**Dependency Injection**

* **It allows us to inject dependencies in different components across our applications.**

**Without needing to know , how dependencies are created or what dependencies they need themselves.**

**What problem does it solves**

**Class car{**

**Constructor(){**

**This.engine= new Engine();**

**This.tires= Tires.getInstance();**

**This.doors.app.get(‘doors’);**

**}**

**}**

**We have Created all the required instances inside constructor.**

**Problem**

* **Constructor know how those object are created . For example he knows Engine Constuctor is used. Singleton interface is used.**
* **Such code is hard to maintain and even harder to test.**

**So how we can make it better.**

* **We can moved the dependency creation out of the constructor and extended the constructor function to expect all the needed dependencies.**

**Class Car{**

**Constructor(engine,tires,doors){**

**This.engine=engine;**

**This.tires=tires;**

**This.doors=doors;**

**}**

**}**

* **So If we want to create a car object now all we have to do is to pass all needed dependencies to the constructor.**

**i.e**

**var car=new Car(**

**new Engine(),**

**new Tires(),**

**new Doors()**

**);**

**Advantage**

* **Dependencies are now decoupled from the class which now allows us to pass mock dependencies while writing tests**

**Such kind of way of passing dependencies are called as Dependency Injection and specifically it’s a constructor Injection.**

**We also have**

1. **Setter injection**
2. **Interface injection**

**But the real problem here is Who will assemble this for us????**

**This is where angular DI framework will comes into picture**

**1.Inject : It’s a decorator that attaches the metadata to our class that is then consumed by the DI system afterwards. Basically we are telling DI that the first constructor parameter should be instance of type Engine,Tires and Doors.**

**Class Car{**

**Constructor(engine:Engine,tires:Tires,doors:Doors){**

**}**

**}**

* **So our class has declared dependency and the DI can read that information to instantiate whatever is needed to create object of car.**

**Injector – it exposes API to create instances of dependencies.**

**How does injector will know how to create object**

* **This is where Providers come into play**

**It provides an object**

**{provide:Car,useClass:car}**

* **It maps token to configuration object.**
* **Token can be type or string**
* **We are specifying here we provide an instance of a class via the class Car.**
* **Providers tells the injector which dependencies are used across the application.**
* **Providers is an instruction which tells how the object of certain token is created.**
* **In the above syntax we can map the token engine to the class OtherEngine which means we now asks for an object of type engine and we get an instance of class OtherEngine which can avoid name collision which was a problem in**

**Providing values**

* **We can provide a simple value using useValue**

**Provide:’someValue’ ,useValue:value**

**Provide factories**

**How to inject a service in Angular**

1. **Create a provider either on your @NgModule,@Component or @Directive using a type or string as provider token.**
2. **Inject the service in the component constructor where its needed using configured token**

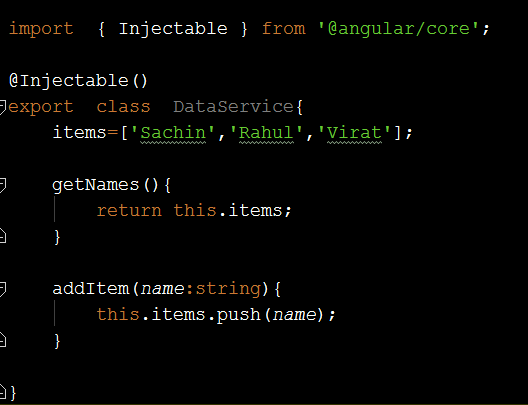
**Show them Demo 1,Demo2**

**Demo 1 :**

1. **Create Log Service , You want to inject.**

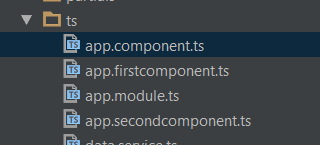
****

1. **Create Data Service**

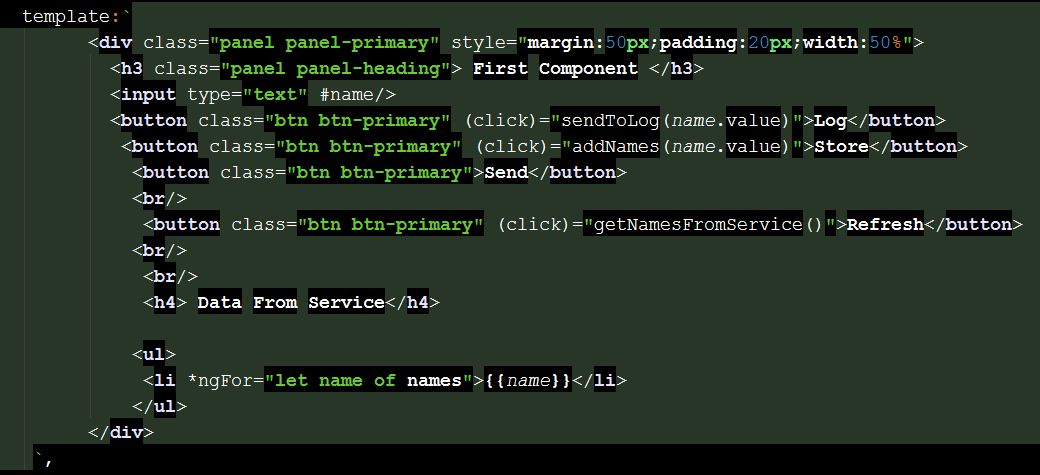
****

1. **Now we want to inject Service , to do this the couple of component we need to create.**

