

# Angular2+ (breaking down complexities)

By George Franklin

Our Sponsors



Limerick AI – (Application and Games) Software Development MeetUp

<http://meetup.com/LimerickAiSD>

Twitter @LimerickAiSD

# Big Thanks To

- For Everyone Joining & Attending
- Pat Carroll BOI
- Deirdre Twomey BOI

- Are you interested in Machine Learning, Data science, Big Data, Statistics - Like Building developing Models With Python or R?
- Artificial Intelligence – Traditional AI? Search, optimization, Puzzles, NLP (Natural Language Processing), NLP(Neuro-linguistic programming)
- Like building Neural Networks i.e. Deep Learning? TensorFlow, Keras, PyTorch , Caffe, Torch, Theano, CNTK ,Chainer
- Games Design Development, Graphics 2d/3d/Modelling Blender.. etc
- General cross platform Software Development? Web,Mobile,Desktop
- IoT/Embedded System – Arduino, raspberry pi, Intel IoT Boards, Automation, Google Nest,
- Web Development? Angular, React,Vue.js, Html5,Css, Sass, WebPack, Javascript, Typescript
- Procedural/ object oriented - C, C++, C# Java, Python, Go
- Functional Languages F#, Scala, Haskell, Erlang, Clojure, Lisp
- Testing/QA – Testing Strategies / interested in Manual/Automated Testing
- Devops – Docker. Kubernetes, Puppet,Chef, Ansible ,Salt, Terraform
- Systems.Network Engineer – Network virtualization, Open Switch, NFV , Cisco, Vmware, openStack, Azure, Aws, Google
- This Meetup is for people interested in learning more about Artificial intelligence, Developing AI Applications or Games and General Software Development across different platforms e.g Web, Mobile, Desktop and IoT Devices. (Windows10 & IoT, Linux / MacOS /IOS / Android).

In Session 1

# Angular2+ (breaking down complexities)

- We will look at the fundamental parts of Angular2/4/5
- This is part of multi-part series (more on that later)
- Prerequisites (software dependencies, some general web development like basic HTML, some programming knowledge )
- We install the required tooling i.e. nodejs, npm, VScode
- We will initialize a new angular project using the Angular Cli
- Examine the project files, run the project using `ng serve -o`, `ng build`. Look at all the various configuration files.
- Cover the very basics of typescript
- Explain and Examine the build process of angular

At the break We  
will do the Raffle

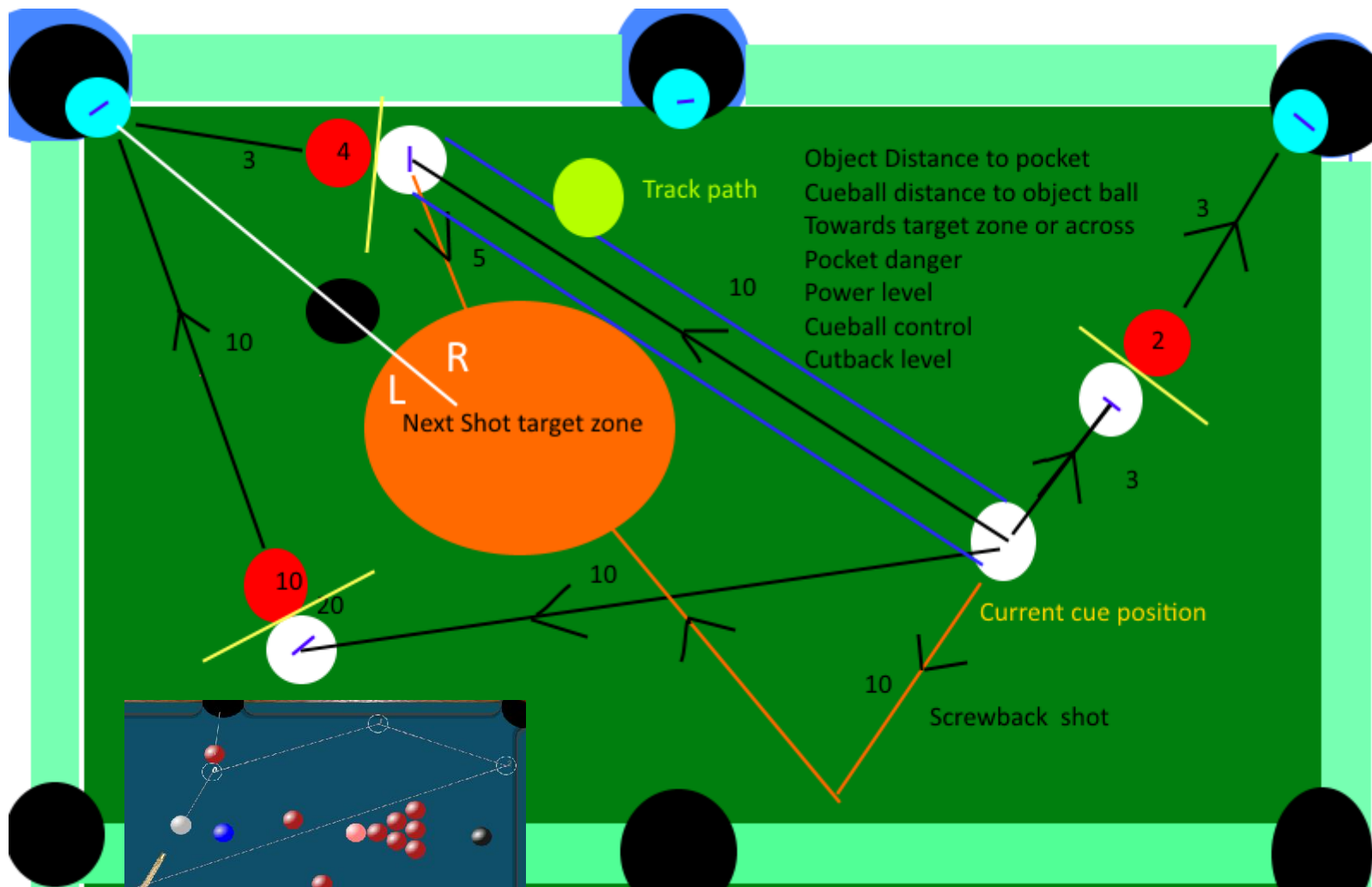
# Session 2

## Credit Card Processor Demo

- Mock out a credit card machine – The user enters a credit card number to make a transaction. We will then mock a retailer's user setting to revoke the card machines ability to process transaction after 3 failed tries. i.e. to stop a customer locking their card. So we use all of the items below.
- Modules, Components, Directives, Services, Pipes
- One Way binding, two Way Binding [(ngModel)], events, @Input,@Output
- Directives @HostBinding, \*ngIf, \*ngFor
- Templates, external, inline, using template containers, ng-content.



