Angular

Superheroic framework

Training Agenda

- ❑TypeScript : Introduction
- ❑Angular : Introduction
- ❑Angular Building Blocks
  - Component
  - Modules
  - Directives
  - Services
  - Pipes
- ❑Bindings
- ❑Forms
- ❑Remote Calls
- ❑Routing
- ❑Authentication / Authorization
- ❑PWA
- ❑NGRX

Why Angular ?

- Angular is easy
- ES2015+ features
- Corporate Care-taker
- Performance and Mobile
- Project architecture and Maintenance
- Component based architecture

What is Angular ?

- Angular is a structural framework for dynamic (SPA) web apps.
- Extension of standard HTML markups.
- Keeps the front-end clean; separation of concerns.

Why Typescript ?

- Microsoft extension for JS
- Object oriented features
- ES6+ features
- Type definition
- Angular itself programmed in TS

TypeScript features

- Classes & Inheritance
- Module system
- Arrow function
- Template String
- Constants and block scope
- Destructuring
- Spread & Rest operator
- Decorator
- Additional types

Understanding Angular Environment Setup

- Node
- TypeScript
- Webpack
- Angular Packages
- RxJS
- ZoneJS

Components

- A component controls a patch of screen real estate that we could call a view, and declares reusable UI building blocks for an application.
- Passing data to/from components
  - Property binding
  - Event binding
  - Two way data-binding
- Nested components
  - Parent to Child
  - Child to Parent
- Data projection
- Component types :
  - Smart components
  - Dump components

ngOnChanges	- called when an input binding value changes
ngOnInit	- after the first ngOnChanges
ngDoCheck	- after every run of change detection
ngAfterContentInit	- after component content initialized
ngAfterContentChecked	- after every check of component content
ngAfterViewInit	- after component's view(s) are initialized
ngAfterViewChecked	- after every check of a component's view(s)
ngOnDestroy	- just before the component is destroyed

Component Life Cycle

Directives

- A Directive modifies the DOM to change appearance, behavior or layout of DOM elements.
- Directive Types :
  - *Component Directive* : directive with template
  - *Attribute Directive* : directives that change the behavior of a component or element but don't affect the template
  - *Structural Directives* : directives that change the behavior of a component or element by affecting how the template is rendered

Pipes

- Pipes are used to filter/format data for template
- Built-in Pipes :
    - Currency
    - Date
    - Uppercase
    - Lowercase
    - Number
    - JSON
    - Percent
    - Async
- Custom pipes
    - Pure
    - Impure

Forms

Template Driven Forms

- Angular infers the Form Object from the DOM
- App logic resides inside the template

Model Driven Forms

- Form is created programmatically and sync with the DOM
- App logic resides inside the component
- Use of FormControl, FormGroup, FormBuilder

DI & Services

Angular's DI system is controlled through **@NgModule**.

Services implement DI concepts in an Angular App.

Services are simple ES6 classes.

Services are registered with Angular App using providers.

Services are Singleton.

Services : Hierarchical Injector

Root Module	Same instance of service is available Application-wide
Root Component	Same instance of service is available for all components (but not for other services)
Other Component	Same instance of service is available for the component and it’s own child components

Services : Hierarchical Injector

Use providedIn: 'root' for services which should be available in whole application as singletons