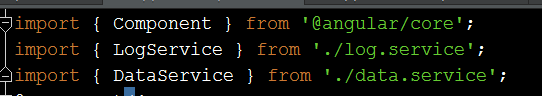
**Create firstComponent and SecondComponent.**

****

1. **Add a Template in Selector of FirstComponent**

****

1. **Import the Service in this component**

****

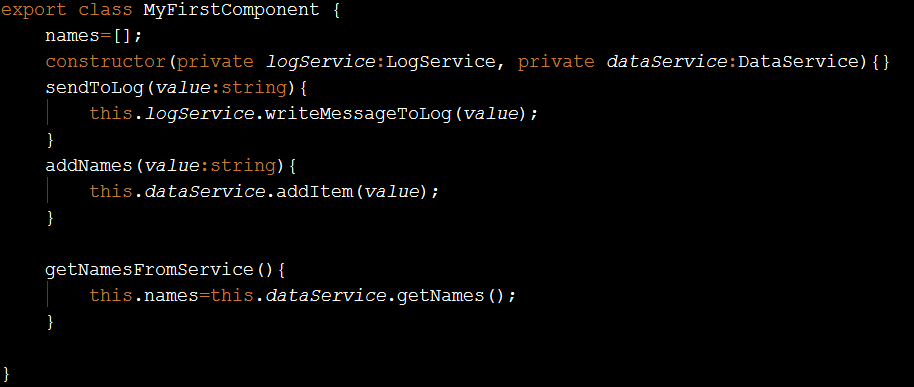
1. **Register this Service with the providers so that it can tell injector how to create instances.**

****

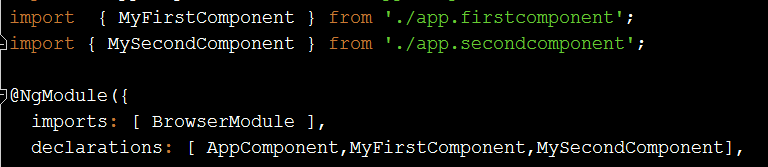
1. **Use Constructor Injection to add the metadata for the dependency which specifies which classes are needed for current execution.**

****

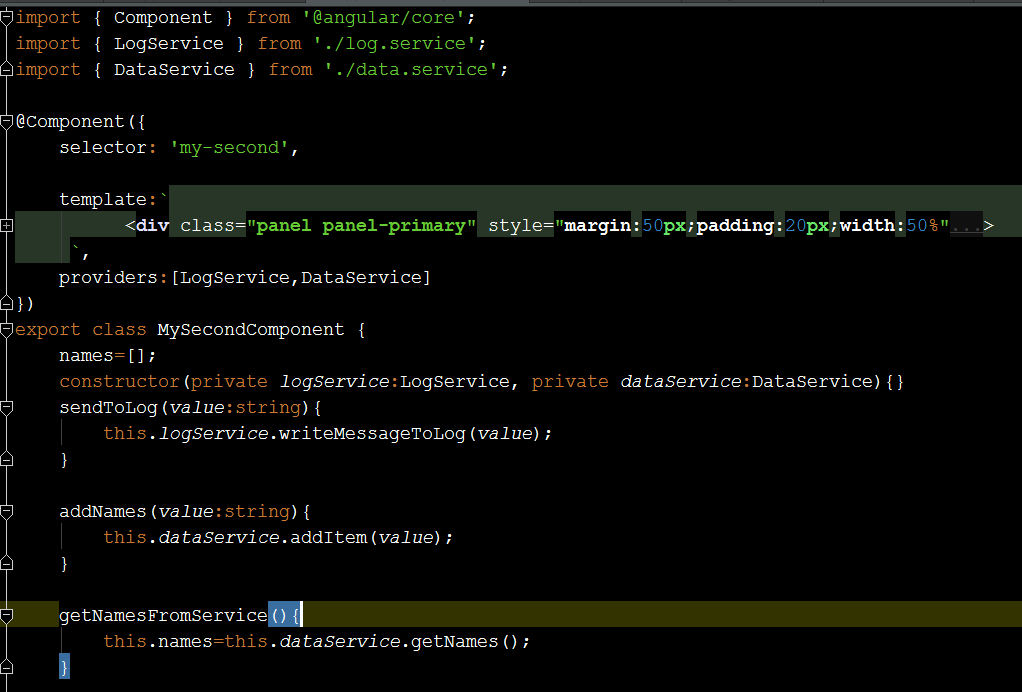
1. **Now use this Services and its method in your MyFirstComponent.**

