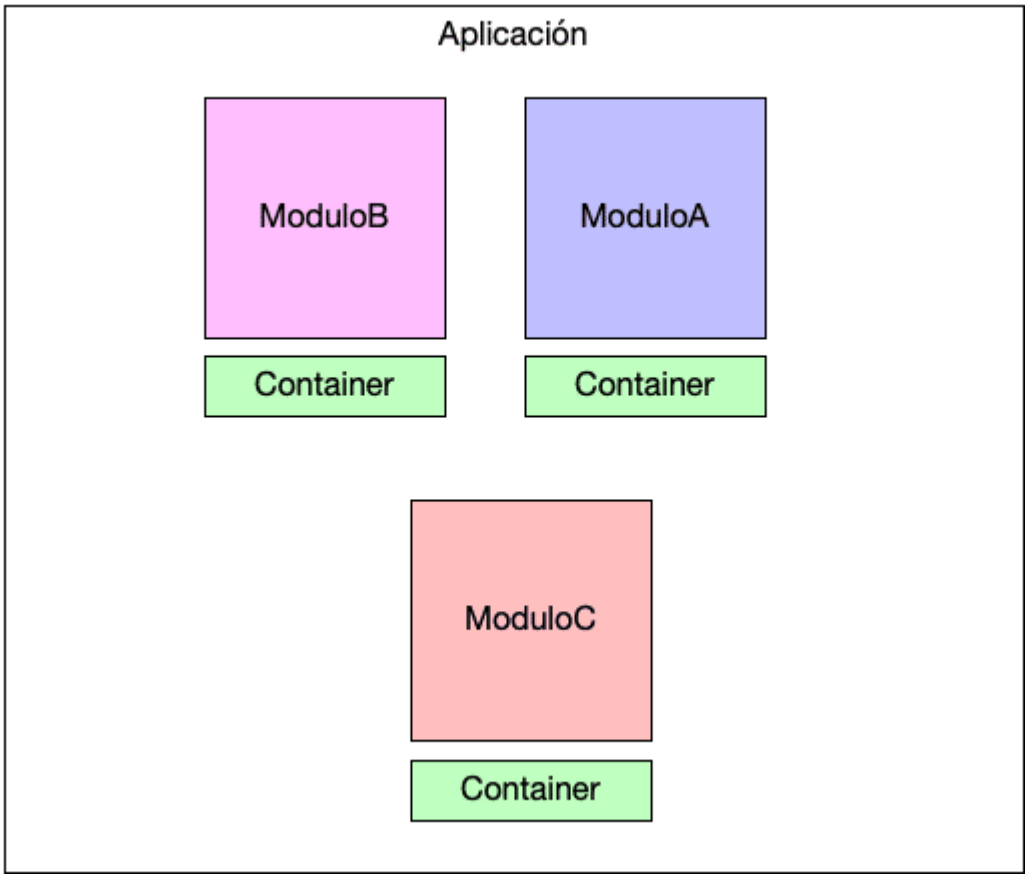
¿MicroServicios vs Monolitos? . Hoy por hoy muchas empresas están empezando a desarrollar arquitecturas de MicroServicios o por lo menos a pensar en ellas . Sin embargo la migración a este tipo de arquitecturas requiere un esfuerzo importante y muchas empresas hoy por hoy simplemente lo valoran y quizás decidan seguir construyendo monolitos hasta que parte de los tecnologías tengan un mayor nivel de madurez , otras en cambio darán el salto. Uno de los temas que debemos tener en cuenta a futuro es que si construimos un monolito , ¿Podemos luego migrarle de forma sencilla o relativamente rápida a una Arquitectura de MicroServicios? .Vamos a verlo :

[ihc-hide-content ihc_mb_type=”show” ihc_mb_who=”4″ ihc_mb_template=”1″]

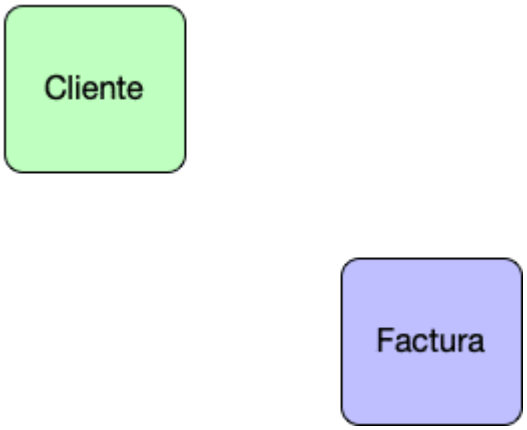
¿MicroAplicaciones o MicroServicios?

