrovertidas y se guían más por hechos que las emociones, las verdes por el contrario son introvertidas y se guían más por sus emociones.

Estas teorías se usan mucho en la actualidad para ayudar a los empleados en sectores donde tienen que tener atención al cliente. Al poder clasificar según la personalidad puedes adecuar tu trato con el cliente a lo que sea más oportuno según la personalidad de este último.

Por ejemplo, si tienes un cliente rojo deberás atenderle rápido y eficientemente porque es lo que suelen valorar más. Pero si tienes un cliente amarillo deberás darle más conversación, preocuparte por sus intereses y demás.

### ¿Qué es C4PI?

Naturalmente nadie va anunciando el color de su personalidad a los cuatro vientos. Es, tras un tiempo interactuando con una persona cuando podemos, con la experiencia y conocimiento adecuados, intuir su color. Lógicamente esto lleva un tiempo y un esfuerzo.

¿Y si pudiéramos saber su color antes incluso de conocer a esa persona?.

Aquí es donde entra C4PI, la idea es realizar una aplicación para que los empleados puedan clasificar a los clientes según su personalidad, simplemente indicando el color predominante que consideran que tiene el cliente. De tal modo que simplemente introduciendo los datos del cliente podamos ver que color creen que tienen y así poder facilitar nuestra interacción con el mismo.

De este modo tendremos nuestra propia versión de C3PO, el famoso robot de la guerra de las galaxias. Recordemos que C3PO era un androide de protocolo y su función era ayudar a los humanos a comunicarse entre ellos. De hay viene el guiño en el nombre de la aplicación.

### ¿ Por que hacer este proyecto ?

Uno de los principales motivos que me ha llevado a elegir la temática del proyecto esta íntimamente ligado con mi carrera profesional. Tengo más de 15 años de experiencia en el sector de la hostelería. Esto me proporcionado una amplia experiencia acerca del trato directo al cliente. Así como la importancia que tiene y la dificultad que conlleva. En alguna ocasión he podido asistir a seminarios sobre la personalidad clasificada por colores. Es algo que me ha resultado realmente útil en mi trato con los clientes. Por estos motivos se me ocurrió aplicar estos conocimientos a mi proyecto. Considero que es una idea que, bien desarrollada, podría resultar muy útil a cualquier persona que tenga que tener un trato con clientes regularmente. No obstante, también se aplican estas teorías para mejorar el trato con tus compañeros de trabajo. Así como a responsables de equipo para poder gestionar mejor a los empleados a su cargo. Aunque en este proyecto me centrare en el trato al cliente.

### Planteamiento

La idea inicial es hacer una web de tal manera que los empleados al registrarse tengan que indicar el hotel y departamento del mismo al que pertenecen. De esta manera la web solo podrá usarse por empleados de empresas registradas en la misma. La información que se maneja de los clientes es bastante sensible. Debido a ello no me parecía buena idea hacer una web totalmente abierta a que cualquier persona pudiera registrarse.

Para los empleados me gustaría que tengan la posibilidad de dar una opinión sobre el cliente y a su vez chequear las opiniones de cualquier cliente.

En principio cada hotel tendrá un cliente administrador por así decirlo al que le daré acceso a un menú específico para que pueda añadir y eliminar empleados del registro. También tendrá acceso al resto de opciones.

## Ciclo de vida

## Análisis

### Tecnologías aplicadas en el proyecto

#### Página web SPA

El proyecto es una SPA(single page application), este modo de realizar webs implica que toda la página se carga en la primera carga y luego se va accediendo a los diversos elementos por los menús. Aunque cambie la URL todos los elementos se cargan al principio. Esto hace que la primera carga sea algo más lenta pero la navegación más fluida. Uno de los stacks más utilizados para la realización de este tipo de páginas es el denominado MEAN(Mongodb,express,angular y node.js). En mi proyecto voy a usar 3 de estas cuatro tecnologías, voy a sustituir mongodb por una base de datos mysql. Express y node se utilizan para la parte del backend y angular para el frontend.

#### Angular

Para la parte del frontend he aplicado angular. Al ser un framework que no hemos visto durante el curso he requerido una formación previa para poder siquiera comenzar el mismo.

He realizado dos cursos online en la academia openwebinars:

<https://openwebinars.net/cert/zzJ56>

<https://openwebinars.net/cert/zzdGD>

Angular ofrece la posibilidad de usar diversas librerías para varios elementos. Yo he utilizado la librería de material que es de las más usadas. Para ello he realizado una investigación previa y una consulta continua de la documentación oficial.

## Node.js express

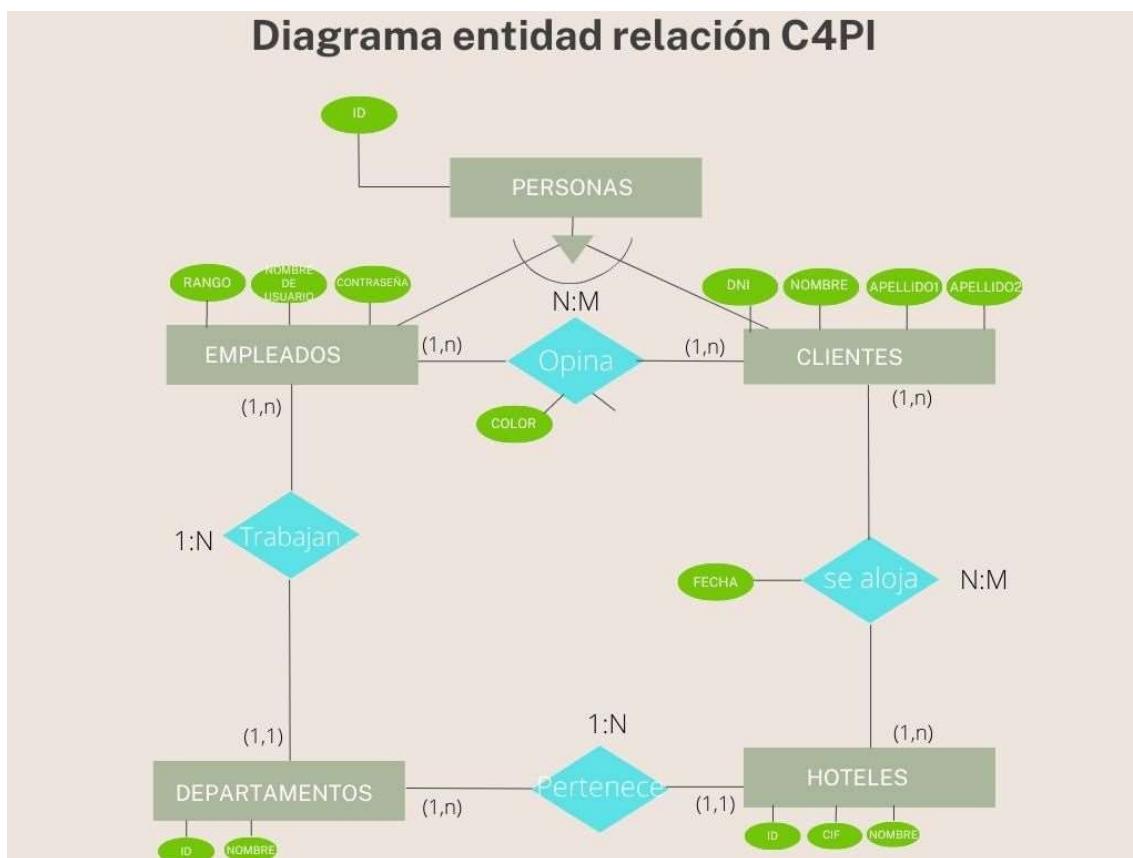
Como he mencionado antes node y express se utilizan para la parte del backend. Estas tecnologías permiten levantar un servidor prescindiendo de servicios como apache o nginx. Las consultas a la base de datos se hacen con node, que es básicamente javascript. Para poder realizar esta parte me ha sucedido lo mismo que con angular, al no haberlo visto durante el curso he tenido que realizar una formación previa, que actualmente sigo cursando, es el curso de udemy del stack MEAN:

[https://www.udemy.com/share/101WIS3@eiO3WqSzwlwscUZPGW0R5O6zizqYSIG8Yem077XNeo\\_9aaV1gXIXe-K68GeNJwztA==/](https://www.udemy.com/share/101WIS3@eiO3WqSzwlwscUZPGW0R5O6zizqYSIG8Yem077XNeo_9aaV1gXIXe-K68GeNJwztA==/)

## Mysql

Para poder realizar una SPA normalmente se usa un framework como angular u otros y node y express para la api del backend. No obstante, el tipo de base de datos utilizado varía mucho y se pueden usar tanto bases de datos relacionales como no relacionales. Para mi proyecto me he decidido por mysql por ser una base de datos que ya conozco y por lo tanto me ha resultado más sencillo.

Previo a la creación de las tablas he realizado un diagrama entidad relación y un modelo relacional.



## Modelo relacional

Empleado(Id,Id\_departamento,rango,login,password)

Cliente(Id,dni,nombre,apellido1,apellido2,nombre)

Departamento(Id,nombre,id\_hotel)

Hotel(Id,nombre,Cif)

Opinión(id\_cliente id\_empleado,color,Fecha,)

Estancia(id\_cliente id\_hotel,fecha)

## Desarrollo

### Backend

#### Services

Para gestionar las llamadas al backend de la aplicación es muy común usar servicios o services.

Estos servicios sirven para hacer las llamadas al servidor, tanto para mandar como recibir información.

Se pueden generar desde la consola con las instrucciones:

*Ng generate service <nombre del servicio>*

Para los servicios uso el paquete de angular rxjs. Este paquete proporciona una serie de objetos para gestionar la circulación de datos entre el backend y el frontend.

Los objetos que utilizo son los siguientes:

Subject : Básicamente es un emisor de eventos, utilizo dos métodos principalmente:

Next(), se utiliza para emitir un nuevo valor y asObservable().

Observable: Este objeto es que devuelve el método asObservable() de Subject, se utiliza para devolver el subject como un escuchador para que solo pueda escucharse pero no emitir eventos. El Observable tiene un método que es subscribe, este método recibe tres funciones para gestionar la emisión de datos, de errores y cuando se completa la suscripción.

Subscription: este es el objeto que devuelve el método subscribe(), se utiliza en el mismo

El método unsubscribe() para evitar fugas de memoria siempre que se destruya el componente.

Para obtener los datos de la API uso el cliente de http que proporciona angular. Este cliente proporciona los métodos get y post( según como sea la solicitud) que devuelven un Observable ,el objeto que comentaba antes, al cual nos suscribimos para obtener la respuesta de la API. Al usar httpclient no hace falta que paremos la subscripción con unsubscribe puesto que ya lo hace Angular por si solo.

En la aplicación por ejemplo tengo un servicio para toda la parte del login y el registro de empleados. También se pueden separar por tablas en la base de datos, por ejemplo, haciendo un servicio para hoteles y otro para departamentos.

Vamos a ver unos ejemplos. Estos serian los métodos utilizados para recoger la información de los hoteles y los departamentos. Se puede ver con deben indicar el tipo de propiedad que recogen, en estos casos es un tipo de propiedad(Hotel y Departamento) que hemos creado en Angular mediante una interface. Esto lo veremos en el siguiente apartado. También vemos que se indica la URL del servidor correspondiente y como recoge los datos.

Cabe destacar que, al gestionar el backend con node, siempre que mandemos alguna información se mandara por POST, que es lo más recomendable.

```
,  
getHoteles() {  
    this.http.get<Hotel[]>('http://localhost:3000/hoteles')  
    .subscribe(  
        (hotelData) => {  
            this.hoteles = hotelData;  
            this.hotelUpdated.next(this.hoteles);  
        }  
    );  
}  
getDepartamentos(idHotel : {envio: number}){  
    this.http.post<Departamento[]>('http://localhost:3000/departamentos', idHotel)  
    .subscribe(  
        (depData) => {  
            this.departamentos = depData;  
            this.DepartamentoUpdated.next(this.departamentos);  
        }  
    );  
}
```

## Models

Como he mencionado antes otra de las funcionalidades que proporciona angular con las interfaces. Son algo así como una clase de Java pero solo con las propiedades, sin constructor ni métodos. Es bastante común utilizarlas para recoger la información de la base de datos utilizando modelos que se asemejen a las tablas.

