# Services

Services are just classes decorated with @Injectable. Instances are provided to injectors. Injectors deliver when needed

# DI

In Angular, the DI framework provides declared dependencies to a class when that class is instantiated.

Why DI

Loosely Coupled Code

Flexible

Unit Testable - Easier to Test

Dependency Inversion Principle

Within Angular’s DI system, instead of directly importing and creating a new instance of a class,

instead we will:

• Register the “dependency” with Angular

• Describe *how* the dependency will be *injected*

• Inject the dependency

Dependency injection in Angular has three pieces:

• the **Provider** (also often referred to as a binding) maps a *token* (that can be a string or a class) to a list of dependencies. It tells Angular how to create an object, given a token. A Provider tells an injector how to create the service.

Provider Recipes

• UseClass

• UseValue

• UseFactory

• UseExisting

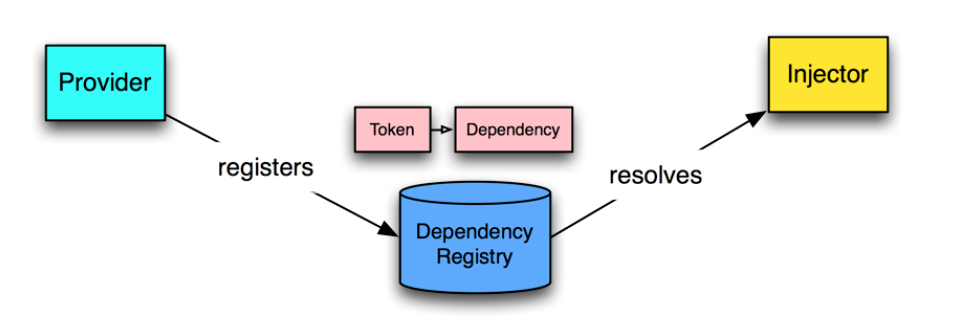
• the **Injector** that holds a set of bindings and is responsible for resolving dependencies and injecting them when creating objects

Deliver provided services when they are requested via constructor injection

Maintain single instance of each service provided.

Determine what to inject based on emitted metadata

• the **Dependency** that is what’s being injected



Angular uses an *injector* to **resolve** a dependency and **create the instance**. This is done for us behind the scenes

**For UserInjector Demo**

The ReflectiveInjector is a concrete implementation of Injector that uses *reflection*

to look up the proper parameter types. While there are other injectors that are possible

ReflectiveInjector is the “normal” injector we’ll be using in most apps.

