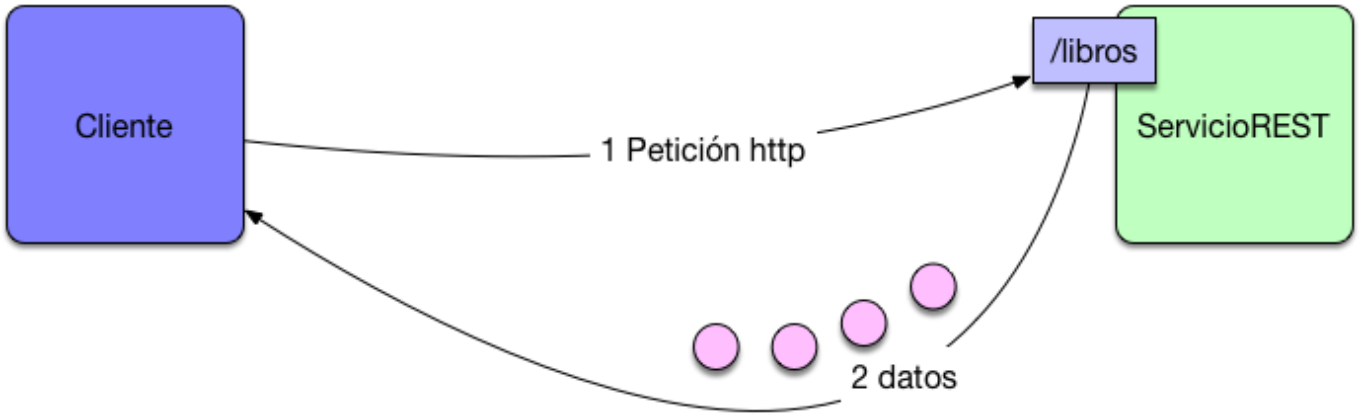
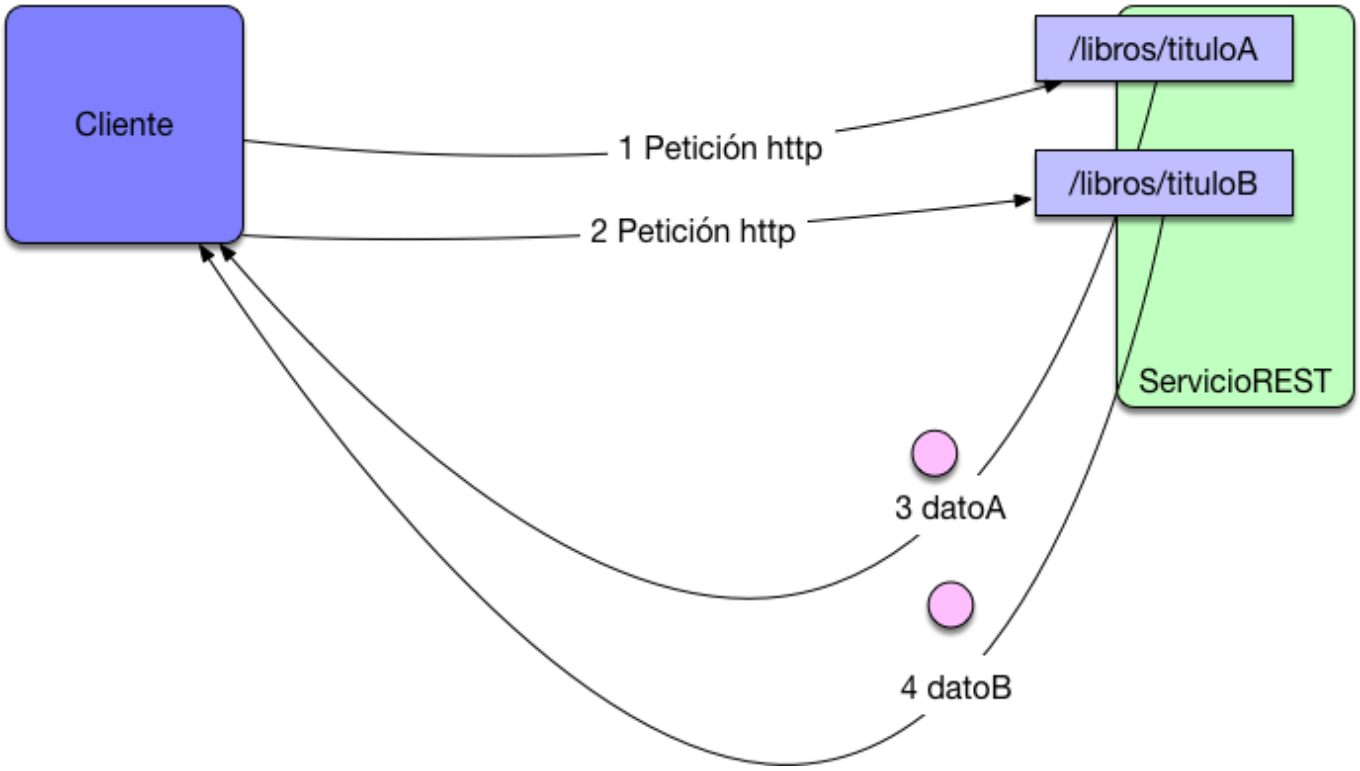
Typescript Promise All es uno de los conceptos más habituales cuando trabajamos con programación asíncrona y promesas en TypeScript. En situaciones sencillas nosotros invocamos a un servicio REST y nos traemos los datos que se encuentran ubicamos en una url .



