Introduction to Single Page Applications (SPA) Development Using Angular 2 and TypeScript

Trayan Iliev

IPT – Intellectual Products & Technologies
e-mail: tiliev@iproduct.org
web: http://www.iproduct.org

Agenda

1. Creating Angular Hello World Application
2. Model-View-Controller (MVC), Model-View-Presenter (MVC), Model-View-ViewModel (MVVM) – MV* patterns
3. Web components
4. Data binding
5. Angular data architecture – Module, Component, Template, Metadata, Binding, Service, Directive, Dependency Injection
6. Component controllers, views & templates
8. Using external template and style files
9. Using Angular Command Line Interface (CLI)
10. Angular by example – TODO application

Where is The Code?

Angular and TypeScript Web App Development code is available @GitHub:

https://github.com/iproduct/course-angular

Angular Command Line Interface (CLI)
[https://github.com/angular/angular-cli]

npm install -g @angular/cli

ng new <*project-name*> --prefix <*custom-project-component-prefix*>
cd <*project-name*>
ng serve
ng serve --port 4201 --live-reload-port 49153

Create Angular 2 compponents using CLI:

Module	ng g module my-new-module [--routing]
Component	ng g component my-new-component
Directive	ng g directive my-new-directive
Pipe	ng g pipe my-new-pipe
Service	ng g service my-new-service --module <module-name>
Guard	ng g guard my-new-guard --module <module-name>

If you prefer *angular-cli* to use *yarn*, instead of *npm*:
ng set --global pacakgeManager=yarn

Angular Hello World Project Structure

- *node_modules ...*
- *src*
  - *app*
    - app.component.ts
    - app.module.ts
  - *assets ...*
  - *environments ...*
  - index.html
  - main.ts
  - polyfills.ts
  - styles.css
- .angular-cli.json
- package.json
- tsconfig.json

app/app.component.ts

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

@Component({
  selector: 'my-app',
  template: '<h2>My First Angular App</h2>'
})

export class AppComponent { }
```

app/app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }   from './app.component';
@NgModule({
  imports:      [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

app/main.ts

```
import { platformBrowserDynamic } from '@angular/platform-
browser-dynamic';
import { AppModule } from './app.module';

const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

index.html: Load App using SystemJS

```
<html>
<head>
    <title>Angular 2 Demo 01</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="app/assets/css/main.css">
    <script src="node_modules/core-js/client/shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/systemjs/dist/system.src.js"></script>
    <script src="systemjs.config.js"></script>
    <script>
      System.import('app').catch(function(err){ console.error(err); });
    </script>
</head>
<body>
    <my-app>Loading...</my-app>
</body>
</html>
```

index.html: Angular CLI (Webpack Template)