Por ejemplo, vemos la interface de Cliente ya la vamos a comparar con su tabla correspondiente en la base de datos.

```

export interface Cliente {
  dni: string;
  nombre: string;
  apellido1: string;
  apellido2: string;
  rojo: { cantidad: number; porcentaje: number };
  verde: { cantidad: number; porcentaje: number };
  azul: { cantidad: number; porcentaje: number };
  amarillo: { cantidad: number; porcentaje: number };
  cantidadOpiniones: number;
}

```

Columnas:		<span style="color: green;">+</span> Agregar	<span style="color: red;">X</span> Borrar	<span style="color: blue;">▲</span> Subir	<span style="color: black;">▼</span> Bajar
#	Nombre	Tipo de datos	Longitud/Conjunto	Sin signo	Permitir...
1	<span style="color: orange;">id</span>	<span style="color: blue;">SMALLINT</span>	5	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	dni	<span style="color: blue;">VARCHAR</span>	15	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3	nombre	<span style="color: blue;">VARCHAR</span>	20	<input type="checkbox"/>	<input checked="" type="checkbox"/>
4	apellido1	<span style="color: blue;">VARCHAR</span>	20	<input type="checkbox"/>	<input checked="" type="checkbox"/>
5	apellido2	<span style="color: blue;">VARCHAR</span>	20	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Como se puede ver coinciden algunas propiedades en ambas y otras solo están en la base de datos (como el id) o en la interface como los colores. Las propiedades que estén en ambas se recogerán cuando se haga la llamada la base de datos, esas y ninguna otra porque no hay sitio donde recogerla. Y las propiedades de la interface que no tengan su correspondencia en la base de datos se quedarán como undefined si no les asignamos ningún valor. En este caso yo usare esas propiedades para indicar que opiniones tienen los clientes.

## RESTFUL API con node.js

Para la RESTFUL(Representational State Transfer) API(Application programming interfaces) he usado node.js y express. Estas tecnologías tienen la particularidad de que no necesitan un servidor, ya sea Apache o Nginx. Algo que también es algo particular de una restful api es el hecho de que esta disociada, por decirlo de algún modo, de la parte de frontend. Lo que significa que podríamos usar esta API para extraer y manipular datos desde, por ejemplo, una aplicación de móvil.

Para la comunicación del frontend con la API se utiliza JSON, es decir para hacer las solicitudes si se necesita enviar un dato y para las respuestas se envían y reciben objetos JSON.

Para realizar la api hay diversas maneras de organizar los ficheros. Por ejemplo, un fichero para proporcionar el servidor de express y otro para los métodos. O uno por cada tabla de la base de datos con sus métodos correspondientes.

En mi caso debido a que la aplicación no es excesivamente grande he decidido usar solo un fichero(app.js) para mayor sencillez.

Al principio declaramos las variables que vamos a utilizar, si vamos a utilizar alguna funcionalidad predefinida (por ejemplo, mysql para la conexión a la base de datos) se indica con require. También configuramos los datos de conexión indicando el puerto donde se escucha el servidor, así como los datos de conexión a la base de datos utilizando el usuario correspondiente.

```
const express = require("express");
const bodyParser = require("body-parser");
const app = express();
var mysql = require("mysql");
const http = require("http");
const port = process.env.PORT || 3000;
app.set("port", port);
const server = http.createServer(app);
var bcrypt = require("bcrypt");
var con = mysql.createPool({
  host: "localhost",
  user: "jorge",
  password: "Nohay2sin3",
  database: "c4pi",
});
```

Otra cosa a tener en cuenta son los headers, al ser una api disociada del front tiene que poder permitir llamadas desde diferentes clientes. En una página web “tradicional” esto no se permite por motivos de seguridad, por este motivo si no cambiamos los headers tendremos un error cors (cross origin resource sharing). En nuestro caso debemos configurar los headers como vemos a continuación:

```
app.use((req, res, next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept"
  );
  res.setHeader(
    "Access-Control-Allow-Methods",
    "GET, POST, PATCH, DELETE, PUT, OPTIONS"
  );
  next();
});
```

Aquí vemos algunos métodos, indicando get o post según corresponda.

```

app.get("/hoteles", (req, res, next) => {
  //res.json(posts);
  // con.connect(function (err) {
  //   if (err) throw err;
  con.query("SELECT * from c4pi.hoteles", function (err, result, fields) {
    if (err) throw err;
    // console.log(result);
    const respuesta = result;
    res.status(200).json(respuesta);
  });
  // });
});

app.get("/clientes", (req, res, next) => {
  con.query("select * from c4pi.clientes", function (err, result) {
    if (err) throw err;
    const respuesta = result;
    res.status(200).json(respuesta);
  });
});

```

Vemos que al no recibir información hemos usado get.

Me voy a detener en el método utilizado para comprobar las contraseñas. Me será de utilidad para explicar un par de cuestiones interesantes:

```

app.post("/comprobacion_password", (req, res, next) => {
  let respuesta = false;
  const login = req.body.nombre;
  const password = req.body.password;
  con.query(
    "SELECT password from c4pi.empleados where login=?",
    [login],
    function (err, result) {
      if (err) throw err;
      if (result.length > 0) {
        respuesta = bcrypt.compareSync(password, result[0].password);
        res.status(200).json(respuesta);
      }
    }
  );
});

```

Primero que todo vemos que se utiliza post dado que recibe datos. Por cuestiones de seguridad, para evitar ataques con inyecciones de código sql no pasamos directamente las propiedades a la base de datos. Se pasan haciendo un binding con una propiedad, en este caso login. También me gustaría destacar que a la hora de almacenar las contraseñas en la base de datos se guarda, como no podía ser de otra manera, la contraseña hasheada y no la contraseña en claro. Para ello usamos la funcionalidad bcrypt de node tanto para insertarlas como para comprobarlas. Aquí vemos como se

```

  app.post("/insercion_empleado", (req, res, next) => {
    const id_departamento = req.body.id_departamento;
    const nombre = req.body.nombre;
    const password = req.body.password;
    let hash = bcrypt.hashSync(password, 5);
    con.query(
      "INSERT into c4pi.empleados(id_departamento,rango,login,password) VALUES (?,0,?,?)",
      [id_departamento, nombre, hash],
      function (err, result, fields) {
        if (err) throw err;
        if (result.affectedRows > 0) {
          respuesta = true;
        } else {
          respuesta = false;
        }
        res.status(200).json(respuesta);
      });
  });
});

```

insertan al crear un empleado:

## Template driven forms

Antes de continuar con los elementos de la aplicación me gustaría detenerme en un aspecto de Angular. Me refiero a la gestión de los formularios. En los dos componentes que veremos a continuación de login y registro de usuario he usado el template driven, que es una de las dos formas de gestionar los formularios (la otra manera la veremos más adelante), esta aproximación es bastante parecida a la tradicional.

Para usar el template driven se usa los siguientes atributos en la etiqueta de form:

```
<form (ngSubmit)="onSubmit()" #formLogin="ngForm">
```

El método onSubmit() será llamado hace hagamos submit en el botón correspondiente.

En cada input debemos poner el atributo ngModel como vemos aquí:

```
<input matInput placeholder=" Nombre" name="nombre" [(ngModel)]="nombre" />
```

Se usa con [()] esto hace que la propiedad del mismo nombre en la lógica del componente tome los datos directamente de este input y viceversa, es decir podríamos cambiar los datos del input desde la lógica del componente.

## Pantalla de inicio

Para la pantalla de inicio he hecho un formulario que permite al usuario introducir su nombre de usuario y contraseña. Esta pantalla está dirigida para los usuarios que ya están registrados en la aplicación. Para los que no lo estén tienen un enlace para dirigirse a la pantalla de registro.

Para la creación de esta pantalla he creado un componente de Angular. Angular divide los proyectos en diversos componentes, cada uno de ellos puede representar una página de la aplicación o una parte de una página. En este primer componente será una página completa, que es la página de inicio como decía.

Para poder generar estos componentes se puede realizar desde la consola con los comandos

*Ng generate component <nombre del componente>*

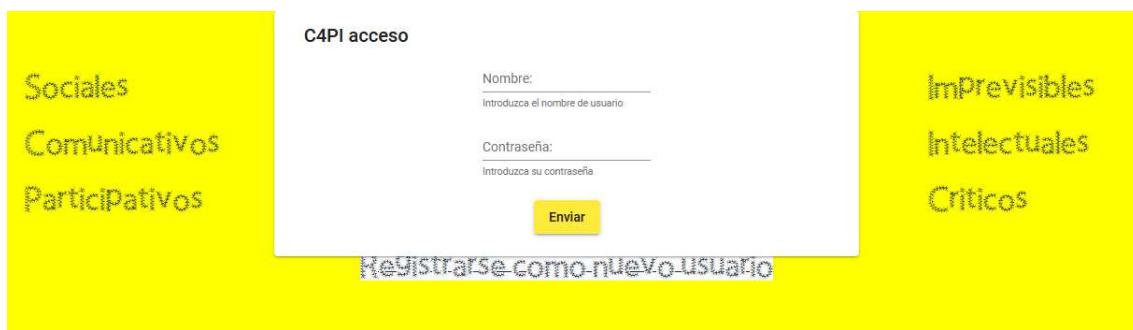
Automáticamente se genera una carpeta con cuatro ficheros, una hoja HTML, una hoja de estilos CSS, un fichero de Typescript con la lógica del componente(sería el equivalente al javascript en una web “tradicional”) y un fichero para testing.

No obstante, en mi proyecto no voy a utilizar los ficheros de testing, para ello siempre que genere un componente se puede añadir las siguientes instrucciones al comando para que omita su creación.

*--skip-tests*

Paso a explicar la hoja de html y typescript, omito la hoja de css poque los estilos los estoy poniendo en la hoja de css general para todo el proyecto dado que se comparten clases en casi todos los componentes del mismo.

#### Visualización



Como comentaba en la parte central se puede ver un formulario con dos campos: nombre y contraseña. El botón de submit y un enlace a la pantalla de registro.

Tanto el color de fondo, la fuente y las propias palabras van cambiando entre los cuatro colores.



## HTML

```
<div id="inicio" class="container-fluid" [ngStyle]="{ 'background-color': color }">
  <div class="row justify-content-center gx-2">
    <div class="col-12 col-md-3 col-lg-2 d-flex justify-content-center align-items-center">
      <div id="palabras1" [ngClass]="fuente" [ngStyle]="{ 'background-color': color }">
        <p>{{ palabrasColor[0] }}</p>
        <p>{{ palabrasColor[1] }}</p>
        <p>{{ palabrasColor[2] }}</p>
      </div>
    </div>
    <div class="col-12 col-md-6 col-lg-4 text-center">
      <form (ngSubmit)="onSubmit()" #formLogin="ngForm">
        <mat-card [ngClass]="theme">
          <mat-card-header>
            <mat-card-title> C4PI acceso </mat-card-title>
          </mat-card-header>
          <mat-card-content>
            <mat-form-field>
              <mat-label>Nombre:</mat-label>
              <input matInput placeholder=" Nombre" name="nombre" [(ngModel)]="nombre" />
              <mat-hint>Introduzca el nombre de usuario</mat-hint>
            </mat-form-field>
          </mat-card-content>
          <mat-card-content>
            <mat-form-field>
              <mat-label>Contraseña:</mat-label>
              <input matInput placeholder="Contraseña" name="password" type="password" [(ngModel)]="password" />
              <mat-hint>Introduzca su contraseña</mat-hint>
            </mat-form-field>
          </mat-card-content>
          <mat-card-actions class="mat-card-button-login">
            <button mat-raised-button color="primary">Enviar</button>
          </mat-card-actions>
          <mat-card-footer id="footer">
            <p *ngIf="errorNombre" class="error">
              No existe ningun usuario con ese nombre
            </p>
            <p *ngIf="errorPassword" class="error">
              La contraseña no es correcta
            </p>
          </mat-card-footer>
        </mat-card>
      </form>
      <a [ngClass]="fuente" [routerLink]="['./', 'registro']" queryParamsHandling="preserve">Registrarse como nuevo usuario</a>
    </div>
    <div class="col-12 col-md-3 col-lg-2 d-flex justify-content-center align-items-center">
      <div id="palabras2" [ngClass]="fuente" [ngStyle]="{ 'background-color': color }">
        <p>{{ palabrasColor[3] }}</p>
        <p>{{ palabrasColor[4] }}</p>
        <p>{{ palabrasColor[5] }}</p>
      </div>
    </div>
  </div>
  <!-- <div class="row justify-content-center gx-2">
    <div class="col-12 col-md-6 col-lg-4 text-center">
      
    </div>
  </div> -->
</div>
```

Antes de nada voy a explicar brevemente el llamado property binding de Angular. Para indicar los atributos se pueden hacer de manera unidireccional(one-way-binding) o bidireccional(double-way-binding). Si se indica el nombre con [] significa que el elemento en html puede recibir la información de la lógica del componente, si se indica con () es al revés, el elemento del html es el que envia la información a la logica, este es que se utiliza para gestionar los eventos por ejemplo (click)="funcion()" indicaría que al pulsar sobre ese elemento ejecutara la función. En el caso del double binding se indica con ambos [()] y envía y recoge información, algo típico son inputs de un formulario.