Sin embargo no siempre las cosas son tan sencillas , hay situaciones en las que no controlamos quien construye los servicios REST y necesitamos realizar varias peticiones para obtener los datos que queremos . Por ejemplo puede ser que exista una URL de /libros que nos devuelve todos los libros y una url /libros/titulo que nos devuelva un libro en concreto. Podemos encontrarnos situaciones en las que necesitemos un subconjunto de libros por ejemplo aquellos que tienen tituloA y tituloB .

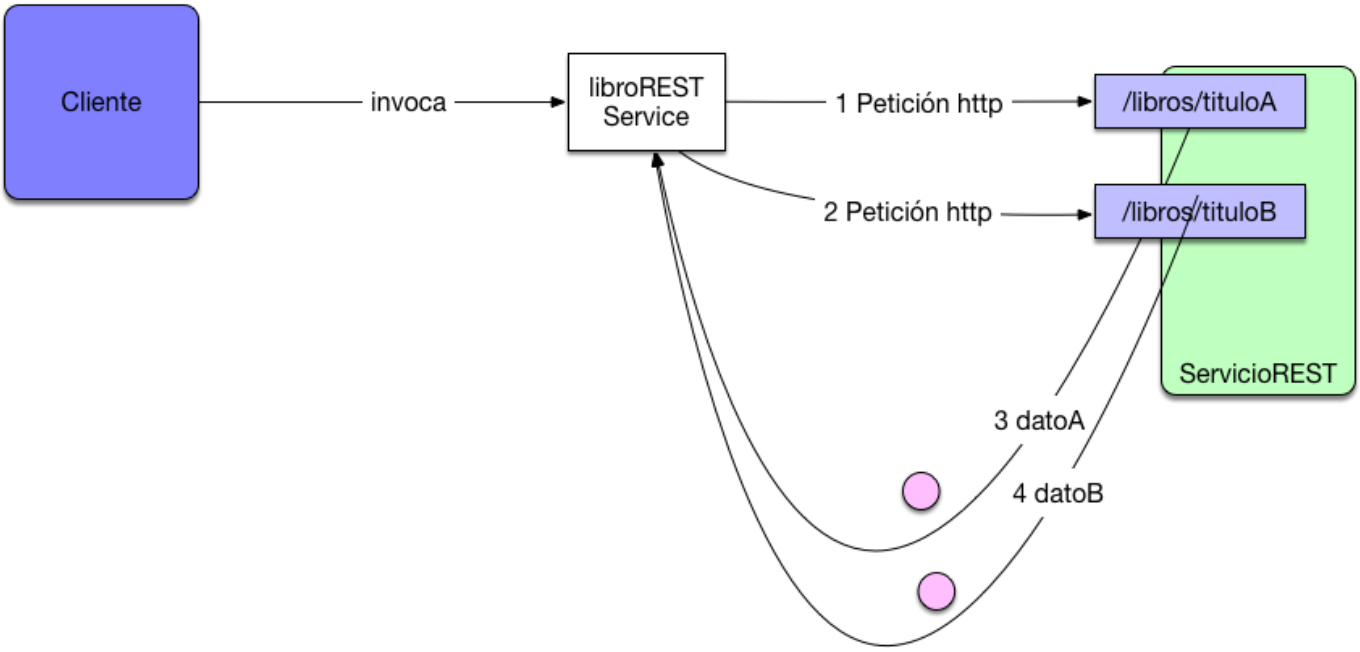


En estas situaciones no nos queda mas remedio que realizar varias peticiones HTTP y combinar los resultados. Vamos a ver un ejemplo sencillo utilizando