Angular:

You’ve Talked About It, You’ve Read About It, Now You’re Ready to Build an Application

Richard Taylor
Tech Lead/Sr. Developer
Logical Advantage
http://www.logicaladvantage.com

@rightincode
http://www.rightincode.com

Angular: What is it?

Angular is a framework for building client applications using HTML and Javascript or a language like TypeScript or Dart that transpiles to Javascript.

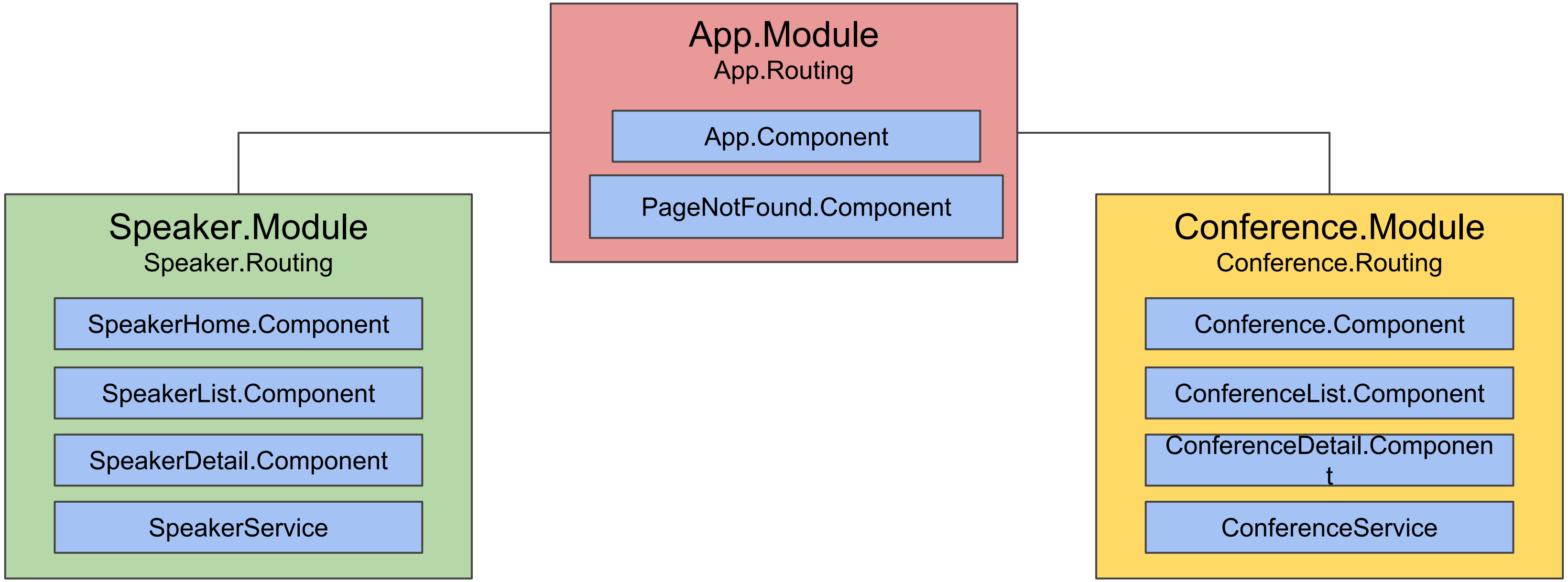
The latest version of Angular, 4.0, was released on March 23, 2017.

Angular 2+ is a complete rewrite of Angular 1.x and is written in Typescript, which was created by Microsoft. Angular 2+ applications can be written in Typescript (recommended by the Angular team), native JavaScript (ECMAScript 5 or 6) and in Dart.

