**Section 9: Using services and Dependency injection**

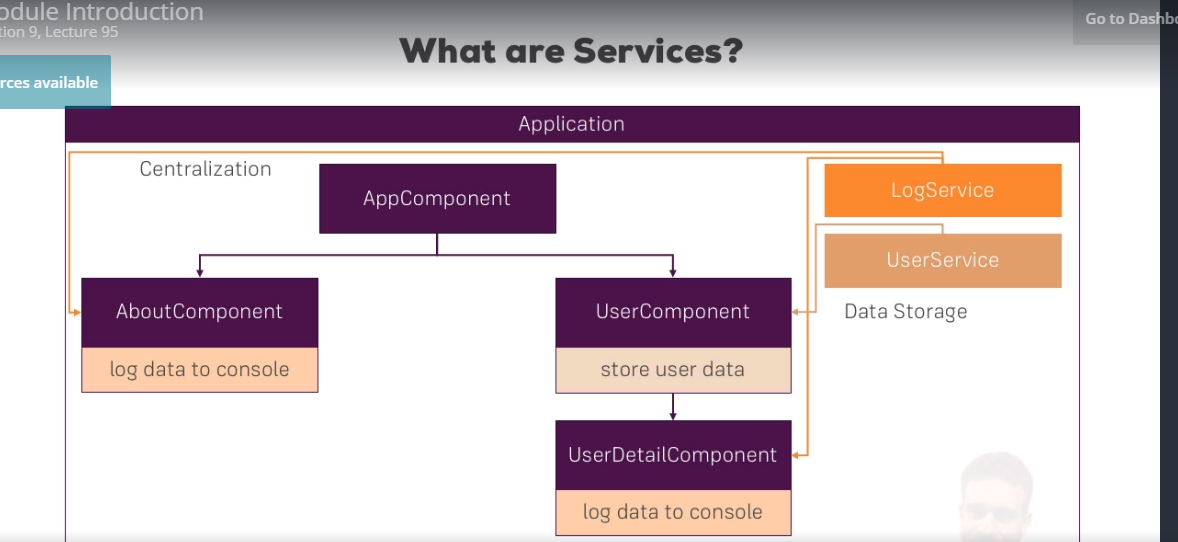
**Section9: Lecture 95//Module Introduction**

1. What are services?

If we have same type of methods in 2 components- then we are duplication the code in these components. There is one more scenario i.e. providing data. These are both the typical use cases for a service.

Service is mainly another piece in your angular app i.e. another class you can add which acts a central repository as a central business unit i.e. something where you can store and also centralize your code in.

1. Details are shown in the image below:



**Section 9: Lecture 96//Why would you need services**

1. There is an example app which we will find attached to this lecture, in this lecture that app is explained.

**Section 9: Lecture 97//Creating a logging Service**

1. We will create a service which will take away the logging task in the sample application we had. So, how do we create a new service?
2. This service will do a generic task so we will place it in the app folder, or we can also place it in the generic or subfolder.
3. So, we will create logging.service.ts file. Now as we know that a component becomes component when we add the @Component decorator, similarly a directive becomes directive when we add @Directive decorator in the .ts file. Now we will not do the similar kind of thing for the service, as service is just normal typescript class.
4. logging.service.ts:
5. export class LoggingService{
6. logStatusChange(status: string){
7. console.log('A server status changed, new status: ' + status);
8. }
9. }

5. new-account.component.ts:

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

//import {LoggingService} from '../logging.service'

@Component({

selector: 'app-new-account',

templateUrl: './new-account.component.html',

styleUrls: ['./new-account.component.css']

})

export class NewAccountComponent {

@Output() accountAdded = new EventEmitter<{name: string, status: string}>();

onCreateAccount(accountName: string, accountStatus: string) {

this.accountAdded.emit({

name: accountName,

status: accountStatus

});

//console.log('A server status changed, new status: ' + accountStatus);

// const service = new LoggingService;

// service.logStatusChange(accountStatus);