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Angular TODO Lab</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <my-app>Loading...</my-app>
</body>
</html>
```

MVC Comes in Different Flavors

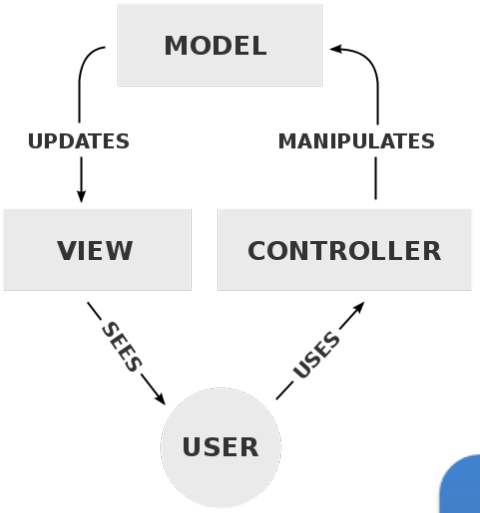
What is the difference between following patterns:

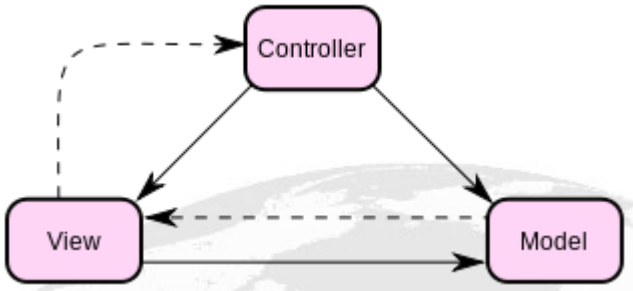
- Model-View-Controller (MVC)
- Model-View-ViewModel (MVVM)
- Model-View-Presenter (MVP)

http://csl.ensm-douai.fr/noury/uploads/20/ModelViewController.mp3

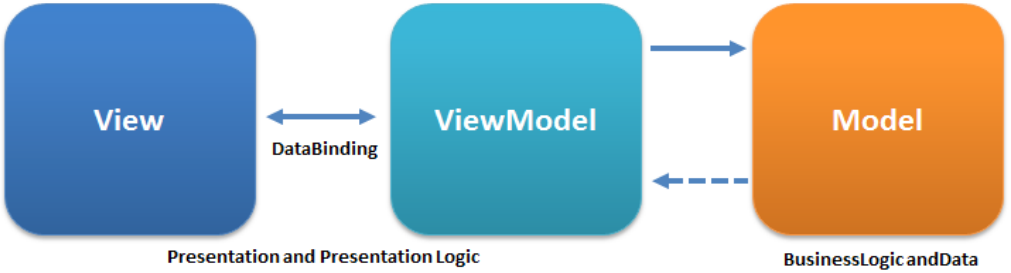
MVC Comes in Different Flavors - 2

- MVC

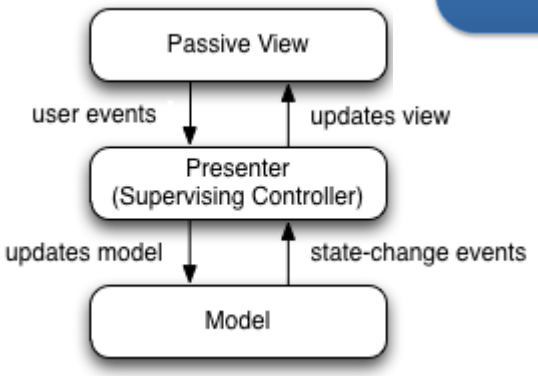




- MVVM



- MVP



Web Components I

- Do Components Exist?
  [http://www.c2.com/cgi/wiki?DoComponentsExist]

- *They have to exist. Sales and marketing people are talking about them.* Components are not a technology. Technology people seem to find this hard to understand. Components are about how customers want to relate to software. They want to be able to buy their software a piece at a time, and to be able to upgrade it just like