Angular: The Main Building Blocks

- Modules
- Components
- Templates
- Metadata
- Data Binding
- Directives
- Services
- Dependency Injection

Speaker Register Application Architecture



Angular: The Module

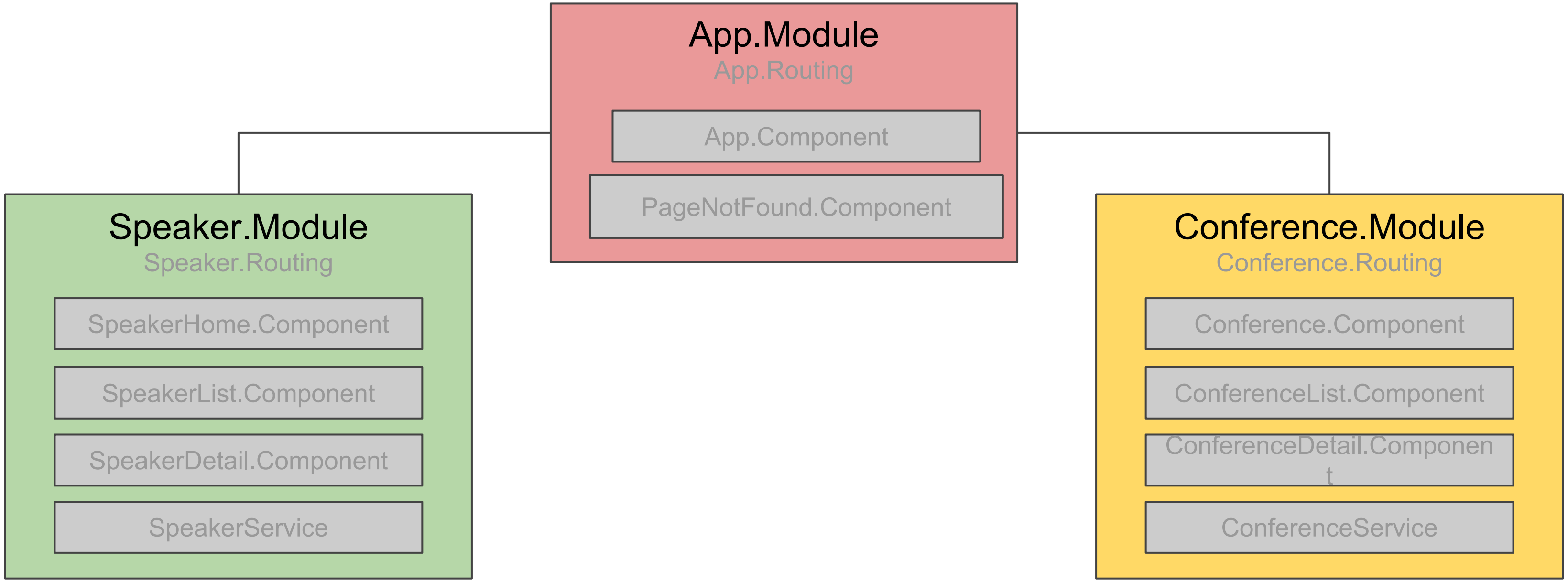
Angular: The Module

- Angular applications are intended to be modular and built from many modules
- A module is a block of code dedicated to a single purpose - think single responsibility
- Modules inherently support re-use
- Modules consolidate components, directives, and pipes into cohesive blocks of functionality focused on a feature area, workflow, or common collection of utilities
- Modules export something of value, for example, a class, function, value, component
- The exported items of a module are imported by other modules
- An Angular module is a class decorated with the @NgModule decorator
- An Angular application contains at least one module, the root module
- Additional modules, named feature modules or shared modules, can be used to separate concerns and better organize an application

Angular: The Module

- Angular launches an application by bootstrapping the root module
- There are two options for bootstrapping an Angular application
  - Dynamic bootstrapping with the Just-In-Time (JIT) compiler
  - Static bootstrapping with the Ahead-Of-Time (AOT) compiler
- Dynamic bootstrapping compiles the application in the browser and then launches the application
- Static bootstrapping compiles the application as a part of the build process.
  - Static bootstrapping produces a much smaller application that launches faster
  - Because the application was pre-complied, the Angular compiler is not sent to the browser because it is no longer needed

Speaker Register Application Architecture



Angular: The Component

Angular: The Component

- A component is a unit of code that controls a section of the screen or view
- Application logic used to support the view is defined in a class
- The class interacts with the view via an API of properties and methods
- A component can have dependencies on modules that export classes, services, and other items
- An Angular component is a class decorated with the @Component decorator

Angular: Component Lifecycle

- Components have a lifecycle that is managed by Angular
- Angular creates, renders, creates/renders children of, checks for data-bound property changes, and destroys components before removing them from the DOM (Document Object Model)
- Angular provides component lifecycle hooks that allow our code to inspect the state of the component at key moments of the lifecycle and to act/re-act when those moments occur
- These moments are accessible in our code by implementing one or more of the lifecycle hook interfaces
- Implementing these interfaces are optional - as long as the method(s) is/are defined, Angular will call it - **using the interface(s) are highly recommended**!

