Mastering Dependency Injection for Reusability

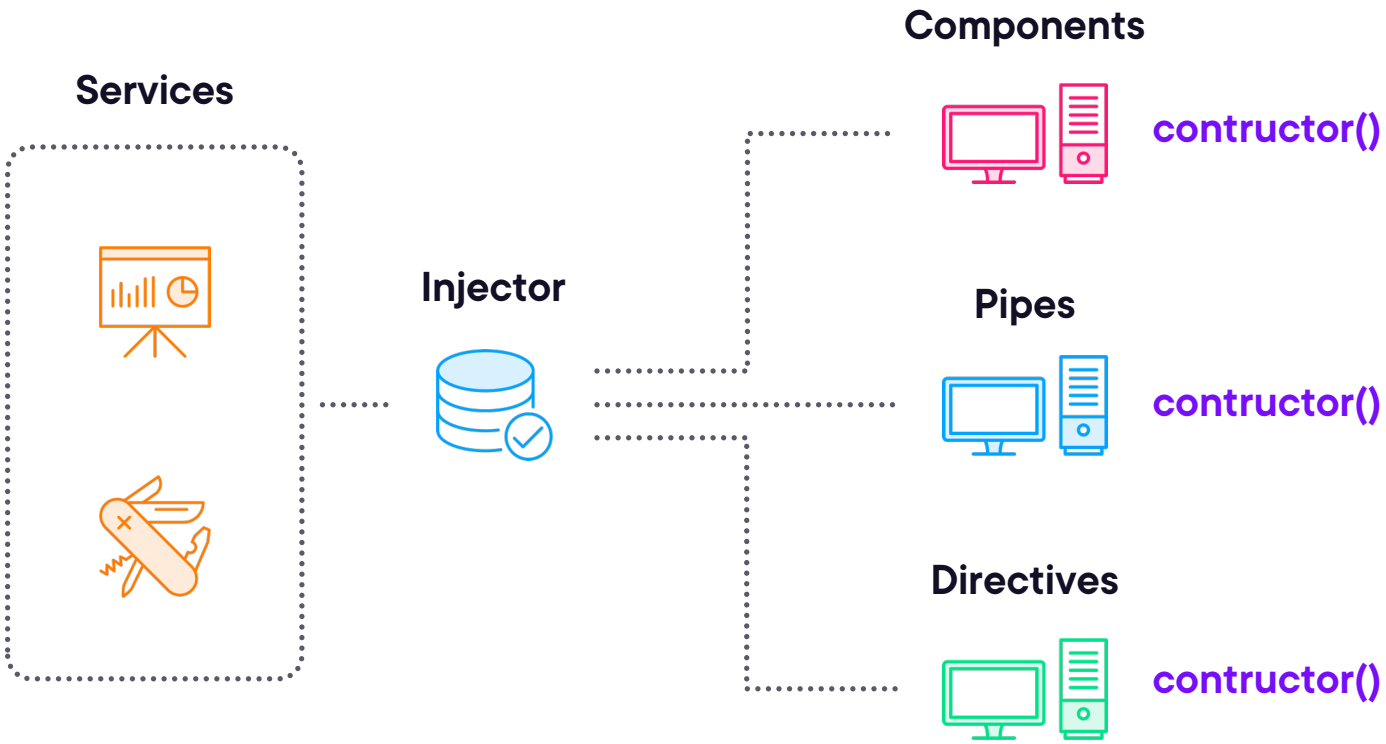
Zachary Bennett
Lead Software Developer

@z_bennett_ | https://www.linkedin.com/in/zbennett10

Dependency Injection

A design pattern wherein you have an external source that provides dependencies to a class.

Angular Dependency Injection



Marking Injectables

The `Injectable` decorator is Angular's way hooking a service up to DI

Component-level

```
@Injectable()
class ProductService {}

. . .

@Component({
  selector: 'product-list',
  template: '...',
  providers: [ProductService]
})
```

Singleton

```
@Injectable({
  providedIn: 'root'
})
class CatalogService {}
```

Multiple services vs. singletons!

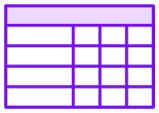
Dependency Injection in Action

Creating Shared Services and Logic Layers

Services for shared code and logic layers should almost always be singletons!

Best Practices for Shared Services

`SharedModule` / `CoreModule`



Use a ‘shared’ or ‘core’ folder

Module providers

Use `providedIn: ‘root’`

`AppModule` **imports**

Component imports

Creating a Shared Service

Advanced DI Patterns for Scalability

Injection Token

Allow you to provide and inject non-class dependencies and/or abstract away specific implementations of classes.

Why Injection Tokens?

Inject primitive values (e.g., strings, numbers, objects)

Define abstract dependencies (interfaces, etc.)

Avoid naming collisions between different library injectables

Providing optional or multi-provider dependencies

Cross-module dependency sharing without tight-coupling

Injection Tokens

Here's an example injection token for an abstract, `ServiceConfig` **interface**

service.config.ts

```ts
import { InjectionToken } from '@angular/core';

// Define the token with a unique string identifier
export const SERVICE_CONFIG = new InjectionToken<ServiceConfig>('SVC_CONFIG');

export interface ServiceConfig {
  apiUrl: string;
  timeout: number;
}
```

Advanced DI Patterns in Action