they can upgrade their stereo. They want new pieces to work seamlessly with their old pieces, and to be able to upgrade on their own schedule, not the manufacturer's schedule. They want to be able to mix and match pieces from various manufacturers. This is a very reasonable requirement. It is just hard to satisfy.

  - RalphJohnson

Web Components II

- Make it possible to build widgets …which can be reused reliably …and which won't break pages if the next version of the component changes internal implementation details.

[http://www.html5rocks.com/en/tutorials/webcomponents/shadowdom/]

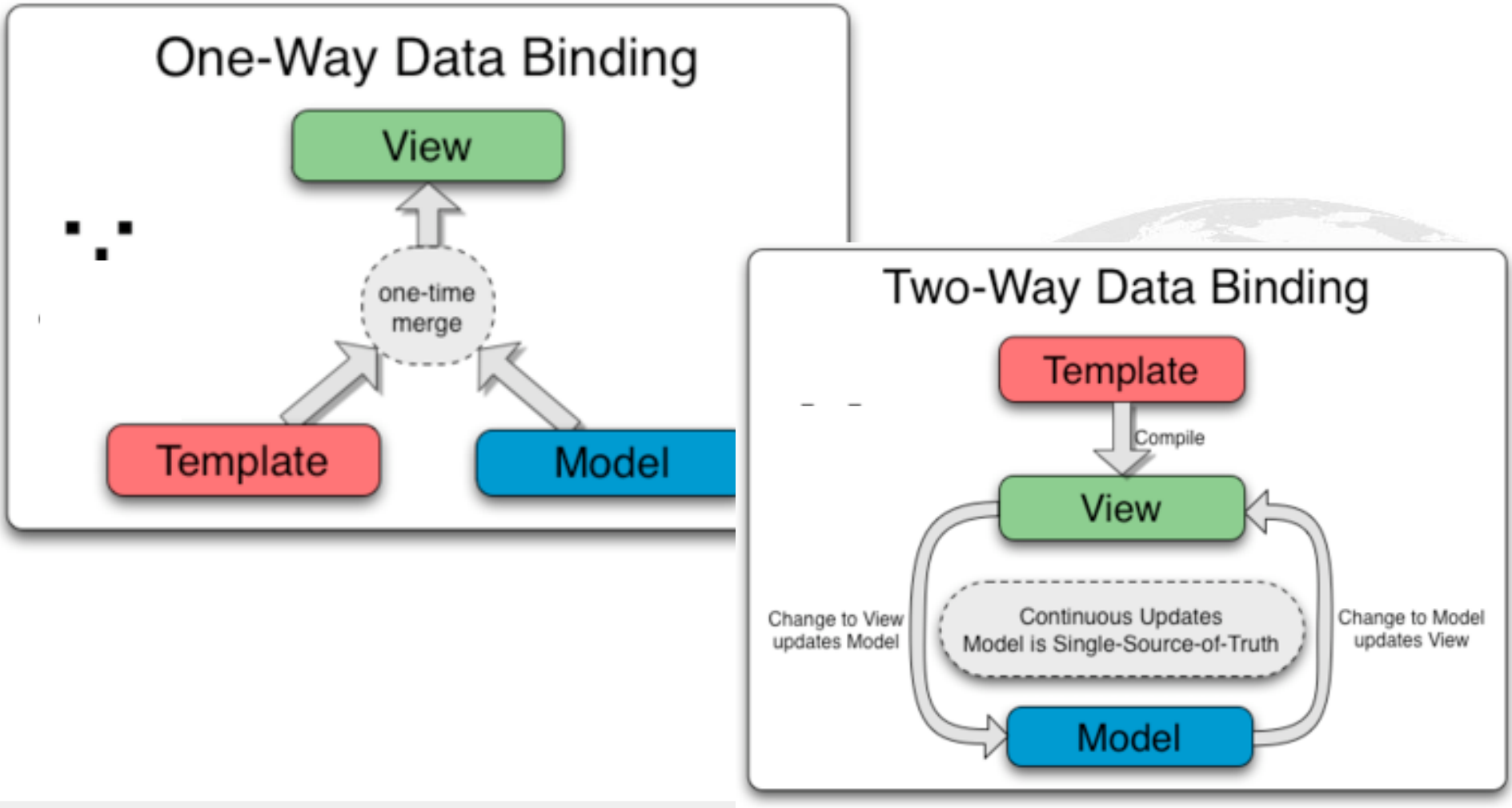
4 emerging W3C specifications:

- **Custom elements** – provide a way for authors to build their own fully-featured DOM elements.
- **Shadow DOM** – combining multiple DOM trees in one hierarchy
- **Template** – declare fragments of HTML that can be cloned and inserted in the document by script
- **HTML imports** – <link rel="import" href="my-custom-cmp.html">

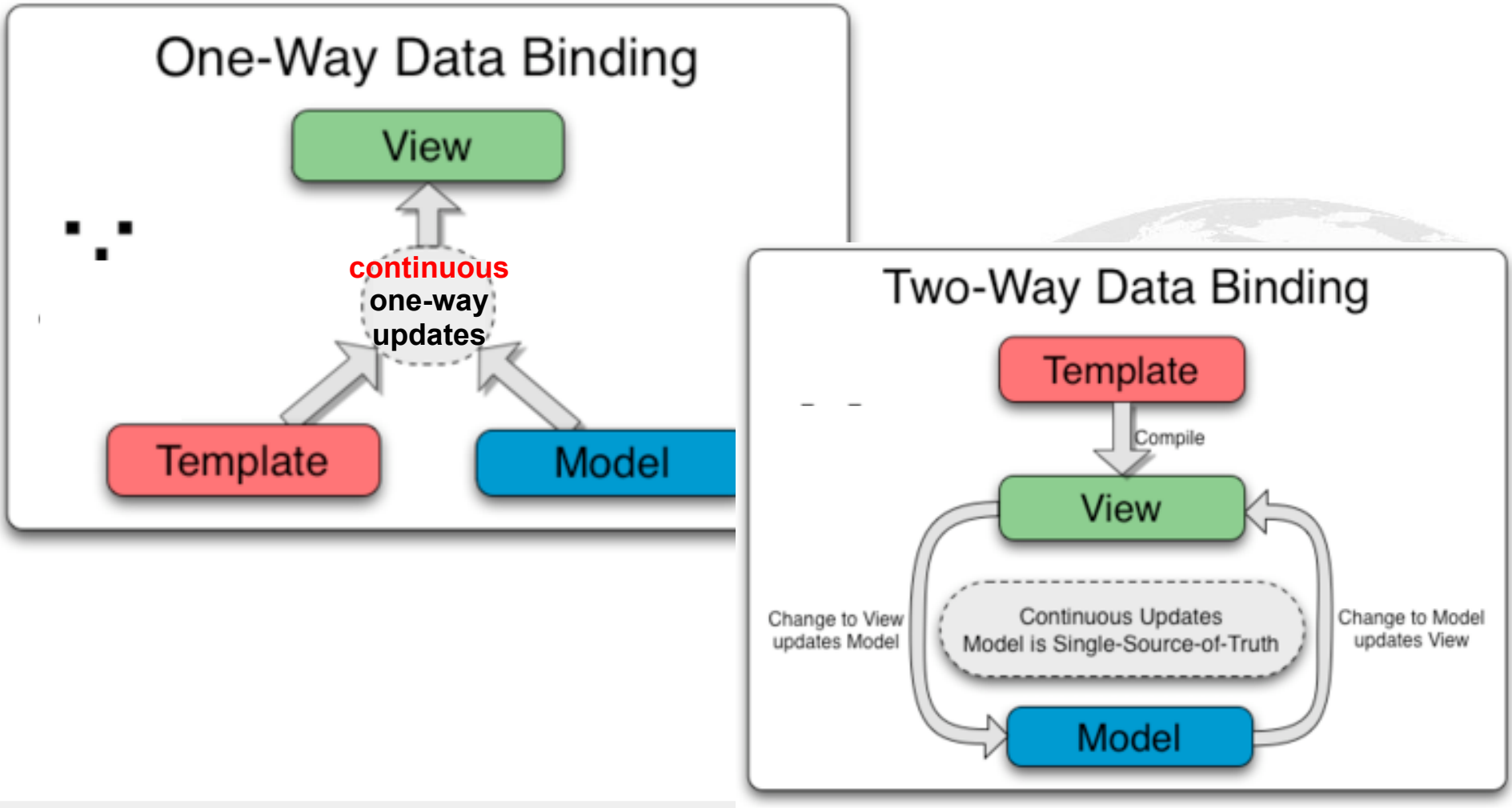
Web Components III