Angular: Component Lifecycle - contd.

- **ngOnInit** - Initialize component after Angular initializes the data-bound input properties
- **ngOnChanges** - Respond after Angular sets a data-bound input property. The method receives a “changes” object of current and previous values
- **ngDoCheck** - Detect and act upon changes that Angular can or will not detect on its own. It is called every change detection run.
- **ngOnDestroy** - Cleanup just before Angular destroys the component. Here we can unsubscribe observables and detach event handlers to avoid memory leaks.

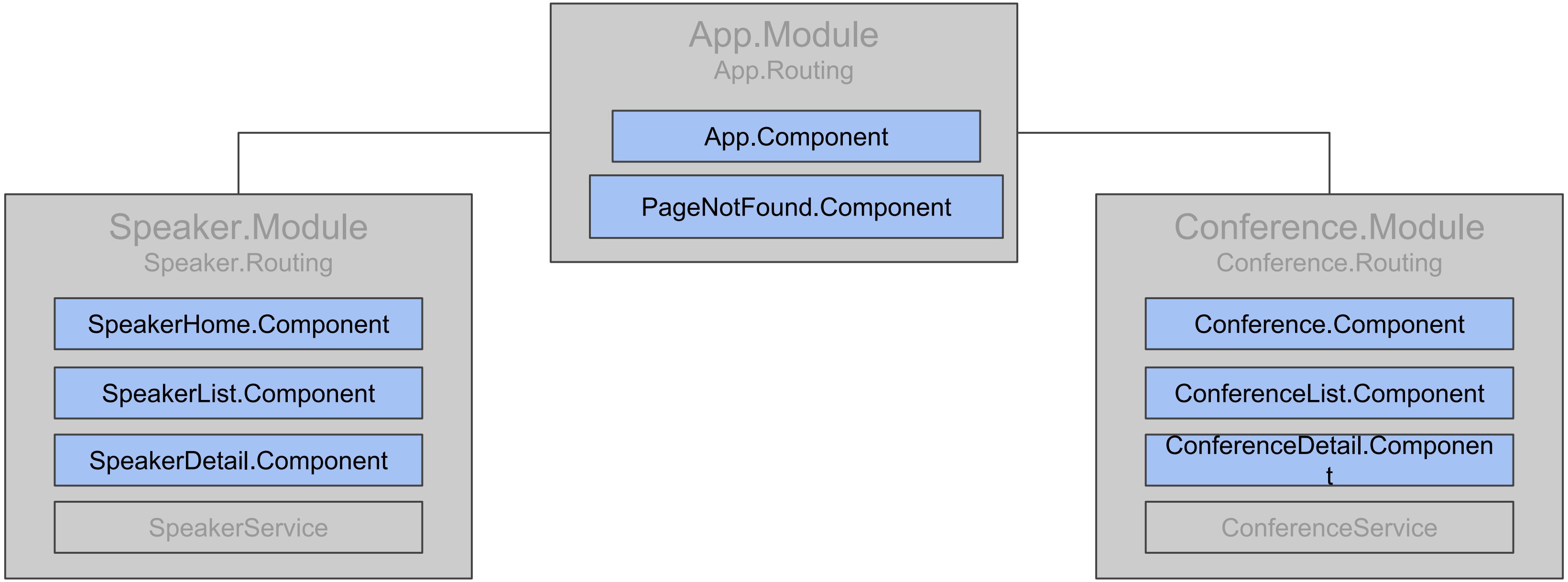
Angular: Component Lifecycle - contd.

- **ngAfterContentInit** - After Angular projects external content into the component view
- **ngAfterContentChecked** - After Angular checks the bindings of the external content that it projected into the component view
- **ngAfterViewInit** - After Angular creates the component’s view(s)
- **ngAfterViewChecked** - After Angular checks the bindings of the component’s view(s)

Angular: Component Lifecycle - contd.

