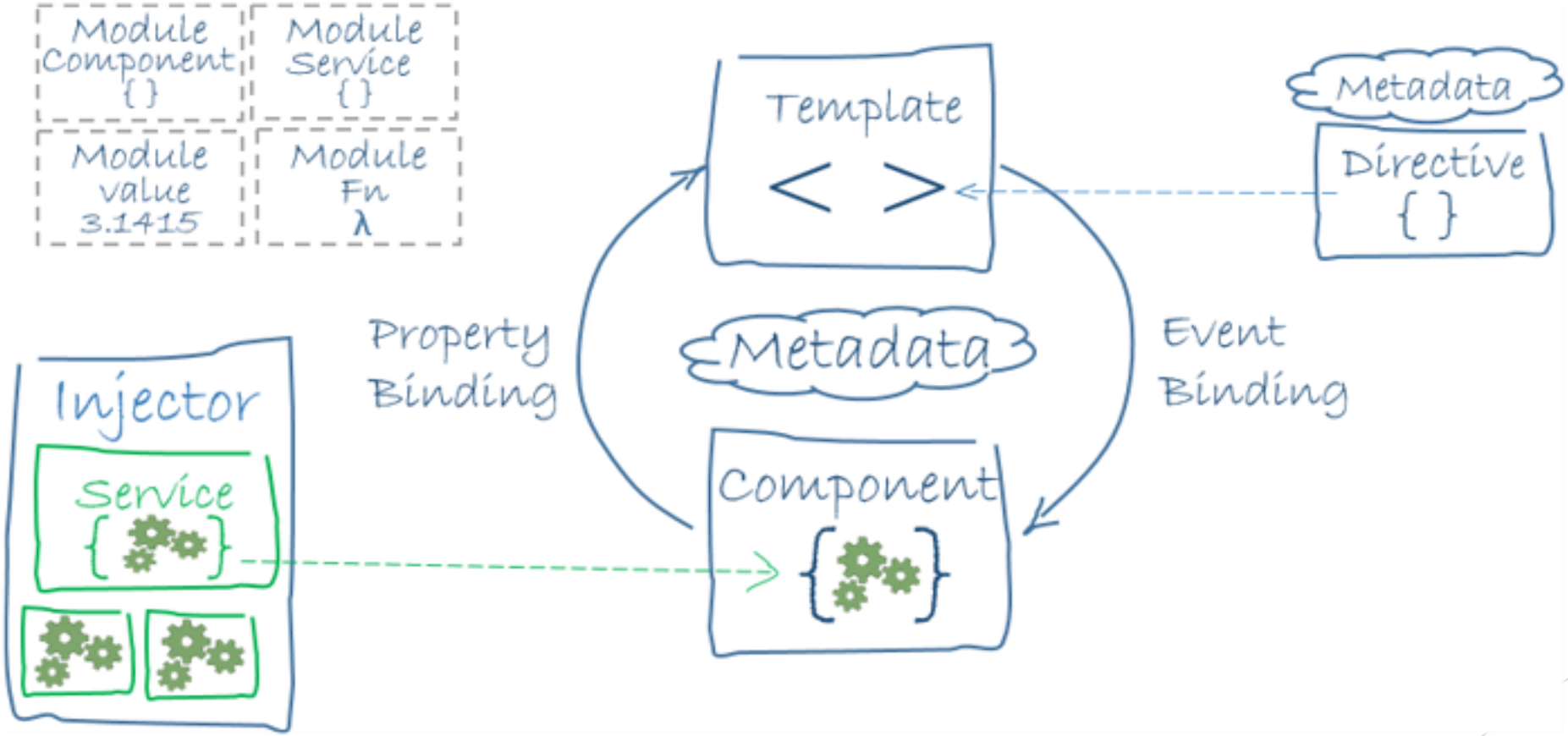
Angular js

By: Vikash Verma

Building blocks of angular

- Components
- Modules
- Templates, directives, and data binding
- Services and dependency injection
- Routing

Architecture



Architecture

- Angular is a platform and framework for building client applications in HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.
- Together, a component and template define an Angular view.
  - A decorator on a component class adds the metadata, including a pointer to the associated template.
  - Directives and binding markup in a component's template modify views based on program data and logic.
- The dependency injector provides services to a component, such as the router service that lets you define navigation among views.

