Components, Modules, Templates and Directives

Introduction to Angular 6

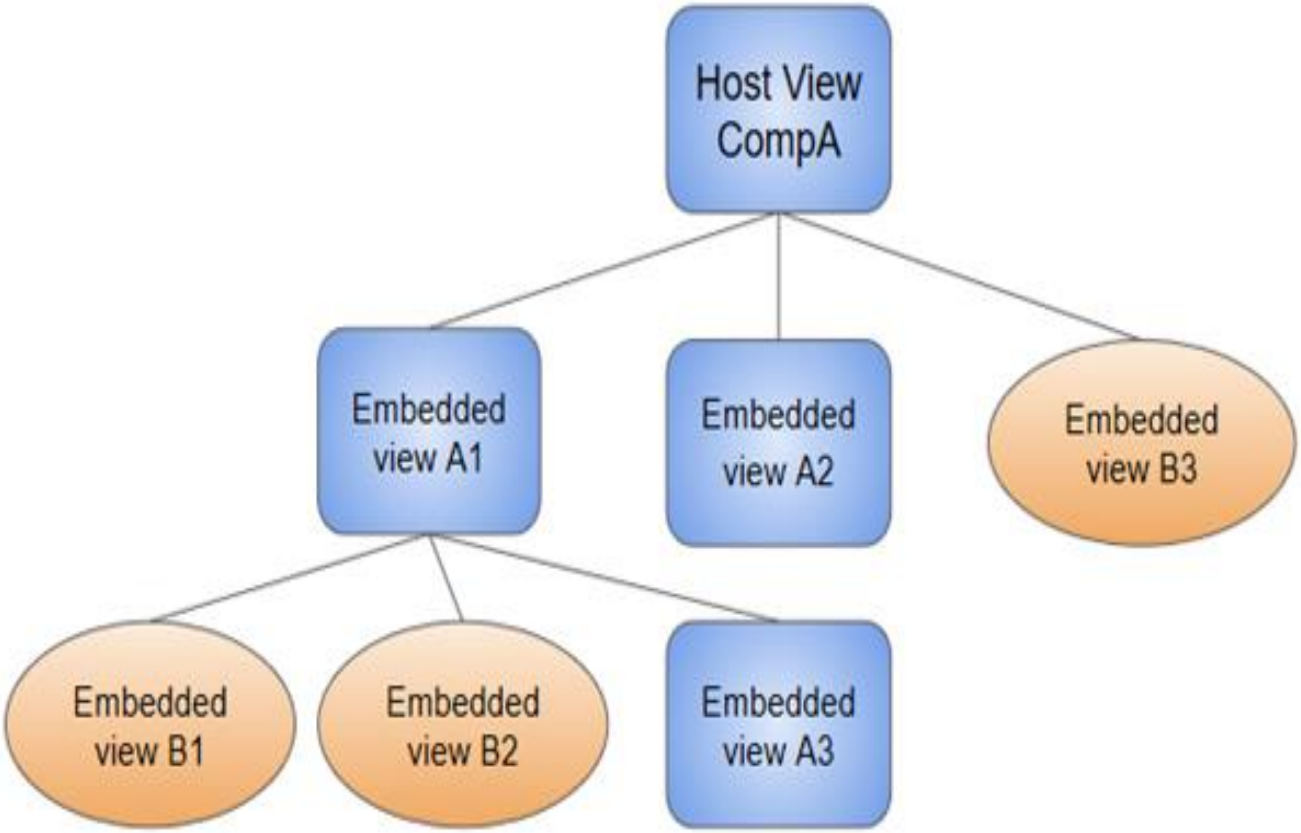
Component overview

- A component defines a class that contains application data and logic
  - is also associated with a template
  - Together, a component and template control a portion of the screen called a view.
- The component class interacts with the view through an API of properties and methods.
- Every Angular application has at least one component, the root component
  - This connects a component hierarchy with the page document object model (DOM)