También cabe destacar que siempre que usemos alguna propiedad de la logica del componente dentro del html(no como atributo) se indica con {}.

Como comenté en anteriores apartados en el proyecto uso la librería de Angular. Esta librería ofrece varias etiquetas predefinidas en html que dan un formato propio, además de otras funcionalidades.

Todas las etiquetas que comienzan con mat pertenecen a la librería de materials.

También uso atributos específicos de Angular que ayudan a realizar la lógica de la aplicación, son todos los que comienzan por ng.

Paso a explicar algunos de ellos.

#### ngClass

Este atributo permite establecer la clase de un modo dinámico, es decir la clase es una propiedad de la hoja de typescript que puede ir modificándose. Luego veremos cuando explique la hoja de Typescript para que la utilice.

#### \*ngIf

Es uno de los atributos más utilizados, al poner este atributo podemos hacer que esa parte del html solo sea visible siempre y cuando se cumpla la condición o condiciones que especifiquemos en el atributo. En este caso lo utilizo con una propiedad que será una bandera de tipo booleano.

#### routerLink

En el enlace uso routerLink en lugar del href tradicional, en este caso específico la ruta relativa al componente al que quiero dirigirlo.

#### TYPESCRIPT

El typescript funciona como una clase, tiene sus propiedades, su constructor y sus métodos.

Paso a explicar algunos de ellos.

## Constructor

```
constructor(public loginservice: LoginService, public router: Router) {
  this.color = '#FFFFFF';
  this.colores = ['#FF0000', '#00FF00', '#0000FF', '#FFFF00'];
  this.palabrasTotales = [
    [
      'Energéticos',
      'Autoritarios',
      'Valientes',
      'Ambiciosos',
      'Pasionales',
      'Realistas',
    ],
    [
      'Creativos',
      'Constantes',
      'Pacientes',
      'Empáticos',
      'Espirituales',
      'Democráticos',
    ],
    [
      'Reflexivos',
      'Analíticos',
      'Tecnológicos',
      'Realistas',
      'Tranquilos',
      'Solitarios',
    ],
    [
      'Sociales',
      'Comunicativos',
      'Participativos',
      'Imprevisibles',
      'Intelectuales',
      'Criticos',
    ],
  ];
  this.themes = ['red-theme', 'green-theme', 'blue-theme', 'yellow-theme'];
  this.fuentes = ['rojo', 'verde', 'azul', 'amarillo']
```

Como se puede ver creo cuatro arrays. Uno con los propios colores en formato hexadecimal, otro con las fuentes que son clases que se usaran para cambiar la propiedad ngClass dinámicamente también los themes que cambian el formato del formulario de materiales.

Y otro con que a su vez son cuatro arrays de strings, cada uno de ellos representa un conjunto de adjetivos que se aplican al color correspondiente de la personalidad.

En el método ngOnInit, que se ejecuta llamo con setInterval a otro método para que este cambiando los elementos y la visualización cada cierto tiempo, este método selecciona aleatoriamente un número del 0 al 3 y lo utiliza para seleccionar el elemento correspondiente de los arrays antes mencionados.

```

    ngOnInit() {
      this.loginservice.getHoteles();
      this.loginSub = this.loginservice
        .getHotelUpdateListener()
        .subscribe((hoteles: Hotel[]) => {
          this.hoteles = hoteles;
        });
      this.cambiaColor();
      setInterval(() => {
        this.cambiaColor();
      }, 3000);
    }

    public cambiaColor(): void {
      let n = Math.floor(Math.random() * this.colores.length);
      this.color = this.colores[n];
      this.palabrasColor = this.palabrasTotales[n];
      this.theme = this.themes[n];
      this_fuente = this.fuentes[n]
    }
  }

```

El resto de métodos se encargan de gestionar la lógica del formulario. Ejecutan llamadas al servicio que a su vez se comunica con la app de node en el backend. Cuando esta todo correcto redirijo al usuario a la pantalla principal. Al ir comprobando si existe el nombre y si la contraseña es la correcta va levantando o no la propiedad correspondiente que indica si hay un error. Esta propiedad es la que uso en con el atributo nglf en el HTML para que muestre el mensaje de error correspondiente.

```

    mandarNombre() {
      this.loginservice.sendLogin({ envio: this.nombre });
      this.loginSub = this.loginservice
        .getLoginUpdatedListener()
        .subscribe((message: boolean) => {
          this.errorNombre = !message;
        });
    }

    mandarPassword() {
      this.loginservice.sendPassword({
        nombre: this.nombre,
        password: this.password,
      });
      this.passwordSub = this.loginservice
        .getPasswordUpdatedListener()
        .subscribe((message: boolean) => {
          this.errorPassword = !message;
        });
    }

    onSubmit() {
      this.mandarNombre();
      if (!this.errorNombre) {
        this.mandarPassword();
      }

      setTimeout(() => {
        this.redireccionar();
      }, 1000);
    }

    redireccionar() {
      if (this.errorNombre === false && this.errorPassword === false) {
        this.loginservice.data = this.nombre;
        this.router.navigate(['/home/main']);
      }
    }
  }

```

## Pantalla de registro

Esta pantalla se corresponde a otro componente de Angular. Su utilidad es poder registrar nuevos empleados en la aplicación, para ello deben introducir su nombre de usuario y contraseña. Así como seleccionar el hotel y el departamento del mismo al que pertenecen.

La selección de hotel y departamento se realiza con dos selects. Hasta que no se selecciona un hotel no aparecen los departamentos de ese hotel.

El fondo es el mismo que en la pantalla de inicio.

#### HTML

Del html voy a destacar tan solo los selects puesto que el resto es prácticamente lo mismo que la pantalla de inicio.

```

<mat-card-content>
  <mat-form-field>
    <mat-label>Selecciona tu hotel</mat-label>
    <mat-select [(value)]="selectedHotel" (selectionChange)="getDepartamentos()">
      <mat-option [value]="hotel.id" *ngFor="let hotel of hoteles">{{hotel.nombre}}</mat-option>
    </mat-select>
  </mat-form-field>
</mat-card-content>
<mat-card-content>
  <mat-form-field>
    <mat-label>Selecciona tu departamento</mat-label>
    <mat-select [(value)]="selectedDepartamento">
      <mat-option [value]="departamento.id" *ngFor="let departamento of departamentos">{{departamento.nombre}}</mat-option>
    </mat-select>
  </mat-form-field>
</mat-card-content>

```

Como se puede ver utilizo el double binding con el value para que recoja el valor seleccionado.

Y algo que no habíamos visto antes y que es muy utilizado en Angular que es el ngFor. Este atributo sirve para hacer una iteración sobre un array que tengamos en el typescript, en este caso los hoteles y sus departamentos.

#### TYPESCRIPT

No hay apenas novedades reseñables con respecto al componente anterior, tan solo vemos como llama a un servicio para darles valores a los arrays antes mencionados. El resto es comprobar nombre, contraseña y demás.

```

ngOnInit(): void {
  this.log.loginService.getHoteles();
  this.loginSub = this.log.loginService.getHotelUpdateListener()
    .subscribe((hoteles: Hotel[]) => {
      this.hoteles = hoteles;
    })
  this.log.cambiaColor();
  setInterval(() => { this.log.cambiaColor(); }, 3000);
}

insertaEmpleado() {
}

```

## Pantalla principal

Una vez cumplimentado el login correctamente o habiéndose registrado como nuevo empleado se pasa a la siguiente pantalla:

NAVBAR
Logout
C4PI
Ver opiniones
Test de personalidad

**Crear nuevo cliente**

Documento: \*

Introduce el numero de documento de usuario

Nombre: \*

Introduce el nombre de usuario

Primer apellido: \*

Introduce el primer apellido del cliente

Segundo apellido:

Introduce el segundo apellido del cliente

**Enviar**

Se creó una nueva opinión de Alex Alphonse con el color azul

Numero de documento	Nombre	Primer apellido	Segundo apellido	Rojo	Verde	Azul	Amarillo	Opiniones totales
21342345G	Pedro	Gutierrez	Rodriguez	100 %	0 %	0 %	0 %	1
x900980	Sarah	Connor		50 %	0 %	50 %	0 %	2
Y982309	Ionnis	Markaris		0 %	0 %	100 %	0 %	1
56456756	Laura	Gomez	Perez	100 %	0 %	0 %	0 %	1
Z983290813	Peter	Crouch		100 %	0 %	0 %	0 %	1
Y98723892	Giorgio	Marini		0 %	0 %	0 %	0 %	0
X9231093	Alex	Alphonse		0 %	0 %	100 %	0 %	1

## REGISTRO-CLIENTE

## VISTA-CLIENTES

La pantalla principal esta dividida en tres componentes de Angular los cuales se indican en la imagen. Cada uno de ellos cuando con su propio html, css y typescript. Esta diferenciación en una misma pantalla da una idea de uno de los principales beneficios de Angular, la reusabilidad. Aquí se puede ver el html de la pantalla principal:

## Main card html

```
1 <div class="container-fluid"></div>
2 <div class="row">
3   <div class="col-12">
4     <app-navbar></app-navbar>
5   </div>
6 
7 </div>
8 
9 <div class="row">
10  <div class="col col-md-6 col-lg-4">
11    <app-registro-cliente></app-registro-cliente>
12  </div>
13  <div class="col col-md-6 lg-8">
14    <app-vista-clientes></app-vista-clientes>
15  </div>
16 </div>
17 </div>
18 </div>
```

Tan solo tiene algo de código usando Bootstrap para ordenar los elementos en pantalla y las etiquetas que hacen referencia a los componentes antes mencionados, que son todas las que comienzan por app.

Como veremos más adelante esto resultará muy útil para rehusar el componente navbar dado que nos deber aparecer en distintas pantallas que se compondrán de diferentes componentes.

Una vez explicado esto voy a dar una pasada sobre los componentes que forman parte de esta pantalla.