Normalmente cuando alguien despliega una aplicación con Spring Boot automáticamente asume que lo que esta desplegando es un MicroServicio. La realidad es que eso no es así nosotros lo que estamos desplegando es una aplicación sobre Spring Boot . La aplicación puede ser grande mediana o pequeña todo se admite . Eso sí la acabaremos desplegando en un container tipo Docker o similar . Por lo tanto nos da la sensación de que es un MicroServicio . Si tuviéramos un MicroServicio nos encontraríamos con una aplicación que se despliega de forma modular en múltiples contenedores y cada contenedor contiene un módulo de esta.



Monolitos y Spring Boot

La clave la tenemos a nivel de la gestión de módulos , vamos a construir una aplicación de Spring Boot que actue como si fuera un monolito . Para ello supongamos que tenemos dos clases totalmente independientes Factura y Cliente que van a pertenecer a dos módulos independientes uno gestionara el ingreso de Clientes y el otro gestionará el pago de las Facturas. Para crear el ejemplo de hola mundo . Necesitaremos al menos 2 clases de negocio:



Veamos su código :

```
package com.arquitecturajava.miaplicacion;

public class Factura {

  private int numero;
  private String concepto;
  private double importe;
  public int getNumero() {
    return numero;
  }

  public void setNumero(int numero) {
    this.numero = numero;
  }


  public String getConcepto() {
    return concepto;
```

```
  }


  public void setConcepto(String concepto) {
    this.concepto = concepto;
  }


  public double getImporte() {
    return importe;
  }


  public void setImporte(double importe) {
    this.importe = importe;
  }


  public Factura(int numero, String concepto, double importe) {
    super();
    this.numero = numero;
    this.concepto = concepto;
    this.importe = importe;
  }
}
```

```
package com.arquitecturajava.miaplicacion;

public class Cliente {

  private String nombre;
  private String empresa;
```

```
  public String getNombre() {
    return nombre;
  }
  public void setNombre(String nombre) {
    this.nombre = nombre;
  }
  public String getEmpresa() {
    return empresa;
  }
  public void setEmpresa(String empresa) {
    this.empresa = empresa;
  }
  public Cliente(String nombre, String empresa) {
    super();
    this.nombre = nombre;
    this.empresa = empresa;
  }
}
```

Una vez. tenemos construidas estas clases únicamente tenemos que generar los servicios y los controllers para que realicen una funcionalidad básica:

```
package com.arquitecturajava.miaplicacion;

import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Service;

@Service
public class ClienteService {
```

```
  public List<Cliente> buscarTodos() {
    Cliente c1= new Cliente("pepe","empresaA");
    Cliente c2= new Cliente("ana","empresaB");
    List<Cliente> lista= Arrays.asList(c1,c2);
    return lista;
  }
}
```

```
package com.arquitecturajava.miaplicacion;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
@RequestMapping("/")
public class ClienteController {

  @Autowired
  ClienteService servicio;
  @RequestMapping("listaclientes")
  public String listaClientes(Model modelo) {
    modelo.addAttribute("clientes", servicio.buscarTodos());
    return "listaClientes";
  }
}
```

Ya tenemos los Clientes , nos quedan las Facturas :

```
package com.arquitecturajava.miaplicacion;

import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
@RequestMapping("/")
public class FacturaController {

  @Autowired
  FacturaService servicio;
  @RequestMapping("listafacturas")
  public String listaFactura(Model modelo) {
    modelo.addAttribute("facturas", servicio.buscarTodos());
    return "listafacturas";
  }
}
```

```
package com.arquitecturajava.miaplicacion;

import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Service;

@Service
public class FacturaService {

  public List<Factura> buscarTodos() {
    Factura f1= new Factura(1,"ordenador",200);
    Factura f2= new Factura(2,"tablet",300);
    List<Factura> lista= Arrays.asList(f1,f2);
    return lista;
  }
```

```
}
```