- Component Lifecycle Sequence
  - **ngOnChanges** - before ngOnInit and when a data-bound input property value changes
  - **ngOnInit** - after the first ngOnChanges
  - **ngDoCheck** - during every Angular change detection cycle
  - **ngAfterContentInit** - after projecting content into the component
  - **ngAfterContentChecked** - after every check of projected component content
  - **ngAfterViewInit** - after initializing the component’s views and child views
  - **ngAfterViewChecked** - after every check of the component’s views and child views
  - **ngOnDestroy** - just before Angular destroys the component

Speaker Register Application Architecture

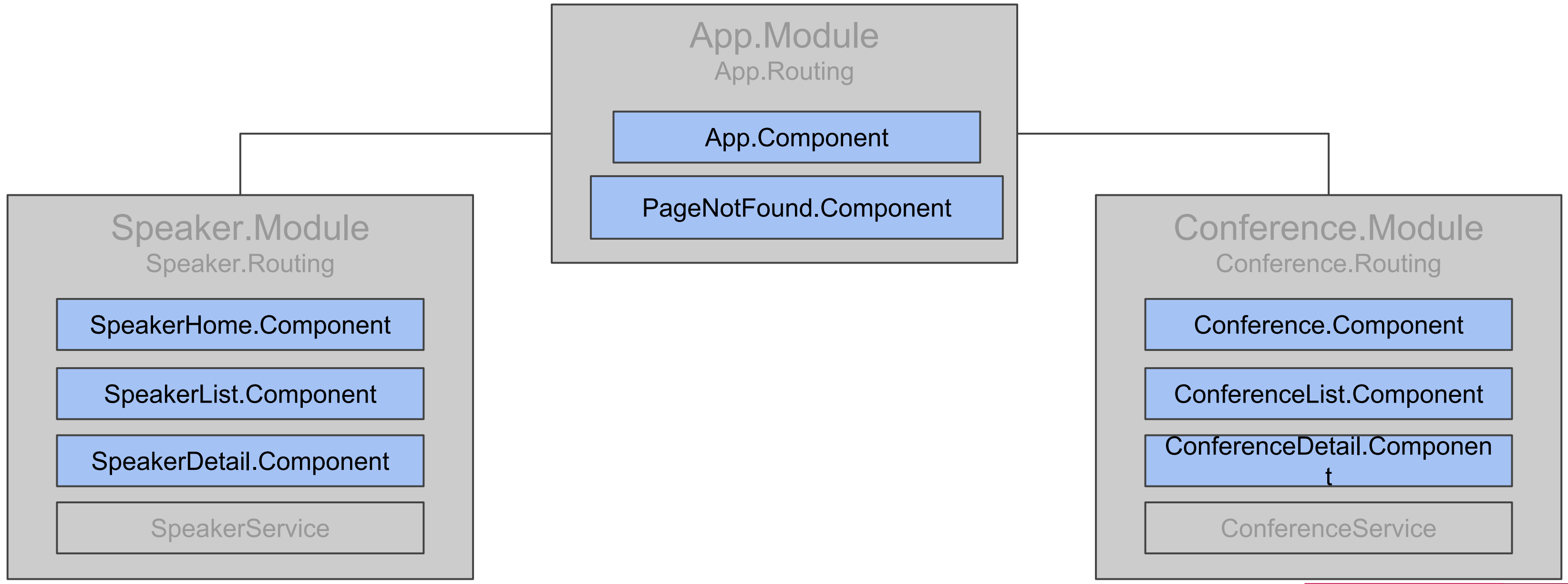


Angular: The Template

Angular: The Template

- A component defines its associated view via a template
- A template contains HTML
- A template can also contain additional markup that is a part of Angular's template syntax
- What the application user sees and can do is managed via components and their associated template
- Templates can contain custom tags that represent other components
- As a result, very complex component trees can be created to support rich user experiences