### Navbar

#### HTML

Para la navbar he usado el elemento de material toolbar, este elemento permite realizar una barra de navegación, típicamente usada como header. Dentro de ello he usado varios buttons que hacen referencia a una función ubicada en la lógica del componente. Y que redirige a cada pantalla correspondiente. La toolbar esta configurada para que sea responsive, cuando la vemos en una pantalla grande la veremos con los nombres de los buttons y demás como vemos a continuación:



Si la vemos desde una pantalla pequeña la veremos con un botón que despliega las diferentes opciones:



Aquí se puede ver el código html:

```

1 <div [ngClass]="'theme">
2   <mat-toolbar [ngClass]="'theme" fxLayout="row" color="primary">
3     <span id="logout" mat-icon-item fxFlex (click)="onLink('home/login')">
4       <svg height="30" viewBox="0 0 30 30" width="30" xmlns="http://www.w3.org/2000/svg">
5         <path d="M8.5 0C7.678 0 7.678 7 1.5v6C0 .665 1 .67 1 0v-6C0-.286.214-.5.5h2C.286 0 .5.214.5.5v2C0 .286-.214.5-.5h-2C-.2
6           </path>
7         </svg>
8       Logout
9     </span>
10    <span id="icono" mat-icon-item fxFlex><svg fill="#000000" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 48 48">
11      <path width="48px" height="48px">
12        <path d="M 24.021484 2.0058594 C 23.796484 2.0058594 23.571094 2.0832813 23.371094 2.2382812 C 21.786094 3.5792813 21.695391 3.6
13          </path>
14        </svg>
15      C4PI
16    </span>
17    <button class="buttonClass" mat-button *ngFor="let item of menuItems" fxHide.xs (click)="onLink(item.link)">
18      {{item.nombre}}
19    </button>
20    <button class="buttonClass" mat-icon-item [matMenuTriggerFor]="dropMenu" fxHide fxShow.xs>
21      <svg fill="#000000" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" width="48px" height="48px">
22        <path d="M 2 5 L 2 7 L 22 7 L 22 5 L 2 5 z M 2 11 L 2 13 L 22 13 L 22 11 L 2 11 z M 2 17 L 2 19 L 22 19 L 22 17 L 2 17 z" />
23      </svg>
24    </button>
25
26
27    <!-- <a routerLink="item.link" queryParamsHandling="preserve"></a> -->
28    <mat-menu #dropMenu="matMenu">
29      <button mat-menu-item (click)="onLink('home/login')">
30        Logout
31      </button>
32      <ng-container class="buttonClass" *ngFor="let item of menuItems">
33        <button mat-menu-item (click)="onLink(item.link)">
34          {{item.nombre}}
35        </button>
36
37        <mat-divider></mat-divider>
38      </ng-container>
39    </mat-menu>
40  </mat-toolbar>
41</div>

```

## CSS

Para la hoja de estilos le he dado unos estilos a los botones para que varan cambiando al posicionarse sobre ellos, como se puede ver:



Además he usado una media query sobre el logout porque no quería que tuviese el mismo formato que es resto de botones y la funcionalidad de la toolbar no funcionaba sobre él.

```

.buttonClass {
    font-size:1.2em;
    font-family: "Futura", Arial, Helvetica, sans-serif;
    width:180px;
    height:50px;
    border-width:1px;
    color:#333333;
    border-color:#333333;
    font-weight:bold;
    border-top-left-radius:6px;
    border-top-right-radius:6px;
    border-bottom-left-radius:6px;
    border-bottom-right-radius:6px;
    box-shadow: 0px 1px 0px 0px #121211;
    text-shadow: 0px 1px 0px #121210;
    background:linear-gradient(#FFFFFF, #b9d7d8);
}

.buttonClass:hover {
    background: linear-gradient(#0e0f0f, #131315);
    color:#FFFFFF;
    box-shadow: 0px 1px 0px 0px #f1f1e8;
    text-shadow: 0px 1px 0px #f4f4f0;
}

#icono,#logout{
    font-size:2.2em;
    font-family: "Futura", Arial, Helvetica, sans-serif;
    color: #333333;
}

#logout:hover{
    cursor: pointer;
}

#logout{
    display: none;
}

@media screen and (min-width: 600px) {
    #logout{
        display: block;
    }
}

```

## Typescript

Para la lógica de este componente creo un array de objetos con las propiedades nombre y link con el nombre y el enlace de los botones. Y también una función que es a la que llaman los botones que redirige al enlace correspondiente.

```

    ngOnInit(): void {
      this.menuItems = [
        { nombre: 'Ver opiniones', link: 'home/main' },
        { nombre: 'Test de personalidad', link: 'home/test' },
        { nombre: 'Menu administrador', link: 'home/admin' },
      ];
      if (localStorage.getItem('rango') !== '1') {
        this.menuItems.pop();
      }
    }
    onLink(e: string) {
      if (e === 'home/login') {
        localStorage.clear();
      }
      this.router.navigate([e]);
    }
}

```

### Registro-cliente

#### Reactive forms

Si recordamos para los componentes del login y registro de empleado uso un tipo de formulario de angular llamado template driven forms.

Para este componente de registro de cliente he usado en cambio el reactive form.

En este caso en la etiqueta form se usa el atributo formGroup dándole un nombre a nuestra elección al formulario, se llama a la función onSubmit pasando como argumento el nombre del formulario.

```
<form [formGroup]="registroCliente" (ngSubmit)="onSubmit(registroCliente)">
```

En cada input se usa el atributo formControlName para asociarles a un nombre que nosotros queremos.

```
<input matInput placeholder="Nombre" name="nombre" formControlName="nombre">
```

En la lógica del componente necesitaremos una variable de tipo FormGroup

```
registroCliente: FormGroup;
```

En el constructor inyectamos como argumento una variable de tipo FormBuilder, usamos esta variable para crear un grupo de objetos, cada uno de estos objetos representa uno de los nombres que pusimos en los formControlName. Como key seria el nombre que pusimos en el html y como valor un array donde se pasa el valor por defecto(en este caso string vacio) y una serie de opciones de validación, en este caso solo uso la opción required que indica que es obligatorio poner algún dato. Pero también se podrían usar otras opciones como minLength u otros.

```

  constructor(private fb: FormBuilder, public mainService: MainService) {
    this.registroCliente = this.fb.group({
      dni: ['', Validators.required],
      nombre: ['', Validators.required],
      apellido1: ['', Validators.required],
      apellido2: '',
    });
  }
}

```

Una vez declarado e inicializado el FormGroup podemos llamar a sus elementos con su nombre correspondiente usando la sintaxis como se a continuación:

```
this.registroCliente.get('dni')
```

Podemos llamar a la propiedad invalid de estos elementos que devuelve un boolean si el contenido del elemento no es válido. También las propiedades dirty y touched para ver si el usuario ha manipulado el elemento o si ha hecho foco sobre el input.

Aquí uso estas propiedades con un \*ngIf para mostrar los posibles errores:

```

<mat-error
  *ngIf="registroCliente.get('nombre')?.invalid && (registroCliente.get('nombre')?.dirty || registroCliente.get('nombre')?.touched)">
  Debe introducir un nombre
</mat-error>

```

En la función onSubmit usamos el mismo formulario que le pasamos como argumento con la propiedad valid para comprobar si todos los inputs cumplen con los validadores que indicamos al construir el formGroup:

```

  ...
  if (form.valid && !this.ExisteDni) {
    this.insertaCliente();
  }
}

```

También uso la función comprobarDni que comprueba si el numero de documento introducido existe ya en la base de datos:

```

  ...
  comprobarDni() {
    let dni = this.registroCliente.get('dni')?.value as unknown as string;
    this.mainService.compruebaDni({ envio: dni });
    this.dniSub = this.mainService
      .getDniUpdatedListener()
      .subscribe((message: boolean) => {
        this.ExisteDni = message;
      });
  }
}

```

Una vez comprobado que este todo correcto y que no este ya creado un cliente con su número de documento se inserta el cliente en la base de datos llamando al servicio correspondiente:

```

  insertaCliente() {
    let dni = this.registroCliente.get('dni')?.value as unknown as string;
    let nombre = this.registroCliente.get('nombre')?.value as unknown as string;
    let apellido1 = this.registroCliente.get('apellido1')
      ?.value as unknown as string;
    let apellido2 = this.registroCliente.get('apellido2')
      ?.value as unknown as string;
    this.mainService.insertaCliente({
      dni: dni,
      nombre: nombre,
      apellido1: apellido1,
      apellido2: apellido2,
    });
    this.cliSub = this.mainService
      .getEmpleadoUpdatedListener()
      .subscribe((message: boolean) => {
        this.errorInsercionCliente = !message;
        if (message) {
          alert('Cliente registrado correctamente');
          window.location.reload();
        }
      });
  };
}

```

Vista-clientes/ver opiniones

**Se creado una nueva opinión de Alex Alphonse con el color azul**

---

Filter

Numero de documento.	Nombre	Primer apellido	Segundo apellido	Rojo	Verde	Azul	Amarillo	Opiniones totales
21342345G	Pedro	Gutierrez	Rodriguez	100 %	0 %	0 %	0 %	1
x980980	Sarah	Connor		50 %	0 %	50 %	0 %	2
Y982309	Ionnis	Markaris		0 %	0 %	100 %	0 %	1
56456756	Laura	Gomez	Perez	100 %	0 %	0 %	0 %	1
Z983290813	Peter	Crouch		100 %	0 %	0 %	0 %	1
Y98723892	Giorgio	Marini		0 %	0 %	0 %	0 %	0
X9231093	Alex	Alphonse		0 %	0 %	100 %	0 %	1

HTML

Para esta vista he usado una tabla, concretamente una tabla de materials.

Al principio del html para que se muestre sobre la tabla ponemos un cuadro de búsqueda que usaremos para filtrar las filas que queremos mostrar en la columna. Este cuadro filtra por cualquier campo que exista en la tabla ya sea nombre apellidos o numero de documento.

```
]}<mat-form-field>
    <input matInput (keyup)="applyFilter($event)" placeholder="Filter">
</mat-form-field>
```

Para la etiqueta de la tabla se usan los atributos mat-table, dataSource que es de donde tomara los datos que veremos en la lógica del componente, y una clase indicando el formato.

```
]}<table id="tablaClientes" mat-table [dataSource]="dataSource" class="mat-elevation-z10">
```

A la hora de definir las celdas usamos las propiedades de cliente que esta en un array de clientes.

```
<ng-container matColumnDef="nombre">
    <th mat-header-cell *matHeaderCellDef> Nombre </th>
    <td mat-cell *matCellDef="let cliente"> {{cliente.nombre}} </td>
</ng-container>
```

En el caso de las celdas donde se indican los colores uso los eventos mouseover, mouseleave y click para diferenciar si se tiene el cursor sobre la celda si se quita y si se pulsa sobre ella. Cada uno de estos eventos llama a una función diferente. También defino el nombre de la columna indicándolo en el th.

```
<ng-container matColumnDef="azul">
    <th mat-header-cell *matHeaderCellDef> Azul </th>
    <td mat-cell *matCellDef="let cliente" class="azulT" (mouseover)="onColor($event)"
        (mouseleave)="onColorLeave($event)"
        (click)="onOpinion(2,cliente.id,cliente.nombre,cliente.apellido1,cliente.apellido2)"
        {{cliente.azul.porcentaje}} % </td>
</ng-container>
```

También indicamos una propiedad que he llamado displayedColumns indicando que columnas va a mostrar la tabla.

```
<tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
<tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
```

## TYPESCRIPT

Uso una serie de variables en la lógica.