}

}

**Section 9: Lecture 98//Injecting and logging the service into components**

1. In the last lecture we created our own service and we understood that we should not instantiate it on our own. We understood that angular offers a great tool which will give access to our own services –its angular’s dependency injector. What is the dependency injector? – Well a dependency is something on which a class of ours depend on. For Ex. the new-account component depends on the logging service because we want to access a method of that service.
2. Dependency injector simply injects this dependency i.e. injects the instance of this class into our component automatically. All we need to do is we require such an instance. So, how do we inform angular that we require such an instance - we add a constructor to the class/ to the component – in this case where we want to use our service.
3. There we will bind it to a property using a typescript shortcut and adding an accessor in front of the name of the argument to instantly create a property with the same name and bind the value to it – so here I will name this logging service, but this name is totally up to you.
4. Now there is an important thing here add a type assignment- this is not optional. Type would be the class name i.e. LoggingService in this case. Make sure to also add the import at the top.
5. This simple task informs angular that we would need the simple task of importing this logging service. Correctly we will ask – well, how do we inform angular? And why does it matter if we write this in a constructor.
6. Think of who gives us this component i.e. account-component? - This a typescript class so somewhere this needs to get instantiated so that something happens in our app.
7. Well, who is responsible for creating our components? – Angular is of course, because we are placing selectors in our templates.
8. Angular finds selectors in our templates, and when angular finds these selectors in our templates it gives us instances of our components. Now, since angular is responsible for instantiating our components – angular will need to construct them correctly.
9. So, if we define in the constructor, that we require some argument, angular will recognize this and now it tries to give us that argument. It tries to give us the type which we define – in this case we have defined the type as LoggingService.
10. So, it knows that we want the instance of the logging service class, because we defined the type here – this why this is important.
11. Now, that is almost enough but not completely enough. Now it knows what we want but it doesn’t know how to give us such an instance. We need to do one additional step – we need to provide a service. Provide – simply means we tell angular – how to create it and that sounds very complicated but it is very simple.
12. All we have to do is add one extra property in @Component decorator i.e. the providers property here - this also takes array like other properties we added before and here we also have to specify the type of what we want to be able to get or to provide – here LoggingService again.
13. When it will build or instantiate such a component – it will check that we need such an instance from the provider – and it will know how to give us such an instance.
14. Now, we can simply in our service anywhere in this component and access our loggingService property which is created automatically and call logStatusChange.
15. Now, let’s also do the same by copying that in the account component.
16. account.component.ts:
17. import { Component, EventEmitter, Input, Output } from '@angular/core';
18. import { LoggingService } from '../logging.service';
19. @Component({
20. selector: 'app-account',
21. templateUrl: './account.component.html',
22. styleUrls: ['./account.component.css'],
23. providers: [LoggingService]
24. })
25. export class AccountComponent {
26. @Input() account: {name: string, status: string};
27. @Input() id: number;
28. @Output() statusChanged = new EventEmitter<{id: number, newStatus: string}>();
29. constructor(private loggingService: LoggingService){}
30. onSetTo(status: string) {
31. this.statusChanged.emit({id: this.id, newStatus: status});
32. //console.log('A server status changed, new status: ' + status);
33. this.loggingService.logStatusChange(status);
34. }
36. }

17. new-account.component.ts:

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

import { LoggingService } from '../logging.service';

//import {LoggingService} from '../logging.service'

@Component({

selector: 'app-new-account',

templateUrl: './new-account.component.html',

styleUrls: ['./new-account.component.css'],

providers: [LoggingService]

})

export class NewAccountComponent {

@Output() accountAdded = new EventEmitter<{name: string, status: string}>();

constructor(private loggingService: LoggingService){}

onCreateAccount(accountName: string, accountStatus: string) {

this.accountAdded.emit({

name: accountName,

status: accountStatus

});

//console.log('A server status changed, new status: ' + accountStatus);

// const service = new LoggingService;

// service.logStatusChange(accountStatus);

this.loggingService.logStatusChange(accountStatus);

}

}

**Section 9: Lecture 99//Creating a Data Service**

1. We saw how to use the logging service that we created in the last lecture. Now let’s, see another typical use case for of the Data Service, which will store and manage our data.
2. Right now we have the accounts array in the AppComponent which stores here in this component and we have this chain of property and event binding to get through the app component. So, that there we can update our account. Let’s create the service for that.
3. The account.service.ts . So, now I will move my accounts from app.component.ts to this service.
4. Now, we have accounts service we should inject it in the app.component.ts and also inside the providers array in the component we need to define the service name.
5. Here, we will define the OnInit method as most of the initialization should not be done inside the constructor.
6. Accounts.component.ts:
7. import { Component, EventEmitter, Input, Output } from '@angular/core';
8. import { LoggingService } from '../logging.service';
9. import { AccountsService } from '../account.service';
10. @Component({
11. selector: 'app-account',
12. templateUrl: './account.component.html',
13. styleUrls: ['./account.component.css'],
14. providers: [LoggingService]
15. })
16. export class AccountComponent {
17. @Input() account: {name: string, status: string};
18. @Input() id: number;
19. //@Output() statusChanged = new EventEmitter<{id: number, newStatus: string}>();
20. constructor(private loggingService: LoggingService, private accountsService: AccountsService){}
21. onSetTo(status: string) {
22. // this.statusChanged.emit({id: this.id, newStatus: status});
23. //console.log('A server status changed, new status: ' + status);
24. this.accountsService.updateStatus(this.id, status);
25. this.loggingService.logStatusChange(status);
26. }
28. }

7. new-account.component.ts:

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

import { LoggingService } from '../logging.service';

import { AccountsService } from '../account.service';

//import {LoggingService} from '../logging.service'

@Component({

selector: 'app-new-account',

templateUrl: './new-account.component.html',

styleUrls: ['./new-account.component.css'],

providers: [LoggingService, AccountsService]

