Services and Dependency Injection

Introduction to Angular 6

Service overview

- Service is a broad category encompassing any value, function, or feature that an app needs.
- A service is typically a class that performs a specific, well-defined purpose.
  - Typically it provides data or logic (that is unrelated to views) that is shared across multiple components.
- Angular distinguishes components from services.
  - By ensuring that a component's functionality is solely view-related and moving all other non view-related functionality to services, you can make your component classes lean and efficient.
- A component can delegate certain tasks to services
  - such as fetching data from the server, validating user input, or logging directly to the console

Service overview

- By defining such processing tasks in a separate injectable service class
  - you make those tasks available to any component
  - facilitates modularity and reusability
  - makes it easier to test a component by using a fake service
- Services can also delegate their tasks and depend on other services to provide them.
- A service class definition is immediately preceded by the @Injectable() decorator.
  - The decorator provides the metadata that allows your service to be injected into client components as a dependency.
  - Also provides a hint to the Angular dependency injection system that the service you are working on might have other dependencies on other services

Dependency injection

- Based on the idea that a component should ask for its dependencies, rather than instantiating them by itself.
  - A dependency can be a service, function or a value.
- The dependency injector provides dependencies to a component so that it can access and use them.
- Angular creates an application-wide injector for you during the bootstrap process, and additional injectors as needed.
  - An injector creates dependencies, and maintains a container of dependency instances that it reuses
  - A provider is an object that tells an injector how to obtain or create a dependency.
- You can make your app more adaptable by injecting different providers of the same kind of service

DI for component instantiation

- When Angular creates a new instance of a component class
  - it determines which services or other dependencies that component needs by looking at the constructor parameter types.
- Angular first checks if the injector has any existing instances of that service.
  - If not, the injector makes one using the registered provider
  - adds it to the injector before returning the service to Angular.
- When all requested services have been resolved and returned
  - Angular calls the component's constructor with those services as arguments

Registering providers

- For any dependency that you need in your app
  - you must register a provider with the app's injector, so that the injector can use the provider to create new instances of that dependency.
- For a service, the provider can be part of the service's own metadata (in the @Injectable() decorator)
  - This makes that service available everywhere
- Can also register providers with specific modules or components (@NgModule() or @Component()) metadata
- By default, the Angular CLI command ng generate service registers a provider with the root injector for your service
- When you provide the service at the root level
  - Angular creates a single, shared instance (singleton) of the service and injects it into any class that asks for it

Providers and the DI hierarchy

- Angular has a hierarchical dependency injection system
  - This supports multiple dependency injectors within the same application
- There is the root injector at the root AppModule level, which is where most services you create will be registered and made available.
- When we add providers at a specific NgModule
  - All components in that module will inherit that injector, taking precedence over that root injector
  - The injector will provide the same instance of that service to all components in that NgModule
  - To register at this level, use the providers property of the @NgModule()decorator,

Providers and the DI hierarchy

- When we add providers at the component level
  - All child components will inherit that injector, taking precedence over the root injector.
  - You get a new instance of the service with each new instance of that component
  - To register at this level, use the providers property of the @Component() metadata.
- Whenever a component asks for a dependency, Angular will check the closest injector in the tree to see whether it can satisfy it.
  - If it can, it will provide it.
  - If not, it will check with the parent injector, all the way to the root injector

Using the DI hierarchy

- Situations when you might want to use this capability of the Angular injector:

- You want to scope services to only certain components and modules, to ensure that a change in the service doesn't impact the entire application.
- You might want different instances of a service in different components
- You want to override a particular service with a more specific implementation for a section of your application.

Injectable() Decorator

- The Injectable decorator is recommended with services
  - hint to Angular DI system that the service you are working on might have other dependencies.
- We can inject any service we want into our component simply by listing it in our constructor
- The name itself doesn't matter; Angular uses the type definition to figure out what service to inject.