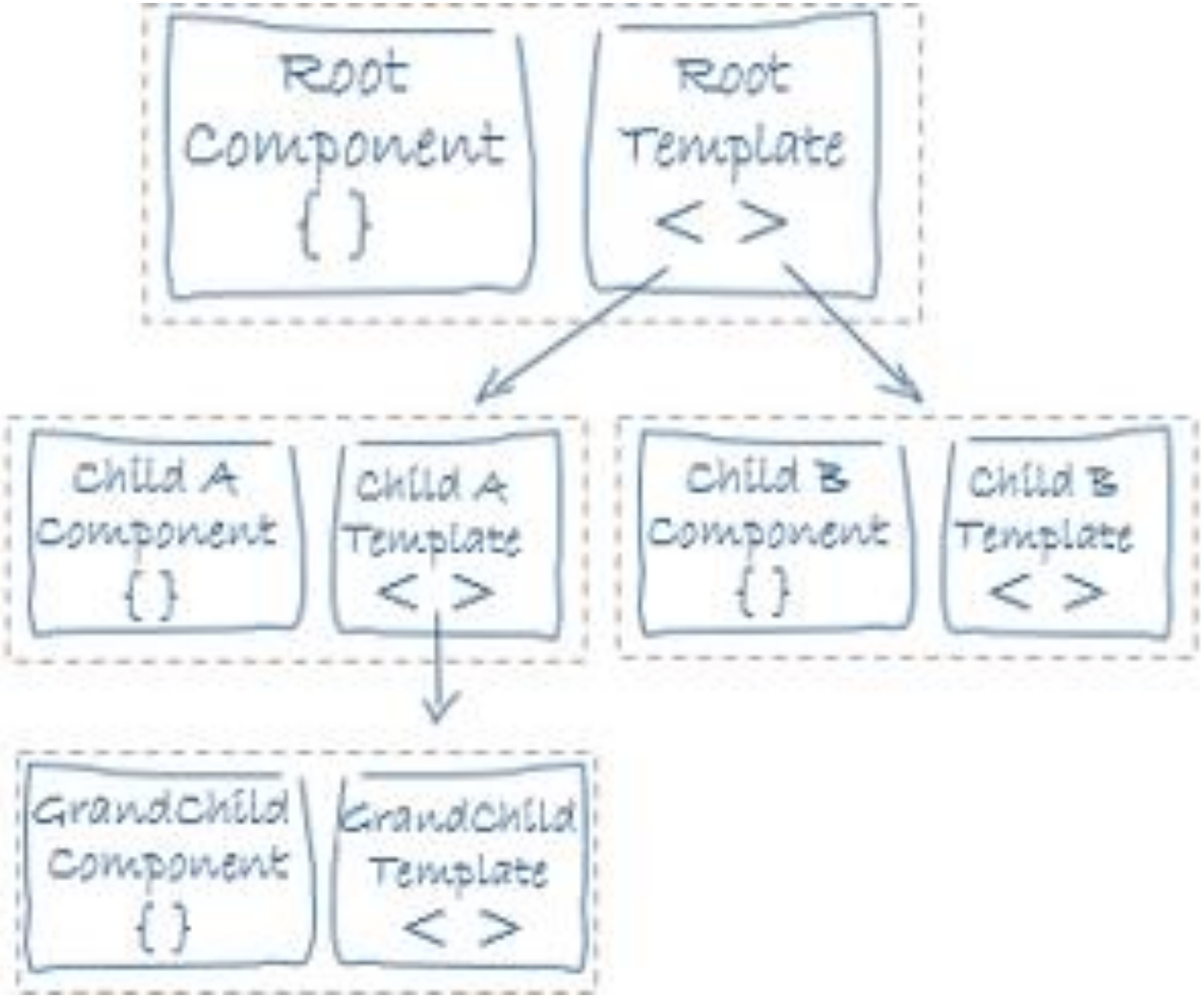
Component hierarchy

- A component and its template is associated with a single view
  - called the host view
- The host view can be the root of a view hierarchy
  - This contains embedded views which are in turn the host views of other components.
- This hierarchy allows you to define complex areas of the screen
  - that can be created, modified, and destroyed as a unit
- A view hierarchy can include views from components in the same Module
  - but it also can include views from components that are defined in different Modules

View hierarchy



Component hierarchy



Component metadata

- The @Component() decorator identifies the class immediately below it as a component
- Some useful @Component configuration options:
- selector:
  - A CSS selector that tells Angular to create and insert an instance of this component wherever it finds the corresponding tag in template HTML.
- templateUrl:
  - The module-relative address of this component's HTML template.
  - you can provide the HTML template inline, as the value of the template property.
  - This template defines the component's *host view*.
- providers:
  - An array of providers for services that the component requires.

