# Dependency Injection

Dependency injection is an important application design pattern. Angular has its own dependency injection framework, and you really can't build an Angular application without it. It's used so widely that almost everyone just calls it DI.

# When to use **NgModule** versus an application component

A provider in an NgModule is registered in the root injector. That means that every provider registered within an NgModule will be accessible in the entire application.

On the other hand, a provider registered in an application component is available only on that component and all its children.

# Singleton services

Dependencies are singletons within the scope of an injector. However, Angular DI is a hierarchical injection system, which means that nested injectors can create their own service instances.

# Why *@Injectable()*?

@Injectable() marks a class as available to an injector for instantiation. Generally speaking, an injector reports an error when trying to instantiate a class that is not marked as @Injectable().

Consider adding @Injectable() to every service class, even those that don't have dependencies and, therefore, do not technically require it. Here's why:

**Future proofing:** No need to remember @Injectable() when you add a dependency later.

**Consistency:** All services follow the same rules, and you don't have to wonder why a decorator is missing.

**Injectors are also responsible for instantiating components , So why doesn't components have @Injectable().**

You can add it if you really want to. It isn't necessary because the Components are already marked with @Component, and this decorator class (like @Directive and @Pipe, which you learn about later) is a subtype of @Injectable(). It is in fact @Injectable() decorators that identify a class as a target for instantiation by an injector.

# Injector providers

A provider provides the concrete, runtime version of a dependency value. The injector relies on providers to create instances of the services that the injector injects into components and other services.

You must register a service provider with the injector, or it won't know how to create the service.

# The **Provider** class and **provide** object literal

providers: [Logger]

The above code is actually a shorthand expression for a provider registration using a provider object literal with two properties:

[{ provide: Logger, useClass: Logger }]

# Alternative class providers

[{ provide: Logger, useClass: BetterLogger }]

# Aliased class providers

[ NewLogger,

// Not aliased! Creates two instances of `NewLogger`

{ provide: OldLogger, useClass: NewLogger}]

[ NewLogger,

// Alias OldLogger w/ reference to NewLogger

{ provide: OldLogger, useExisting: NewLogger}]

# Value providers

When we have some ready-made object to inject.

# Factory providers

Sometimes you need to create the dependent value dynamically, based on information you won't have until the last possible moment. Maybe the information changes repeatedly in the course of the browser session.

**Factory**

import { OrderService } from '../services/order/order.service';

import { LoginService } from '../services/login/login.service';

const orderServiceFactory = (loginService: LoginService) => {

return new OrderService(loginService.isLoggedin);

};

export let orderServiceProvider = {

provide: OrderService,

useFactory: orderServiceFactory,

deps: [LoginService]

};

**Component :**

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

import { OrderService } from '../services/order/order.service';

import { Orders } from './order';

import { orderServiceProvider } from '../factory/product.service.provider';

@Component({

selector: 'app-order',

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

styleUrls: ['./order.component.css'],

providers: [orderServiceProvider]

})

export class OrderComponent implements OnInit {

@Input() customerId: number;

**Service:**

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

import { Orders } from '../../order/order';

import { OrderDetails } from '../../order/order.details';

import { Observable } from 'rxjs/Observable';

import 'rxjs/add/operator/map';

import 'rxjs/add/operator/catch';

@Injectable()

export class OrderService {

orders: Orders[] = [{

amount: 100,

customerId: 1,

orderDate: new Date('10-Nov-2016'),

details: [],

orderId: 1,

orderNumber: '001-ABC'

}];

orderDetails: OrderDetails[];

constructor(private isAuthorized: boolean) { }

getAllOrders() {

console.log(this.isAuthorized);

return Observable.of(this.orders);

}

getOrderByCustomerId(customerId: number) {

return Observable.of(this.orders.filter(data => data.customerId === customerId));

}

getOrderDetails(orderId: number) {

return this.orderDetails;

}

}