Modules

- Angular NgModules differ from and complement JavaScript (ES2015) modules.
- An NgModule declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities.
- An NgModule can associate its components with related code, such as services, to form functional units.
- Every Angular app has a root module, conventionally named AppModule, and resides in a file named app.module.ts which provides the bootstrap mechanism that launches the application.
- Like JavaScript modules, NgModules can import functionality from other NgModules, and allow their own functionality to be exported and used by other NgModules. For example, to use the router service in your app, you import the Router NgModule.

NgModule 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.
- The most important properties are as follows.
  - **declarations**: The components, directives, and pipes that belong to this NgModule.
  - **exports**: The subset of declarations that should be visible and usable in the component templates of other NgModules.
  - **imports**: Other modules whose exported classes are needed by component templates declared in this NgModule.
  - **providers**: Creators of services that this NgModule contributes to the global collection of services; they become accessible in all parts of the app. (You can also specify providers at the component level, which is often preferred.)
  - **bootstrap**: The main application view, called the root component, which hosts all other app views. Only the root NgModule should set the bootstrap property.

Feature modules vs. root modules

- A feature module is an organizational best practice, as opposed to a concept of the core Angular API.
- A feature module delivers a cohesive set of functionality focused on a specific application need such as a user workflow, routing, or forms. While you can do everything within the root module, feature modules help you partition the app into focused areas.
- A feature module collaborates with the root module and with other modules through the services it provides and the components, directives, and pipes that it shares.

Components

- Every Angular application has at least one component, the root component that connects a component hierarchy with the page document object model (DOM).
- Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.
- The @Component **decorator** identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

Component metadata

- The @Component decorator identifies the class immediately below it as a component class, and specifies its metadata.
- The metadata for a component tells Angular where to get the major building blocks that it needs to create and present the component and its view.
- @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. Alternatively, 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.

Decorators

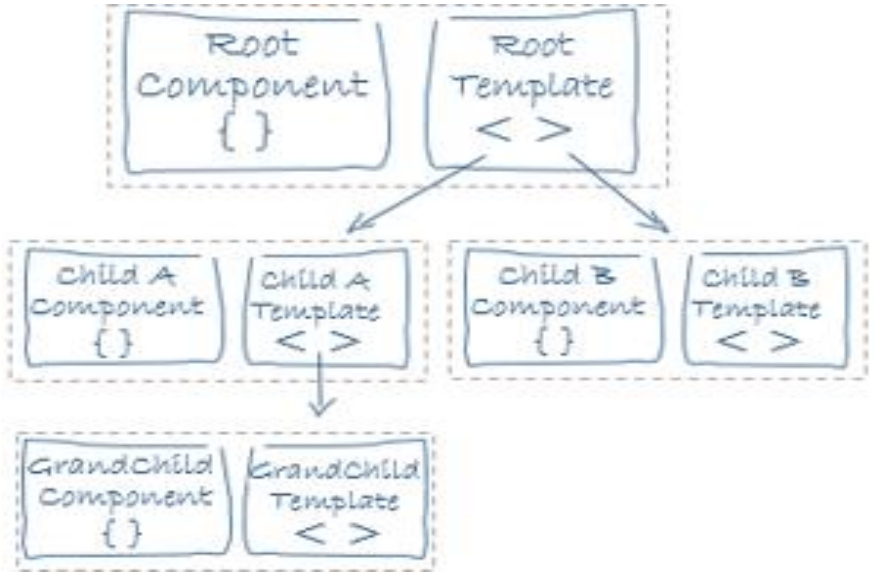
- Decorators are functions that modify JavaScript classes.
- Angular defines a number of decorators that attach specific kinds of metadata to classes, so that the system knows what those classes mean and how they should work.

