

Angular

Superheroic framework



Training Agenda

☐ TypeScript : Introduction

☐ Angular : Introduction

☐ Angular Building Blocks

- Component
- Modules
- Directives
- Services
- Pipes

☐ Data Binding

☐ Forms

☐ HTTP

☐ Routing

☐ Authentication / Authorization

Angular is easy

ES2015+ features

Corporate Care-taker

Performance and Mobile

Project architecture and Maintenance

Component based architecture

Why Angular ?

Angular is a structural framework for dynamic (SPA) web apps.

Extension of standard HTML markups.

Keeps the front-end clean; separation of concerns.

What is
Angular ?

Microsoft extension for JS

Object oriented features

ES6+ features

Type definition

Angular itself programmed in TS

Why Typescript ?

TypeScript features

- Classes & Inheritance
- Module system
- Arrow functions
- Template String
- Constants and block scopes
- Destructuring
- Spread & Rest operators
- Decorators
- Additional types

Understanding Angular Environment Setup

- Node
- TypeScript
- 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 into components
 - Property binding
 - Event binding
 - Two way data-binding
- Nested components
- 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

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

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

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.

DI & Services

Hierarchical Injector

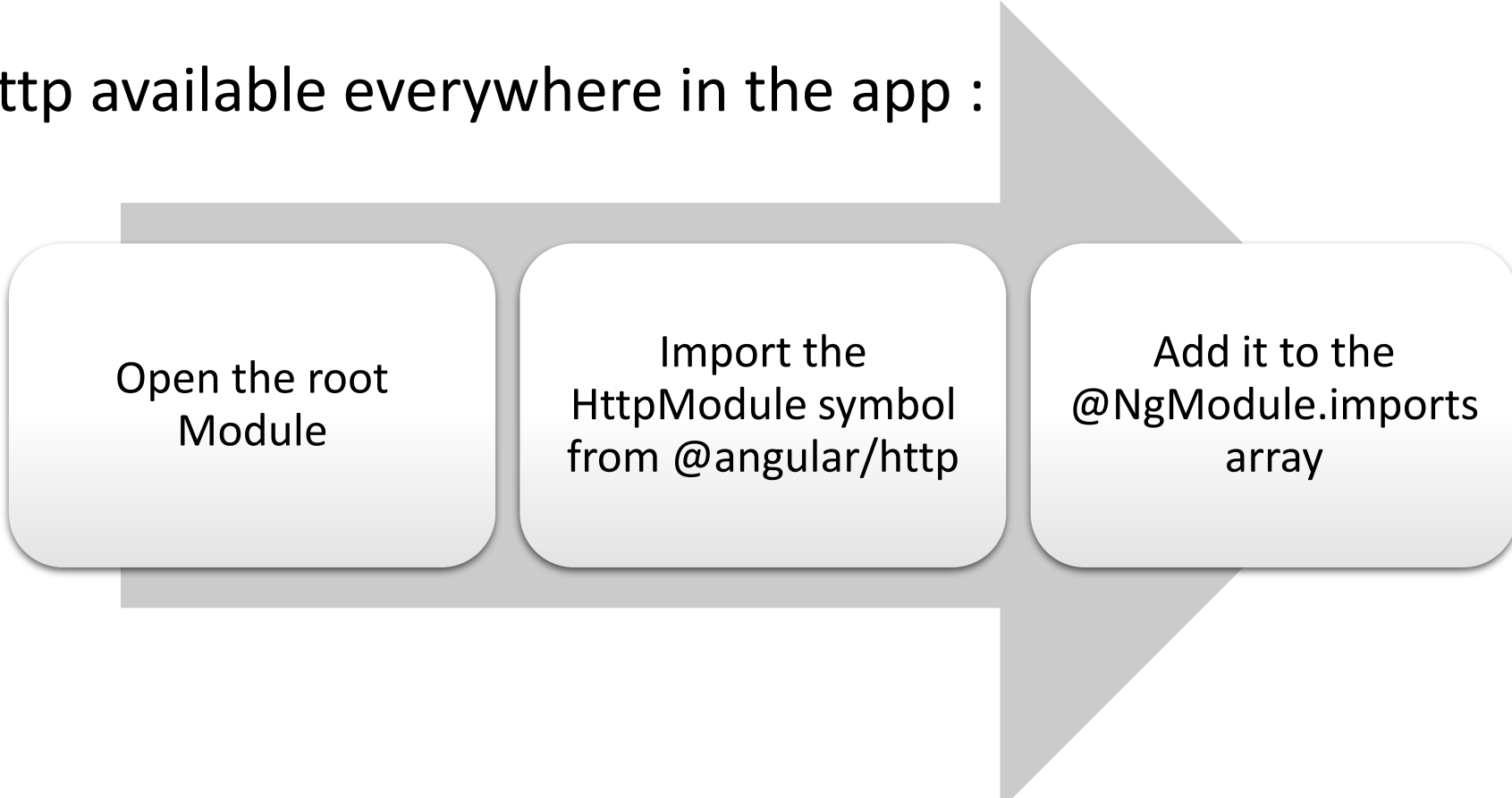
Root Module	Same instance of service is available Application-wide
-------------	--