# Session 3

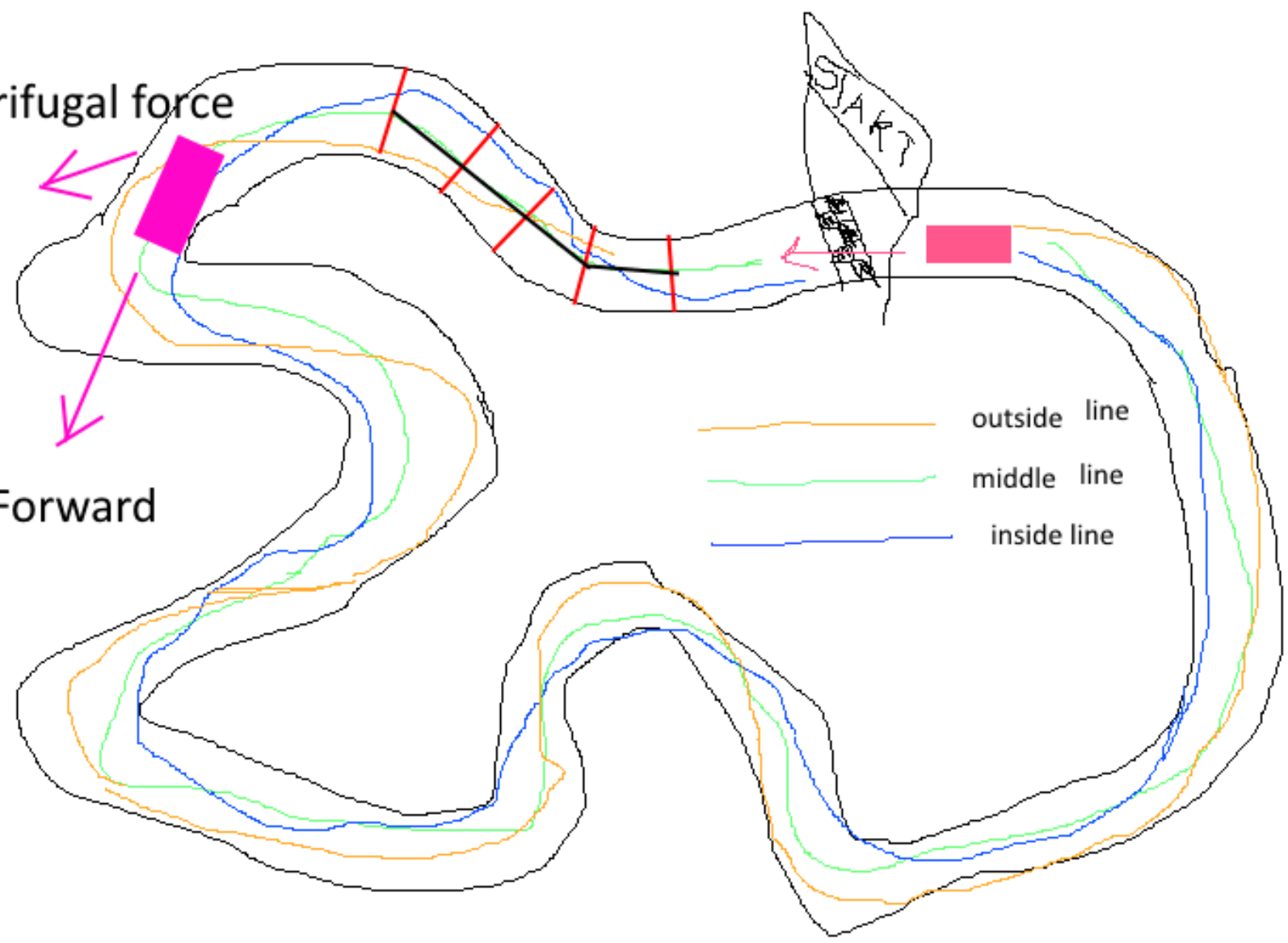


centrifugal force

Forward

START

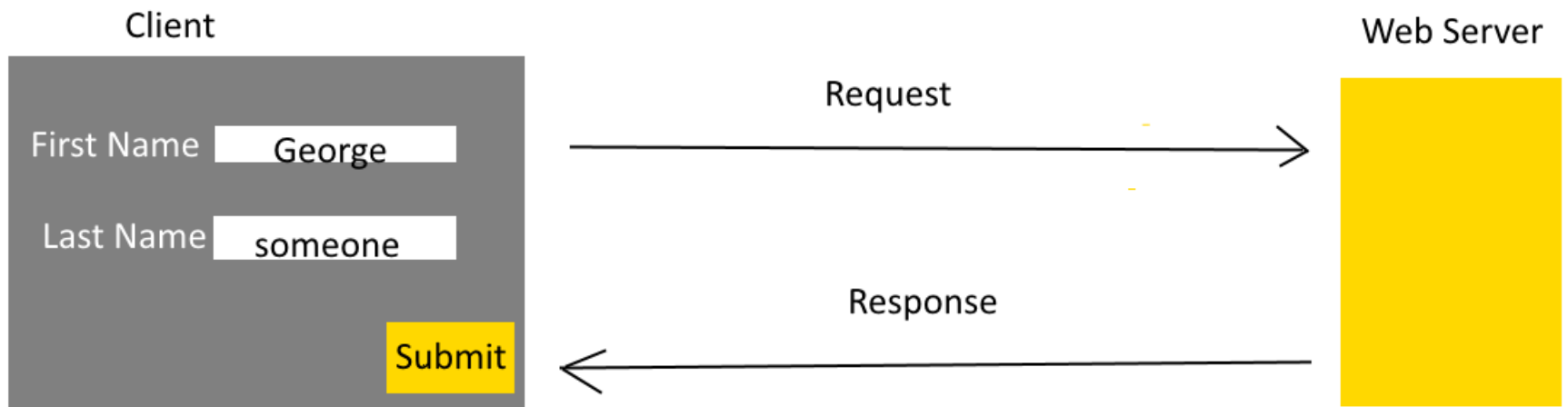
- outside line
- middle line
- inside line