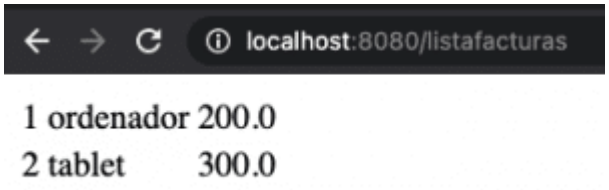
ThymeLeaf y Plantillas

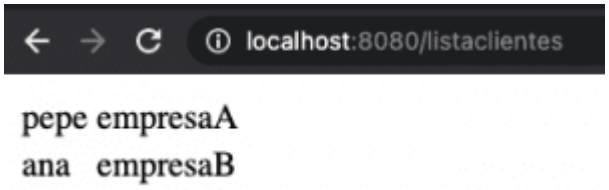
Para que todo nos funcione correctamente tenemos evidentemente que añadir también las plantillas de Thymeleaf:

```
<html xmlns:th="http://www.thymeleaf.org">
<body>
  <table>
    <tr th:each="cliente: ${clientes}">
      <td th:text="${cliente.nombre}" />
      <td th:text="${cliente.empresa}" />
    </tr>
  </table>
</body>
</html>
```

```
<html xmlns:th="http://www.thymeleaf.org">
<body>
  <table>
    <tr th:each="factura : ${facturas}">
      <td th:text="${factura.numero}" />
      <td th:text="${factura.concepto}" />
      <td th:text="${factura.importe}" />
    </tr>
  </table>
</body>
</html>
```

Si ejecutamos cada una de las páginas veremos el resultado :

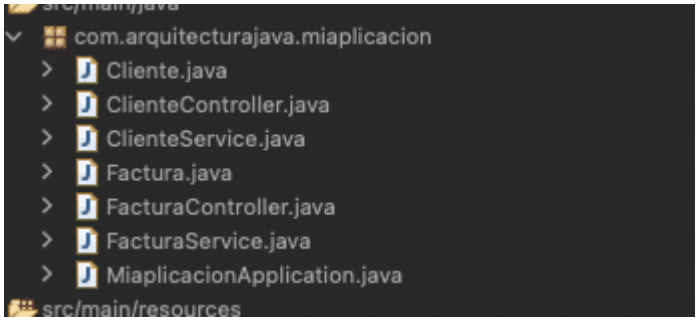




Tenemos todo correctamente desplegado y la aplicación funciona sobre Spring Boot , pero sigue siendo un monolito.