Module Overview

- Angular apps are modular.
- Modules are containers for a cohesive block of code dedicated to an application domain.
  - They can contain components and service providers
  - They can import functionality that is exported from other Modules
  - Also export selected functionality for use by other Modules
- Every app has at least one Module class
  - the root module named AppModule
  - resides in a file named app.module.ts
- Launch your app by bootstrapping the root module.

Module Overview

- A small application might have only one Module
  - most apps have many more feature modules
- The root Module can include child Modules in a hierarchy of any depth.
- Organizing your code into distinct functional modules
  - helps in managing development of complex applications and designing for reusability
  - take advantage of lazy-loading to minimize the amount of code that needs to be loaded at startup

Module metadata

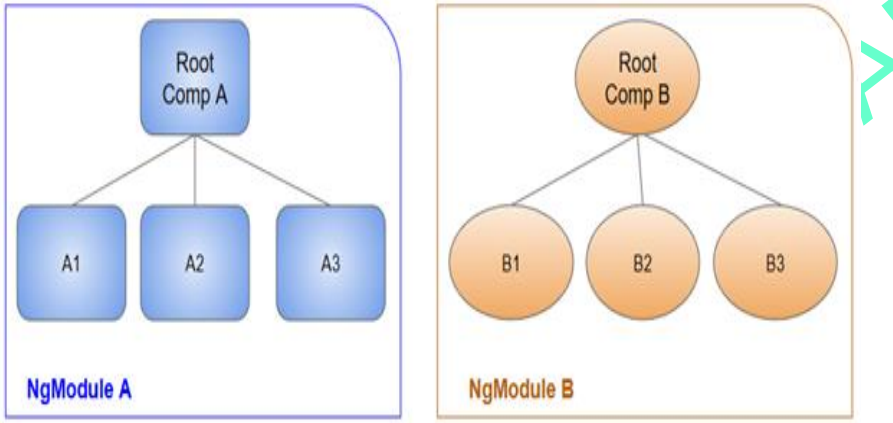
- An NgModule is defined by a class decorated with @NgModule().
  - The @NgModule() decorator is a function that takes a single metadata object, whose properties describe the module.
- declarations:
  - The components, directives, and pipes that belong to this NgModule.
  - For any component to be used within the context of a module, it has to be imported into your module file and declared here.
  - This ensures that the component is visible to all other components within the module.
  - The Angular CLI will automatically add your component or directive to the module when you create a component through it.