Dos arrays para guardar los clientes y las opiniones:

```
clientes: Cliente[] = [];
opiniones: Opinion[] = [];
```

Las subscripciones para las llamadas a la API

```
private clientesSub!: Subscription;
private opinionesSub!: Subscription;
private opinionSub!: Subscription;
```

La propiedad que controla las columnas que sacara la tabla y una propiedad dataSource para cargar los datos.

```

displayedColumns: string[] = [
  'dni',
  'nombre',
  'apellidol',
  'apellido2',
  'rojo',
  'verde',
  'azul',
  'amarillo',
  'cantidadOpiniones',
];
dataSource: any;

```

Y luego diversas propiedades para controlar la visualización:

```

filaSeleccionada: string = '';
empleado!: Empleado;
selectedOpinion!: Opinion;
respuestaSetOpinion: string = '';
claseRespuesta: string = '';
colorNombre: string = '';

```

En el constructor inyectamos los servicios que usaremos para la comunicación con la API, en el ngOnInit recogemos el id del empleado que tenemos en localStorage y cargamos los clientes llamando al servicio correspondiente.

```

constructor(
  public MainService: MainService,
  public loginService: LoginService
) {}

ngOnInit(): void {
  console.log(Number(localStorage.getItem('id')));
  this.cargaClientes();
}

cargaClientes() {
  this.loginService.data;
  this.MainService.getClientes();
  this.clientesSub = this.MainService.getClientesUpdatedListener().subscribe(
    (clientes: Cliente[]) => {
      this.clientes = clientes;
      for (let cliente of this.clientes) {
        cliente.rojo = { cantidad: 0, porcentaje: 0 };
        cliente.verde = { cantidad: 0, porcentaje: 0 };
        cliente.azul = { cantidad: 0, porcentaje: 0 };
        cliente.amarillo = { cantidad: 0, porcentaje: 0 };
        cliente.cantidadOpiniones = 0;
      }
    }
  );
  this.MainService.getOpiniones();
  this.opinionesSub =
    this.MainService.getOpinionesUpdatedListener().subscribe(
      (opiniones: Opinion[]) => {
        this.opiniones = opiniones;
      }
    );
  setTimeout(() => {
    this.cargarOpiniones();
  }, 500);
}

```

Cargamos las opiniones y modificamos en el array de clientes los campos cantidad y porcentaje de cada color así como las opiniones totales. Al final cuando ya esta completo el array lo

cargamos en una propiedad MatTableDataSource que es la que usa la tabla de Angular para tomar los datos.

```
cargarOpiniones() {
  console.log('carga opiniones');
  for (let cliente of this.clientes) {
    cliente.cantidadOpiniones = 0;
    for (let opinion of this.opiniones) {
      if (cliente.id == opinion.id_cliente) {
        switch (opinion.color) {
          case 0:
            cliente.rojo.cantidad = opinion.cantidad;
            break;
          case 1:
            cliente.verde.cantidad = opinion.cantidad;
            break;
          case 2:
            cliente.azul.cantidad = opinion.cantidad;
            break;
          case 3:
            cliente.amarillo.cantidad = opinion.cantidad;
            break;
        }
        cliente.cantidadOpiniones += opinion.cantidad;
      }
    }
    cliente.rojo.porcentaje = Math.floor(
      (cliente.rojo.cantidad / cliente.cantidadOpiniones) * 100
    );
    cliente.verde.porcentaje = Math.floor(
      (cliente.verde.cantidad / cliente.cantidadOpiniones) * 100
    );
    cliente.azul.porcentaje = Math.floor(
      (cliente.azul.cantidad / cliente.cantidadOpiniones) * 100
    );
    cliente.amarillo.porcentaje = Math.floor(
      (cliente.amarillo.cantidad / cliente.cantidadOpiniones) * 100
    );
    cliente.rojo.porcentaje = isNaN(cliente.rojo.porcentaje)
      ? 0
      : cliente.rojo.porcentaje;
    cliente.azul.porcentaje = isNaN(cliente.azul.porcentaje)
      ? 0
      : cliente.azul.porcentaje;
    cliente.verde.porcentaje = isNaN(cliente.verde.porcentaje)
      ? 0
      : cliente.verde.porcentaje;
    cliente.amarillo.porcentaje = isNaN(cliente.amarillo.porcentaje)
      ? 0
      : cliente.amarillo.porcentaje;
  }
  this.dataSource = new MatTableDataSource(this.clientes);
}
```

Las consultas en la API para recuperar los datos de clientes y las opiniones son las siguientes:

```

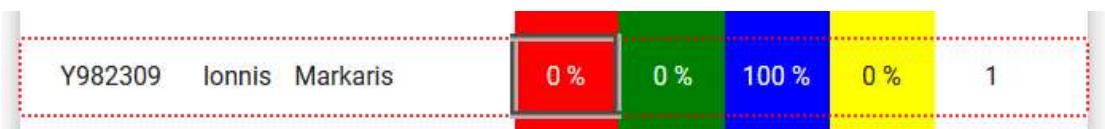
]app.get("/clientes", (req, res, next) => {
  con.query("select * from c4pi.cliente", function (err, result) {
    if (err) throw err;
    const respuesta = result;
    res.status(200).json(respuesta);
  });
});

[app.get("/opiniones", (req, res, next) => {
  con.query(
    "select id_cliente, color, count(color) as cantidad from c4pi.opinion group by id_cliente, color",
    function (err, result) {
      if (err) throw err;
      const respuesta = result;
      res.status(200).json(respuesta);
    }
);
});

```

A continuación están los métodos onColor y onColorLeave que son los que controlan los eventos mouseover y mouseleave de las celdas de colores respectivamente.

Estos métodos se usan para controlar la visualización de tal manera que cuando dejas el cursor sobre la celda se cambian las clases de la fila y de la propia celda para resaltar la propia celda con un borde más pronunciado y la fila con un borde puntos con el color correspondiente, se muestra así:



### Clases de CSS

```

.rojoRow{
  border: 2px dotted red;
}
verdeRow{
  border: 2px dotted green;
}
azulRow{
  border: 2px dotted blue;
}
amarilloRow{
  border: 2px dotted yellow;
}
.rojot:hover, .verdet:hover, .azult:hover, .amarillot:hover{
  border: 5px outset black !important;
}
.rojot{
  background-color: red;
  color: white !important;
  text-align: center !important;
  min-width: 10%;
  cursor: pointer;
}

.azult{
  background-color: blue;
  color: white;
  text-align: center !important;
  min-width: 10%;
  cursor: pointer;
}
.verdet{
  background-color: green;
  color: white;
  text-align: center !important;
  min-width: 10%;
  cursor: pointer;
}
.amarillot{
  background-color:yellow;
  text-align: center !important;
  min-width: 10%;
  cursor: pointer;
}

```

## Métodos de typescript

```
[ onColor(e: any) {
    let letra = e.target.className.replace('mat-cell cdk-cell ', '').charAt(1);
    let fila = e.target.parentNode;

    switch (letra) {
        case 'o':
            fila.className = fila.className + ' rojoRow';
            break;
        case 'z':
            fila.className = fila.className + ' azulRow';
            break;
        case 'e':
            fila.className = fila.className + ' verdeRow';
            break;
        case 'm':
            fila.className = fila.className + ' amarilloRow';
            break;
    }
}

onColorLeave(e: any) {
    let fila = e.target.parentNode;
    fila.className = fila.className.replace('rojoRow', '');
    fila.className = fila.className.replace('azulRow', '');
    fila.className = fila.className.replace('verdeRow', '');
    fila.className = fila.className.replace('amarilloRow', '');
}
```

Esta es la función que controla el cuadro de búsqueda

```
applyFilter(e: any) {
    let filterValue = e.target.value;
    this.dataSource.filter = filterValue.trim().toLowerCase();
}
```

Y, por último, el método que carga la nueva opinión o la actualiza en la base de datos.

Lo he configurado de tal modo que el empleado no pueda opinar más de una vez sobre el mismo cliente. Si esta opinando sobre un cliente del cual ya había opinado lo que hace es cambiar la opinión pero no crea una nueva. Avisa de ellos con un mensaje encima de la tabla.

```

onOpinion(
  color: number,
  id: number,
  nombre: string,
  apellido1: string,
  apellido2: string
) {
  this.selectedOpinion = {
    id_cliente: id,
    color: color,
    id_empleado: Number(localStorage.getItem('id')),
    cantidad: 0,
  };
  switch (color) {
    case 0:
      this.colorNombre = 'rojo';
      break;
    case 1:
      this.colorNombre = 'verde';
      break;
    case 2:
      this.colorNombre = 'azul';
      break;
    case 3:
      this.colorNombre = 'amarillo';
      break;
    default:
      break;
  }
  let confirmacion = confirm(
    `Quiere opinar que el caracter de ${nombre} ${apellido1} ${apellido2} es del color ${this.colorNombre}?`
  );
  if (confirmacion) {
    this.MainService.setOpinion(this.selectedOpinion);
    this.opinionSub = this.MainService.getOpinionUpdatedListener().subscribe(
      (respuesta: number) => {
        console.log(respuesta);
        switch (respuesta) {
          case 0:
            this.respuestaSetOpinion =
              'No se ha podido introducir la opinión';
            this.claseRespuesta = 'error';
            break;
          case 1:
            this.respuestaSetOpinion = `Se ha actualizado su opinión de ${nombre} ${apellido1} ${apellido2} al color ${this.colorNombre}`;
            this.claseRespuesta = 'ok';
            break;
          case 2:
            this.respuestaSetOpinion = `Se creado una nueva opinión de ${nombre} ${apellido1} ${apellido2} con el color ${this.colorNombre}`;
            this.claseRespuesta = 'ok';
            break;
        }
        this.cargaClientes();
      }
    );
  }
}

```

El método en la API que uso para insertar nuevas opiniones primero controla si ya existe una opinión con el id de cliente y de empleado en cuyo caso realiza un update y, si no existe, realiza un insert.

```

  app.post("/insertaOpinion", (req, res, next) => {
    const id_cliente = req.body.id_cliente;
    const id_empleado = req.body.id_empleado;

    const color = req.body.color;
    con.query(
      "SELECT * FROM c4pi.opinion WHERE id_cliente = ? and id_empleado = ?",
      [id_cliente, id_empleado],
      function (err, result, fields) {
        if (err) throw err;
        if (result.length > 0) {
          existeOpinion = true;
        } else {
          existeOpinion = false;
        }

        if (existeOpinion) {
          con.query(
            "UPDATE c4pi.opinion set color = ? where id_cliente = ? and id_empleado = ?",
            [color, id_cliente, id_empleado],
            function (err, result, fields) {
              if (err) throw err;
              if (result.affectedRows > 0) {
                respuesta = 1;
              } else {
                respuesta = 0;
              }
              res.status(200).json(respuesta);
            }
          );
        } else {
          con.query(
            "INSERT into c4pi.opinion VALUES(?, ?, ?, CURDATE())",
            [id_cliente, id_empleado, color],
            function (err, result, fields) {
              if (err) throw err;
              if (result.affectedRows > 0) {
                respuesta = 2;
              } else {
                respuesta = 0;
              }
              res.status(200).json(respuesta);
            }
          );
        }
      }
    );
  });
}

```

## Menú administrador

Para registrar un empleado como administrador utilice un campo llamado rango con dos valores 0 para empleados normales y 1 para administradores. En principio cada hotel solo puede tener un administrador. No he habilitado la opción para que un empleado pueda registrarse como administrador. Entiendo que por cuestiones de seguridad es algo que debería gestionar la empresa usuario directamente con el gestor de la aplicación.

No obstante para realizar las pruebas y poder mostrar la funcionalidad extra que se proporcionara a los administradores he creado un usuario administrador de prueba, utilizo este código de node:

```

function insertAdmin() {
  let hash = bcrypt.hashSync(process.env.HOTEL_ADMIN_PASSWORD, 5);
  con.query(
    "INSERT into c4pi.empleado(id_departamento,rango,login,password) VALUES(?, ?, ?, ?)",
    [process.env.DEP_ID, 1, "admin", hash],
    function (err, result, fields) {
      if (err) throw err;
      if (result.affectedRows > 0) {
        console.log("admin introducido correctamente");
      }
    }
  );
}

insertAdmin();

```

Cuando se usuario administrador realiza el login le aparecerá en la barra de navegación el botón para acceder a su menú específico, el resto de las opciones las tiene igualmente habilitadas:

The screenshot shows a web application interface. At the top, there is a dark header bar with the text "Menu administrador". Below this, the main content area has a light gray background. On the left, there is a "Logout" button. In the center, there is a logo consisting of a stylized eye icon followed by the text "C4PI". To the right of the logo, there are several menu items: "Ver opiniones", "Test de personalidad", and "Menu administrador". Below these menu items, the text "Lista de empleados del hotel Novotel Madrid Center" is displayed. Underneath this text, there is a table-like structure with a header row containing "Nombre" and "Eliminar empleado". The data rows show three employees: "jorge", "clara", and "Antonio", each with a small trash can icon next to their name.

Nombre	Eliminar empleado
jorge	
clara	
Antonio	

Como se puede ver es un menú sencillo donde se informa al administrador de los empleados de su hotel registrados en la aplicación. Se le proporciona la opción de borrarlos.