****

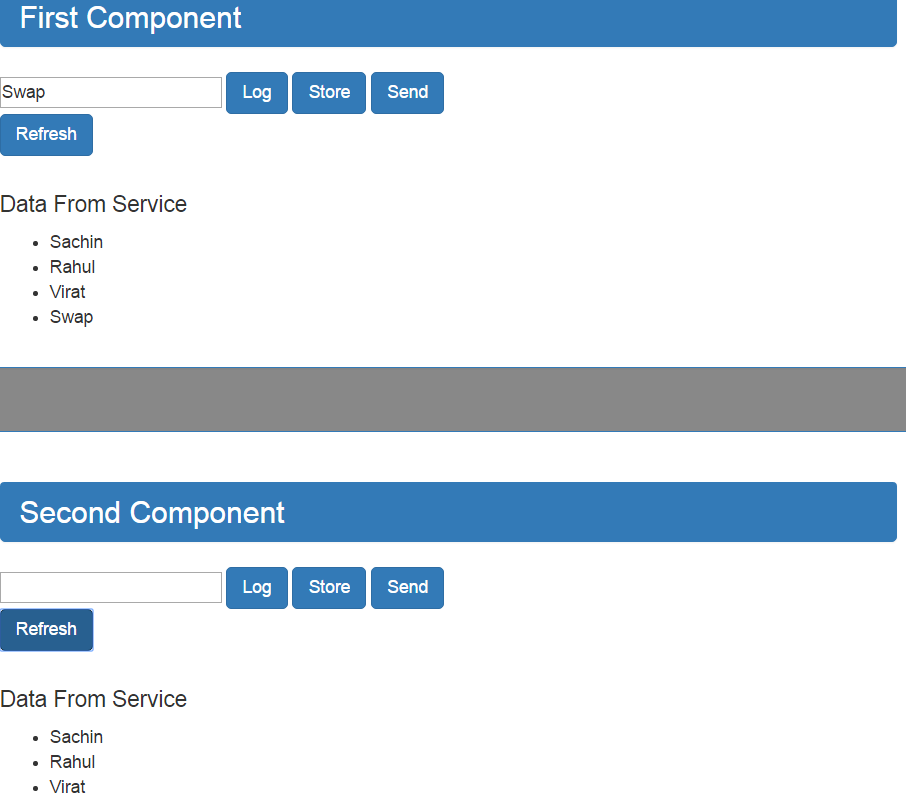
1. **Register this Component at the module level**

****

1. **Repeat same procedure for the SecondComponent.**

****

**Output**

****

**This proves because we have injected at the component level 2 different instances are created hence changes in one instances will not be available to Other instance.**

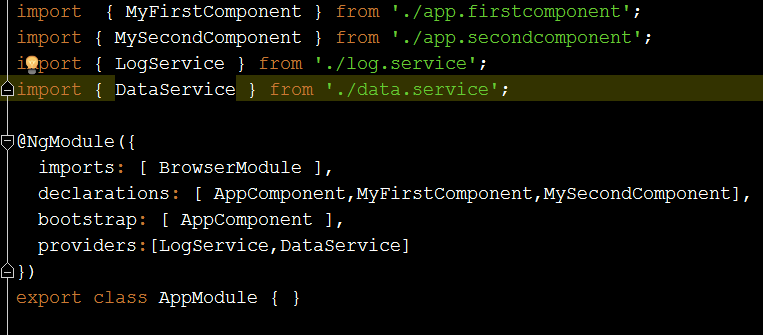
**How to Solve it??**

**Demo 2: Lets Do Module Level Provider Registration**

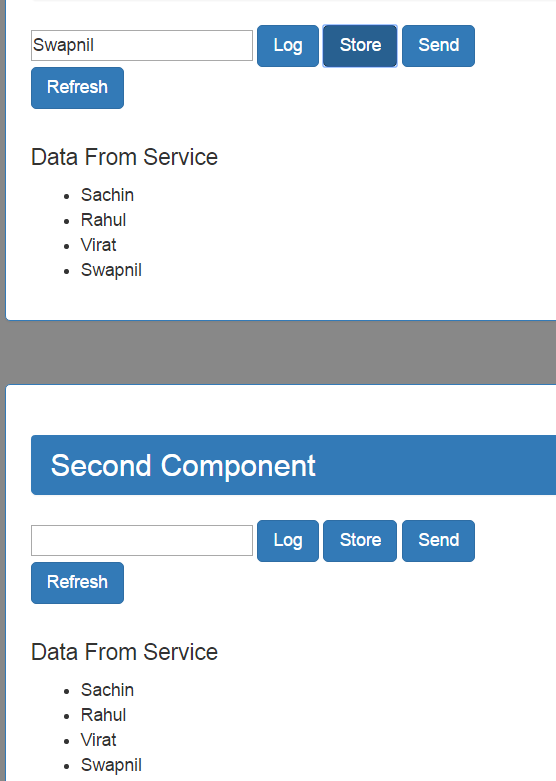
1. **Remove the Providers from individual components that is first and second component**

****

1. **Register this providers at the module level.**

****

1. **See the output**

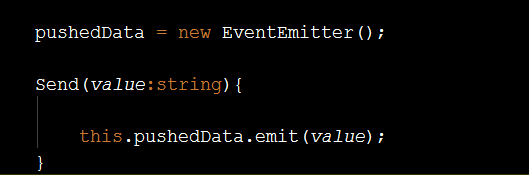
****

**Data entered in the first component is automatically available to second component it means only one instance of the Service object is created.**

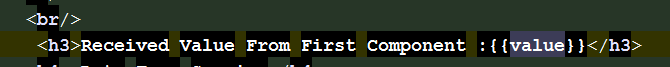
**Now how to add working of Send Function. Here we need communication between 2 Components through Services hence we need to use something called ass EventEmitter which emits the value**