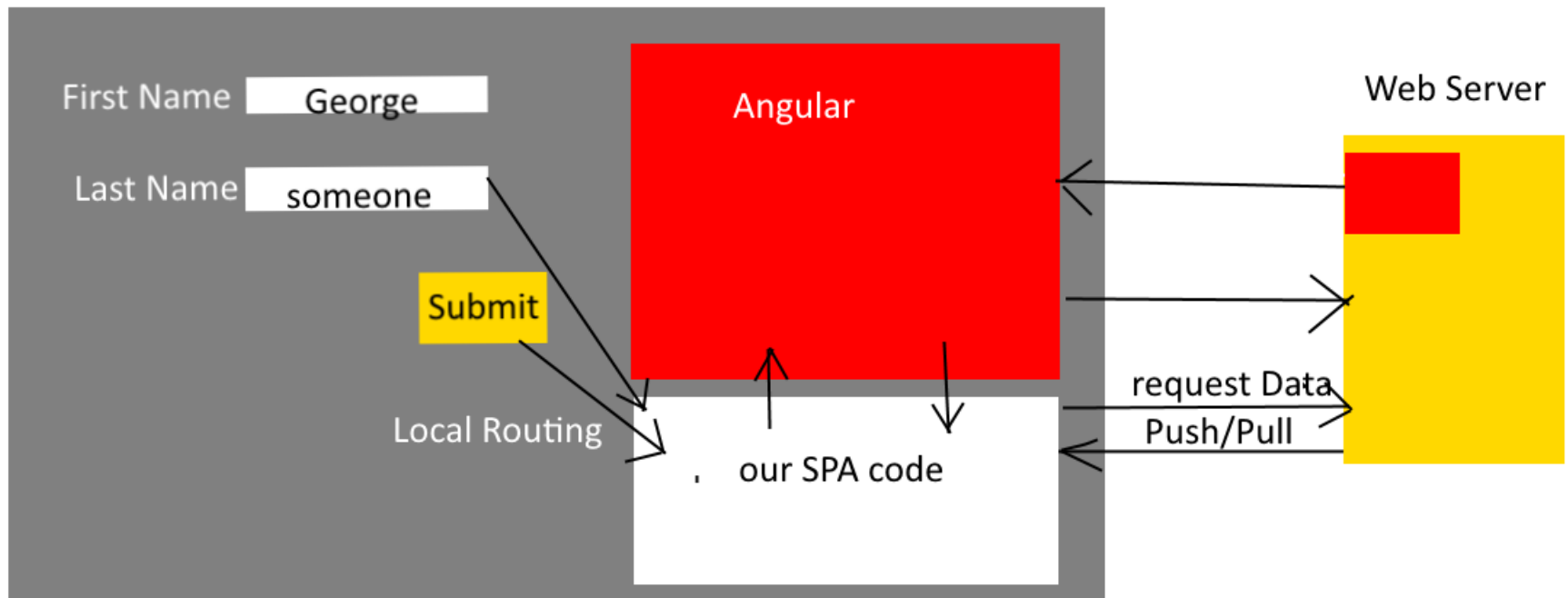
Lets Gets Started

# Angular (SPA)s

## The big Picture



### SPA (Single Page Application)



# The Tooling

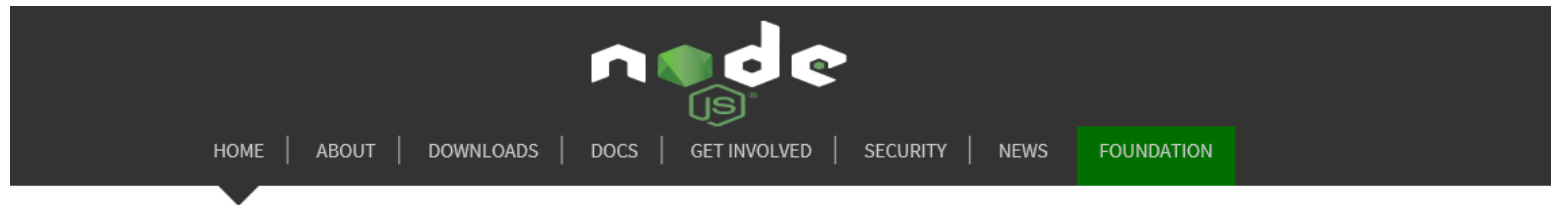
- We need **Nodejs** as all the tooling and support of packages are shipped and maintained via NPM (Node Package Manager)
- **Angular Cli** is a NPM package that allows us to scaffold out a working application and add different components to it.
- **Typescript** compiler NPM package to transcompiler from .TS typescript files to Javascript. .JS files
- **Visual Studio Code** but you can use **Visual Studio** or a **JetBrain IDE**

# Nodejs



# The Tooling Nodejs

- We need **Nodejs** as all the tooling and support of packages are shipped and maintained via NPM (Node Package Manager)
- <https://nodejs.org/en/>



Node.js® is a JavaScript runtime built on **Chrome's V8 JavaScript engine**. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, **npm**, is the largest ecosystem of open source libraries in the world.

**Spectre and Meltdown in the context of Node.js.**

Download for Windows (x64)

**8.9.4 LTS**

Recommended For Most Users

**9.6.1 Current**

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the **LTS schedule**.

Sign up for **Node.js Everywhere**, the official Node.js Weekly Newsletter.

Visual studio  
Code

# The Tooling (VSCode)

- We will use VScode as it has great support for Typescript
- <https://code.visualstudio.com/>

Visual Studio Code Docs Updates Blog Community Extensions FAQ

Search Docs

Download

Version 1.20 is now available! Read about the new features and fixes from January.

## Code editing. Redefined.

Free. Open source. Runs everywhere.

Download for Windows  
Stable Build

Other platforms and Insiders Edition

By using VS Code, you agree to its license and privacy statement.