```
<template id="custom-tag-tpl">
   <style>
      h1 { color: blue; }
   </style>
   <h1>My Custom Component</h1>
</template>

var CustomCmpProto = Object.create(HTMLElement.prototype);
CustomCmpProto.createdCallback = function() {
        var template = document.querySelector('#custom-tag-tpl');
        var clone = document.importNode(template.content, true);
        this.createShadowRoot().appendChild(clone);
};
var MyCmp = document.registerElement('custom-cmp', {prototype: CustomCmpProto});
document.body.appendChild(new MyCmp());
```

Data Binding I



Data Binding II



Source: AngularJS Developer Guide:
https://docs.angularjs.org/guide/databinding
License: CC BY-3.0

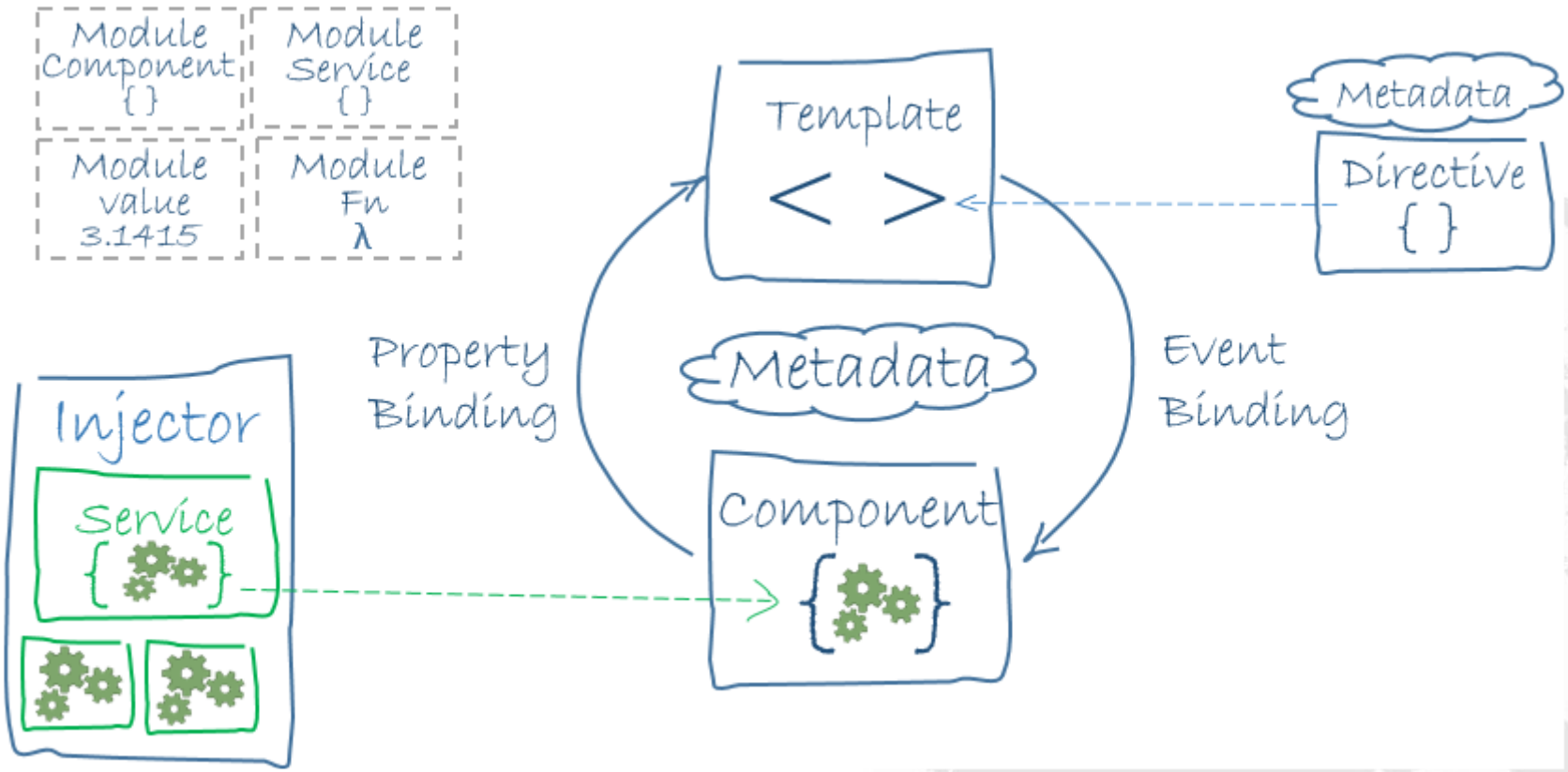
Advantages of Angular 2+

- **Speed** – Angular 2+ is dramatically faster than AngularJS with support for fast initial loads through server-side pre-rendering, offline compile for fast startup, and ultrafast change detection and view caching for smooth virtual scrolling and snappy view transitions.
- **Browsers** – Angular 2+ supports IE 9, 10, 11, Microsoft Edge, Safari, Firefox, Chrome, Mobile Safari, and Android 4.1+.
- **Cross-platform** – By learning Angular, you'll gain the core knowledge you'll need to build for a full range of platforms including desktop and mobile web, hybrid and native UI mobile installed apps, and even installable desktop applications.

Angular Application Development

- Data architecture in Angular – overview, main types of components: Module, Component, Template, Metadata, Data Binding, Service, Directive, Dependency Injection.
- Component controllers, views & templates
- Using external template and style files.
- Angular by example – simple TODO application

Angular Data Architecture



Source: Angular 2 Developer Guide: Architecture overview
https://angular.io/docs/ts/latest/guide/architecture.html

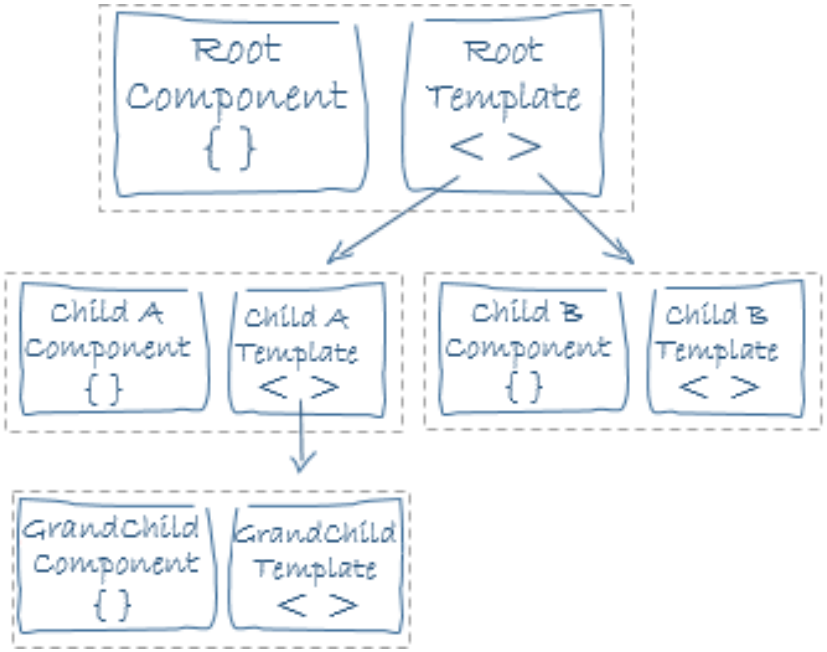
Example: app.module.ts

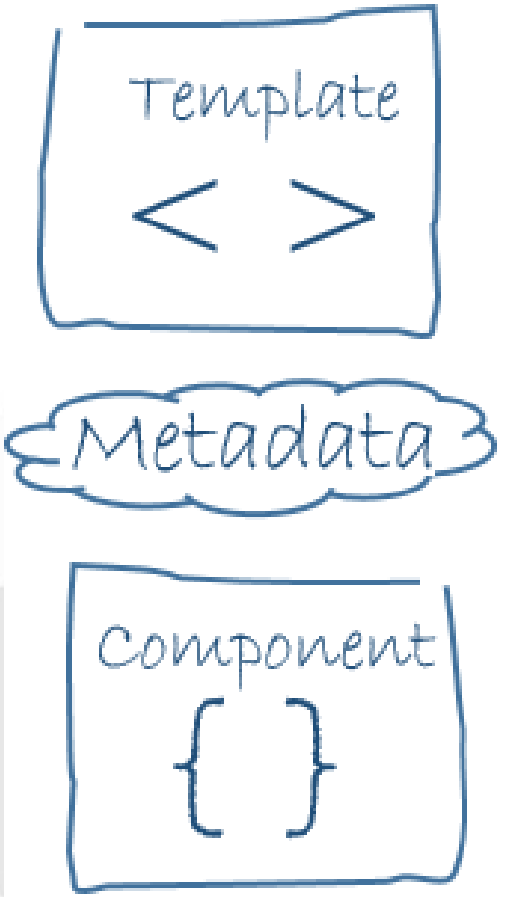
```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }   from './app.component';
@NgModule({
  imports:      [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Angular Modules using @NgModule Decorator

- **declarations** - the view classes that belong to this module (components, directives, and pipes).
- **exports** - the subset of declarations that should be visible and usable in the component templates of other modules.
- **imports** - other modules whose exported classes are needed by component templates declared in this module.
- **providers** - creators of services that this module contributes to the global collection of services.
- **bootstrap** - the main application view, called the root component, that hosts all other app views (root module only).

Components (View Models) & Templates (Views)





Source: Angular 2 Developer Guide: Architecture overview
https://angular.io/docs/ts/latest/guide/architecture.html, License: CC BY 4.0.