**Demo 3: How to Pass data from one component to another component via Services.**

1. **Add EventEmitter that will emit the values received. Add this in dataService.**

****

1. **Add the HTML in Second Component to display the value**

****

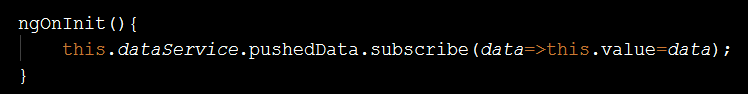
1. **Now Subscribe the value which is emitting from the Service in Second Component for this demo , you need to do some changes**
2. **Add OnInit in Import**

****

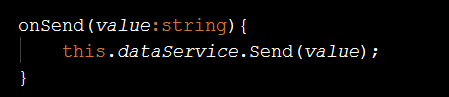
1. **Implement Oninit**

****

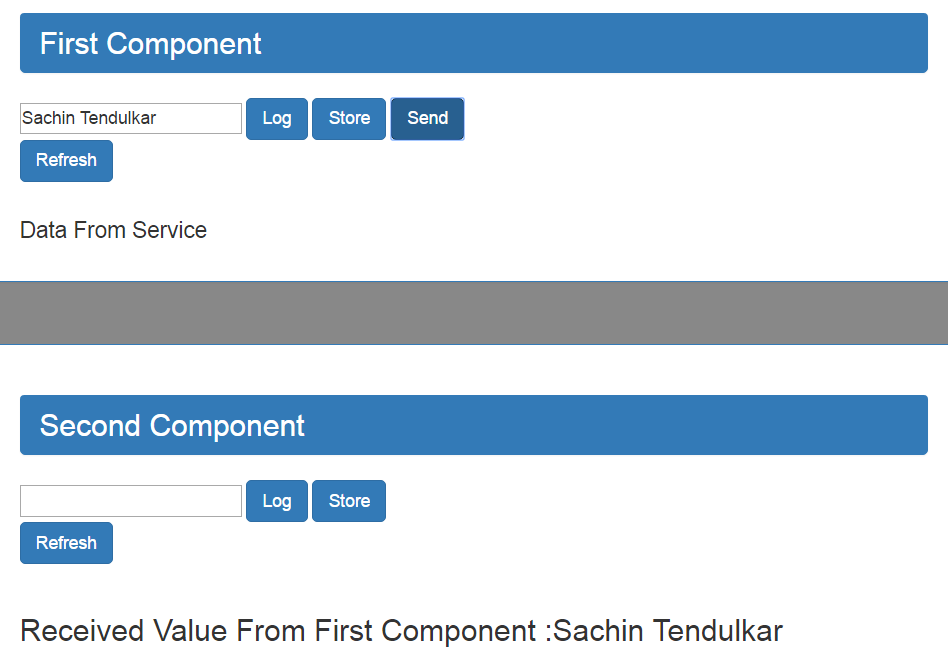
1. **Add ngOnInit function and use Subscribe function**

****

1. **Now Send the value from FirstComponent.**

****

**Output:**

****

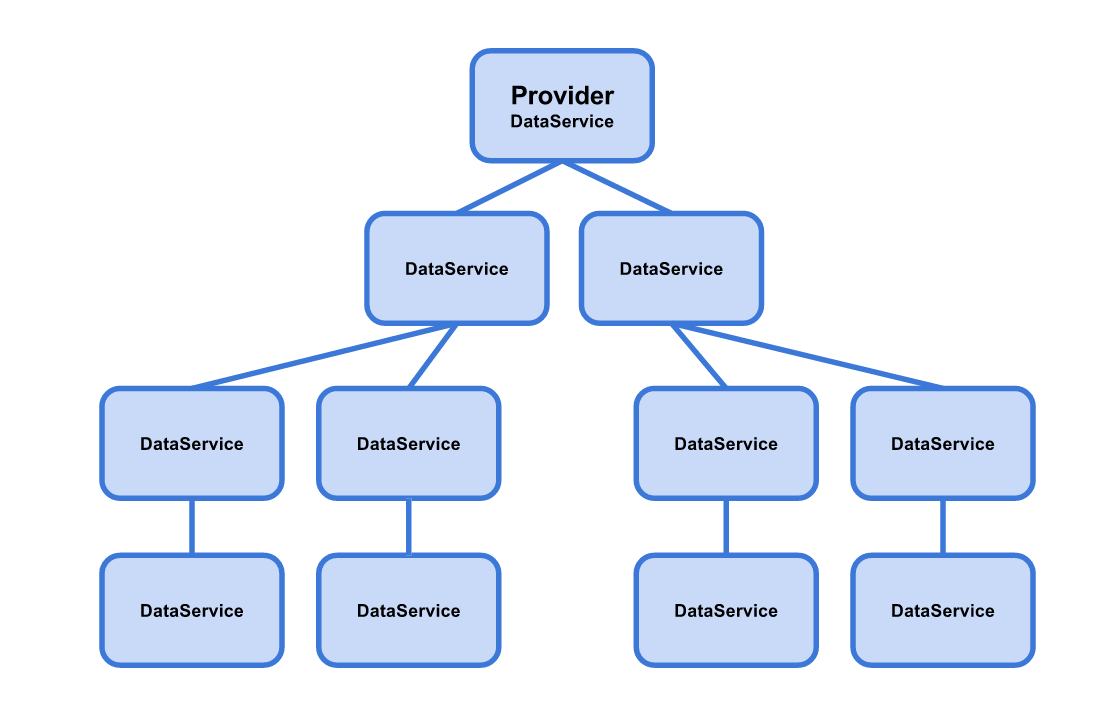
**Hierchial Injection with Class Provider,Value Provider and Factory Provider**