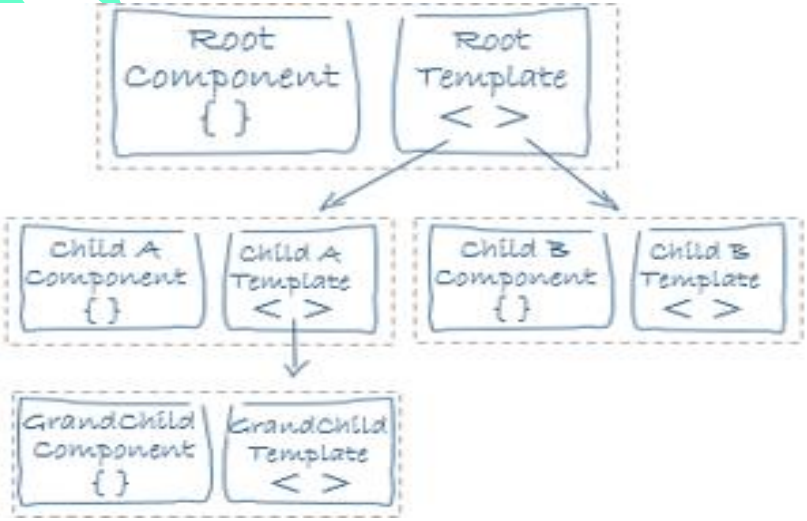
Module metadata

- exports:
  - The subset of declarations that should be visible and usable in the component templates of other NgModules.
- imports:
  - Other modules whose exported classes and services are needed by component templates declared in this NgModule.
  - You must export a component to allow it to be accessed outside of the module where the component is declared
- providers:
  - Creators of services that this NgModule contributes to the global collection of services
- bootstrap:
  - The main application view, called the root component, which hosts all other app views.
  - Only the root NgModule should set the bootstrap property

Modules and components

- A root NgModule always has a root component that is created during bootstrap
  - can include any number of additional components
- All of these share a compilation context.





NgModules and JavaScript modules

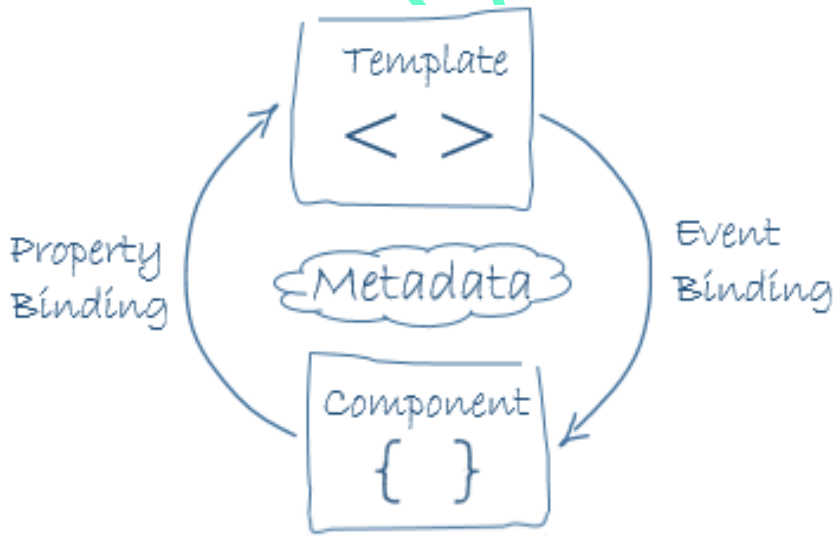
- The NgModule system is different from and unrelated to the ES2015 module system
  - These are complementary module systems that you can use together to write your apps
- Angular loads as a collection of JavaScript library modules.
  - Angular library name begins with the @angular prefix
- You import NgModules from Angular libraries using JavaScript import statements

Template overview

- A template combines standard HTML with Angular markup based on template syntax
  - This alters the HTML based on your app's logic and state as well as the DOM of the HTML
  - Together, a component and template control a portion of the screen called a view.
- Template syntax involves:
  - Data binding to connect your application data and the DOM
  - Pipes to transform data before it is displayed
  - Directives to apply app logic to what gets displayed
- Angular evaluates the directives
  - resolves the binding syntax in the template to modify the HTML elements and the DOM