En el html simplemente se muestra una tabla de materials con un filtro como el que vimos en la pantalla principal.

En la lógica del componente el método ngOnInit que se ejecuta al principio se comprueba que el empleado se administrador y, si es así, se llama al servicio que recoge los datos de empleados del hotel correspondiente.

```

ngOnInit(): void {
  if (localStorage.getItem('rango') !== '1') {
    this.router.navigate(['home/login']);
  }
  let id = localStorage.getItem('id') as unknown as number;
  const idEnvio = { id: id };
  this.mainService.getEmpleados(idEnvio);
  this.empleadosSub = this.mainService
    .getEmpleadosUpdatedListener()
    .subscribe((response: Empleado[]) => {
      this.empleados = response;
      this.hotel = this.empleados[0].hotel;
      this.dataSource = new MatTableDataSource(this.empleados);
      console.log(this.empleados);
    });
}

```

En el backend la consulta se realiza aquí:

```

app.post("/empleados", (req, res, next) => {
  const id = req.body.id;
  con.query(
    "select e.id,e.login,e.rango,e.color,h.nombre as hotel from c4pi.empleado e
    JOIN departamento d on e.id_departamento=d.id join hotel h on h.id=d.id_hotel where h.id IN (d.id_hotel)
    AND d.id IN(SELECT id_departamento from c4pi.empleado where id = ?) AND e.id!=?",
    [id, id],
    function (err, result) {
      if (err) throw err;
      const respuesta = result;
      res.status(200).json(respuesta);
    }
  );
});

```

Lógicamente contiene un método para gestionar el borrado de empleados:

```

  deleteEmpleado(id: number, nombre: string) {
    console.log('empleado a eliminar' + id);
    let confirmacion = confirm(
      `Esta seguro de que quiere eliminar al empleado ${nombre}`
    );
    if (confirmacion) {
      this.mainService.deleteEmpleado({ id: id as unknown as number });
      this.deleteEmpleadoSub = this.mainService
        .getDeleteEmpleadoUpdatedListener()
        .subscribe((respuesta: boolean) => {
          this.errorDelete = !respuesta;
          this.respuestaDelete = this.errorDelete
            ? `No se ha podido borrar al empleado ${nombre}`
            : '';
          if (respuesta) {
            window.location.reload();
          }
        });
    }
  }
}

```

Y en el backend:

```

  app.post("/deleteEmpleado", (req, res, next) => {
    const id = req.body.id;
    con.query(
      "DELETE from c4pi.empleado where id = ?",
      [id],
      function (err, result, fields) {
        if (err) throw err;
        if (result.affectedRows > 0) {
          respuesta = true;
        } else {
          respuesta = false;
        }
        res.status(200).json(respuesta);
      }
    );
  });
}

```

## Test

La pantalla de test le proporciona la opción de realizar un test de personalidad al empleado.

### Interfaz usuario

Aquí se puede ver:

The screenshot shows a web-based personality test interface. At the top, there's a navigation bar with 'Logout' and 'C4PI' logo, followed by 'Test personalidad'. Below the navigation, there's a section titled 'Instrucciones' (Instructions) with some general text. The main part consists of a grid of statements, each with two radio buttons for rating. The statements are as follows:

- 1A: 0 [radio] En general soy abierto cuando se trata de conocer a la gente personalmente y establecer relaciones. Introduce la puntuación del 0 al 3.
- 1B: 0 [radio] En general no me resulta fácil establecer relaciones personales con la gente. Introduce la puntuación del 0 al 3.
- 2A: 0 [radio] Normalmente reacciono despacio y de manera deliberada. Introduce la puntuación del 0 al 3.
- 2B: 0 [radio] Normalmente reacciono rápidamente y con espontaneidad. Introduce la puntuación del 0 al 3.
- 3A: 0 [radio] No dejo que los demás malgasten mi tiempo. Introduce la puntuación del 0 al 3.
- 3B: 0 [radio] No pongo reparos a los demás cuando necesitan de mi tiempo. Introduce la puntuación del 0 al 3.
- 4A: 0 [radio] Me presento a mí mismo en reuniones sociales. Introduce la puntuación del 0 al 3.
- 4B: 0 [radio] Espero que los demás se presenten a mí en reuniones sociales. Introduce la puntuación del 0 al 3.
- 5A: 0 [radio] Suelo tomar decisiones basándome en hechos y en evidencias. Introduce la puntuación del 0 al 3.
- 5B: 0 [radio] Suelo tomar decisiones basadas en sentimientos, experiencias o relaciones. Introduce la puntuación del 0 al 3.
- 6A: 0 [radio] Suelo ser paciente con mi interlocutor cuando se expresa / habla más despacio que yo. Introduce la puntuación del 0 al 3.
- 6B: 0 [radio] Puedo ser impaciente cuando mi interlocutor se expresa / habla más despacio que yo. Introduce la puntuación del 0 al 3.

At the bottom right of the form area, there is a 'Enviar' (Send) button.

En la parte de arriba he puesto un menú desplegable que permite ver las instrucciones para realizar el test:

This screenshot shows the 'Instrucciones' (Instructions) section of the test. It contains the following text:

Para cada uno de los siguientes pares de afirmaciones (6) distribuir tres puntos entre las dos alternativas A y B, basándose en cómo actuaría en situaciones cotidianas. Aunque en algunos casos las dos opciones le puedan parecer válidas, asignar más puntos a la opción que mejor representa su comportamiento la mayor parte del tiempo.

Si A le caracteriza y B no le caracteriza, asignar 3 puntos a A y 0 a B.

Si A le resulta más característico que B pero B le resulta plausible en algunas ocasiones, asignar 2 puntos a A y uno a B.

Si B le caracteriza más que A, pero A le resulta plausible, asignar 2 puntos a B y uno a A.

Si B le caracteriza y A no le caracteriza para nada, asignar 3 puntos a B y 0 a A.

En todos los casos los puntos asignados a ambas opciones deben sumar tres.

Como se puede ver en las instrucciones el test se compone de varios pares de opciones entre las cuales hay que seleccionar una repartir una puntuación de 3 según lo que más te represente. Para facilitar la experiencia de usuario el formulario va informando si no has puesto justo 3 puntos entre las dos opciones:

<b>1A:</b> 1 <input type="radio"/> En general soy abierto cuando se trata de conocer a la gente personalmente y establecer relaciones.  La suma de ambas opciones tiene que ser 3. Introduzca la puntuación del 0 al 3	<b>1B:</b> 1 <input type="radio"/> En general no me resulta fácil establecer relaciones personales con la gente.  La suma de ambas opciones tiene que ser 3. Introduzca la puntuación del 0 al 3
--	--

Si lo has puesto correctamente desaparece el mensaje de error y el nombre de la opción aparece en azul:

<b>1A:</b> 1 <input type="radio"/> En general soy abierto cuando se trata de conocer a la gente personalmente y establecer relaciones.  Introduzca la puntuación del 0 al 3	<b>1B:</b> 2 <input type="radio"/> En general no me resulta fácil establecer relaciones personales con la gente.  Introduzca la puntuación del 0 al 3
--	--

Además el botón de enviar solo se habilita cuando se han llenado correctamente las opciones y si deshabilita si no es así.

## HTML

Para el menú desplegable uno el componente accordion de materials:

```
<div class="row align-items-center">
  <div class="col-12">
    <mat-accordion>
      <mat-expansion-panel (opened)="panelOpenState = true" (closed)="panelOpenState = false">
        <mat-expansion-panel-header>
          <mat-panel-title> Instrucciones </mat-panel-title>
        </mat-expansion-panel-header>
        <p>
          Para cada uno de los siguientes pares de afirmaciones (6) distribuir tres puntos entre las dos alternativas A y B, basándose en cómo actuaría en situaciones cotidianas. Aunque en algunos casos las dos opciones le puedan parecer válidas, asignar más puntos a la opción que mejor representa su comportamiento la mayor parte del tiempo.
        </p>
        <p>
          Si A le caracteriza y B no le caracteriza, asignar 3 puntos a A y 0 a B.
        </p>
        <p>
          Si A le resulta más característico que B pero B le resulta plausible en algunas ocasiones, asignar 2 puntos a A y uno a B.
        </p>
        <p>
          Si B le caracteriza más que A, pero A le resulta plausible, asignar 2 puntos a B y uno a A.
        </p>
        <p>
          Si B le caracteriza y A no le caracteriza para nada, asignar 3 puntos a B y 0 a A.
        </p>
        <p>
          En todos los casos los puntos asignados a ambas opciones deben sumar tres.
        </p>
      </mat-expansion-panel>
    </mat-accordion>
  </div>
</div>
```

Para este formulario uso el tipo reactive forms que explique en un apartado anterior.

Respecto a las opciones visualmente uso los componentes de materials que usé en anteriores componentes. Cada input de número llama al método correspondiente pasando el título de la opción, el mensaje de error se gestiona con un array de errores.

```
<mat-card>
  <mat-card-content>
    <mat-form-field>
      <mat-label>1A:</mat-label>

      <input matInput type="number" placeholder="puntuacion" name="1A" formControlName="1A" min="0" max="3"
        (change)="onChangeNumber('1A')" />
      <span>En general soy abierto cuando se trata de conocer a la gente
        personalmente y establecer relaciones.</span>
      <mat-hint>Introduzca la puntuación del 0 al 3</mat-hint>
      <div class="errorNumeros" *ngIf="errorNumeros[0]">
        La suma de ambas opciones tiene que ser 3.
      </div>
    </mat-form-field>
    <mat-form-field>
      <mat-label>1B:</mat-label>
      <input matInput type="number" placeholder="puntuacion" name="1B" formControlName="1B" min="0" max="3"
        (change)="onChangeNumber('1B')" />
      <span>En general no me resulta fácil establecer relaciones personales
        con la gente.
      </span>
      <mat-hint>Introduzca la puntuación del 0 al 3</mat-hint>
      <div class="errorNumeros" *ngIf="errorNumeros[0]">
        La suma de ambas opciones tiene que ser 3.
      </div>
    </mat-form-field>
  </mat-card-content>
</mat-card>
```

El botón de enviado usa una propiedad para comprobar si está todo correcto y activarse. En este caso usa el atributo disabled de angular:

```
<mat-card-actions class="mat-card-button-login">
  <button mat-raised-button type="submit" [disabled]="testIncompleto" color="primary">
    Enviar
  </button>
</mat-card-actions>
```

Y, por último, un par de opciones que muestran si se ha introducido correctamente el color:

```
<div class="row align-items-center">
  <div class="col-12 text-center">
    <h3 class="ok" *ngIf="!errorSetColor">{{respuestaSetColor}}</h3>
    <h3 class="error" *ngIf="errorSetColor">{{respuestaSetColor}}</h3>
  </div>
</div>
```

## Typescript

Este componente se gestiona principalmente con dos métodos, OnChangeNumber que es el método que gestiona la visualización cuando se van cambiando la puntuación de las diferentes opciones. Y OnSubmit que gestiona el botón de enviar.