clases de TypeScript y TypeScript Promise All para combinar los resultados en el mundo de Angular.

```
import { Injectable } from '@angular/core';
import {HttpClient} from "@angular/common/http";
import { Libro } from './negocio/libro';
@Injectable({
  providedIn: 'root'
})
export class LibroRESTService {
```

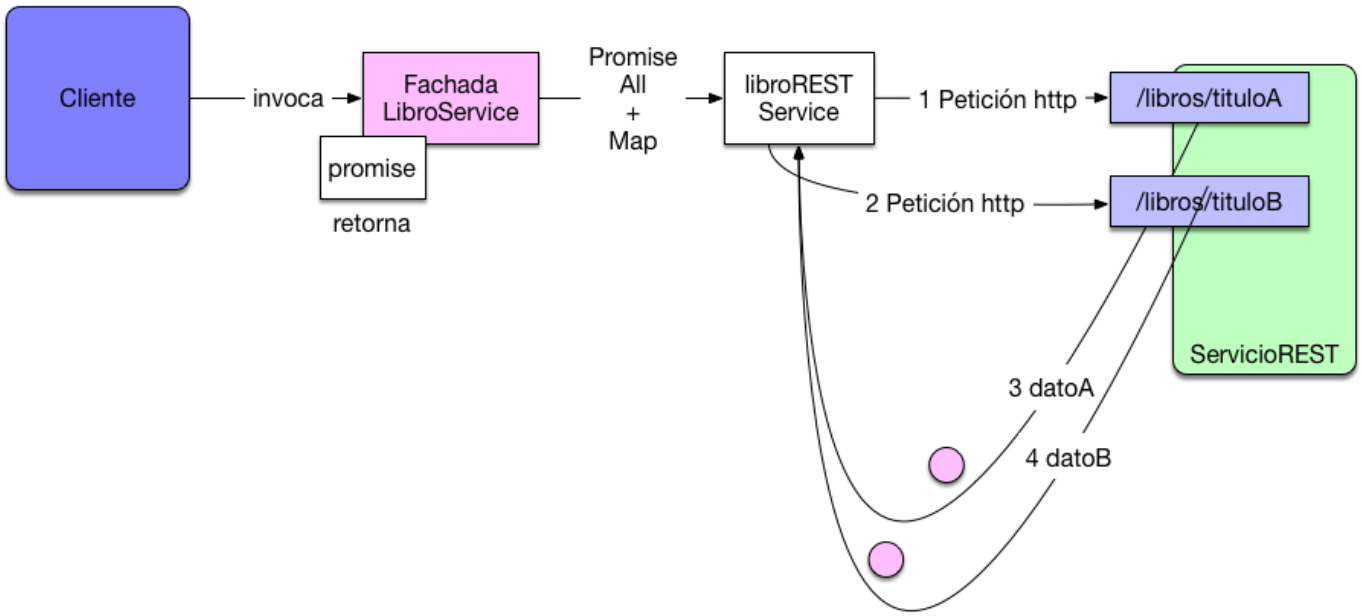
```
  constructor(private miservicio:HttpClient) { }

  findAll():Promise<Libro[]>{
    return this.miservicio
    .get<Libro[]>("http://localhost:3000/libros").toPromise();
  }

  findOne(titulo:string):Promise<Libro> {

    return
this.miservicio.get<Libro>(`http://localhost:3000/libros/${titulo}`).t
oPromise();
  }

}
```