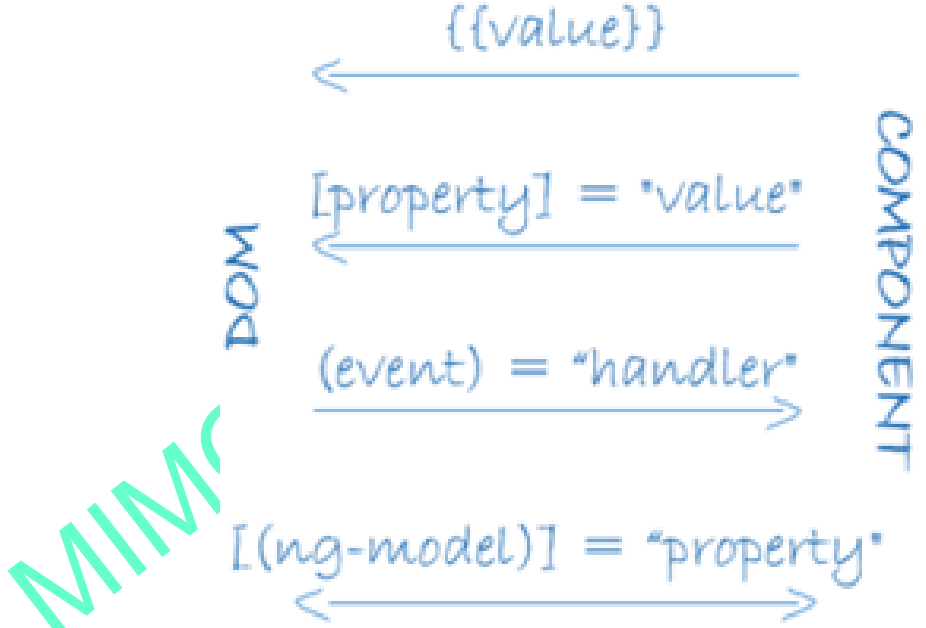
Data binding

- There are two types of data binding:
- Event binding
  - lets your app respond to user input in the target environment by updating your application data.
- Property binding
  - lets you interpolate values that are computed from your application data into the HTML.



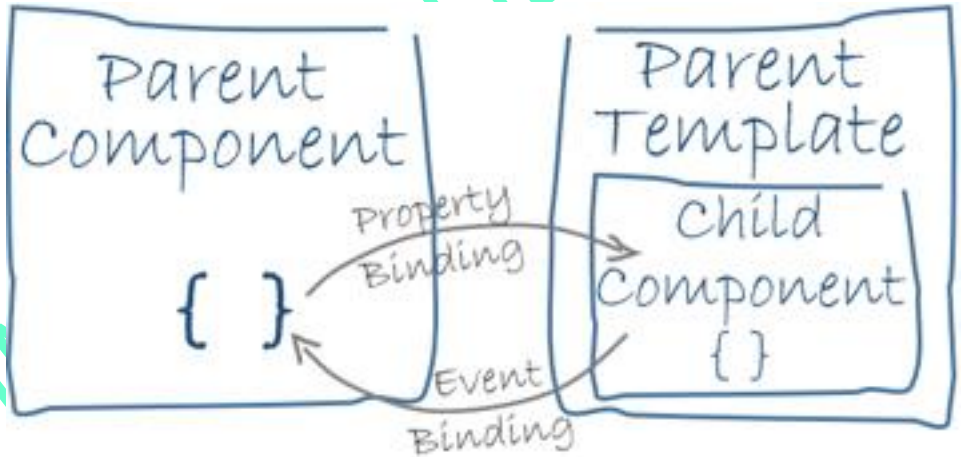
Two way data binding

- Angular supports two-way data binding
  - Changes in the DOM properties are also reflected in program data and vice versa
- Two-way data binding is used mainly in template-driven forms
  - Combines property and event binding in a single notation



Binding across component hierarchy

- Data binding is also important for communication between parent and child components.
- Angular processes all data bindings once for each JavaScript event cycle
  - from the root of the application component tree through all child components.



Pipes

- Angular pipes let you declare display-value transformations in your template HTML
  - predefined pipes available for common transformations such as the date and currency pipe.
- You can chain pipes
  - sending the output of one pipe function to be transformed by another pipe function
- A pipe can also take arguments that control how it performs its transformation.
- A class with the @Pipe decorator
  - defines a function that transforms input values to output values for display in a view

Directives

- A directive allows you to attach some custom functionality to elements in your HTML
  - It provides both functionality and UI logic
  - When Angular renders them, it transforms the DOM according to the instructions given by directives.
- A directive is a class with a @Directive() decorator
  - The metadata for a directive associates the decorated class with a selector element that you use to insert it into HTML.
- In templates, directives typically appear within an element tag as attributes
  - either by name or as the target of an assignment or a binding.

Directive types

- Structural directives
  - alter DOM layout by adding, removing, and replacing elements in the DOM.
- Attribute directives
  - alter the appearance or behavior of an existing element
  - In templates they look like regular HTML attributes.
- Angular provide predefined directives of both kinds
  - you can define your own using the @Directive() decorator

Overall architecture