Templates, directives, and data binding

- A template combines HTML with Angular markup that can modify HTML elements before they are displayed.
- Template directives provide program logic, and binding markup connects your application data and the DOM.
- 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.

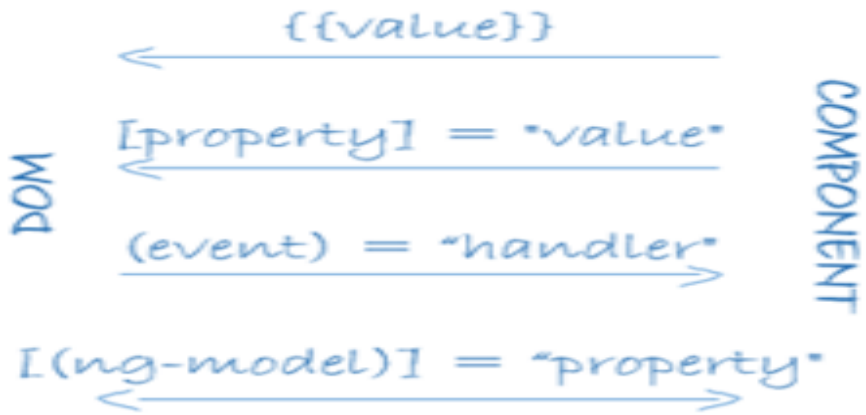
Templates and views

- You define a component's view with its companion template. A template is a form of HTML that tells Angular how to render the component.
- Views are typically arranged hierarchically, allowing you to modify or show and hide entire UI sections or pages as a unit
- The template immediately associated with a component defines that component's host view.



Data binding

- Without a framework, you would be responsible for pushing data values into the HTML controls and turning user responses into actions and value updates.
- Writing such push and pull logic by hand is tedious, error-prone.
- **Angular supports two-way data binding**, a mechanism for coordinating the parts of a template with the parts of a component. Add binding markup to the template HTML to tell Angular how to connect both sides.



Binding types

- Data from model to view use ( )
- View to Model use [ ]
- For two way binding use [( )]

Pipes

- Angular pipes let you declare display-value transformations in your template HTML.
- A class with the @Pipe decorator defines a function that transforms input values to output values for display in a view.
- Syntax: {{interpolated_value | pipe_name}}
- Example Today is {{today | date}}

Directives

- Angular templates are dynamic. When Angular renders them, it transforms the DOM according to the instructions given by directives.
- A directive is a class with a @Directive() decorator
- There are two other kinds of directives:
  - structural
  - attribute

Structural directives

- Structural directives alter layout by adding, removing, and replacing elements in the DOM.
- For Example
  - *ngFor is an iterative;
  - *ngIf is a conditional;

Attribute directives

- Attribute directives alter the appearance or behavior of an existing element.
- In templates they look like regular HTML attributes, hence the name.
- The ngModel directive, which implements two-way data binding, is an example of an attribute directive.
- ngModel modifies the behavior of an existing element (typically <input>) by setting its display value property and responding to change events.

Services and dependency injection

- For data or logic that isn't associated with a specific view, and that you want to share across components, you create a service class.
- A service class definition is immediately preceded by the @Injectable() decorator.
- The decorator provides the metadata that allows other providers to be injected as dependencies into your class.
- **Dependency injection (DI)** lets you keep your component classes lean and efficient.
  - They don't fetch data from the server, validate user input, or log directly to the console; they delegate such tasks to services.

Routing

- The Angular Router NgModule provides a service that lets you define a navigation path among the different application states and view hierarchies in your app.
- It is modeled on the familiar browser navigation conventions:
  - Enter a URL in the address bar and the browser navigates to a corresponding page.
  - Click links on the page and the browser navigates to a new page.
  - Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.
- The router maps URL-like paths to views instead of pages.
- When a user performs an action, such as clicking a link, that would load a new page in the browser, the router intercepts the browser's behavior, and shows or hides view hierarchies.