El concepto de TypeScript interface es bastante más flexible que el concepto clásico de Java Interface y nos permite realizar operaciones curiosas con él a nivel de arquitectura. Vamos a ver un ejemplo sencillo que incluya el manejo de un servicio REST que nos devuelve un DTO (Data Transfer Object) con Angular. El primer paso es construir un servicio REST. Para ello nos vamos a apoyar en express.js y node ya que nos permite un código que entienda todo el mundo.

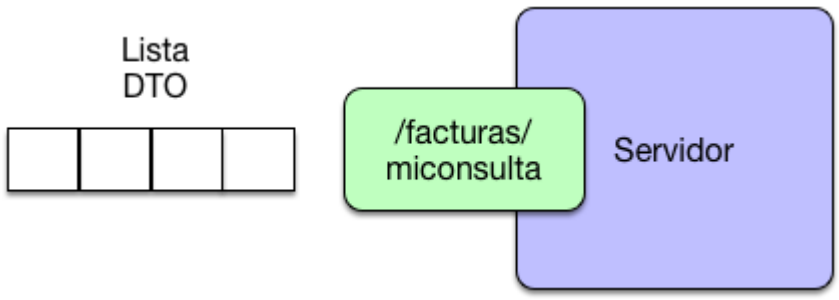
```
var express = require('express');
var cors = require('cors')
listaFacturacion=[];
var app = express();
app.use(cors());

listaFacturacion.push({numero:1,cliente:"pedro","importe":200,direccio
n:"micalle",vip:true});
listaFacturacion.push({numero:2,cliente:"juan","importe":300,direccion
:"otra calle",vip:true});
listaFacturacion.push({numero:3,cliente:"miguel","importe":400,direcci
on:"ninguna calle",vip:false});

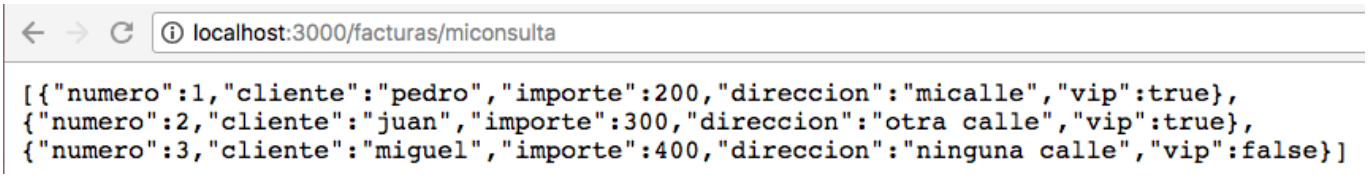

app.get('/facturas/miconsulta', function (req, res) {
  res.send(listaFacturacion);
});


app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

En este caso el servicio REST nos devuelve una lista con Data Transfer Object .

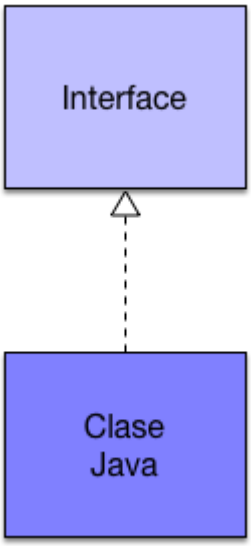


Es decir un conjunto de datos que digamos es una combinación de varios objetos de negocio y que necesitamos presentar en el interface de usuario.

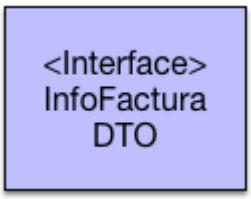


TypeScript Interface e implementaciones

En el mundo Java necesitaríamos construir una clase que contenga todos los campos que se incluyen en la respuesta JSON.



Esto no supone ningún problema. Sin embargo nos obliga a construir más clases Java en nuestro código que no pertenecen de forma clara al modelo de negocio. ¿Cómo podemos enfocar en TypeScript? . Al tratarse de un lenguaje más moderno y estar basado en la flexibilidad de JavaScript podemos simplemente definir un interface para nuestro DTO, no hace falta tener una clase.



Veamos su código:

```
export interface InfoFacturaDto {

    numero:number;
    cliente:string;
```

```
    importe:number;
    direccion:string;
    vip:true;

}
```

Este interface contiene todas las propiedades que necesita la estructura JSON . Por lo tanto no nos queda más que construirnos un componente invocar al servicio REST y clarificar que lo que devolvemos es de este tipo de interface.

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import {InfoFacturaDto} from "../info-factura-dto";

@Component({
  selector: 'app-hola002',
  templateUrl: './hola002.component.html',
  styleUrls: ['./hola002.component.css']
})
export class Hola002Component implements OnInit {

  lista:InfoFacturaDto[];
  constructor(private http: HttpClient) {

this.http.get<InfoFacturaDto[]>("http://localhost:3000/facturas/micons
ulta").subscribe((datos)=> {

            this.lista= datos as InfoFacturaDto[];


       });
```

```
  }

  ngOnInit() {

  }

}
```

No hace falta tener ningún tipo de implementación del interface para que este funcione y nos muestre los datos. Recordemos que el componente de angular necesitara una estructura *ngFor para imprimir los datos.

```
  {{elemento.numero}}
  {{elemento.cliente}}
  {{elemento.importe}}
  {{elemento.direccion}}
  {{elemento.vip}}
```

Acabamos de usar un typescript interface para gestionar información JSON de un servicio REST.

1 pedro 200 micalle true

2 juan 300 otra calle true

3 miguel 400 ninguna calle false

Otros artículos relacionados:

1. Angular router y su configuración
2. Angular Modules y el uso de servicios
3. Angular async pipe y observables
4. Angular 5 Hello World y su funcionamiento

Links Externos:

1. Angular