angular 2.0 building blocks

- Module
- Component
- Template
- Metadata
- Data Binding
- Service
- Directive
- Dependency Injection

1. Module

- Angular apps are modular and so in general we assemble our application from many modules. A typical module is a cohesive block of code dedicated to a single purpose.
- A module exports something of value in that code, typically one thing such as a class.
- Some modules are libraries of other modules.
- Angular itself ships as a collection of library modules called “barrels”. Each Angular library is actually a public facade over several logically related private modules.
- The angular2/core library is the primary Angular library module from which we get most of what we need.
- There are other important Angular library modules too such as angular2/common, angular2/router, and angular2/http.
- e.g import {Component} from ‘angular2/core’;
- The key take aways are:
    - Angular apps are composed of modules.
    - Modules export things — classes, function, values — that other modules import.
    - We prefer to write our application as a collection of modules, each module exporting one thing.
    - The first module we write will most likely export a component.

2. Component

- Perhaps the first module we meet is a module that exports a component class. The component is one of the basic Angular blocks, we write a lot of them
- Most applications have an AppComponent. By convention, we’ll find it in a file named

app.component.ts

- Components are the collection of templates, styles, selector configurations etc.
- Angular creates, updates, and destroys components as the user moves through the application.
- Each components is a typescript class which again includes variable, functions, prop declaration etc.

3. Template

- We define a Component's view with its companion template. A template is a form of HTML that tells Angular how to render the Component.
- A template looks like regular HTML much of the time … and then it gets a bit strange.
- Template will have all diff type databinding and html DOM information.

4. Metadata

- Metadata tells Angular how to process a class.
- e.g

```
@Component({
  selector:    'hero-list',
  templateUrl: 'app/hero-list.component.html',
  directives:  [HeroDetailComponent],
  providers:   [HeroService]
})
```

Here we see the @Component decorator which (no surprise) identifies the class immediately below it as a Component class.

selector - a css selector that tells Angular to create and insert an instance of this component where it finds a <hero-list> tag in parent HTML. If the template of the application shell (a Component) contained

templateUrl - the address of this component's template

directives - an array of the Components or Directives that this template requires.

providers - an array of dependency injection providers for services that the component requires. This is one way to tell Angular that our component's constructor requires a HeroService so it can get the list of heroes to display.

5. Data Binding

- There are four forms of data binding syntax.
- Each form has a direction - to the DOM, from the DOM, or in both directions.
- We can group all bindings into three categories by the direction in which data flows. Each category has its distinctive syntax:
    - One Way (from component -> View) - Binds Property, Attributes, Class, Style.

```
{{expression}}
[target] = "expression"
bind-target = "expression"
```

    - One Way (from View -> Component) - Binds Events

```
(target) = "Statement"
on-target="statement"
```

    - Two way binding

```
[(target)] = "expression"
bindon-target ="expression"
```

For More Details - https://angular.io/docs/ts/latest/guide/template-syntax.html

6. Directive

- Our Angular templates are dynamic. When Angular renders them, it transforms the DOM according to the instructions given by a directive.
- A directive is a class with directive metadata. In TypeScript we'd apply the @Directive decorator to attach metadata to the class.
- While the component is technically a directive, it is so distinctive and central to Angular applications that we chose to separate the component from the directive in our architectural overview.

- There are two other kinds of directives as well that we call “structural” and “attribute” directives.
- Structural directives alter layout by adding, removing, and replacing elements in DOM.
    - e.g *ngFor , *ngIf etc.
- Attribute directives alter the appearance or behavior of an existing element. In templates they look like regular HTML attributes, hence the name.
    - e.g. [(ngModel)]

7. Service

- “Service” is a broad category encompassing any value, function or feature that our application needs.
- Almost anything can be a service. A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.
    - Examples include:
        - logging service
        - data service
        - message bus
        - tax calculator
        - application configuration
- Most common usage of services are to bind components with Databases or any similar part which delivers data to components and that way, components will be independent from Data Layer.

8. DI (Dependency Injection)

- “Dependency Injection” is a way to supply a new instance of a class with the fully-formed dependencies it requires. Most dependencies are services. Angular uses dependency injection to provide new components with the services they need.
- dependency injection is wired into the framework and used everywhere.
- the Injector is the main mechanism.
    - an injector maintains a container of service instances that it created.
    - an injector can create a new service instance using a provider.
- a provider is a recipe for creating a service.
- we register providers with injectors.

9. Other Stuff

- Animation: A forthcoming animation library makes it easy for developers to animate component behavior without deep knowledge of animation techniques or css.
- Bootstrapping: A method to configure and launch the root application component.
- Change Detection: Learn how Angular decides that a component property value has changed and when to update the screen
- Zones: Change Detection uses zones to intercept asynchronous activity and run its change detection strategies.
- Events: The DOM raises events. So can components and services.
- Router: With the Component Router service, users can navigate a multi-screen application in a familiar web browsing style using URLs.
- Forms: Support complex data entry scenarios with HTML-based validation and dirty checking.
- Http: Communicate with a server to get data, save data, and invoke server-side actions with this Angular HTTP client.
- Lifecycle Hooks: We can tap into key moments in the lifetime of a component, from its creation to its destruction, by implementing the “Lifecycle Hook” interfaces.
- Pipes: Services that transform values for display. We can put pipes in our templates to improve the user experience.

  ```
  price | currency:'USD':true
  ```

- Testing: Angular provides a testing library for “unit testing” our application parts as they interact with the Angular framework.

Author

Bhavin Patel