INTRODUCING ANGULAR

Ori Calvo, 2017

oric@trainologic.com

http://trainologic.com

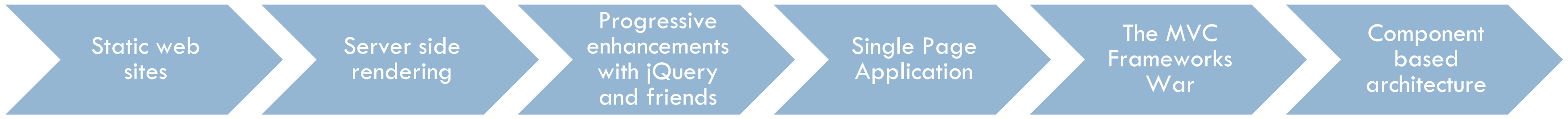
trainologic

Objectives

- Angular History
- Getting Started with Angular
- Identify Angular dependencies
- Develop basic Angular component
- Use @angular/cli

trainologic

Industry Trends



Angular

- Almost 3 years of development
- Now at version 4
- AngularJS is based on concepts rooted at 2009
- Angular aims to "upgrade" AngularJS with new 2016/2017 concepts
- Not backward compatible
- Does support side by side execution with AngularJS

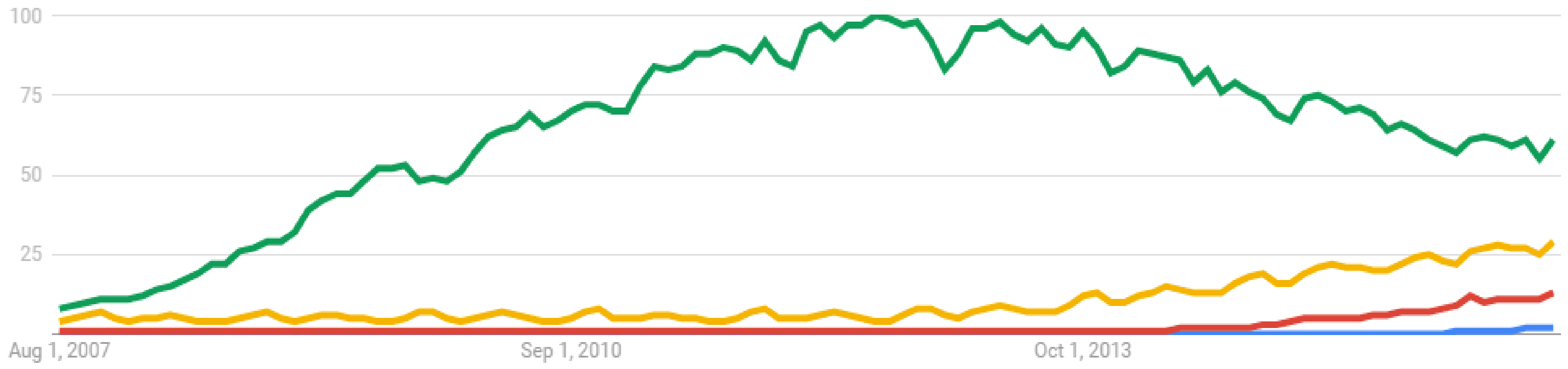
trainologic

New Concepts

- Component based architecture
- Unidirectional data flow
- Server side rendering
- Running inside web workers
- Native development
- Pre compilation of views
- Observables
- Hierarchical Dependency Injection

trainologic

Angular vs. Others



- jQuery
- Angular
- React
- Angular2

Getting Started

- The easiest way is to use @angular/cli
- Hold your horses … lets do it manually
  - Module
  - Component
  - Bootstrapping
  - Polyfills
  - Typescript
  - Webpack

Installing Angular Dependencies

- Start with
  - `npm install @angular/platform-browser-dynamic`
- Fix all “UNMET PEER DEPENDENCY”
  - @angular/platform-browser-dynamic
  - @angular/core
  - @angular/compiler
  - @angular/platform-browser
  - @angular/common
  - rxjs
  - zone.js

Angular Polyfills

- Depends on your browser
- At minimum
  - reflect-metadata
    - Reflect API
  - zone.js
    - Not really a polyfill
    - Helps Angular handle asynchronous code

Angular “Minimal” Ingredients

- Module
- Component
- Bootstrapping

trainologic

Angular Module

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { ClockComponent } from "./clock.component";

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

Enjoy the public content of other modules

Make these components available to the application

The component to be loaded when this module is bootstrapped

Angular Module

- Consolidates components, directives and pipes into cohesive blocks of functionality
- Provides services
- Can be lazy loaded
- Usually per feature or per library
- Has public/private interfaces

trainologic

Angular Component

Component metadata is injected using decorators

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

@Component({
  selector: "my-app",
  template: "<h1>Hello Angular 2</h1>"
})
export class AppComponent {
}
```

HTML element name

The template that will be injected into the component host element

Angular Component

- The term “controller” is no longer being used by Angular
  - Resembles the industry shift from MVC to component based architecture
- A component consist of
  - Name
  - Logic
  - Template
  - Styles
  - Metadata

trainologic

Boostrapping

Browser is not the only supported platform

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

platformBrowserDynamic().bootstrapModule(AppModule);
```

Why not just name it “bootstrap“ ?