File Edit View Goto Help

EXTENSIONS

@popular

- C# 1.2.2 356K ★★★★★  
C# for Visual Studio Code (p...  
Microsoft Install
- Python 0.11.2 211K ★★★★★  
Linting, Debugging (multi-t...  
Don Jayamanne Install
- Debugger for Chrome 1.48  
Debug your JavaScript code...  
Microsoft JS Diagno... Install
- C/C++ 0.7.1 143K ★★★★★  
Complete C/C++ language ...  
Microsoft Install
- Go 0.6.39 99K ★★★★★  
Rich Go language support f...  
lukehoban Install
- ESLint 0.10.1 88K ★★★★★  
Integrates ESLint into VS Co...  
Dirk Baumer Install

app.ts

```
1 import app from './app';
2 import debugModule = require('debug');
3 import http = require('http');
4
5 const debug = debugModule('node-express-typescript:server');
6
7 // Get port from environment and store in Express.
8 const port = normalizePort(process.env.PORT || '3000');
9 app.set('port', port);
10
11 // create
12 const server = export
13 server.listen
14 server.on
15 server.on
16
17 /**
18 * Normal
19 */
20 function normalizePort(val: any): number|string|boolean {
21   let port = parseInt(val, 10);
22 }
```

package.json

README.md

Ln 9, Col 21 Spaces: 2 UTF-8 LF TypeScript

# Typescript

- TypeScript is a programming language which was developed by Microsoft. Version 0.8 launched for public use in October 2012
- Development at Microsoft lead by Anders Hejlsberg, the lead architect of c# - also the creator of Delphi and Turbo Pascal etc. (as you all know !!!)
- Maintained by Microsoft, the language comes with an addition of class-based object oriented programming and as optional static typing i.e. or just type JavaScript.
- It is possible to create JavaScript applications for client or server side using the TypeScript.
- In addition, the language has support for definition files containing type information of current JavaScript libraries. This feature is quite similar to header files in the C or C++ languages which describe the structure of current object files.

# Create Project Directory for typescript Demo

- Open a command or powershell prompt
- // make a vscodeprojects directory for your projects
- > md c:\users\george\documents\vscodeprojects
- // cd into the directory
- > cd c:\users\george\documents\vscodeprojects
- // make a directory called typescript and cd into it
- > md typescript
- >cd typescript
- > c:\users\george\documents\vscodeprojects\typescript

# The Tooling - Typescript

- So with the prerequisite of **nodejs** installed
- next install typescript globally with:
- **npm install -g typescript**
- next check the version installed with:
- **tsc -v**
- **NOTE: I now have 2.7.2 (2.6.2 or greater is fine)**

⌵ C:\Users\george>tsc -v  
Version 2.6.2

# The Tooling - Typescript

- Let's now generate tsconfig.json to control tsc settings

Try to run in your console the following to check the version:

```
$ tsc -v
```

If the version is older than 1.6 you will need to update:

```
$ npm install -g typescript
```

Remember that you need to install node.js to use npm.

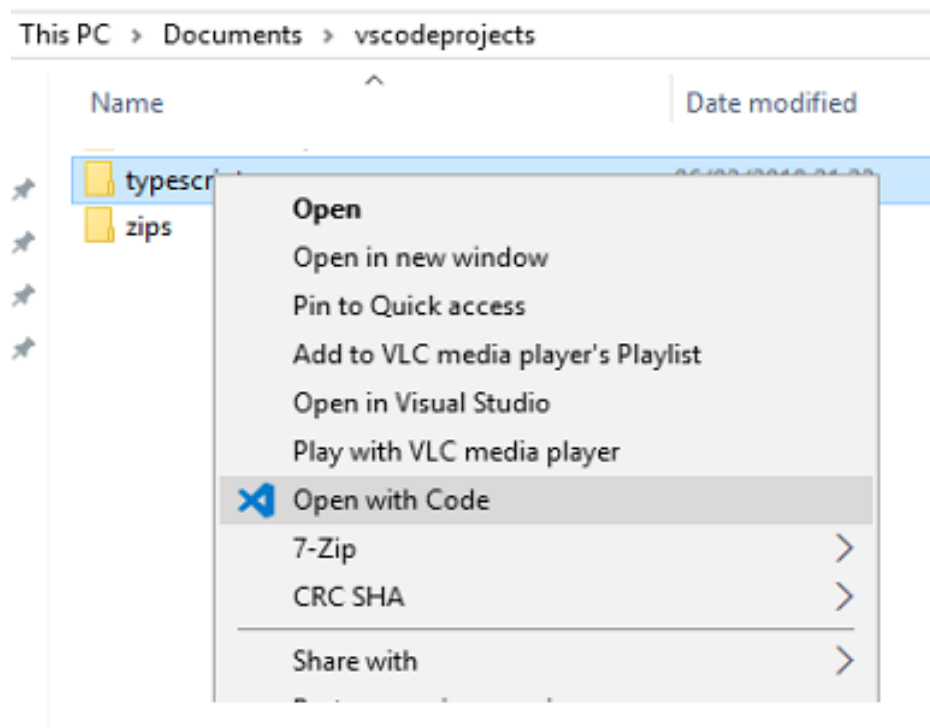
The correct command is `--init` not `init`:

```
$ tsc --init
```

- Lets learn some Typescript and we will use Vscode

# Now open your Project Directory in Vscode

- Now to open Vscode in your current directory on windows type: `code .` (code followed by a fullstop) not `vscode` !!!
- **code .**
- Or you can let VSCode open the folder in windows explorer by right clicking and selecting open with vs code



- Lets learn some Typescript and we will use Vscode



## Next we cover the basics of Typescript

- Set options in the tsconfig file : for dist output & ES version. Look at Tslint and the extensions in VSCode
- Review how to change Tslint settings or disable in file with `/* tslint:disable */` or enable `/* tslint:enable */`
- How to set a watch to invoke TSC to compile our files
- Write some basic typescript (enough to understand and work with angular 2/4/5)
- Import & export classes. Define/instantiate Classes, private/ public methods, functions, instance/static variables, var & let.
- Interfaces, Enums, Unions.
- Examine the TSC Typescript compiler output and compare to Javascript, with regards to the tsconfig ES setting. IIFE(Immediately Invoked Function Expressions).

# Typescript Basics and VSCode Tooling Demo

# The VSCode Tooling extensions

The screenshot displays the VS Code Extensions Marketplace interface. The left sidebar shows the 'EXTENSIONS' view with a search bar and a list of installed extensions. The main panel shows the 'INSTALLED' tab with a list of extensions, including 'Git History', 'Git Merger', 'GitLens', 'Ionide-fsharp', 'Live Server', 'npm', 'npm Intellisense', 'Python', and 'TSLint'. A tooltip is visible over the 'npm Intellisense' extension, stating 'This extension is recommended'.