Example: app.component.ts

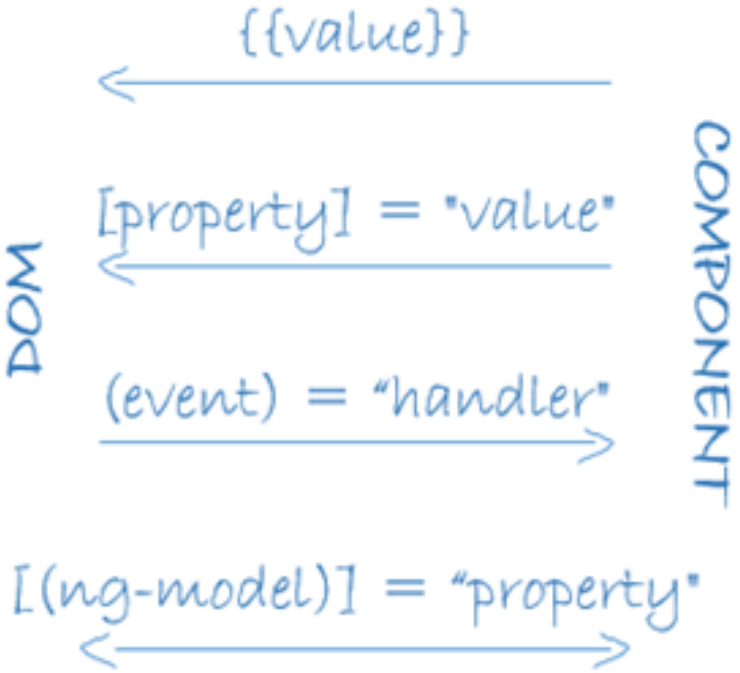
```
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `
    <h1>{{title}}</h1>
    <h2>My favorite hero is: {{myHero}}</h2>
    `

})
export class AppComponent {
  title = 'Tour of Heroes';
  myHero = 'Windstorm';
}
```

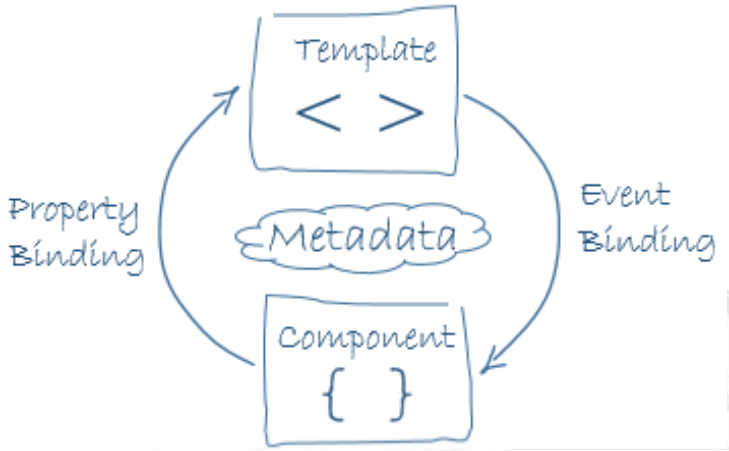
← Metadata

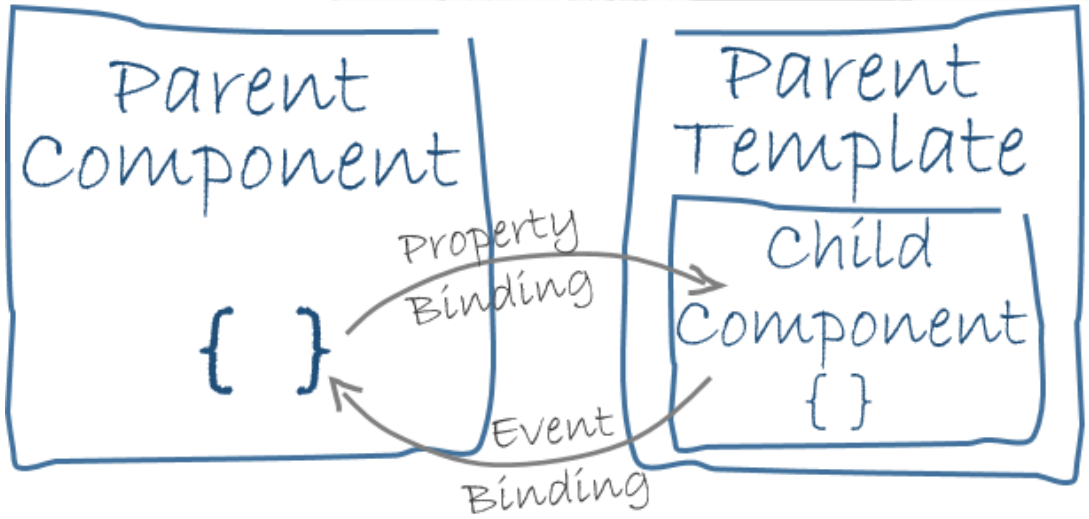
← Template

← Component

Angular 2 Data Binding Types







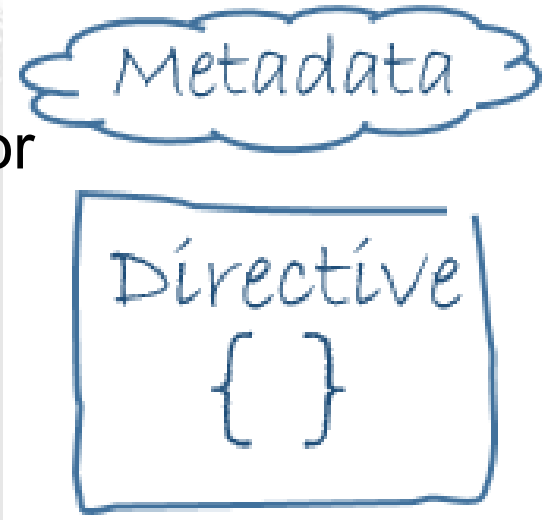
Angular 2 Directives

- Structural directives – alter (add, remove replace) DOM elements – Example:

```
<li *ngFor="let hero of heroes"></li>
<hero-detail *ngIf="selectedHero"></hero-detail>
```