Bootstrapping

- No automatic bootstrapping ☺
- You must tell Angular when to initialize the application
  - Allows for easier integration with 3rd party libraries
- Just like AngularJS you specify the root module and Angular does the magic

trainologic

Configure Typescript

- Run npm install typescript
- Add tsconfig.json
- Angular depends on Typescript decorator support
  - experimentalDecorators
  - emitDecoratorMetadata
- Use lib: [“dom”, “es2015”] to support standard libraries that Angular uses

trainologic

tsconfig.json

Convert import to require

Resolving non relative modules according to NodeJS convention

Angular relies heavily on decorators metadata

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  }
}
```

trainologic

Compile Your code

- Typescript compiler is located under node_modules
  - node_modules/.bin/tsc
- Just execute it and it will read all options from the tsconfig.json
- Ensure you don't get any compilation errors
- You may add a package.json scripts command

```
{
  "scripts": {
    "tsc": "tsc"
  }
}
```

trainologic

Compiled main.js

```
"use strict";

var platform_browser_dynamic_1
        = require('@angular/platform-browser-dynamic');

var app_module_1
        = require('./app.module');

platform_browser_dynamic_1.platformBrowserDynamic()
        .bootstrapModule(app_module_1.AppModule);

//# sourceMappingURL=main.js.map
```

The Typescript compiler uses require instead of import

trainologic

Module Loader

- The keyword import is not yet supported by browsers
- We need to convert it to different syntax
- The common practice is to use CommonJS modules
- The Typescript compiler can transform "import" to "require"

Module Loader

- Two popular libraries for loading CommonJS modules inside the browser
  - WebPack – 8M downloads per month
  - SystemJS – 0.5M downloads per month
- Angular prefers Webpack
  - Many options
  - Large eco system
  - Too complex ☹

trainologic

Webpack

- npm install webpack
- Create webpack.config.js

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

module.exports = {
    entry: './main.js',
    output: {
        filename: 'bundle.js',
        path: path.resolve(__dirname, 'dist')
    }
};
```

- node_modules/.bin/webpack --watch

index.html

- Webpack creates bundles under dist folder
- We need to manually include the bundles inside the HTML

```
<body>
    <app-root></app-root>

    <script src="dist/bundle.js"></script>
</body>
```

- Can be automated using html-webpack-plugin
  - Out of scope

@angular/cli

- Even when using Webpack, implementing build scripts is considered a complex task
- So the Angular team created an abstraction layer on top of Webpack
  - So now you need to learn both …
- Starting with @angular/cli is easy
- At the long term you understand that customization capabilities resides inside Webpack and not inside angular/cli

trainologic

@angular/cli

- Very opinionated
- A complete technology stack
- Strict directory structure
- Supports unit testing + E2E
- Development server
- Production build
- Scaffolding

@angular/cli Getting Started

- Install CLI tool globally
  - npm install -g @angular/cli
  - yarn global add @angular/cli
- Verify installation: ng –v
- Create new project

Create new project

- ng new my-project
- A new directory is created with all source files
  - package.json
  - tsconfig.json
  - .angular-cli.json
  - e2e – End to end testing
  - src/app – Component & Services
  - src/assets – Runtime assets
  - More …

trainologic

ng new options

- --directory: Name of directory to create, by default this is the application name
- --prefix: Component selector prefix
  - Can be overridden per component
- --inline-style: Do not generate CSS file
  - Can be overridden per component
- --inline-template: Do not use inline templates
  - Can be overridden per component

.angular-cli.json

- This is @angular/cli configuration file
- Use is to customize aspects of @angular/cli
- For example,
  - defaults/serve/port
  - apps[0]/prefix
  - app[0]/environments

ng serve

- Same as npm start
- Starts a development server on port 4200
- JavaScript bundles are created in memory
- Bundles are injected into Index.html
- Any change to the file system triggers re-build
- Use --open option to open a browser
  - Can fix the “npm start” command

trainologic

--routing

- Commonly used cli command option to create a new project and automatically add a routing file in order to implement routing in angular app
- ng new myapp --routing

```
src
-app
----app.component.css
----app.component.html
----app.component.spec.ts
----app.component.ts
----app.module.ts
----app-routing.module.ts
-assets
-environments
-favicon.ico
-index.html
-polyfills.ts
-main.ts
-styles.css
-test.ts
-tsconfig.app.json
-typings.d.ts
-tsconfig.spec.json
```

The project files tree after the command.
A routing module file is now available

trainologic

ng generate

- Assists in creating features to the app such as components, modules, services, pipes & directives
- Some options are derived from project level definition
- Some options can be re-defined
- Also have other options such as:
  - --inline-template use an inline template instead of a separate HTML file
  - --inline-style use inline styles instead of a separate CSS file
  - --prefix change prefix selector

--flat

- Do not generate a parent directory when generating a new component
- ng g component contactList --flat
- Probably you will want to use it when defining a new root component per feature module
  - To be consistent with app.component.ts

trainologic

assets

- By default all static files are rejected
  - Except Webpack bundles
- Solution,
  - Put the asset inside the assets directory
  - The directory is part of production build
- In case of images consider using background-image
  - Thus the image is bundled

SCSS

- By default @angular/cli uses simple CSS files
- You may fix that
  - defaults/styleExt → scss
- You should also rename app/styles.css

trainologic

src/style.css

- A global CSS that is injected into index.html
- Use it to
  - Define styling prior Angular load
  - Global application theme

More Commands

- https://github.com/angular/angular-cli/wiki
- ng lint
- ng test
- ng e2e
- ng build
- ng get/set
- ng eject

trainologic

@angular/cli stories

- https://github.com/angular/angular-cli/wiki/stories
- HMR
- Proxy
- Routing
- Bootstrap
- Many more

Summary

- @angular/cli is an abstraction layer on top of Webpack
- As such it makes life easier (short term)
- Consider use ng eject and work directly with Webpack configuration