**EXTENSIONS**

Search Extensions in Marketplace

**INSTALLED** 20

- Angular 5 Snippets - TypeScript, Htm...** 5.2.16 1.8M ★ 5  
165 Angular Snippets (TypeScript, Html, Angular Material,...)  
Mikael Morlund
- angular2-inline** 0.0.17 116K ★ 4.5  
Visual Studio Code language extension for javascript/type..  
Nate Wallace
- Auto Import** 1.5.3 318K ★ 4  
Automatically finds, parses and provides code actions and..  
steoates
- C#** 1.14.0 5.1M ★ 4  
C# for Visual Studio Code (powered by OmniSharp).  
Microsoft
- Code Runner** 0.9.0 2.4M ★ 4.5  
Run C, C++, Java, JS, PHP, Python, Perl, Ruby, Go, Lua, Gro..  
Jun Han
- Color Picker** 0.4.5 363K ★ 3  
Helper with GUI to generate color codes such as CSS colo..  
anseki
- Debugger for Chrome** 4.1.0 5.9M ★ 4.5  
Debug your JavaScript code in the Chrome browser, or an..  
Microsoft
- Easy icon theme** 0.3.1 47K ★ 5  
A simple icon pack, made to be easy to customise  
James Maquire

**INSTALLED** 20

- Git History** 0.4.0 2.3M ★ 4.5  
View git log, file history, compare branches or commits  
Don Jayamanne
- Git Merger** 0.3.7 24K ★ 5  
Simplifies the git merge process  
Shahar Kazaz
- GitLens — Git supercharged** 8.0.2 3.4M ★ 5  
Supercharge the Git capabilities built into Visual Studio C..  
Eric Amodio
- Ionide-fsharp** 3.17.2 726K ★ 5  
F# Language Support  
Ionide
- Live Server** 3.2.1 431K ★ 5  
Launch a development local Server with live reload featur..  
Ritwick Dey
- npm** 0.3.3 1.3M ★ 4.5  
npm support for VS Code  
egamma
- npm Intellisense** 1.3.0 555K ★ 5  
Visual Studio Code plugin that autocompletes npm modu..  
Christian Kohler
- Python** 2018.1.0 6.9M ★ 4.5  
Linting, Debugging (multi-threaded, remote), Intellisense,..  
Microsoft
- TSLint** 1.0.28 4.4M ★ 5  
TSLint for Visual Studio Code  
egamma

This extension is recommended

# TypeScript Support

- It now has wide support in IDE like Visual Studio Code, Visual studio 2017 and in JetBrains IDE products.
- It is Distributed via npm - Current version is 2.7.2  
([www.npmjs.com/package/typescript](http://www.npmjs.com/package/typescript))
- <https://angular-2-training-book.rangle.io>
- <https://www.gitbook.com/book/rangle-io/ngcourse2/details>
- <https://angular-2-training-book.rangle.io/handout/features/>

# A few words about ECMAScript/Javascript

- The ECMAScript specification is a standardized specification of a scripting language developed by Brendan Eich of Netscape; initially it was named Mocha, later LiveScript, and finally JavaScript. In December 1995, Sun Microsystems and Netscape announced JavaScript in a press release. In March 1996, Netscape Navigator 2.0 was released, featuring support for JavaScript.
- Javascript releases are code names ES\_ and the year of release.
- ES5 was released in 2009 – the common version supported by all browsers. ES5 Added “strict mode”
- ES6 - ECMAScript 2015 also referred to as fat arrow () => ☺
- Currently, the standard is to publish a new ES specification version once a year. ES6 was published in 2015 and ES7 was published in 2016.
- ECMAScript 8 or ECMAScript 2017 was officially released at the end of June.

## Note about standards

- ES7 has support for decorators
- Great blog post by Addy Osmani
- <https://medium.com/google-developers/exploring-es7-de>

```
function superhero(target) {  
  target.isSuperhero = true;  
  target.power = 'flight';  
}
```

```
@superhero  
class MySuperHero() {}
```

```
console.log(MySuperHero.isSuperhero); // true
```

# TRANSCOMPILING/TRANS-COMPILERS

- So what is it well e.g. we can take perhaps some Javascript written in ES6 and run it through a trans-compiler and output ES5 which is better supported in Browsers. Two well known trans-compilers used over the last few years were google/traceur-compiler & Babel

## Babel is a JavaScript compiler.

Use next generation JavaScript, today.

### Put in next-gen JavaScript

```
var name = "Guy Fieri";  
var place = "Flavortown";  
  
`Hello ${name}, ready for ${place}?`;
```

Template literals are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them. They were called "template strings" in prior editions of the ES2015 specification

### Get browser-compatible JavaScript out

```
var name = "Guy Fieri";  
var place = "Flavortown";  
  
"Hello " + name + ", ready for " + place + "?";
```

[Check out the REPL to experiment more!](#)

# IIFE(Immediately Invoked Function Expressions)

- Self-invoking functions in JavaScript (or Immediately Invoked Function Expressions) -- (pseudo code of the revealing pattern)

```
var module = (function (name,amount) {  
    // private  
    var pname = name;  
    var pbankaccount =amount;  
    var getAccount = function (){  
        return pbankaccount  
    }  
    return {  
        // public  
        getBankaccount : getAccount  
    };  
})(name,amount));
```



# Modular/Component Design. Use Case

# Use Case – The Azure Portal

- Lets just use the Azure portal as a means to under a large complex system.
- Note: \*Whilst the azure portal does not use angular, we will just highlight some features used in the building of it to illustrate the idea of a modular/component design.

# Under the Hood of the new Azure Portal



- Justin Beckwith - (<http://jbecks.com> )
- <http://jbeckwith.com/2014/09/20/how-the-azure-portal-works/>
- In this blog article he explains the building of the new Azure portal in 2014, when he worked at Microsoft. Now works at Google.
- The first version of the portal was built using a Asp.net MVC application. But due to scalability and building components between teams they moved to a new pattern.
- The new portal consisted of using iframes, so that each part of the portal could operate independently. So that each development team could develop in the safe knowledge that changes to the their section would not interfere with the global application.
- So lets looks at the old and the new version ....