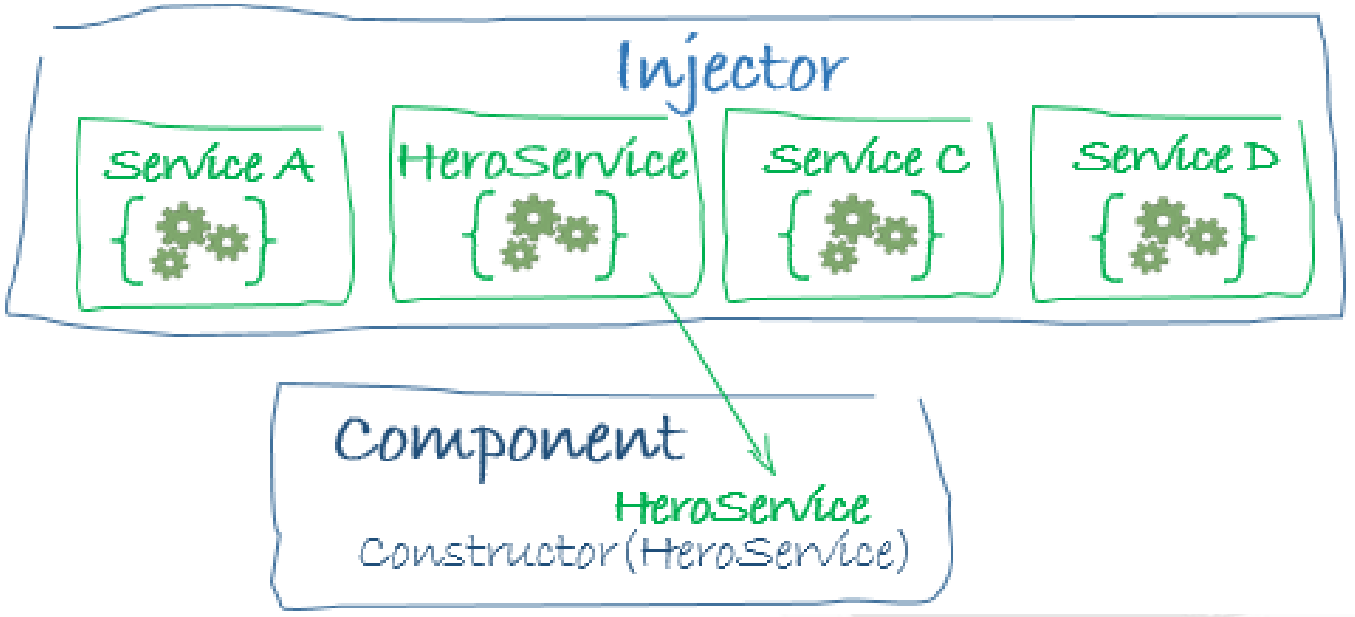
- Attribute directives – alter the appearance or behavior of an existing element – Example:

```
<input [(ngModel)]="hero.name">
```



Services & Dependency Injection (DI)

- Dependency Injection (DI) using constructors
- Hierarchical dependency injectors – module or component provided services



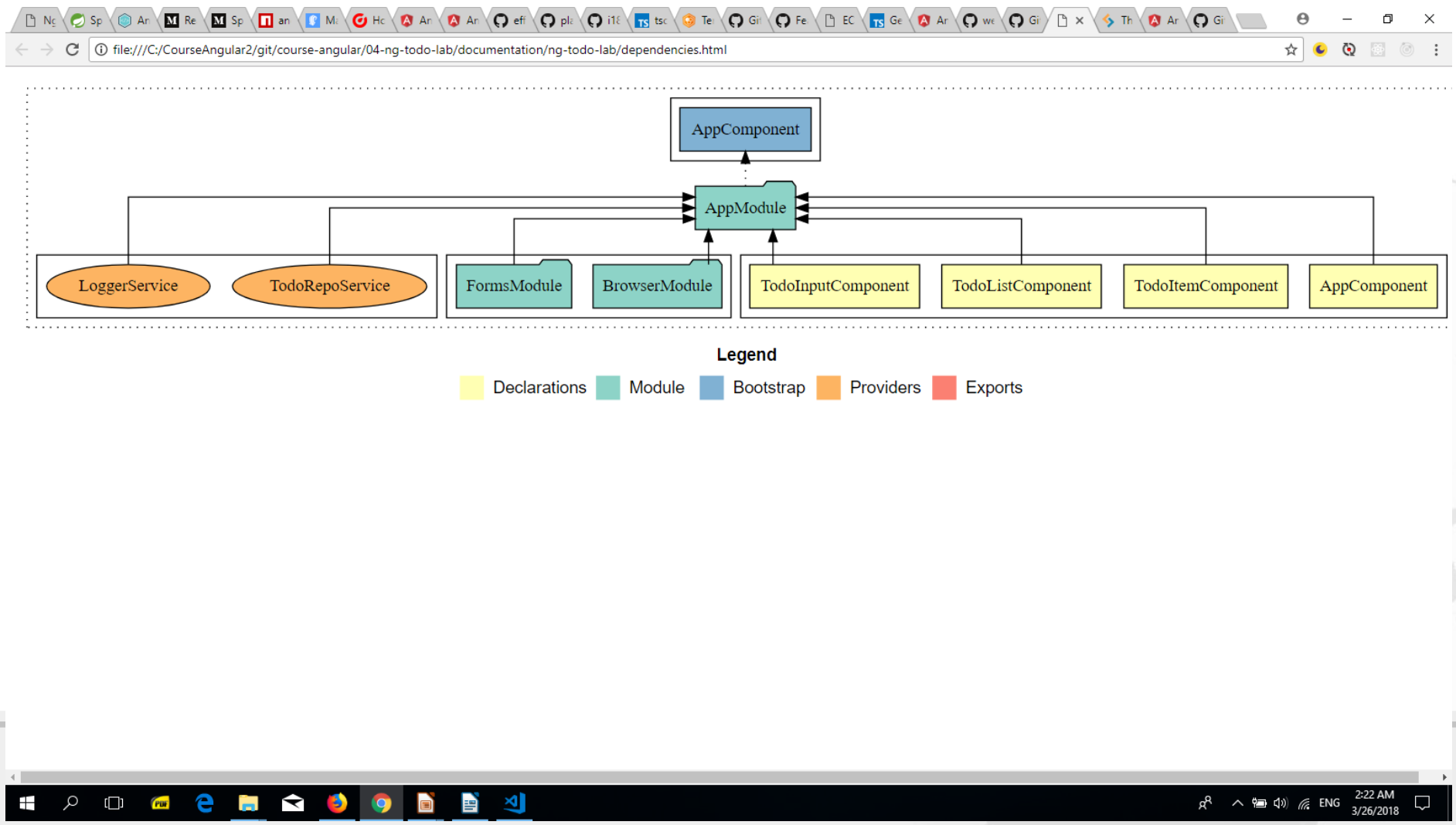
Source: Angular 2 Developer Guide: Architecture overview
https://angular.io/docs/ts/latest/guide/architecture.html, License: CC BY 4.0.