Speaker Register Application Architecture



Angular: Metadata

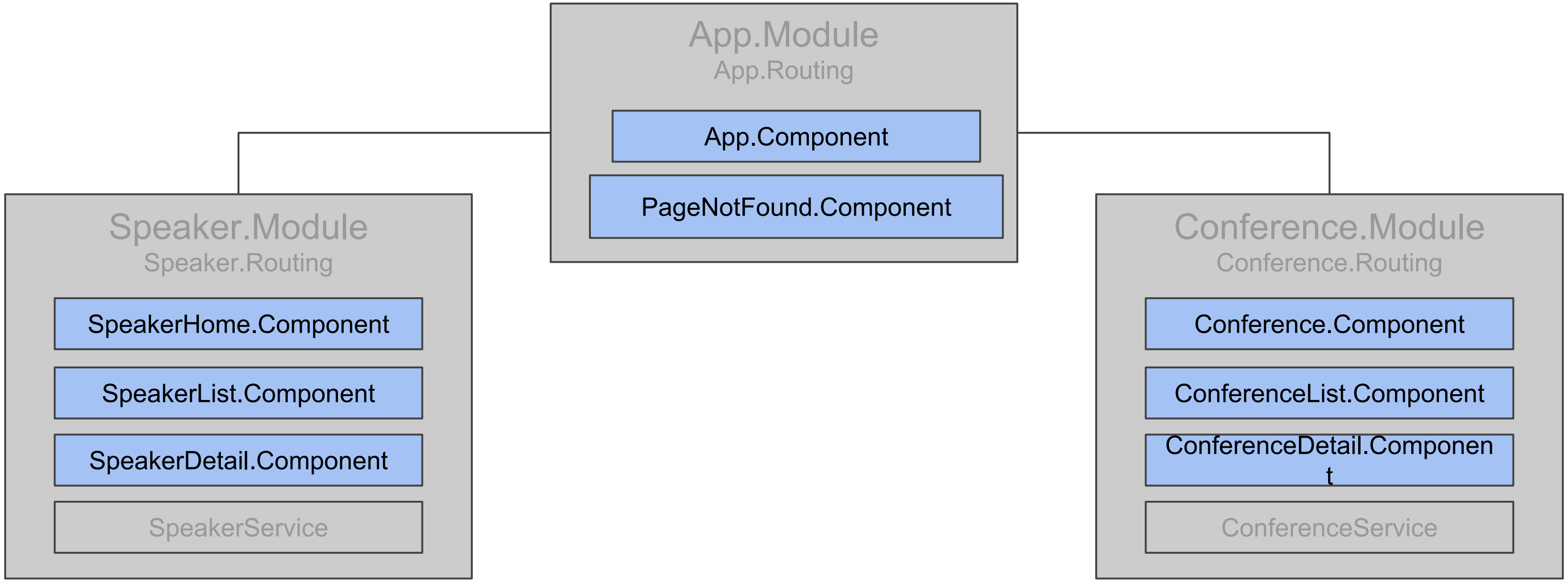
Angular: Metadata

- Metadata is used to tell Angular a class is a module or component
- Think of metadata as “configuration” for a class
- In Typescript, we attach metadata by using a decorator
- A decorator is a function that often has configuration parameters

Angular: Metadata

- @Component - a decorator that identifies the exported class as a component class
- It takes a configuration object that provides information to Angular so that the component can be displayed properly
- Selector - a css selector that tells Angular to create and insert an instance of this component where it finds the matching HTML tag
- Template URL - the template for the view associated with the component
- Directives - an array of components or directives this component requires
- Providers - an array of dependency injection providers for services that is component requires