**Problem**

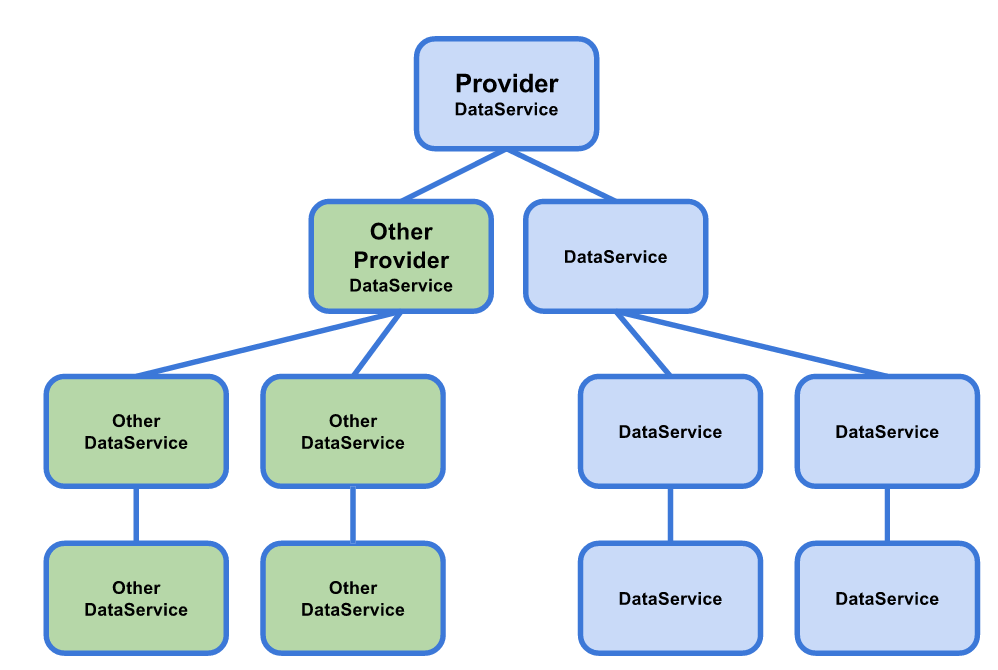
* **Can I bypass a provider to get dependency from another ancestor provider??**

**Understand it??**

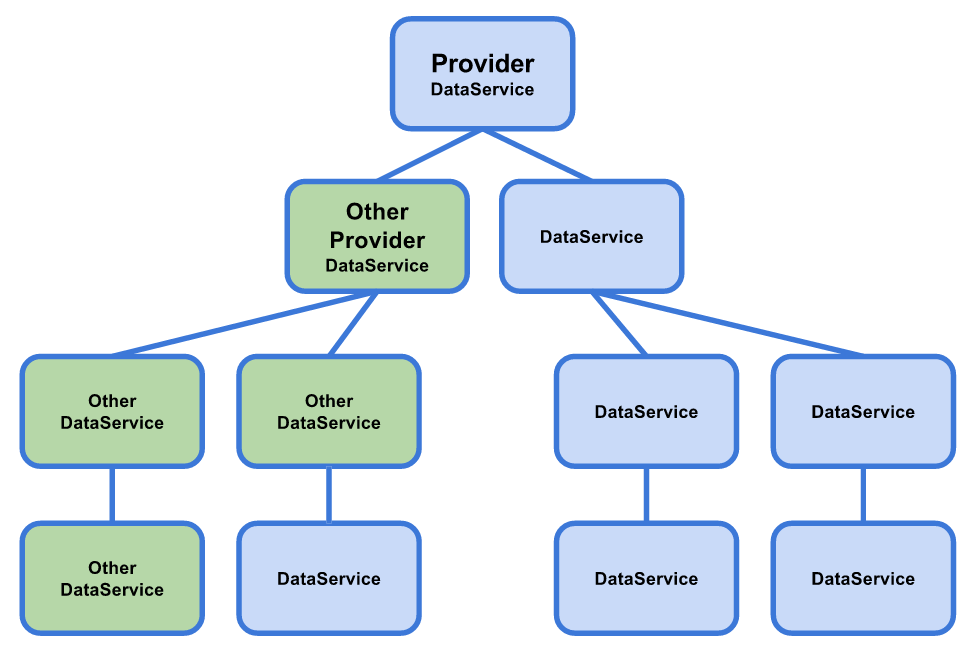
* **Every Component Comes up with his own injector . This helps us to configure what is going to be created when we ask for dependencies on a component level.**
* **If you apply some injector on the component then all of its subcomponent will get exactly same instance because there injector will be looking upwards for a tree provider until they find one.**

****

* **Suppose we add some injector at the some child component then child component will get different instance then one configured in NgModule.**