Angular 2 Features & Services

- Animations – animate component behavior
- Change detection – zones to intercept asynchronous activity
- Events – publishing and subscribing to events (EventEmitter)
- Forms – complex data entry with validation and dirty checking
- HTTPClient – communicate with a server to get/update data
- Lifecycle hooks – tap into key lifetime events of a component,
- Pipes – transforming display data: **price | currency:'USD':true**
- Router – single-page application navigation
- Testing – running unit tests on Ng2 application components

Source: Angular 2 Developer Guide: Architecture overview
https://angular.io/docs/ts/latest/guide/architecture.html, License: CC BY 4.0.

Angular TODO Application Demo



Angular Ahead of Time (AOT) Compilation [https://angular.io/guide/aot-compiler]

- Before the browser can render the application, the **components** and **templates** must be converted to executable JavaScript by the **Angular compiler**.
- The **just-in-time (JIT) compiler** compiles the app in the browser, at runtime, every time the application loads.
- The **ahead-of-time (AOT) compiler** runs **once at build time** using **one set of libraries**; with **JIT** it **runs every time** for every user at runtime using a **different set of libraries**.
- The **ahead-of-time (AOT) compiler** can catch template errors early and improve performance by compiling at build time.

Ahead of Time (AOT) Compilation Example

[https://blog.craftlab.hu/multiple-solutions-for-angular-ahead-of-time-aot-compilation-c474d9a0d508]

AOT compilation turns HTML template:

```
<h1>Hello World!</h1>
```

into runable TS code like this:

```
const parentRenderNode:any =
    this.renderer.createViewRoot(this.parentElement);
this._el_0=import3.createRenderElement( this.renderer,parentRenderNode,'h1',
    import3.EMPTY_INLINE_ARRAY,(null as any));
this._text_1 = this.renderer.createText(this._el_0,'Hello World!',(null as any));
this._text_2 = this.renderer.createText(parentRenderNode,'\n',(null as any));
```

Benefits of Ahead of Time (AOT) Compilation [https://angular.io/guide/aot-compiler]

- Smaller application size – the Angular compiler is excluded from the code to be downloaded by the client
- Faster component rendering – the component templates are already compiled
- Fewer asynchronous requests – the pre-compiled templates are already inlined in the TS code
- Template parse errors detected earlier – at build time, as already mentioned
- Better security – with no templates to read and no risky client-side HTML and JavaScript evaluation, there are fewer opportunities for injection attacks

AOT Tooling Alternatives

- **ngc** – the Angular compiler is excluded from the command line, instead of the TypeScript compiler (tsc):

```
"node_modules/.bin/ngc" -p tsconfig-aot.json
```

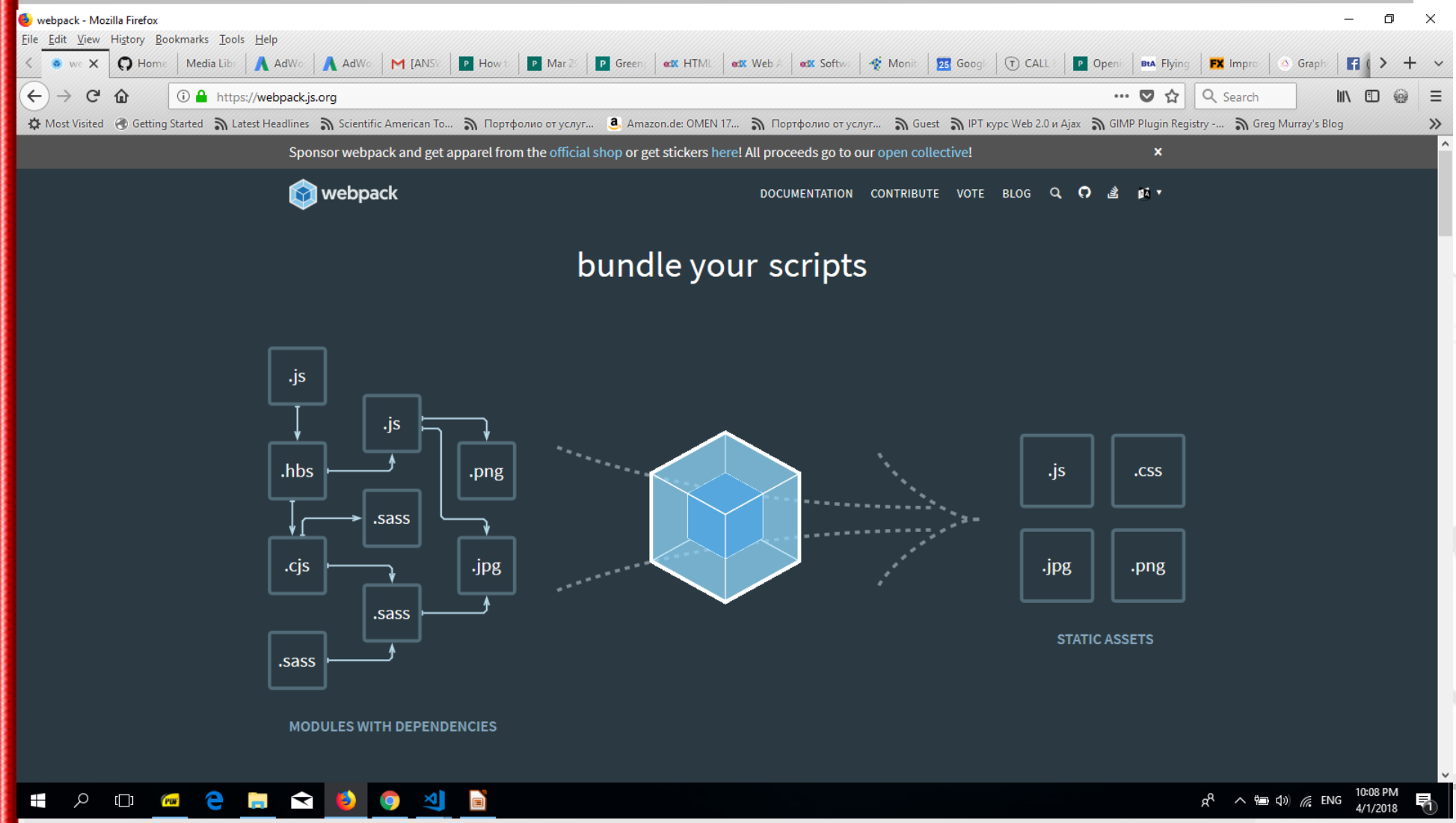
- Angular CLI and **@ngtools/webpack plugin** – the easiest way using Webpack toolchain, better support for style inlining, no need for separate AOT version of bootstrap file (main.ts), no output to disk of separate **.ngfactory.ts* files, but because of this it is not good for AOT compatible package publishing:

```
ng --dev --aot      (development) OR
ng --prod           (production, --aot by default)
```