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
-----------------	---

HTTP

- Http is Angular's mechanism for communicating with a remote server over HTTP.
- To make Http available everywhere in the app :

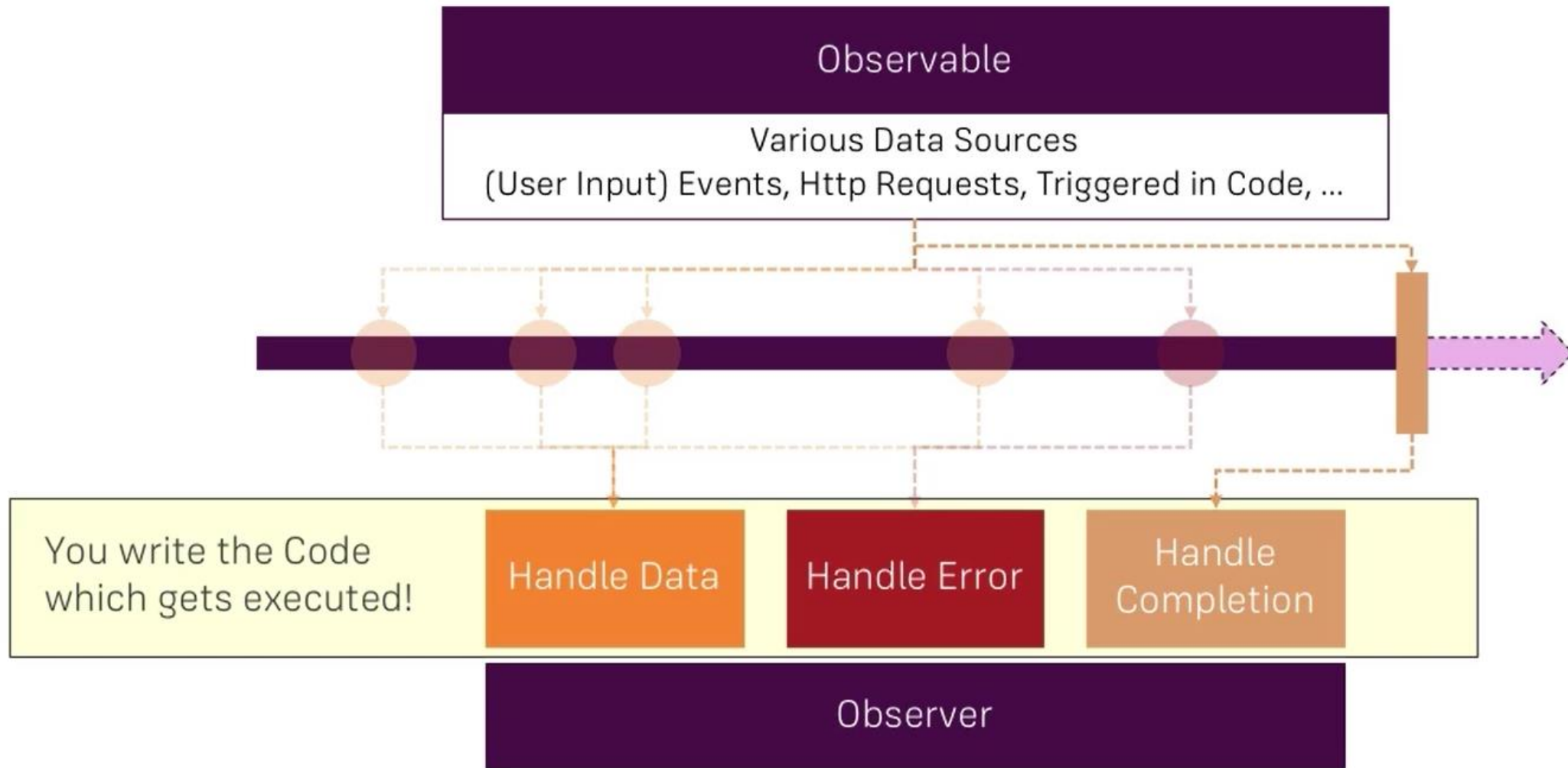


Open the root
Module

Import the
HttpModule symbol
from @angular/http

Add it to the
@NgModule.imports
array

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



Open the root
AppModule

Import the
HttpClientModule
symbol from
`@angular/common/http`

Add it to the
`@NgModule.imports`
array

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)