****

* **So all left part will get their dependency instance from the root provider and than the right one.**
* **Suppose now we want to get the dependency instance of the root provider bypassing the neares provider**

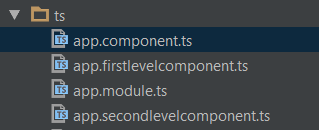
****

**Solution : We have some Providers Strategy that we can use that define how dependencies are created.**

1. **Creating alias token with useExisting**

**Demo 3 :**

1. **Add 2 more component firstlevelcomponent and secondlevelcomponent.**

****

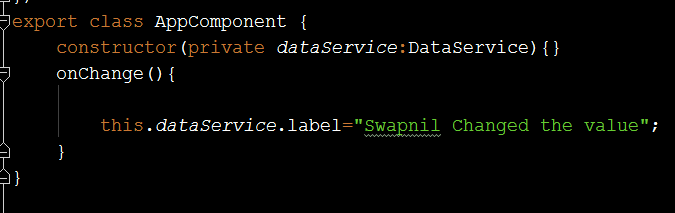
1. **Import dataService**

****

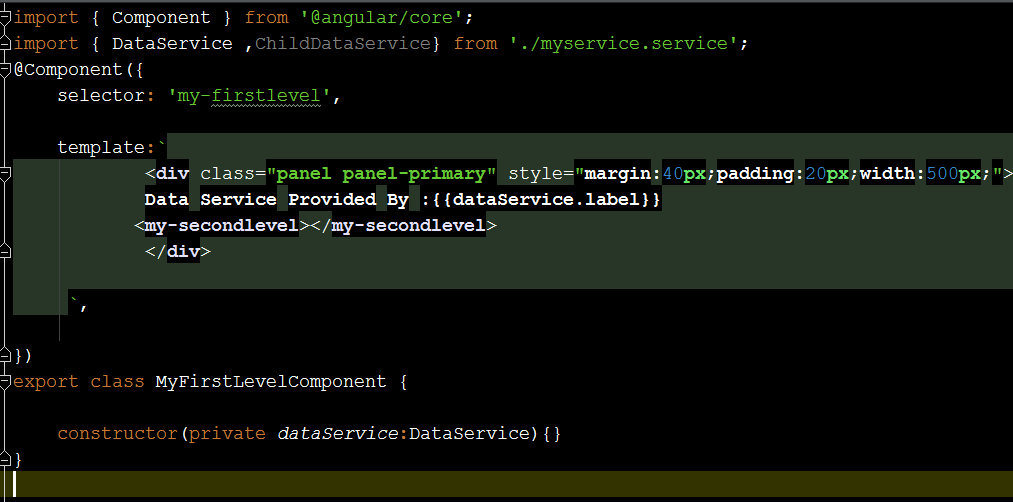
1. **Add template to display the data from service**