But There Are Linitations What Works with AOT [https://github.com/rangle/angular-2-aot-sandbox]

Test	AoT With `ngc`	AoT With `@ngtools/webpack`	JiT
control	✅	✅	✅
form-control	✅	✅	✅
func-in-string-config	✅	✅	✅
jquery	✅	✅	✅
template-variable	✅	✅	✅
template-expression	✅	✅	✅
mut-property-decorator	✅	❌	✅
nomut-property-decorator	✅	❌	✅
angular-redux-store	✅	✅	✅
ngrx	✅	✅	✅
ngrx-compose	✅	✅	✅
arrow-function-exports	❌	❌	✅
default-exports	❌	❌	✅
form-control-error	❌	❌	✅
func-as-variable-export	❌	❌	✅
func-declaration-export	✅	✅	✅
func-in-declarations	❌	❌	✅
func-in-providers	❌	❌	✅
func-in-providers-useFactory	❌	❌	✅
func-in-providers-useValue	❌	❌	✅
func-in-routes	❌	❌	✅
interpolated-es6	❌	❌	✅
property-accessors	❌	❌	✅
private-contentchild	❌	❌	✅
private-hostbinding	❌	❌	✅
private-input	❌	❌	✅

Webpack Builder & Bundler Tool



Creating New Project: NPM + WebPack

[https://angular.io/guide/webpack]

```
mkdir my-project
cd my-project
npm init
npm install webpack webpack-dev-server  --save-dev
touch index.html src/index.js webpack.config.js
npm install awesome-typescript-loader angular-router-loader
    angular2-template-loader --save-dev
npm install css-loader style-loader css-to-string-loader --save-dev
npm install file-loader url-loader html-loader --save-dev
npm install extract-text-webpack-plugin html-webpack-plugi --save-dev
```

In package.json:

```
"scripts": {
    "start": "webpack-dev-server --inline --hot",
    "watch": "webpack --watch",
    "build": "webpack -p"
},
```

Simple webpack.config.js I

```
const path = require('path');

module.exports = {
  context: path.resolve(__dirname, 'src'),
  entry: './index.js',
  output: {
      path: path.resolve(__dirname, 'dist'),
      filename: 'bundle.js'
  },

  ...
```

Simple webpack.config.js II

```
module: {
    rules: [{
      test: /\.ts$/,
      loaders: ['awesome-typescript-loader',
      'angular-router-loader', 'angular2-template-loader']
    }, {
      test: /\.html$/,
      loader: 'html-loader'
    }, ...
    {
      test: /\.css$/,
      include: helpers.root('src', 'app'),
      loader: ['css-to-string-loader','css-loader']
    }]
  }
  ...
}
```

Webpack Loaders and Plugins

- Loaders are transformations (functions running in node.js) that are applied on a resource file of your app
- You can use loaders to to load ES6/7 or TypeScript
- Loaders can be chained in a pipeline to the resource. The final loader is expected to return JavaScript
- Loaders can be synchronous or asynchronous
- Loaders accept query parameters – loader configuration
- Loaders can be bound to extensions / RegExps
- Loaders can be published / installed through npm
- Plugins – more universal than loaders, provide more features

WebPack Loaders
[https://webpack.js.org/loaders/]

- awesome-typescript-loader - turns TypeScript into ES 5 or 6
- babel-loader - turns ES6 code into vanilla ES5 using Babel
- file-loader - emits the file into the output folder and returns the url
- url-loader - like file loader, but returns Data Url if file size <= limit
- extract-loader - prepares HTML and CSS modules to be extracted into separate files (alt. to ExtractTextWebpackPlugin)
- html-loader - exports HTML as string, requiring static resources
- style-loader - adds exports of a module as style to DOM
- css-loader - loads css file resolving imports and returns css code
- sass-loader - loads and compiles a SASS/SCSS file
- postcss-loader - loads and transforms a CSS file using PostCSS
- raw-loader - lets you import files as a string

WebPack Main Plugins

- ExtractTextWebpackPlugin - extracts CSS from your bundles into a separate file (e.g. app.bundle.css) → mini-css-extract-plugin
- CompressionWebpackPlugin - prepare compressed versions of assets to serve them with Content-Encoding
- I18nWebpackPlugin - adds i18n support to your bundles
- HtmlWebpackPlugin - simplifies creation of HTML files (index.html) to serve your bundles
- ProvidePlugin - automatically loads modules, whenever used
- UglifyJsPlugin – tree transformer and compressor which reduces the code size by applying various optimizations
- ~~CommonsChunkPlugin~~ - generates chunks of common modules shared between entry points and splits them to separate bundles

Thanks for Your Attention!

Questions?