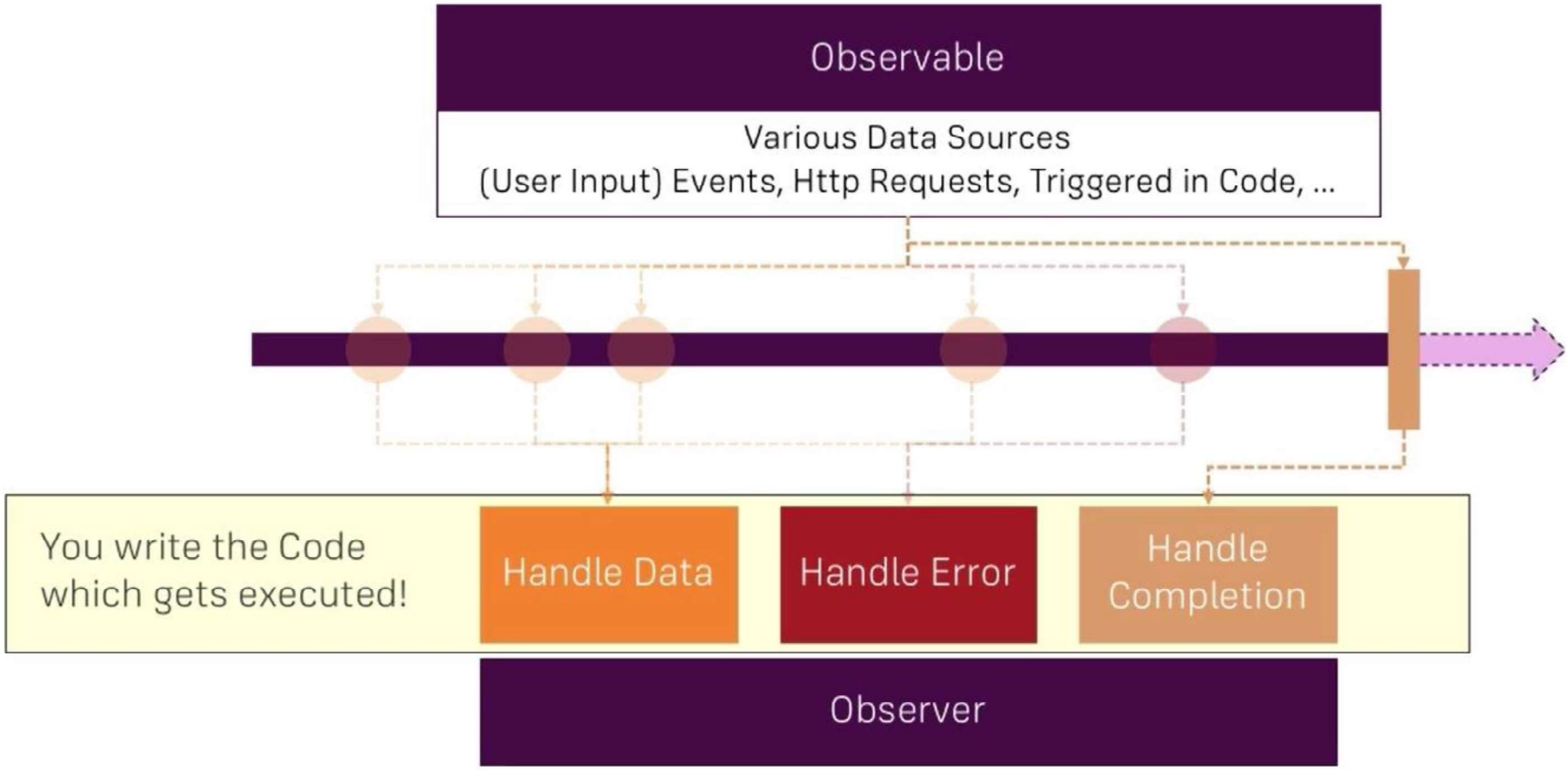
Never use `providedIn: EagerlyImportedModule`, you don’t need it and if there is some super exceptional use case then go with the providers: [] instead

Use `providedIn: LazyServiceModule` to prevent service injection in the eagerly imported part of the application or even better use `providers: [LazyService]` in the LazyModule.

If we want to use LazyServiceModule then we have to import it in LazyModule to prevent circular dependency warning. LazyModule will then be lazy loaded using Angular Router for some route in a standard fashion.