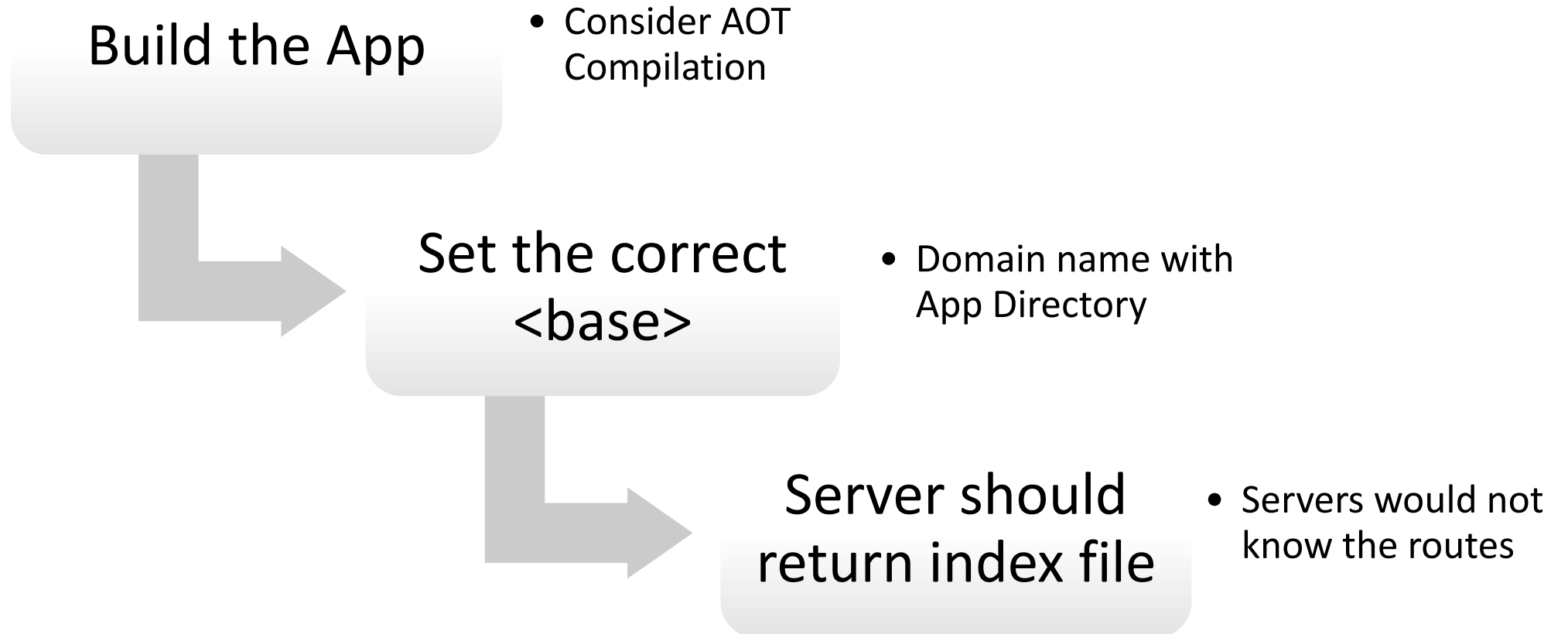
Query Parameters

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

Deployment Steps :



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 'do' operator

Securing Angular Apps

- Best Practices :
 - Keep current with the latest Angular library releases
 - Don't modify your copy of Angular.
 - Avoid Angular APIs marked in the documentation as “*Security Risk.*”
- Preventing cross-site scripting (XSS)
 - Angular treats all values as untrusted by default.
 - Angular sanitizes and escapes untrusted values.
 - Interpolated content is always escaped
 - Angular recognizes the *binded value* as unsafe and automatically sanitizes it
 - Never generate template source code by concatenating user input and templates, instead use the offline template compiler (template injection)

Securing Angular Apps

Trusting Safe Values : To mark a value as trusted, inject *DomSanitizer* and call one of the following methods -

bypassSecurityTrustHtml

bypassSecurityTrustScript

bypassSecurityTrustStyle

bypassSecurityTrustUrl

bypassSecurityTrustResourceUrl

HTTP-level vulnerabilities : Angular has built-in support to help prevent two common HTTP vulnerabilities -

Cross-Site Request Forgery (CSRF or XSRF)

Cross-Site Script Inclusion (XSSI) / JSON vulnerability

Optimizing Angular App Performance

Using onPush change detection strategy

Avoid computing values in templates

Using lazy loading

Disable change detection (if required)

References

Books

- Rangle's Angular2 Training Book
- Ngbook2

Web

- <http://angular.io>
- <http://rangle.io>
- <http://www.egghead.io>
- <http://www.youtube.com>