MicroServicios vs Monolitos

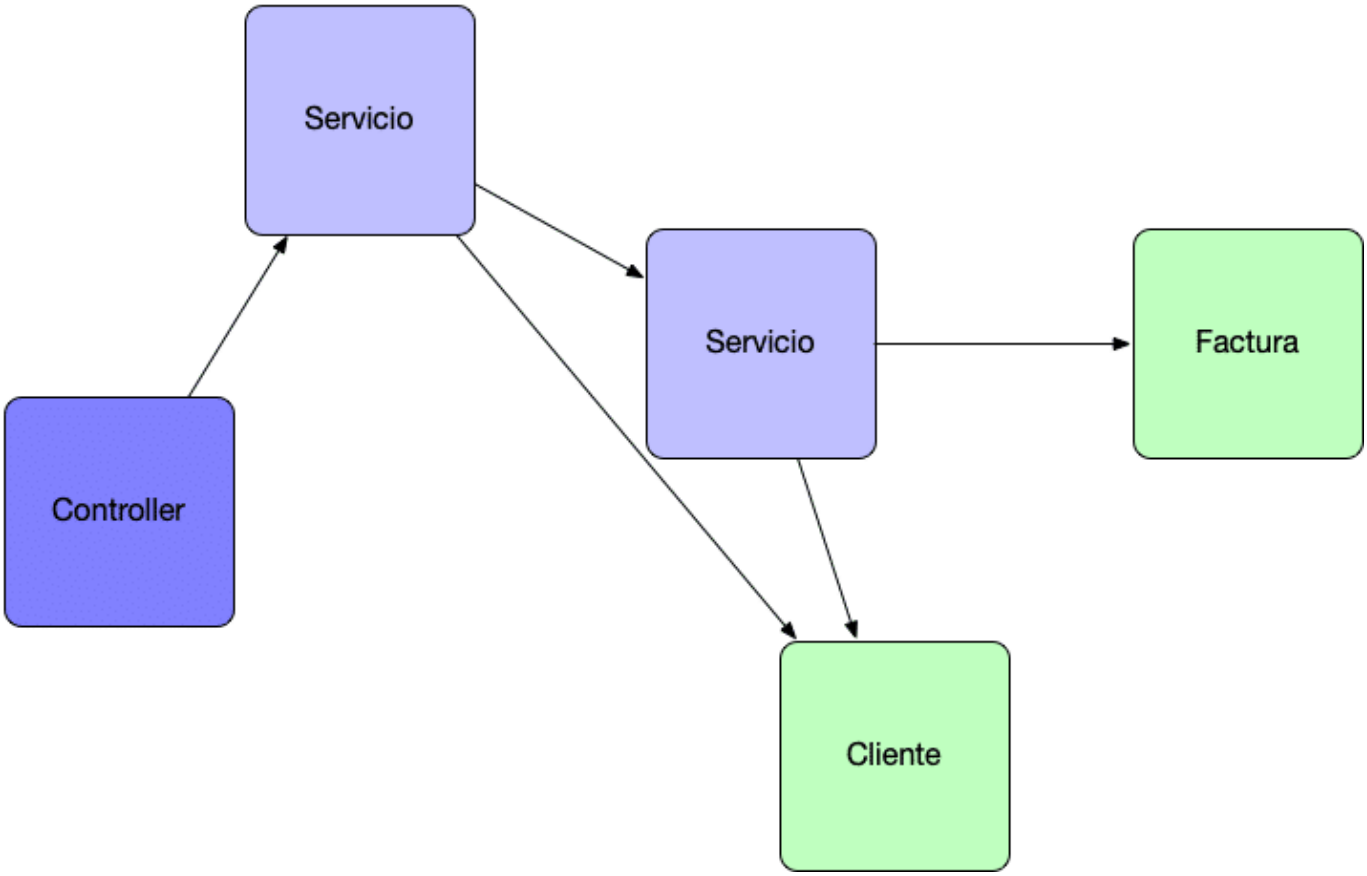
¿Cómo podemos enfocarlo hacia MicroServicios? . Realmente el esfuerzo de construir una arquitectura de MicroServicios es grande . Pero podemos mantener el monolito y realizar unos cambios de bajo impacto que nos acerquen a una futura Arquitectura de MicroServicios y facilite la migración. En estos momentos tenemos una estructura de packages de este estilo:



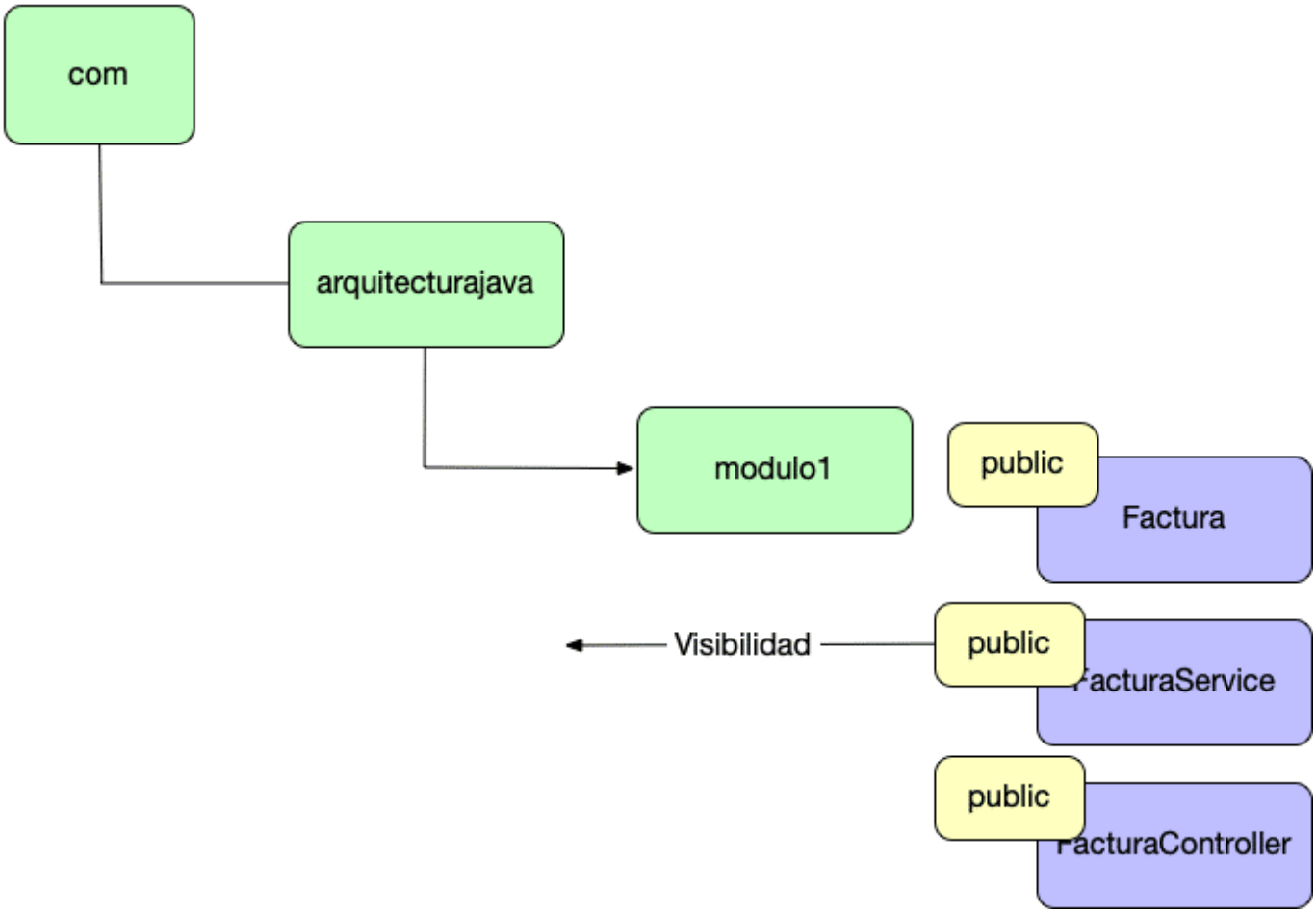
El primer cambio aconsejable sería modularizar la estructura de packages de tal forma que cada modulo tenga sus propias clases . Un posible enfoque es :

Packages Visibilidad y Modularización

De esta forma estamos más organizados , cada grupo de clases esta en su package . Sin embargo cualquier servicio puede acceder a cualquier otro o cualquier controlador a cualquier servicio. Seguimos estando en una arquitectura de Monolito.