Speaker Register Application Architecture



Angular: Data Binding

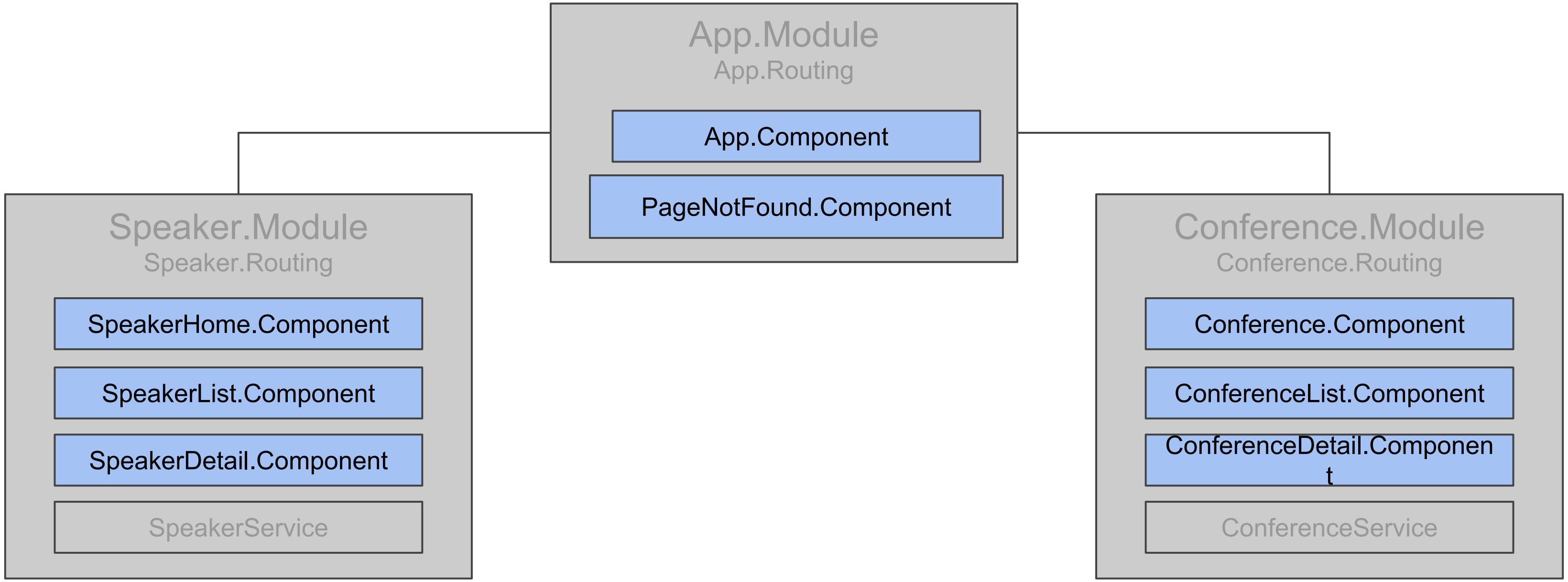
Angular: Data Binding

- Angular uses data binding to coordinate the template with the component
- Data binding markup in the template tells Angular how to make the connections
- There are four forms of data binding syntax
  - Interpolation
  - Property Binding
  - Event Binding
  - Two-way Data Binding

Angular: Data Binding

- Interpolation - displays the components property value
- Property Binding - used to set a property of a view element to the value of a component - one-way: component property -> view element
- Event Binding - calls a method of the component, in this case responding to a click event - one-way: view element -> component property (i.e. event handler)
- Two-way data binding - data flow between the input control and the component property