Aquí lo que acabamos de definir es una sencilla clase que dispone de varios métodos uno es findAll() que nos busca todos los libros y nos devuelve una promesa. Por otro lado tenemos un método findOne que lo que hace es nos devuelve una promesa para una url determinada (con un titulo concreto) .El primer método no tiene ningún misterio ya que simplemente se le invoque y procesamos el resultado.Sin embargo el segundo método tiene mas historia ya que le podemos necesitar llamar varias veces de forma asíncrona y agrupar los resultados.



TypeScript Promise All y fachadas

¿Como podemos agrupar todas esas peticiones asíncronas que realizamos para que parezcan una sola?. Es aquí donde TypeScript Promise All nos puede ayudar. Vamos a diseñar una clase de Fachada que se encargue de abordarlo.



Esta clase contendrá la funcionalidad que fusiona un conjunto de promesas (llamadas asíncronas como si fuera una sola)

```
import { Injectable } from '@angular/core';
import { LibroRESTService } from './libro-rest.service';
import { Libro } from './negocio/libro';

@Injectable({
  providedIn: 'root'
})
export class FachadaLibrosService {

  constructor(private libroREST:LibroRESTService) {
   }
   findLibrosPorTitulos(...titulos):Promise<Libro[]> {
```

```
    let promesas=titulos.map((titulo)=>{

          return this.libroREST.findOne(titulo);

    })

   return Promise.all(promesas);
  }
}
```

En este caso la clase se encarga de por cada título realizar una llamada asíncrona y convertir todo en un array de promesas que es retornado para que nuestro componente de angular lo renderice.

```
import { Component, OnInit } from '@angular/core';
import { LibroRestPromesaService } from '../libro-rest-
promesa.service';
import { Libro } from '../negocio/libro';
import { LibroRESTService } from '../libro-rest.service';
import { FachadaLibrosService } from '../fachada-libros.service';

@Component({
  selector: 'app-hola017',
  templateUrl: './hola017.component.html',
  styleUrls: ['./hola017.component.css']
})
export class Hola017Component implements OnInit {

  listaLibros:Libro[]=[];
```

```
  constructor(private servicio:FachadaLibrosService) {


   servicio.findLibrosPorTitulos("titulo1","titulo2").then((datos)=> {

      let lista:Libro[]=[];
      datos.forEach((e)=>lista.push(e));
      this.listaLibros=lista;
})


  }

  ngOnInit() {
  }

}
```

El componente nos mostrará el resultado en el interface de usuario con los dos títulos solicitados apoyandose en el array que devuelve el método promiseAll como resultado:

```
titulo1 autor1 100 misterio
titulo2 autor2 100 romantica
```

La programación asíncrona siempre aborda situaciones complejas y a veces es difícil saber como utilizar esta de forma correcta.

Otros artículos relacionados

1. JavaScript Promise y la programación asíncrona
2. JavaScript Streams vs Promises
3. El concepto de JavaScript Function Composition
4. TypeScript