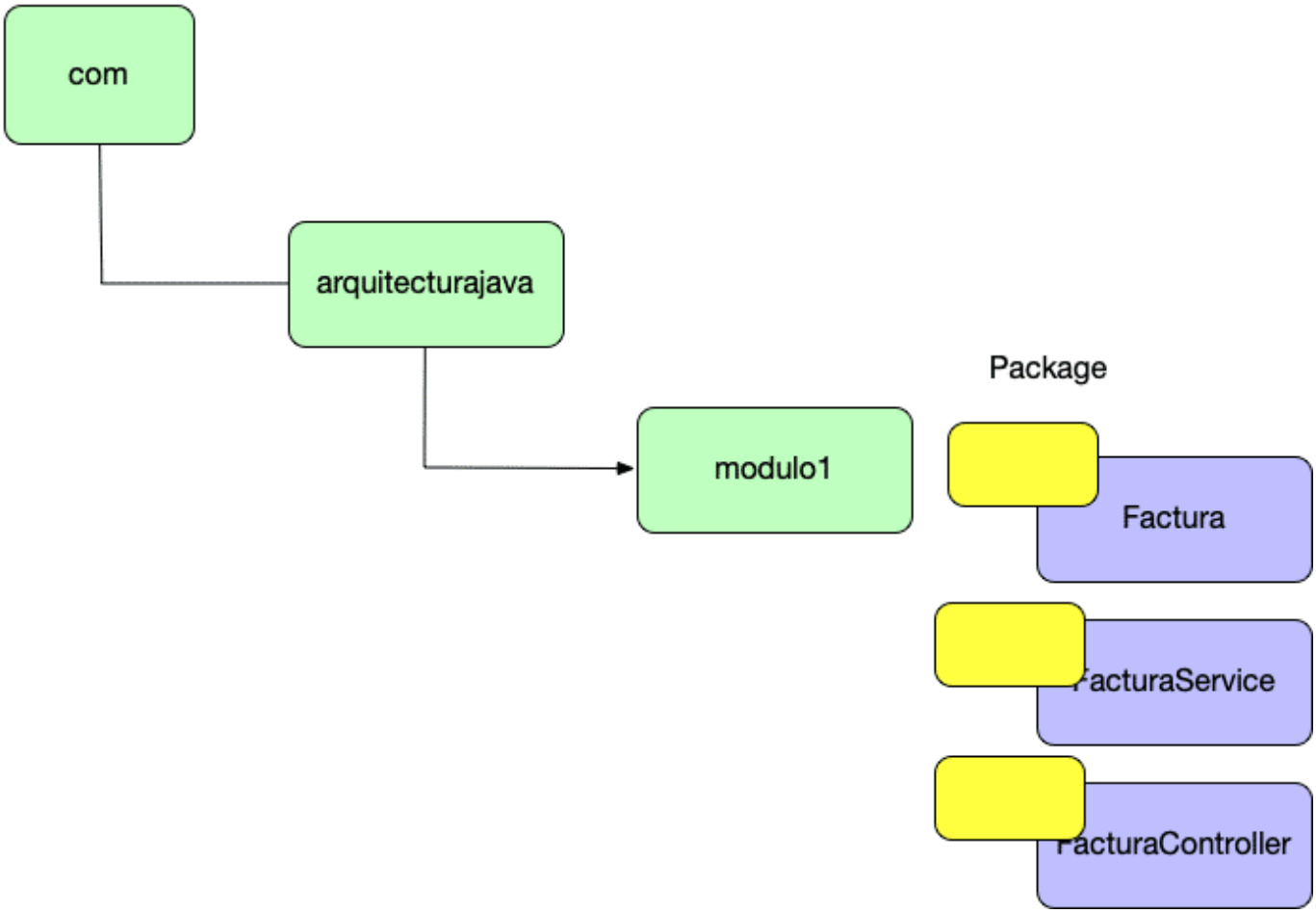


Para lograr cambios importantes podemos cambiar la visibilidad de nuestras clases es decir ahora mismo nuestras clases son todas públicas . Podemos cambiar su ámbito y dejarlas como visibilidad de package ya que ahora mismo estan en public.

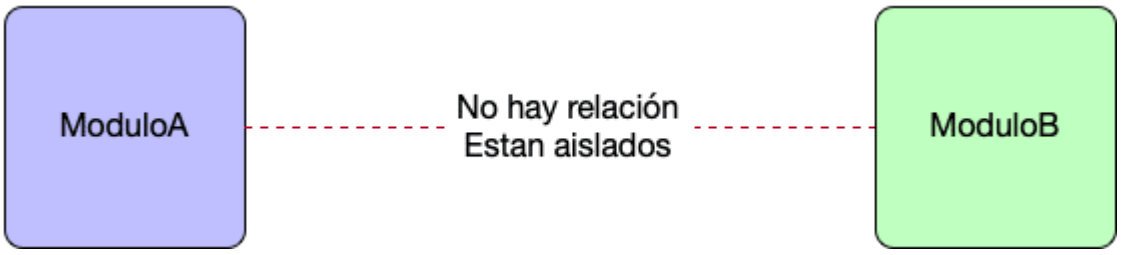


MicroServicios vs Monolitos y visibilidad

Si cambiamos la visibilidad habremos aumentado el nivel de aislamiento entre diferentes módulos.



Siendo cada uno de ellos totalmente aislado para los otros :



Eso sí , si construimos pruebas unitarias deberemos ubicarlas con la misma nomenclatura de packages . Comencemos a enfocar nuestras arquitecturas a un futuro ligado a los MicroServicios aunque ahora no los utilicemos

Otros artículos relacionados:

1. MicroServicios
2. Curso Spring Boot y MicroServicios
3. El porqué de los MicroServicios

[/ihc-hide-content]