Speaker Register Application Architecture

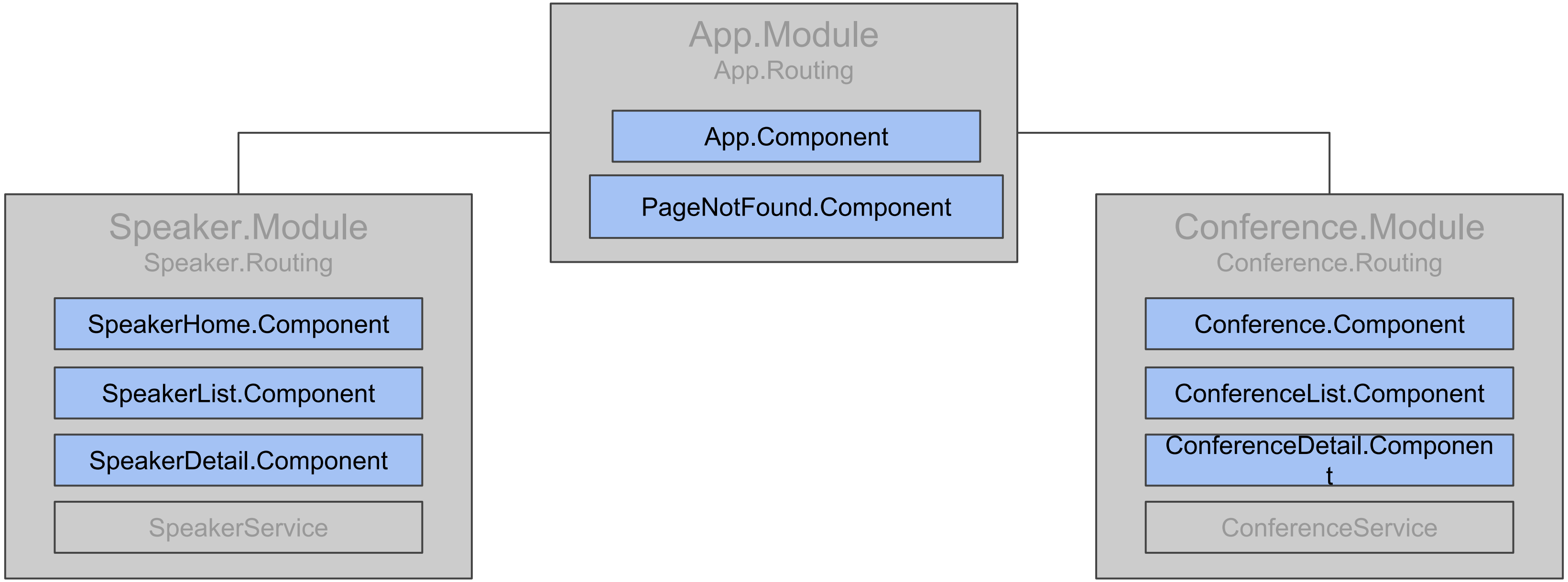


Angular: The Directive

Angular: The Directive

- A directive is a class with directive metadata (as opposed to component metadata)
- We use the @Directive decorator to attach the metadata to the class
- Technically a component is a directive - a directive with a template
- A structural directive alters layout by adding, removing, and replacing elements in the DOM (*ngFor, *ngIf)
- An attribute directive alters the appearance or behavior of an existing element (ngStyle, ngClass)
- Custom directives can be created - i.e. our components are a type of directive

Speaker Register Application Architecture

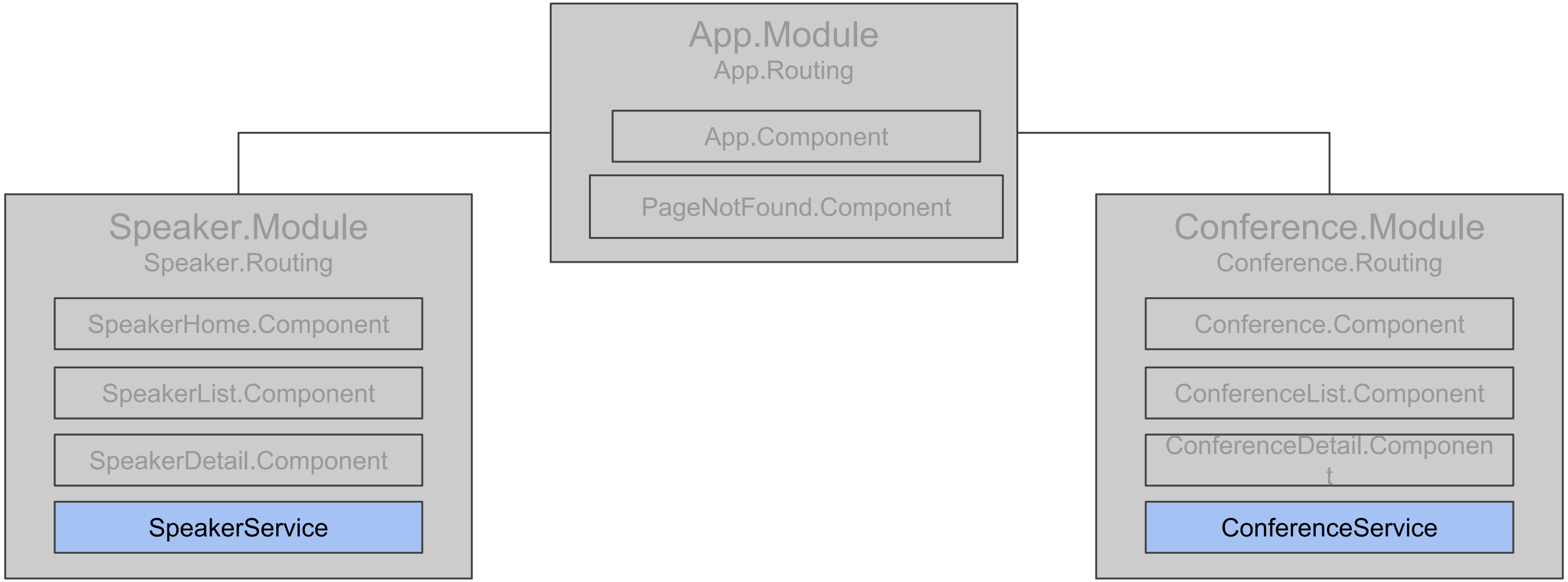


Angular: The Service

Angular: The Service

- A service represents any value, function, or feature that an application needs
- Almost anything can be a service
  - Logging
  - Data
  - Message bus
  - Configuration
- Angular has no formal definition of a service