# The old Azure Portal

- Built using a Asp.net MVC application

Microsoft Azure | CREDIT STATUS | Subscriptions | thebeckwith@live.com

ALL ITEMS

WEB SITES 13

VIRTUAL MACHINES 0

MOBILE SERVICES 0

CLOUD SERVICES 0

SQL DATABASES 1

STORAGE 5

HDINSIGHT 0

MEDIA SERVICES 0

SERVICE BUS 0

VISUAL STUDIO ONLINE 1

CACHE 0

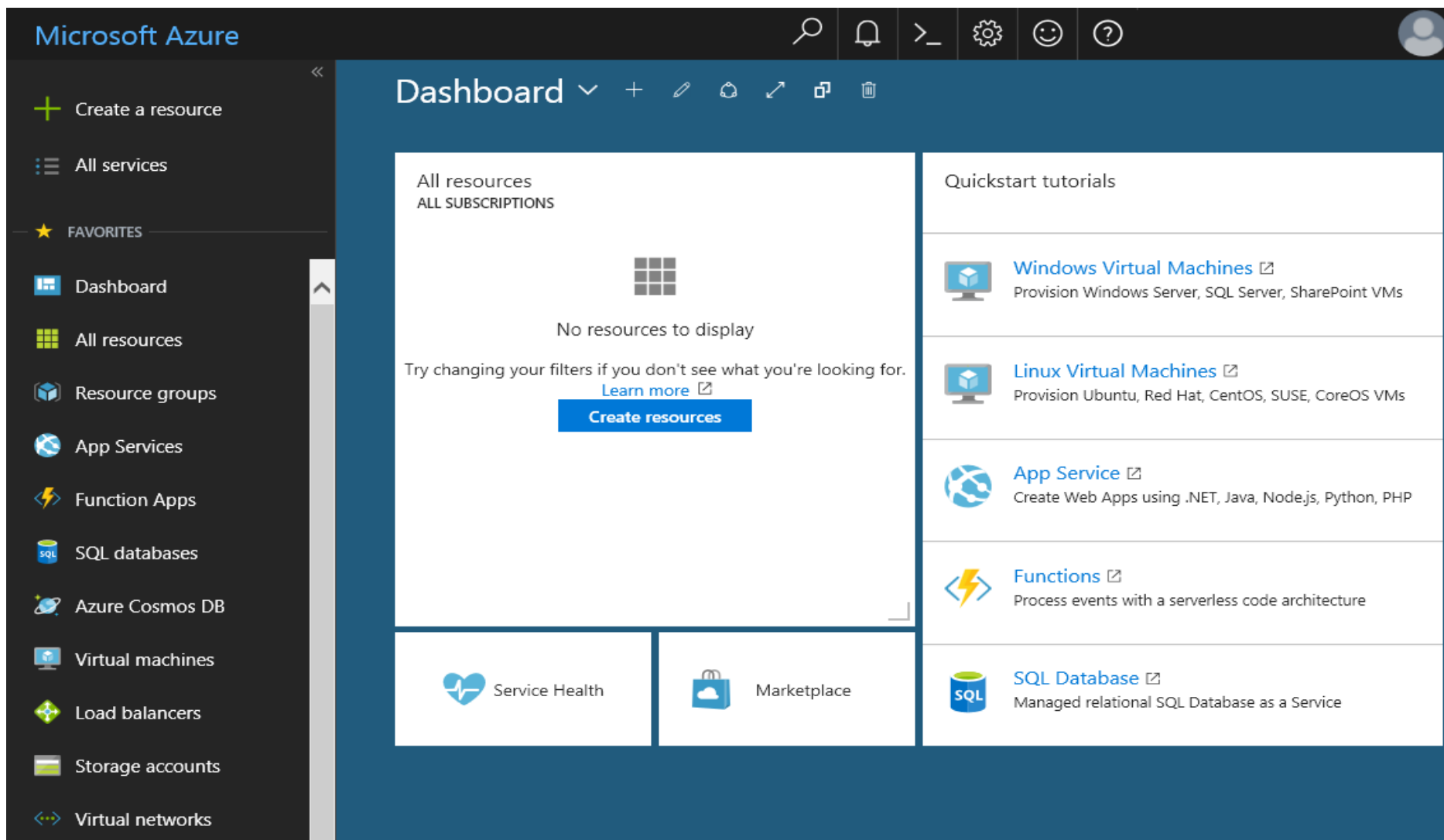
### web sites

NAME	STATUS	SUBSCRIPTION	LOCATION	MODE	URL
DemoSaaSWeb →	✓ Running	Windows Azure MSDN - Visual Studio Ultimate	West US	Free	demosaaasyweb.azurewebsites.net
AwesomeFunDemo	✓ Running	Windows Azure MSDN - Visual Studio Ultimate	West US	Free	awesomefundemo.azurewebsites.net
wazstagram	✓ Running	Windows Azure MSDN - Visual Studio Ultimate	West US	Free	wazstagram.azurewebsites.net
PhotoCloud	✓ Running	Windows Azure MSDN - Visual Studio Ultimate	West US	Free	photocloud.azurewebsites.net
myawesomedemosite	✓ Running	Visual Studio Ultimate with MSDN	West US	Free	myawesomedemosite.azurewebsites.net
svgicons	✓ Running	Visual Studio Ultimate with MSDN	West US	Free	svgicons.azurewebsites.net
AwesomeWebSiteNow	✓ Running	Visual Studio Ultimate with MSDN	West US	Standard	awesomewebsitenow.azurewebsites.net
theawesomestsitemade	✓ Running	Visual Studio Ultimate with MSDN	West US	Standard	theawesomestsitemade.today.azurewebsites.net
sdrdemotime123	✓ Running	Visual Studio Ultimate with MSDN	West US	Standard	sdrdemotime123.azurewebsites.net
hashtagredis	✓ Running	Visual Studio Ultimate with MSDN	East US	Free	hashtagredis.azurewebsites.net
instawesome	✓ Running	Visual Studio Ultimate with MSDN	East US	Free	instawesome.azurewebsites.net
bestdemoever	✓ Running	Visual Studio Ultimate with MSDN	East US	Free	bestdemoever.azurewebsites.net
hashtagredisapi	✓ Running	Visual Studio Ultimate with MSDN	East US	Free	hashtagredisapi.azurewebsites.net

+ NEW | BROWSE | STOP | RESTART | MANAGE DOMAINS | DELETE | ?