User Service Demo

First remove the user.service.ts file and user.component.ts files as well

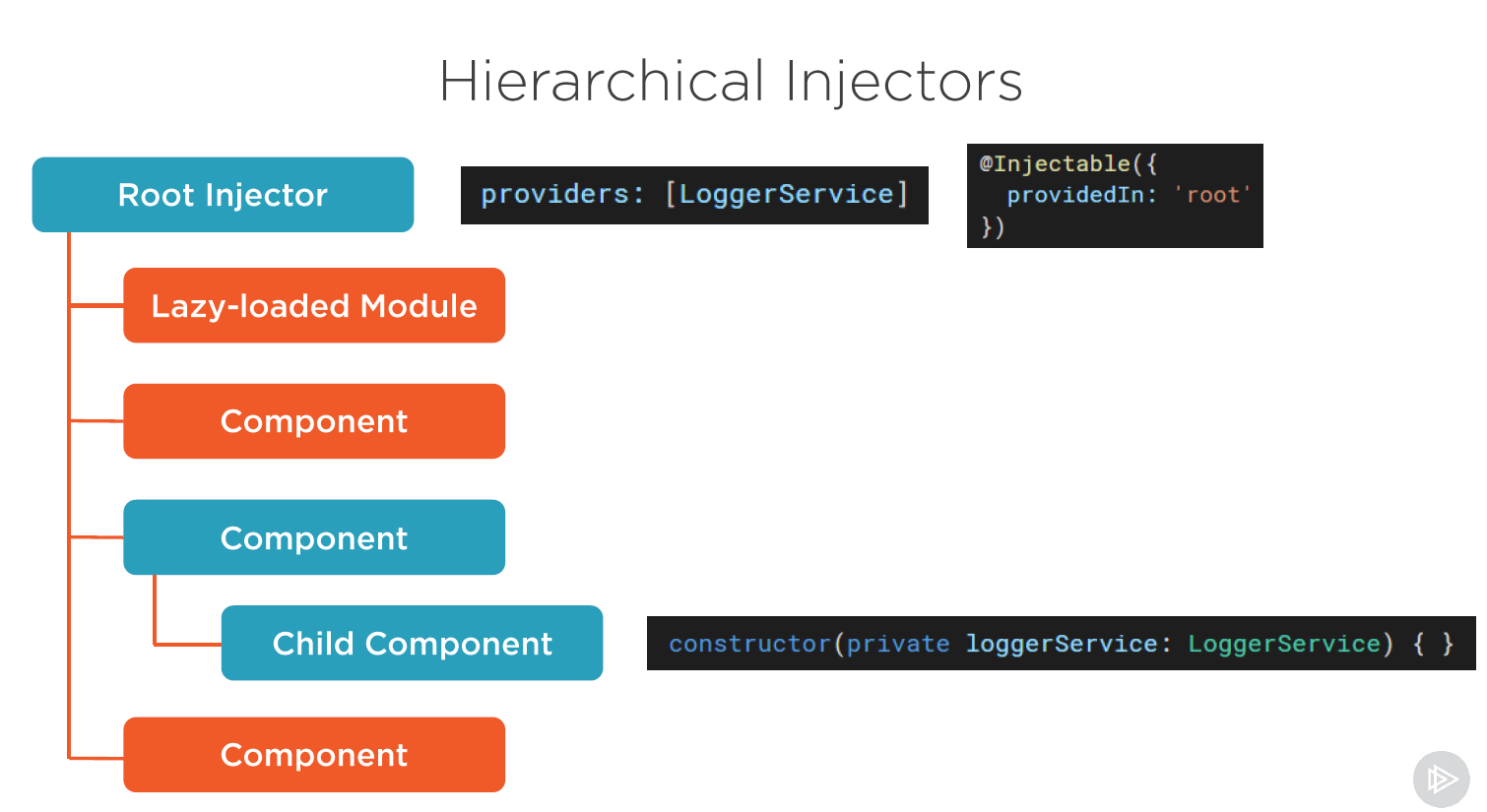
Let’s convert our UserService to be *injectable* as a singleton across our app. First, we’re going to add it to the providers key of our NgModule:

@Injectable()

Add in App.module providers

UserService

Hierarchical Injectors



**Http**

Import Angular http Service

Create the data access service

Inject the Angular http service

Import observable and the observable operator

Write the code to issue each http request

Angular Material Themes –

<https://material.io/design/color/the-color-system.html#tools-for-picking-colors>

**Security**

<https://developer.okta.com/blog/2017/04/17/angular-authentication-with-oidc#authentication-with-the-okta-auth-sdk>

Secure Practices

* [Prevent cross-site scripting (XSS)](https://www.syncfusion.com/blogs/post/top-5-best-practices-angular-app-security.aspx#prevent-cross-site-scripting)
* [Block HTTP-related vulnerabilities](https://www.syncfusion.com/blogs/post/top-5-best-practices-angular-app-security.aspx#block-http-related-vulnerabilities)
* [Avoid risky Angular APIs](https://www.syncfusion.com/blogs/post/top-5-best-practices-angular-app-security.aspx#avoid-risky-angular-apis)
* [Don’t customize Angular files](https://www.syncfusion.com/blogs/post/top-5-best-practices-angular-app-security.aspx#dont-customize-angular-files)
* [Stay updated with the latest Angular library](https://www.syncfusion.com/blogs/post/top-5-best-practices-angular-app-security.aspx#stay-updated-with-latest-angular-library)