Speaker Register Application Architecture

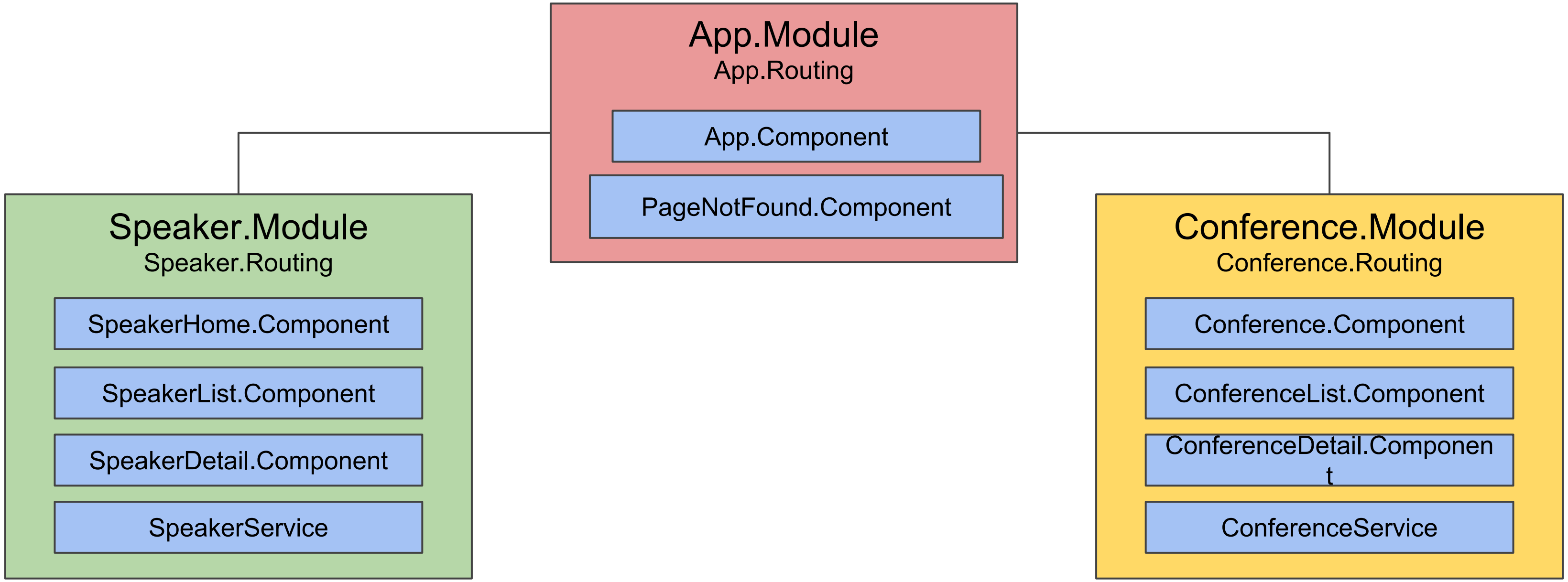


Angular: Dependency Injection

Angular: Dependency Injection

- Dependency injection is a way to supply a new instance of a class to components that require the class
- Typically components request classes they depend on via its constructor
- Angular locates the dependent class/classes, instantiates them, and provides them to the dependent class
- Dependency injection is a key part of the Angular framework and used just about everywhere

Speaker Register Application Architecture



Angular: Other Important Features and Services

Angular: Other Important Features and Services

● Animations	● Bootstrap (launching the root application component)
● Change Detection	● Component Router
● Events	● Forms
● HTTP	● Pipes
● Testing	

Angular: Resources

- Source Code: https://github.com/rightincode/speakerregister
- Official Site: https://angular.io/
- Plurasight: http://www.pluralsight.com
  - John Papa - Angular 2: First Look
  - Deborah Kurata - Angular 2: Getting Started
  - Joe Eames - Angular 2: Preparing for and Migrating Applications to Angular 2
- Typescript: https://www.typescriptlang.org/
- Angular Style Guide: https://angular.io/styleguide

Questions???