# The new Azure Portal

- New portal consisted of using iframes



# Highlight regions

The screenshot shows the Microsoft Azure dashboard interface. Red rectangular highlights are placed on several key areas:

- Top Bar:** The "Microsoft Azure" logo on the left and the help icon (a question mark in a circle) on the right.
- Left Navigation Panel:** The "Resource groups" menu item, which is highlighted with a red box.
- Dashboard Header:** The "Dashboard" title and its associated icons (plus, edit, refresh, link, share, and delete).
- Main Content Area:**
  - The "All resources" section, which includes the text "ALL SUBSCRIPTIONS", a grid icon, the message "No resources to display", a link to "Learn more", and a "Create resources" button.
  - The "Quickstart tutorials" section, which lists several options: "Windows Virtual Machines", "Linux Virtual Machines", "App Service", "Functions", and "SQL Database".
  - The "Service Health" and "Marketplace" tiles at the bottom of the main content area.

So Lets talk Angular

# Angular TimeLine

- Angular 1 or AngularJS as it known today – released in 2010
- It is still maintained - stable is 1.6.9 as of 3rd February 2018
- It was popular until the Angular team did a pivot and decided to reinvent Angular .



# Why Angular 2 (second Generation)

## Improvements in Angular2/4/5

- 1. Change updates – new way to update components on the webpage.
- 2. The modularity of the angular modules got better with looser coupling and deployed SPA is smaller due to just including the required parts. (plus precompiling some parts)
- 3. Typescript being a superset of Javascript has allowed people with an C#/Java or an basic OOP background to be productive.

## Angular 2/4/5 – 6 in Beta

- The Naming convention (explained on the next slide)
- Now branded just as Angular !!! ( No confusion right).
- “Angular is an TypeScript-based open-source front-end web application platform led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS”.
- Angular 2 Initial release: 14 September 2016; (16 months ago)
- Stable release: 5.2.3 / 31 January 2018; (17 days ago)
- Preview release: 6.0.0-beta.2 / 31 January 2018; (17 days ago)

# Angular 2 (second Generation) uses Semantic Versioning

CODE STATUS	STAGE	RULE	EXAMPLE #
First Release	New Product	Start with 1.0.0	1.0.0
Bug fixes, other minor changes	Patch Release	Increment the third digit	1.0.1
New Features that don't break existing features	Minor release	Increment the middle digit	1.1.0
Changes that break backward compatibility	Major release	Increment the first digit	2.0.0

## ☞ Semver for Consumers

**Major**

**Minor**

**Patch**

As a consumer, you can specify which kinds of updates your app can accept in the **package.json** file.

If you were starting with a package 1.0.4, this is how you would specify the ranges:

- Patch releases: **1.0** or **1.0.x** or **~1.0.4**
- Minor releases: **1** or **1.x** or **^1.0.4**
- Major releases: **\*** or **x**

```
},  
"devDependencies": {  
  "CLI tool for Angular": "1.7.1",  
  "core": "~0.4.2",  
  "@angular/cli": "1.6.3",  
  "@angular/compiler-cli": "^5.0.0",  
  "@angular/language-service": "^5.0.0",  
  "@types/jasmine": "~2.5.53",  
  "@types/jasminewd2": "~2.0.2",  
  "@types/node": "~6.0.60",  
  "code-linter": "1.0.0"  
}
```

this

Yo

this

\*

# More about package.json prefixs

- Save your default prefix using npm config set save-prefix=''
- <https://www.npmjs.org/doc/files/package.json.html>
- <https://docs.npmjs.com/misc/semver#x-ranges-12x-1x-12->
- ^ upto major version
- ~ upto to minor version
- >version Must be greater than version
- >=version greater or equal to version
- <version
- <=version
- \* any version
- Latest – obtains the latest version.
- It is also possible to specify an exact range of versions, like  
1.2.0 || >=1.2.2 <1.3.0

# Angular

## Startup process

# The Angular building blocks

- Modules – these files are a way to group components, directives, services and import other modules. Every SPA need at least one root module and one root component
- Components – define a selector tag to be used elsewhere with the html. The output of this selector tag is the Html mark-up that you place in the component's template. The component also has a typescript class to provide the data for your component or to respond to events e.g. like a button click event or text change event.
- Directives – are the same as components BUT don't have a Template. Just like components they do have a typescript class and therefore can provide some supporting function in terms of properties or events processing and are used by other components or by basic HTML tags.
- Pipes – are meant to filter or transform data. With the idea that the outcome of one pipe can be fed into another.
- Services – are a way to provide supporting services to components e.g. Shared data or a service may go out and make a remote call. Services provide the means to mediate shared operations. Services are inserted into a component's constructor by means of dependency injection.

# The root Component(app-root)

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Angularcli2template</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>
  <app-root></app-root>
</body>
</html>
```

# The root Component(app-root)

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Angularcli2template</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>
  <h1>My Angular App</h1>
</body>
</html>
```



# Components

*app/app.component.ts*

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

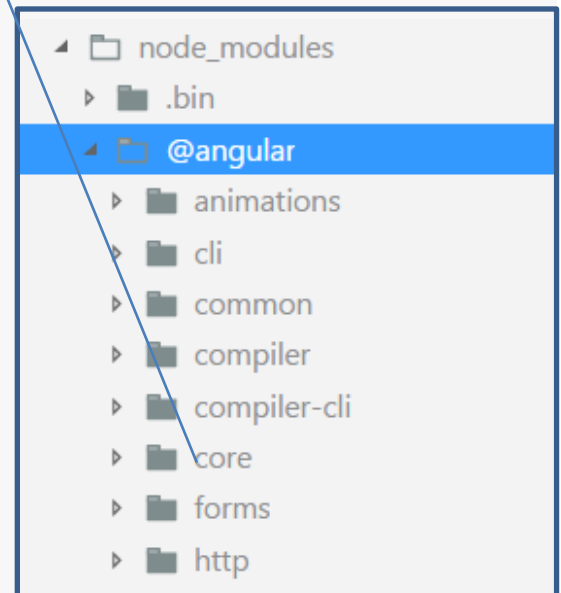
@Component({
  selector: 'app-root',
  template: '<h1>My Angular App</h1>'
})
export class AppComponent {}
```

## *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 { }
```