****

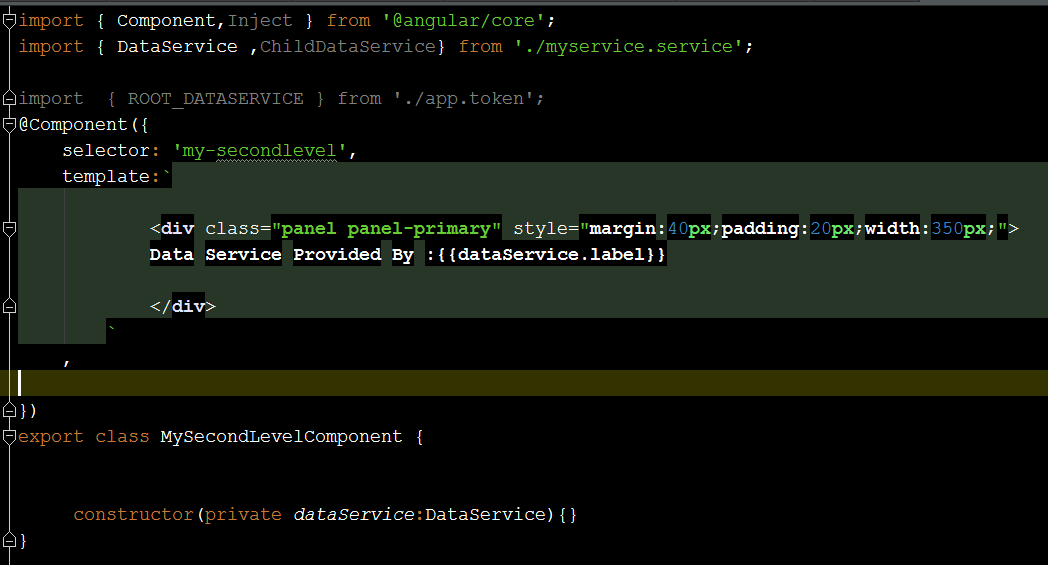
1. **Perform constructor Injection**

****

1. **Create MyFirstComponent**

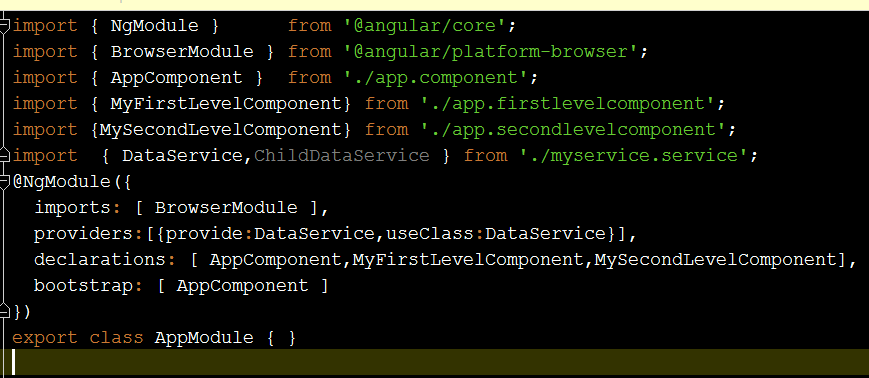
****

1. **Create SecondLevelComponent**

****

**If you check all of these component have same template**

1. **Now Inject the dependency through AppModule File**

****

**Check this providers and see I have injected it at the AppModule level so instance isadded to the root of component**

Provider

DataService

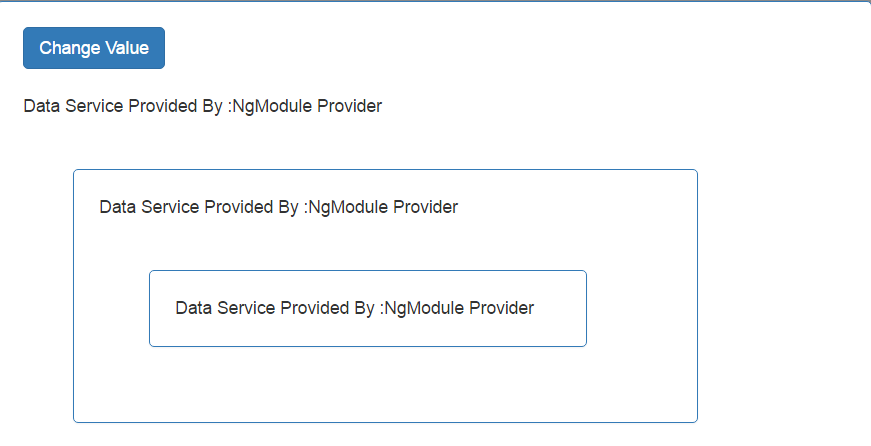
Provider

DataService

Provider

DataService

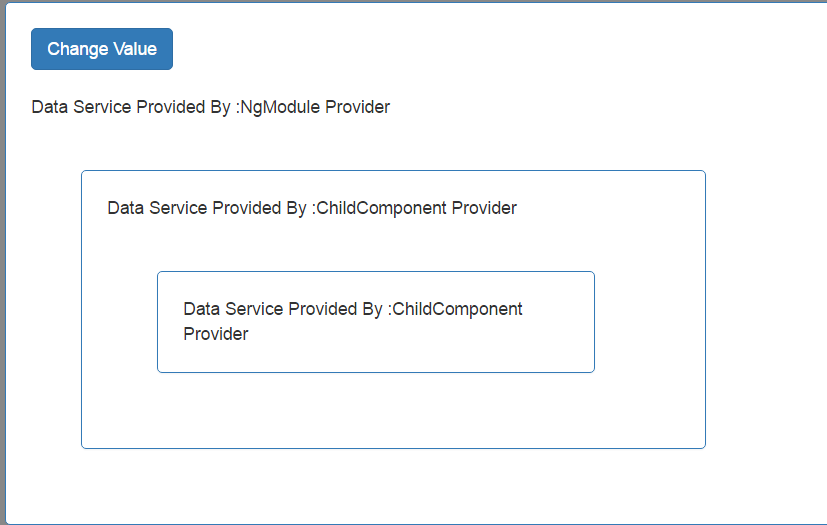
Output :



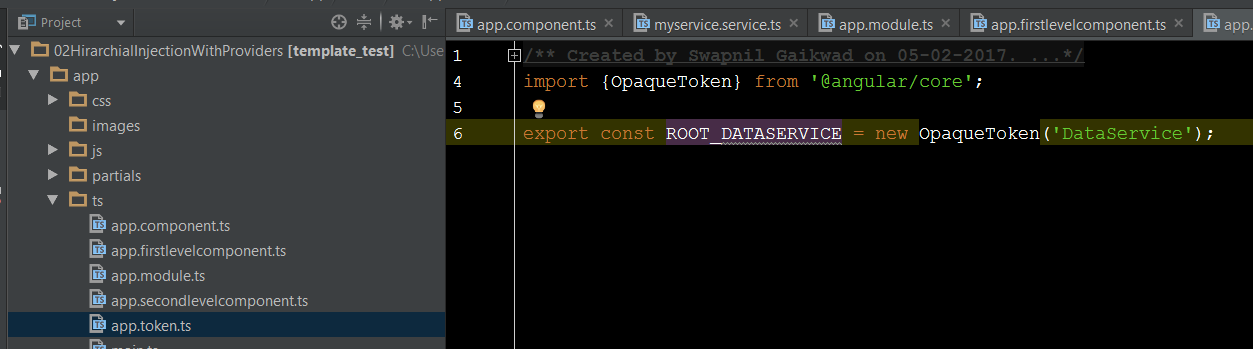
1. Now inject some other provider in FirstLevelComponent. We have injected ChilDataService



