Mi problema radica en que tengo peticiones a base de datos complejas y quiero que mientras el usuario esta en otra ruta se vayan cargando esos datos como lo podria hacer? Quiero usar resolver y forkJoin.  
  
Para cargar datos complejos en segundo plano mientras el usuario navega por tu aplicación, puedes usar servicios Angular y observables. Aquí tienes un enfoque para lograrlo:

1. \*\*Crear un Servicio de Datos\*\*: Este servicio se encargará de hacer las peticiones a la base de datos y almacenar los datos.

2. \*\*Usar `forkJoin` para Peticiones Simultáneas\*\*: Utiliza `forkJoin` de RxJS para realizar múltiples peticiones a la vez.

3. \*\*Pre-Cargar los Datos\*\*: Usa `Resolve` en Angular Router para pre-cargar los datos antes de que el usuario acceda a la ruta que los necesita.

### Paso 1: Crear el Servicio de Datos

Primero, crea un servicio para manejar las peticiones y almacenamiento de datos.

#### `data.service.ts`

```typescript

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable, BehaviorSubject, forkJoin } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class DataService {

private data1Subject = new BehaviorSubject<any>(null);

private data2Subject = new BehaviorSubject<any>(null);

data1$ = this.data1Subject.asObservable();

data2$ = this.data2Subject.asObservable();

constructor(private http: HttpClient) {}

fetchData(): void {

const request1 = this.http.get('/api/data1');

const request2 = this.http.get('/api/data2');

forkJoin([request1, request2]).subscribe(

([data1, data2]) => {

this.data1Subject.next(data1);

this.data2Subject.next(data2);

},

error => {

console.error('Error fetching data', error);

}

);

}

}

```

### Paso 2: Crear un Resolver

Un resolver en Angular puede pre-cargar los datos antes de activar la ruta.

#### `data.resolver.ts`

```typescript

import { Injectable } from '@angular/core';

import { Resolve } from '@angular/router';

import { DataService } from './data.service';

@Injectable({

providedIn: 'root'

})

export class DataResolver implements Resolve<void> {

constructor(private dataService: DataService) {}

resolve(): void {

this.dataService.fetchData();

}

}

```

### Paso 3: Configurar las Rutas

Configura las rutas para usar el resolver.

#### `app-routing.module.ts`

```typescript

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { LayoutComponent } from './layout/layout.component';

import { ChildComponent } from './child/child.component';

import { DataResolver } from './data.resolver';

const routes: Routes = [

{

path: '',

component: LayoutComponent,

children: [

{

path: 'child',

component: ChildComponent,

resolve: {

data: DataResolver

}

}

]

}

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

```

### Paso 4: Usar los Datos en el Componente

Finalmente, usa los datos cargados en tus componentes.

#### `child.component.ts`

```typescript

import { Component, OnInit } from '@angular/core';

import { DataService } from './data.service';

@Component({

selector: 'app-child',

templateUrl: './child.component.html',

styleUrls: ['./child.component.css']

})

export class ChildComponent implements OnInit {

data1: any;

data2: any;

constructor(private dataService: DataService) { }

ngOnInit(): void {

this.dataService.data1$.subscribe(data => {

this.data1 = data;

});

this.dataService.data2$.subscribe(data => {

this.data2 = data;

});

}

}

```

### Paso 5: Iniciar la Carga de Datos

Para iniciar la carga de datos cuando el usuario esté en otras rutas, puedes llamar al método `fetchData` en el constructor del componente de layout:

#### `layout.component.ts`

```typescript

import { Component, OnInit } from '@angular/core';

import { DataService } from './data.service';

@Component({

selector: 'app-layout',

templateUrl: './layout.component.html',

styleUrls: ['./layout.component.css']

})

export class LayoutComponent implements OnInit {

constructor(private dataService: DataService) { }

ngOnInit(): void {

// Iniciar la carga de datos

this.dataService.fetchData();

}

}

```

### Conclusión

Con esta configuración, tus datos se cargarán en segundo plano mientras el usuario navega por la aplicación. Los resolvers aseguran que los datos necesarios estén disponibles antes de que el usuario acceda a las rutas que los requieren, mejorando la experiencia del usuario y optimizando la carga de datos.