})

export class NewAccountComponent {

//@Output() accountAdded = new EventEmitter<{name: string, status: string}>();

constructor(private loggingService: LoggingService, private accountsService: AccountsService){}

onCreateAccount(accountName: string, accountStatus: string) {

// this.accountAdded.emit({

// name: accountName,

// status: accountStatus

// });

//console.log('A server status changed, new status: ' + accountStatus);

// const service = new LoggingService;

// service.logStatusChange(accountStatus);

this.accountsService.addAccount(accountName, accountStatus);

this.loggingService.logStatusChange(accountStatus);

}

}

8. account.service.ts

export class AccountsService{

accounts = [

{

name: 'Master Account',

status: 'active'

},

{

name: 'Testaccount',

status: 'inactive'

},

{

name: 'Hidden Account',

status: 'unknown'

}

];

addAccount(name: string, status: string){

this.accounts.push({name: name, status: status});

}

updateStatus(id :number, status:string){

this.accounts[id].status = status;

}

}

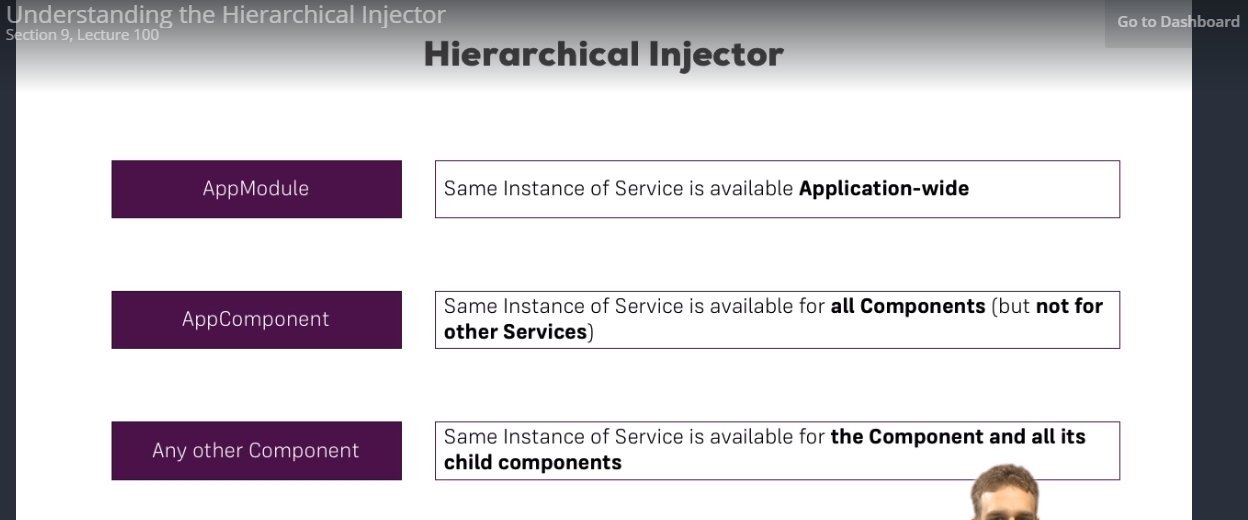
1. app.component.ts:
2. import { Component, OnInit } from '@angular/core';
3. import { AccountsService } from './account.service';
4. @Component({
5. selector: 'app-root',
6. templateUrl: './app.component.html',
7. styleUrls: ['./app.component.css'],
8. providers: [AccountsService]
9. })
10. export class AppComponent implements OnInit {
12. accounts: {name: string, status: string}[] = [];
13. constructor(private accountsService: AccountsService){}
14. ngOnInit(){
15. this.accounts = this.accountsService.accounts;
16. }
17. // onAccountAdded(newAccount: {name: string, status: string}) {
19. // }
20. // onStatusChanged(updateInfo: {id: number, newStatus: string}) {
22. // }
23. }

**Section 9: Lecture 100//Understanding the Hierarchical Injector**

1. In the last lecture we created our accounts service and hooked it up in our app, but still our app seems to be broken.
2. Hierarchical Injector:

The angular dependency injector is actually a hierarchical injector - that means that if we provide a service in some place of our app let’s say on one component - the angular framework knows how to create an instance of the service for this component and all its child components and also all the child components of the child components will receive the same instance of that service.

1. There are other places where we can provide a service too. The highest possible level is the app module. The app module does have the providers array. Then the same instance of the service would be available in our whole app. It would also be available for other services - yes we can inject services into services.
2. The next level would be the AppComponent and all its child components will have the same instance of the service. And this is true for any component.
3. The instances can only be propagated to the child components and not to the parent components i.e. they can be propagated only down and not up.
4. The lowest level is the single component with no child component.



**Section 9: Lecture 101//How many Instances of Service Should Be?**

1. In the last lecture we learnt about the hierarchical Injector