Use `providers: []` inside of @Component or @Directive to scope service only for the particular component sub-tree which will also lead to creation of multiple service instances (one service instance per one component usage)

Observables : An Overview

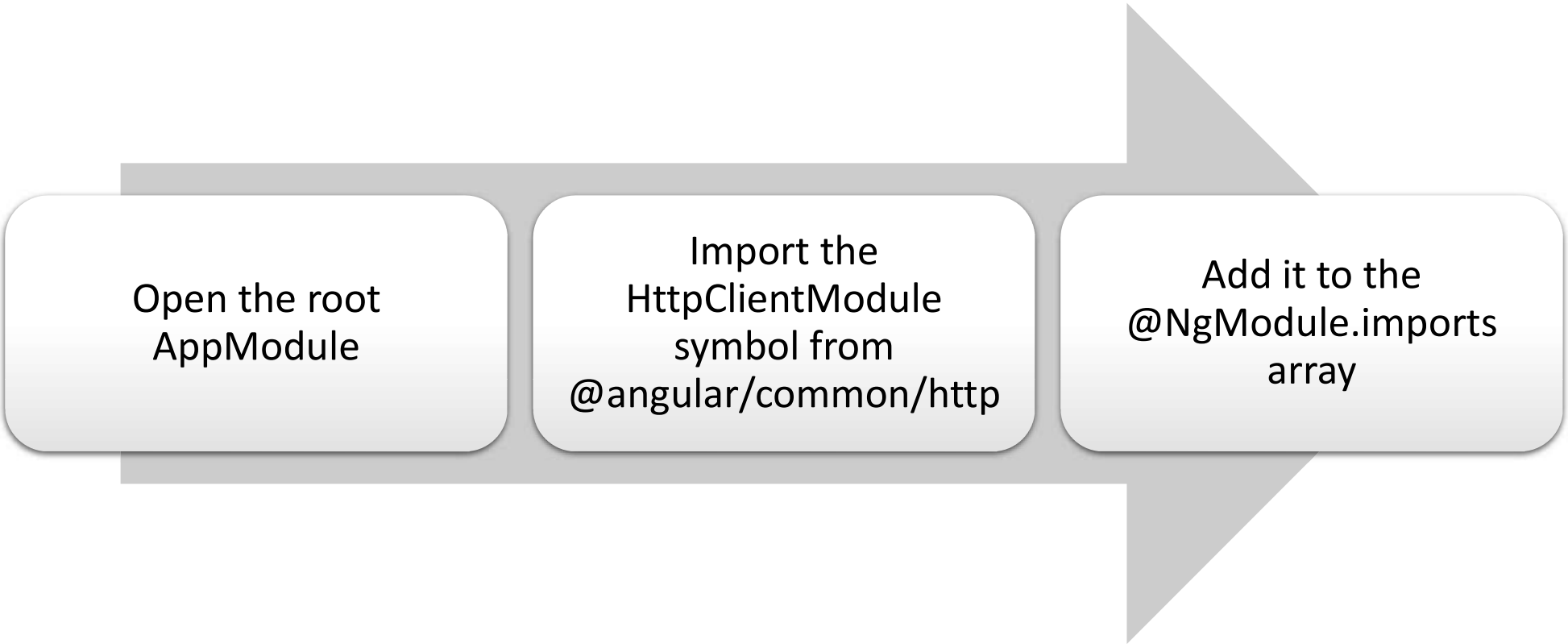


HttpClient

The HttpClient in @angular/common/http offers a simplified client HTTP API for Angular applications that rests on the XMLHttpRequest interface exposed by browsers.

Benefits of HttpClient:	
	Typed request and response objects
	Request and response interception
	Observable APIs
	Streamlined error handling.

HttpClient : Unlocking



Routing allows to:

- Maintain the state of the application
- Implement modular applications
- Implement the application based on the roles (certain roles have access to certain URLs)

5 steps routing:

- Checking the base href tag in index file
- Configuring routes with components
- Tell angular about routing app
- Setting up the routing links
- Provide space on template to load the component

Routing

Programmatic navigation

Child routing

Routes with parameters

Route guard (Authentication)

Query Parameters

Routing (Cntd..)

Modules

- A module is a mechanism to group components, directives, pipes and services that are related
- Module Types -
  - Root Module : one per application
  - Feature Module : depends on application features
- Modules can be instantiate lazily

Debugging Angular Apps

- Prevent Bugs with TypeScript
- Using Debugger Statements to Stop JavaScript Execution
- Inspect Data with the JSON pipe
- Console Debugging
- Augury Chrome Plugin
- Debugging RxJS Observables using ‘tap’ operator