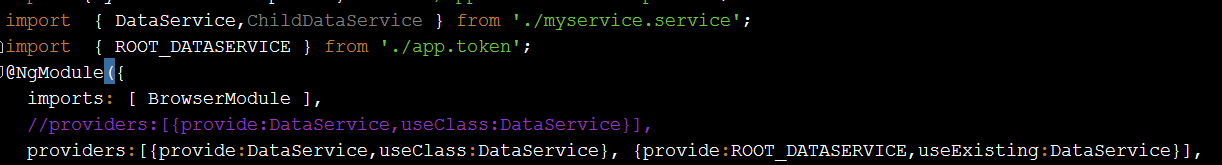
1. Output after injection is : Notice the child component of the first level automatically takes the instances of the ancestor



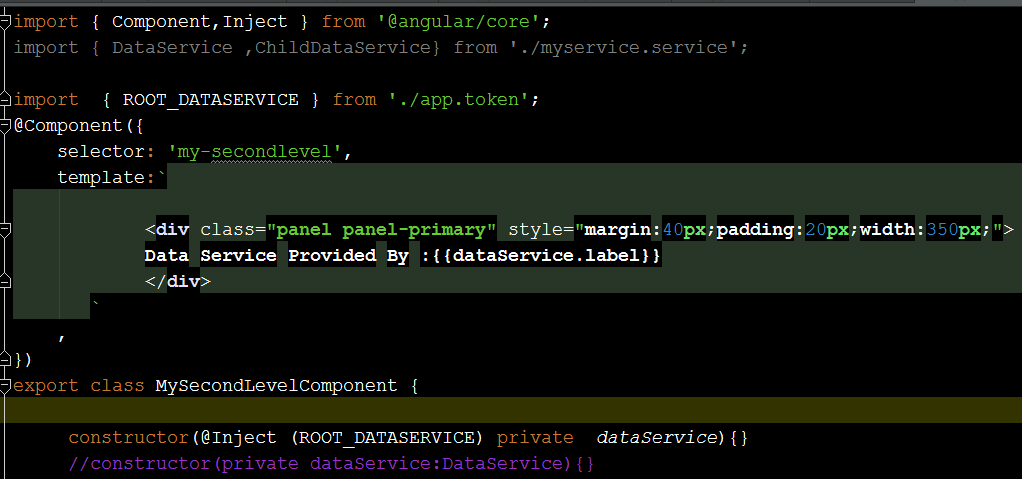
1. Now suppose we want to use RootProvider instance bypassing the nearest ancestor. Then what we need to Inject that same instance of dataService through some token.
2. Add app.token file with the configured Type



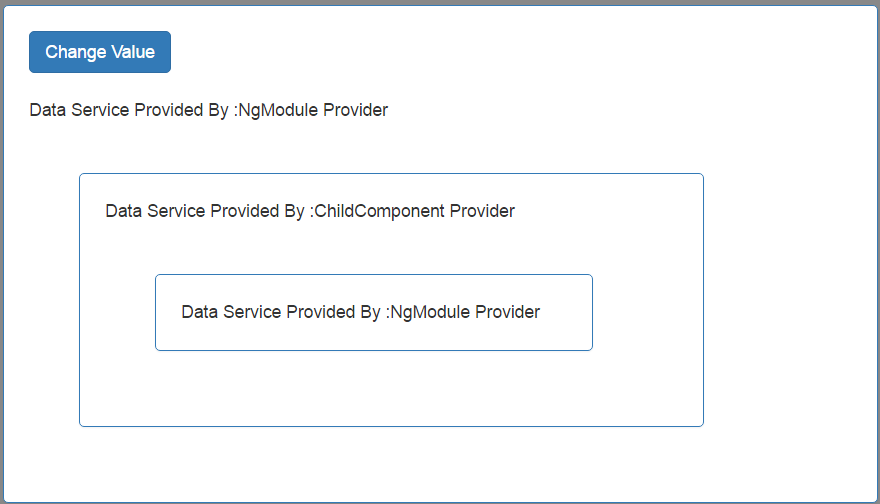
1. Change the app.module and add the ROOT\_DATASERVICE into the provider.



1. Import ROOT\_DATASERVICE into the SecondlevelComponent and Inject it using @Inject . Notice Inject



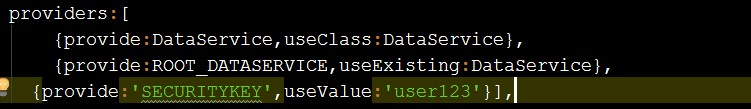
Ourput :



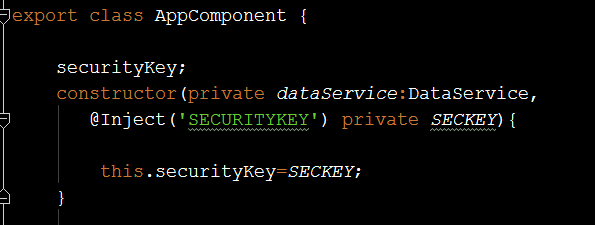
Demo 4:

How to inject Values : Values as Provider

1. Go to AppModule and add a provider for that



1. Inject it using @Inject metadata



1. Print the value

{{securityKey}}