Overview of Angular

BUILDING WEB APPLICATIONS USING ANGULAR

Contents

- The eight building blocks of angular
- Bringing them together
- Observables

Core Components

- Angular has eight building blocks we will discuss
  - Modules
  - Components
  - Templates
  - Metadata
  - Data binding
  - Directives
  - Services
  - Dependency injection

Modules (Angular modules / NgModules)

- Angular is modular, beginning with the root module, often called AppModule
- Most applications will have many feature based modules, encapsulating related code
- Takes form as a class with an @NgModule decorator which can describe:
  - Declarations: the view classes that this module declares (components, directives and pipes)
  - Exports: the declarations that are visible to other modules
  - Imports: modules whose exported classes are needed in this module
  - Providers: creators of services that this module adds to the global collection of services
  - Bootstrap: only set by the root module, this declares the main application view.
- These are not the same as JavaScript modules, which we use throughout Angular applications to complement Angular Modules.

Modules

```
@NgModule({
    declarations: [
        AppComponent
    ],
    imports: [
        BrowserModule,
        FormsModule,
        HttpModule
    ],
    providers: [],
    bootstrap: [AppComponent]
})

export class AppModule { }
```

Components

- Components control parts of the view
- A basic component will be made up of 3 building blocks all encapsulated within a single module
  - A class with a @Component decorator which contains the application logic
  - An HTML template
  - CSS styles
- The class interacts with the view through an API of properties and methods
- Angular creates, populates and destroys components as the user interacts with the application

Components

```
import { Component } from '@angular/core';

@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    title = 'app works!';
}
```

Templates

- Templates are used to tell Angular how to render the component
- They are written in normal HTML with some additions from Angular

```
<h1>
    {{title}}
</h1>
```

Metadata

- Did you notice how Modules and Components are just Classes?
- Metadata is how we tell Angular what this class is
- In TypeScript we attach this metadata through the use of decorators
- For both our Module and Component decorates we passed a required configuration object

```
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    title = 'app works!';
}
```

```
@NgModule({
    declarations: [
        AppComponent
    ],
    imports: [
        BrowserModule,
        FormsModule,
        HttpModule
    ],
    providers: [],
    bootstrap: [AppComponent]
})
export class AppModule { }
```

Data binding

- Data binding is the mechanism through which we link parts of a component with parts of the template
- Depending on the form we use, we can send data
  - from the component to the template
  - From the template to the component
  - In both directions!

```
<h1>
    {{title}}
</h1>
```

Directives

- Components are a type of Directive, a directive with a template
- The remaining two types of directives are structural and attribute
  - Structural directives alter layout through adding and/or removing DOM elements
  - Attribute directives alter layout or behaviour of existing DOM elements
- There are built-in directives but we can also write our own (like Components!)

Services

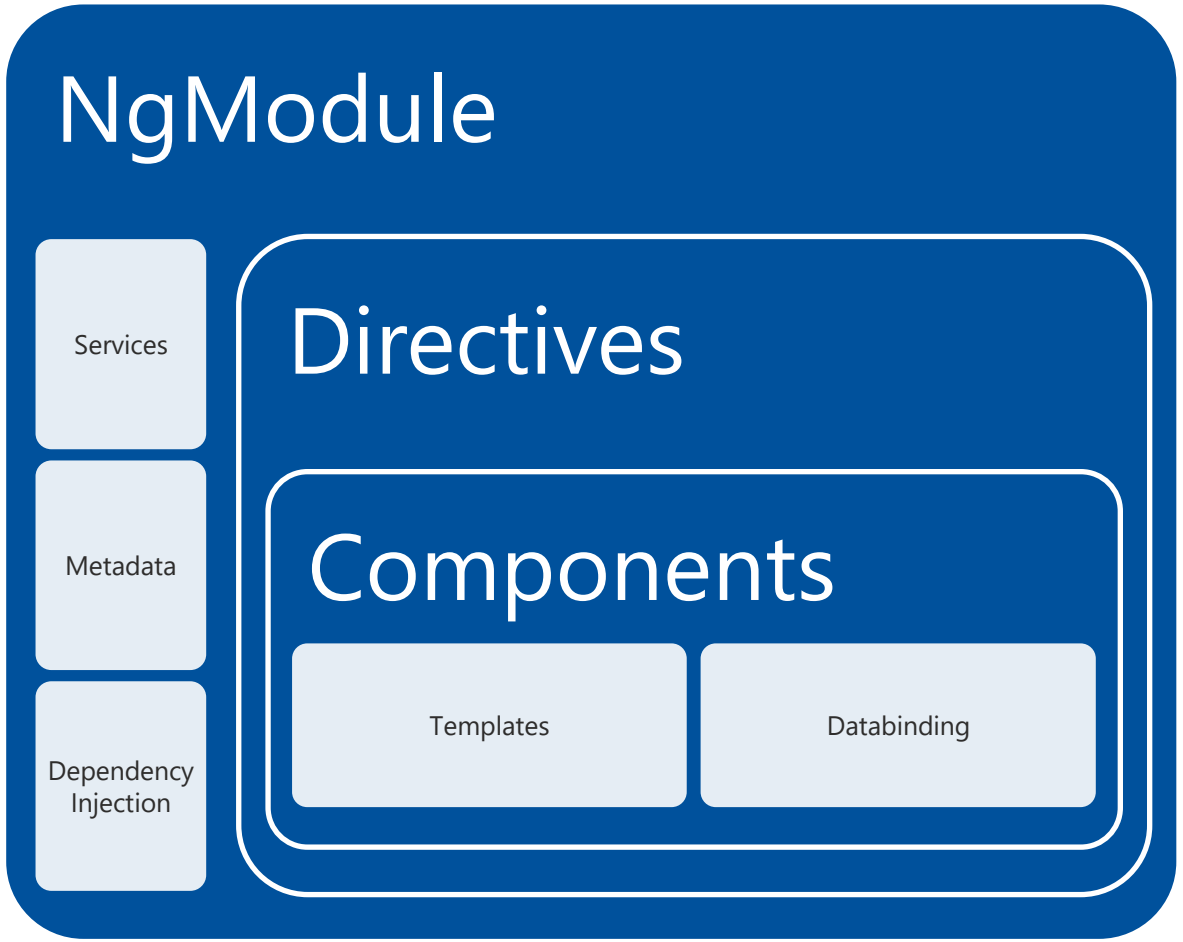
- Component's job is to bridge between the view and the application logic
- All non-trivial tasks should be carried out by services
- A service is not defined by Angular and Angular provides no place to 'register' services
- A service is *typically* a Class but could actually be anything

```
export class Logger {
    error(thing: any) {
       console.log(thing);
}
```

Dependency Injection

- Modules, components, directives, services – with this many building blocks floating around things will get confusing with dependencies upon dependencies
- Angular handles dependencies using Dependency Injection (DI)
- The Injector holds instances of services it has previously created, and should a requested service not be within then it creates a new instance
- Typically we are injecting services into components by passing them in as parameters to the component's constructor function
- We register providers with the injector, in this case the service class itself
- If you register them in a module they are available for that module – generally it's a good idea to register them with the root module so the same service instance is available everywhere
- If you register a provider with a component then a new instance is created with each new instance of the component

Bring them all together



Exercise

TOUR OF ANGULAR