### OnChangeNumber

Primero declaro una serie de variables, cabe destacar el number y la letra que recojen el número y letra de la opción, recordemos que recibe el título que puede ser 1B O 5A por ejemplo.

```
let element;
let value;
let number = key.charAt(0);
let letra = key.charAt(1);
let valuePar;
let numberValid = false;
let error: boolean = false;
let indiceLinea: number = 0;
```

Recoge el número introducido por el usuario como el nombre del formulario y usando get con la clave, esto es del reactive form de angular:

```
if (this.test.get(key) !== null) {
    element = this.test.get(key);
}
if (element !== null) {
    value = element?.value;
}
```

Recoge el valor de su casilla par, es decir si ha recibido la 1A comprueba la 1B o viceversa, para después comprobar si suman entre ambas exactamente 3.

```
if (letra === 'A') {
    valuePar = this.test.get(number + 'B');
} else {
    valuePar = this.test.get(number + 'A');
}

if (valuePar !== null) {
    if (value + valuePar.value === 3) {
        numberValid = true;
    }
}
```

Para controlar la visualización tengo que controlar los elementos de angular mat-label y mat-form-field-ripple que son la etiqueta del input y la línea que sale debajo. No quiero que aparezcan rojas si esta todo correcto ni azules si algo esta mal. Para ello las recojo en un array de elementos html y, con un índice que recojo previamente según la opción que haya seleccionado el usuario, les pongo la clase que tiene el formato correcto.

```

        error = value + valuePar.value === 3;
    switch (number) {
        case '1':
            this.errorNumeros[0] = error;
            indiceLinea = letra === 'B' ? 0 : 1;
            break;
        case '2':
            this.errorNumeros[1] = error;
            indiceLinea = letra === 'B' ? 2 : 3;
            break;
        case '3':
            this.errorNumeros[2] = error;
            indiceLinea = letra === 'B' ? 4 : 5;
            break;
        case '4':
            this.errorNumeros[3] = error;
            indiceLinea = letra === 'B' ? 6 : 7;
            break;
        case '5':
            this.errorNumeros[4] = error;
            indiceLinea = letra === 'B' ? 8 : 9;
            break;
        case '6':
            this.errorNumeros[5] = error;
            indiceLinea = letra === 'B' ? 10 : 11;
            break;
    }
}

let indice2 = indiceLinea % 2 === 0 ? indiceLinea + 1 : indiceLinea - 1;
const linea = document.getElementsByClassName('mat-form-field-ripple')[indiceLinea];
const linea2 = document.getElementsByClassName('mat-form-field-ripple')[indice2];
const titulo = document.getElementsByTagName('mat-label')[indiceLinea];
const titulo2 = document.getElementsByTagName('mat-label')[indice2];
if (linea !== undefined && linea2 !== undefined) {
    linea.className = error ? 'lineaIncorrecta' : 'lineaCorrecta';
    linea2.className = error ? 'lineaIncorrecta' : 'lineaCorrecta';
}
if (titulo !== undefined && titulo2 !== undefined) {
    titulo.className = error ? 'tituloIncorrecto' : 'tituloCorrecto';
    titulo2.className = error ? 'tituloIncorrecto' : 'tituloCorrecto';
}

```

Por último suma todas las puntuaciones introducidas para ver si se ha completado el test y activar el botón de enviar.

```

let valor = 0;
for (let i = 1; i < 7; i++) {
    valor += this.test.get(i + 'A')?.value as unknown as number;
    valor += this.test.get(i + 'B')?.value as unknown as number;
}

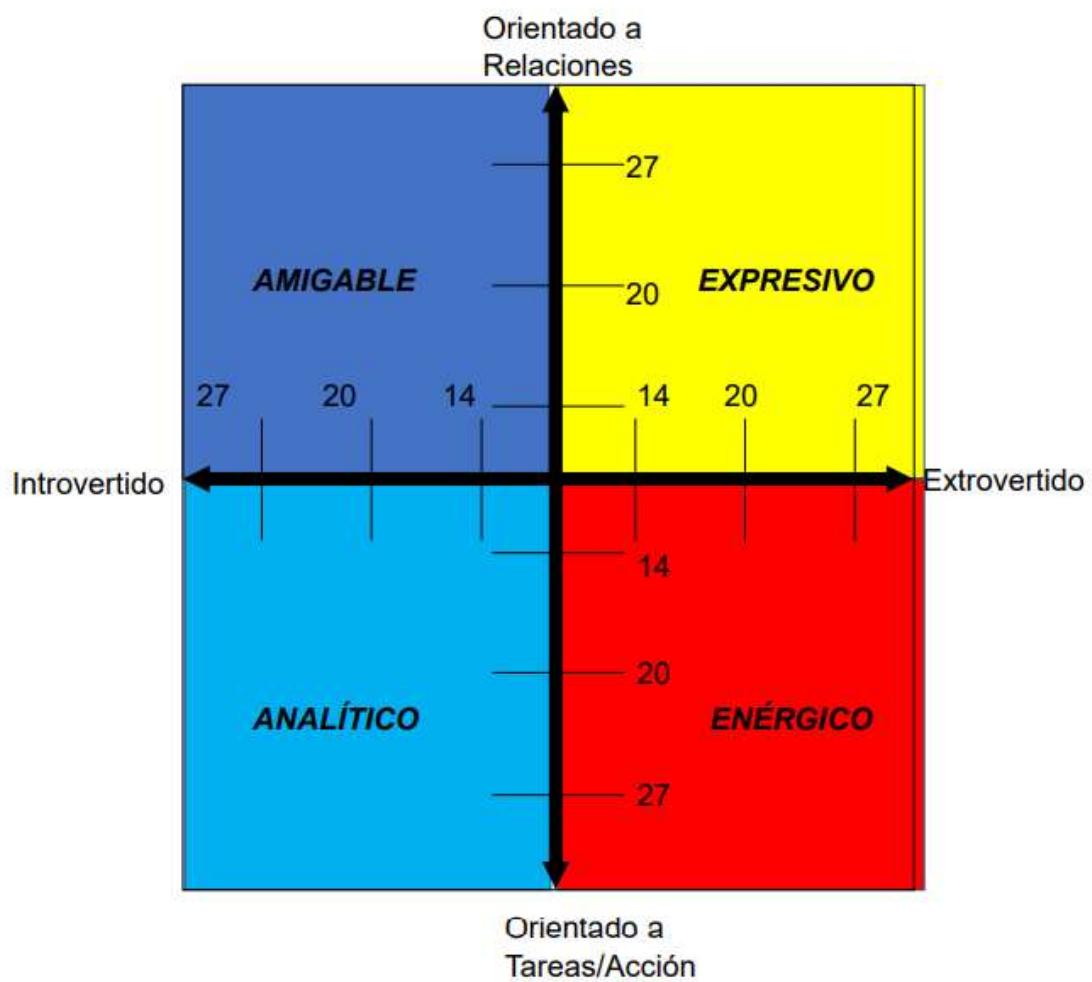
if (valor === 6 * 3) {
    this.testIncompleto = false;
} else {
    this.testIncompleto = true;
}

```

onSubmit

Para poder recoger el color del cliente utilice las instrucciones del test que se pueden ver a continuación, el color superior izquierda debería ser verde. Aunque el cuestionario oficial se compone de 18 pares de opciones he decidido reducirlo a 6 para hacerlo un poco más dinámico de cara a la demostración.

<b>PUNTUACIÓN DEL CUESTIONARIO</b>			
<b>R</b>	<b>T</b>	<b>E</b>	<b>I</b>
1A	1B	2B	2A
3B	3A	4A	4B
5A	5B	6B	6A
7B	7A	8A	8B
9A	9B	10B	10A
11B	11A	12A	12B
13A	13B	14B	14A
15B	15A	16A	16B
17A	17B	18B	18 <sup>a</sup>
<b>Total R:</b>	<b>Total T:</b>	<b>Total E:</b>	<b>Total I:</b>
Compare las puntuaciones R y T. Escribir la puntuación más alta de las dos en el espacio a continuación y señalar la letra correspondiente: _____			
Compare las puntuaciones E e I. Escribir la puntuación más alta de las dos en el espacio a continuación y señalar la letra correspondiente: _____			



Para ello uso 4 variables que son números que recogen los valores que vemos arriba, recorro las opciones del test y voy sumando cada valor introducido por el usuario a la variable correspondiente.

```

onSubmit(form: FormGroup) {
  let valorA = 0;
  let valorB = 0;
  let relaciones = 0;
  let tareas = 0;
  let extrovertido = 0;
  let introvertido = 0;
  let color = -1;
  let colorNombre: string;
  for (let i = 1; i < 7; i++) {
    valorA = this.test.get(i + 'A')?.value as unknown as number;
    valorB = this.test.get(i + 'B')?.value as unknown as number;
    switch (i) {
      case 1:
        relaciones += valorA;
        tareas += valorB;
        break;
      case 2:
        extrovertido += valorB;
        introvertido += valorA;
        break;
      case 3:
        relaciones += valorB;
        tareas += valorA;
        break;
      case 4:
        extrovertido += valorA;
        introvertido += valorB;
        break;
      case 5:
        relaciones += valorA;
        tareas += valorB;
        break;
      case 6:
        extrovertido += valorB;
        introvertido += valorA;
        break;
    }
  }
}

```

Una vez recogidos los valores compruebo que dos de ellos identifican mas al usuario y le asigno su color correspondiente usando el servicio correspondiente.

```

        if (introvertido > extrovertido) {
            //verde introvertido+relaciones
            //azul introvertido+tareas
            color = relaciones > tareas ? 1 : 2;
        } else {
            //amarillo extrovertido+relaciones
            //rojo extrovertido+tareas
            color = relaciones > tareas ? 3 : 0;
        }
        console.log(color);
        switch (color) {
            case 0:
                colorNombre = 'rojo';
                break;
            case 1:
                colorNombre = 'verde';
                break;
            case 2:
                colorNombre = 'azul';
                break;
            case 3:
                colorNombre = 'amarillo';
                break;
        }
    let id_empleado = localStorage.getItem('id') as unknown as number;
    this.mainService.setColorEmpleado({ id: id_empleado, color: color });
    this.setColorEmpleadoSub = this.mainService
        .getSetColorUpdatedListener()
        .subscribe((respuesta: boolean) => {
            if (respuesta) {
                localStorage.setItem('color', color as unknown as string);
                this.errorSetColor = false;
                this.respuestaSetColor = `Se ha actualizado correctamente su color a ${colorNombre}`;
                // setTimeout(() => {
                //     window.location.reload();
                // }, 2000);
            } else {
                this.errorSetColor = true;
                this.respuestaSetColor = 'No se ha podido actualizar su color';
            }
        });
    });
}

```

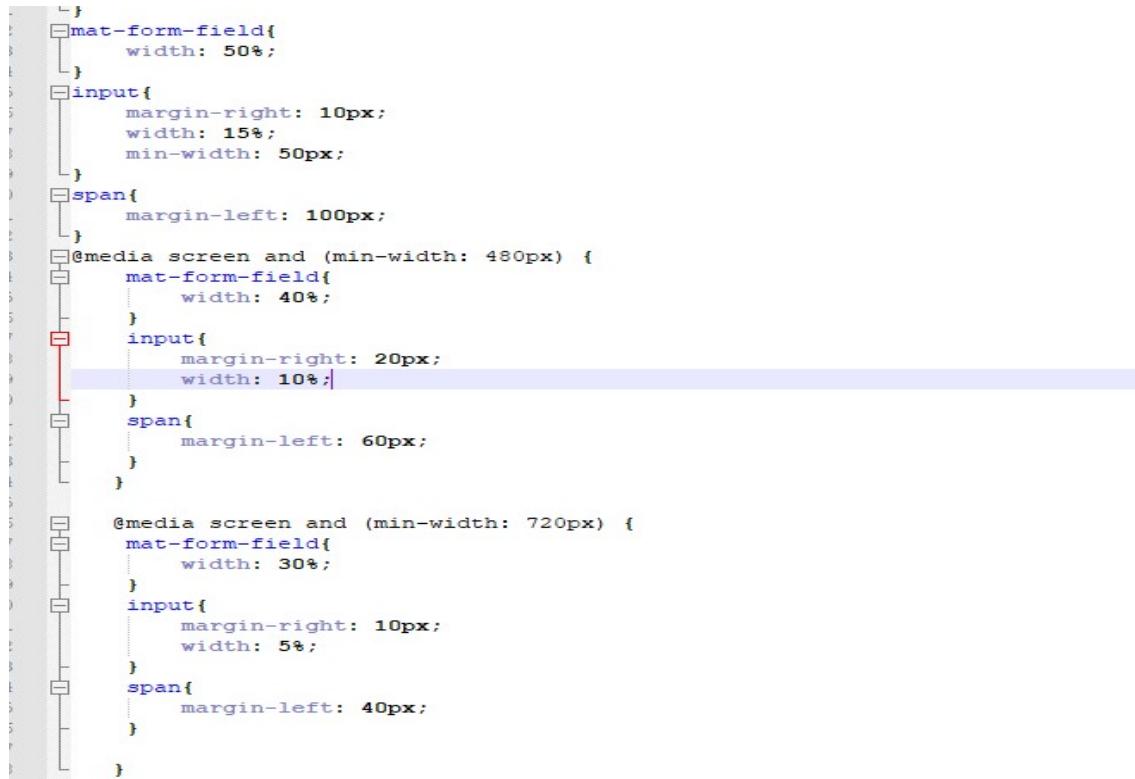
## CSS

En la hoja de estilos uso una serie de clases para controlar la visualización de errores:

```
        }
    .errorNumeros{
        font-size: small;
        color: red;
        margin-top: 10px;
    }
    .lineaCorrecta{
        background-color: blue !important;
        color: blue !important;
    }
    .lineaIncorrecta{
        background-color: red !important;
        color: red !important;
    }
    .tituloCorrecto{
        color: blue !important;
    }
    .tituloIncorrecto{
        color: red !important;
    }
    .ok{
        color: green;
        font-size: large;
    }
    .black{
        font-size: 2em;
        color: #F7F9F9;
        background-color: rgba(0, 0, 0, 0.75);
    }
    - - - - -
```

Y una serie de media queries para controlar el tamaño de los inputs y el texto de las opciones.

```
        }
    mat-form-field{
        width: 50%;
    }
    input{
        margin-right: 10px;
        width: 15%;
        min-width: 50px;
    }
    span{
        margin-left: 100px;
    }
    @media screen and (min-width: 480px) {
        mat-form-field{
            width: 40%;
        }
        input{
            margin-right: 20px;
            width: 10%;
```



```
        }
        span{
            margin-left: 60px;
        }
    }

    @media screen and (min-width: 720px) {
        mat-form-field{
            width: 30%;
        }
        input{
            margin-right: 10px;
            width: 5%;
        }
        span{
            margin-left: 40px;
        }
    }
```

## Seguridad

Voy a explicar varias medidas de seguridad implementadas tanto en el back como el front.

### Sanitización de las interpolaciones de datos con Angular

Cuando usamos la interpolación de elementos con Angular, ya sea con atributos como ngClass o directamente con los símbolos {{}} para insertar algún dato desde la lógica del componente, Angular sanea automáticamente estos elementos que insertamos en el DOM. Esto lo hace para prevenir ataques cross-site scripting, es decir que se pueda injectar código malicioso en nuestra web. Por este motivo es desaconsejable realizar manipulaciones del DOM como se suele hacer con javascript, usando las propiedades y funciones de document. Es por ello que he usado lo mínimo indispensable este tipo de manipulaciones y he usado casi siempre las interpolaciones de angular que he venido explicando en anteriores apartados.

### Bindeo de datos

Para usar cualquier dato en una consulta a la base de datos se bindea previamente, por ejemplo en esta función vemos como no se pasa el dni directamente en la consulta:

```
app.post("/comprobacion_dni", (req, res, next) => {
  const dni = req.body.envio;
  let respuesta;

  con.query(
    "SELECT dni from c4pi.cliente where dni=?",
    [dni],
    function (err, result) {
      if (err) throw err;
      if (result.length > 0) {
        respuesta = true;
      } else {
        respuesta = false;
      }
      res.status(200).json(respuesta);
    }
  );
});
```

### Hasdeo de contraseñas

Antes de usar las contraseñas en la base de datos usamos la extensión de node bcrypt que permite hashearlas antes de almacenarlas.

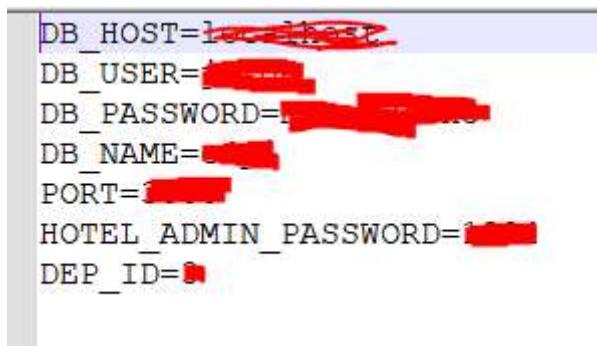
```
const nombre = req.body.nombre;
const password = req.body.password;
let rango = 0;
let hash = bcrypt.hashSync(password, 5);
con.query(
  "INSERT into c4pi.empleado(id_departamento,rango,login,password) VALUES(?, ?, ?, ?)",
  [id_departamento, rango, nombre, hash],
  function (err, result, fields) {
    if (err) throw err;
```

## Fichero de environment

Para evitar subir datos sensibles al repositorio de GitHub, que para mas inri es público, he usado un documento para poner los datos de conexión. Esto permite hacerlo la extensión dotenv de node. Notese en la siguiente imagen como no pongo directamente los datos de conexión:

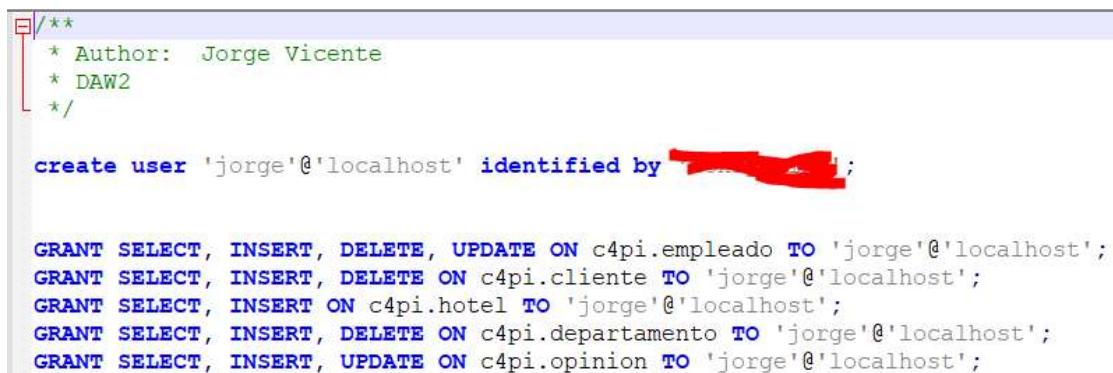
```
var con = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
});
```

En su lugar los toma de un fichero de environment que lógicamente no subo al repositorio.



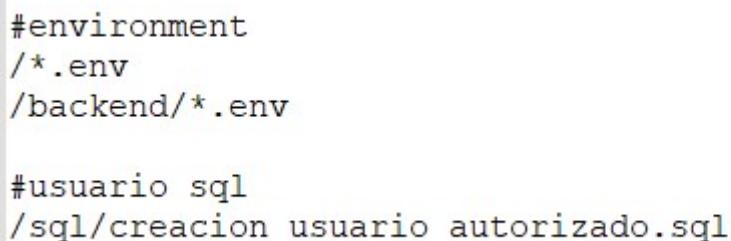
DB\_HOST=██████████  
DB\_USER=██████████  
DB\_PASSWORD=██████████  
DB\_NAME=██████████  
PORT=██████████  
HOTEL\_ADMIN\_PASSWORD=██████████  
DEP\_ID=██████████

Tampoco subo al repositorio el script de creación de usuario de seguridad en la base de datos.



```
/**  
 * Author: Jorge Vicente  
 * DAW2  
 */  
  
create user 'jorge'@'localhost' identified by ██████████;  
  
GRANT SELECT, INSERT, DELETE, UPDATE ON c4pi.empleado TO 'jorge'@'localhost';  
GRANT SELECT, INSERT, DELETE ON c4pi.cliente TO 'jorge'@'localhost';  
GRANT SELECT, INSERT ON c4pi.hotel TO 'jorge'@'localhost';  
GRANT SELECT, INSERT, DELETE ON c4pi.departamento TO 'jorge'@'localhost';  
GRANT SELECT, INSERT, UPDATE ON c4pi.opinion TO 'jorge'@'localhost';
```

Para ello añado los ficheros mencionados al .gitignore:



```
#environment  
/*.env  
/backend/*.env  
  
#usuario sql  
/sql/creacion_usuario_autorizado.sql
```

## Unsubscribe

Como comentaba en anteriores apartados para realizar las conexiones al back usamos el objeto Subscription de Angular. Este tipo de objeto tiene un método que ayuda a prevenir fugas de memoria. Por este motivo siempre que realizamos una subscripción la guardamos primero en una variable como en este ejemplo:

```
this.setColorEmpleadoSub = this.mainService
  .getColorUpdatedListener()
  .subscribe((respuesta: boolean) => {
    if (respuesta) {
      localStorage.setItem('color', color as unknown as string);
      this.errorSetColor = false;
      this.respuestaSetColor = `Se ha actualizado correctamente su color a ${colorNombre}`;
      // setTimeout(() => {
      //   window.location.reload();
      // }, 2000);
    } else {
      this.errorSetColor = true;
      this.respuestaSetColor = 'No se ha podido actualizar su color';
    }
});
```

Luego en el método ngOnDestroy que se ejecuta siempre que se destruye el componente usamos el método unsubscribe sobre esa subscripción:

```
ngOnDestroy() {
  if(this.setColorEmpleadoSub !== undefined){
    this.setColorEmpleadoSub.unsubscribe();
  }
}
```

Esto lo hago con todas las suscripciones.

## Auth guard

Como ya he comentado aunque Angular funciona como una Single Page Application si que se pueden asignar los componentes a diferentes URL. Esto puede producir fallas de seguridad si un usuario introduce manualmente una URL que le de acceso a alguna parte de la web a la que no debería poder acceder.

Una manera de prevenir esto es con el servicio Auth guard, este servicio permite crear algún método con las comprobaciones que queramos para prevenir el acceso a determinadas rutas.

Lo vemos aquí que comprueba si el usuario se ha registrado, en caso de no estarlo le redirige a la pantalla de login:

```


    '',
    export class AuthGuardService implements CanActivate {
      constructor(public loginService: LoginService, public router: Router) {}
      canActivate() {
        if (this.usuarioAutentificado()) {
          return true;
        } else {
          this.router.navigate(['/home/login']);
          return false;
        }
      }
      usuarioAutentificado() {
        if (
          localStorage.getItem('id') === undefined ||
          localStorage.getItem('id') === null
        ) {
          return false;
        } else {
          return true;
        }
      }
    }
  

```

Para activar este servicio en las rutas deseadas lo debemos especificar en app-routing indicando la propiedad CanActivate con el servicio auth guard

```


const appRoutes: Routes = [
  {
    path: '',
    pathMatch: 'full',
    redirectTo: 'home/login',
  },
  { path: 'home/login', component: LoginComponent },
  { path: 'home/registro', component: RegistrationComponent },
  {
    path: 'home/main',
    component: MainCardComponent,
    canActivate: [AuthGuardService],
  },
  {
    path: 'home/test',
    component: TestComponent,
    // canActivate: [AuthGuardService],
  },
  {
    path: 'home/admin',
    component: AdminComponent,
    canActivate: [AuthGuardService],
  },
  {
    path: '**',
    redirectTo: 'home/login',
  },
];
  

```

## Bibliografía

Documentación oficial Angular

<https://angular.io/docs>

Biblioteca materials de Angular

<https://v7.material.angular.io/>

Creación de temas personalizados con angular

<https://www.digitalocean.com/community/tutorials/angular-angular-material-custom-theme>

Significado de las Fuentes

<https://www.creadictos.com/significados-tipografias/>

Como añadir una fuente

<https://reactgo.com/add-fonts-to-angular/>

Creación de documento environment

<https://www.freecodecamp.org/news/how-to-use-node-environment-variables-with-a-dotenv-file-for-node-js-and-npm/>

Creación navbar responsiva

<https://zoaibkhan.com/blog/create-a-responsive-toolbar-in-angular-using-flex-layout/>

Botones con CSS

<https://cssbuttongenerator.com/>

Seguridad con Angular

<https://angular.io/guide/security>

“Si uno no entiende a otra persona tiende a considerarlo un loco”

Carl Jung