# The Angular Boot Strap process main.ts

main.ts

```
1  import { enableProdMode } from '@angular/core';
2  import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4  import { AppModule } from './app/app.module';
5  import { environment } from './environments/environment';
6
7  if (environment.production) {
8    enableProdMode();
9  }
10
11  platformBrowserDynamic().bootstrapModule(AppModule)
12    .catch(err => console.log(err));
13
```

environment.ts

```
1  // The file contents for the current environment will overwrite these during build.
2  // The build system defaults to the dev environment which uses `environment.ts`, but if you do
3  // `ng build --env=prod` then `environment.prod.ts` will be used instead.
4  // The list of which env maps to which file can be found in `.angular-cli.json`.
5
6  export const environment = {
7    production: false
8  };
9
```

The first thing that we notice is that our module is importing the **BrowserModule** as an explicit dependency. The **BrowserModule** is a **built-in** module that exports basic **directives**, **pipes** and **services**. Unlike previous versions of Angular, we have to explicitly import those dependencies to be able to use directives like **\*ngFor** or **\*ngIf** in our **templates**.


*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 { }
```

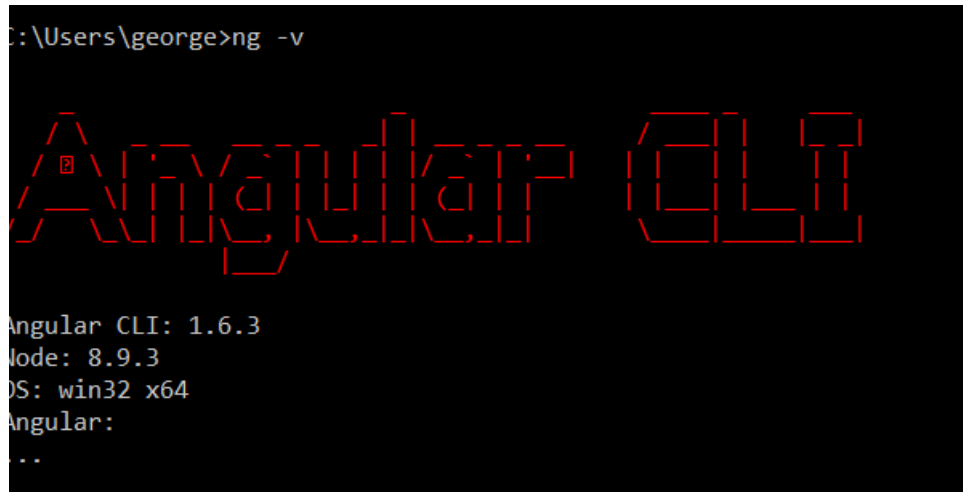
# Modules and Components

- Lets just differentiate between Modules and Components in Angular 2/4/5.
- Component have templates, they are the small units of work that can hold our visual html elements.
- An Angular application must have 1 root Module, but can have 0 or more extra modules (referred to as feature modules)
- Modules exist as a means to group components together logically and provides a separate of concerns. Only the module with that component knows of it's existence.
-  For one module to use another module we need to export a module and then import it. ( Most importantly Angular modules are not and I repeat not ES6 modules. )

# Angular Cli

# The Tooling Angular Cli

- **Angular Cli** is a NPM package that allows us to scaffold out a working application and add different components to it.
- To see if you already have angular-cli installed type `ng -v`
- To install the latest version of angular cli type
- `npm install -g @angular/cli@latest`
- confirm installation with `ng -v` (note: I have an older version)



```
C:\Users\george>ng -v

Angular CLI
Angular CLI: 1.6.3
Node: 8.9.3
OS: win32 x64
Angular:
...
```

- To see a list of your global packages type :
- `npm list -global --depth 0`

# The Tooling (Important remove old angular version)

- If you have an old version of angular cli , then you have to remove the old one. Note the old package was called angular-cli
- The new package is called @angular/cli

You can run these two snippets to upgrade from angular-cli to the new package @angular/cli.

```
npm uninstall -g angular-cli  
npm cache clean  
npm install -g @angular/cli@latest
```

```
rm -rf node_modules dist  
npm uninstall --save-dev angular-cli  
npm install --save-dev @angular/cli@latest  
npm install  
ng update
```



# Angular Cli command

- `ng new <your app name>` i.e. `ng new mywebapp`
- (component creates a directory by default) i.e. `name/name.ts`
- Create a component : `> ng generate component name`
- (Modules, Directives, Services, Pipes – don't create a directory by default). But you can specify to i.e. `ng generate module mydir\name`
- Create a feature module : `> ng generate module name`
- Create a directive with : `> ng generate directive name`
- Create a service with : `> ng generate service name`
- Create a pipe with `> ng generate pipe name`

# Angular Cli command

- You can use shorts with single letters
- `ng new <your app name>` i.e. `ng n mywebapp`
- Create a component : `> ng g c name`
- 
- Create a feature module : `> ng g m name`
- Create a directive with : `> ng g d name`
- Create a service with : `> ng g s name`
- Create a pipe with `> ng g p name`

- `--app` (aliases: `-a`) default : 1st app. Specifies app name to use
- `--change-detection` (aliases: `-cd`) Specifies change detection strategy
- `--flat` default : false. - Flag to indicate if a dir is created
- `--export` default : false. - if declaring module exports the component.
- `--inline-style` (aliases: `-is`) default : false. Specifies if the style will be in the ts file
- `--inline-template` (aliases: `-it`) default : false. Specifies if the template will be in the ts file.
- `--module` (aliases: `-m`) Allows specification of the declaring module's file name (e.g 'app.module.ts').
- `--prefix` Specifies whether to use the prefix.
- `--skip-import` default : false. Allows for skipping the module import.
- `--spec` Specifies if a spec file is generated.
- `--view-encapsulation` (aliases: `-ve`). Specifies the view encapsulation strategy

# Angular Cli command options

- You can dry run a command with : `-d` short dry run
- Create a component : `> ng g c name -d`
- The `-d` command shows the files it will create, their full paths.  
if the files exist it will also inform you of that fact.
- 
- You can force a component to be created as a flat file, instead of being placed in a directory with `--flat`
- Create a component as a flatfile : `> ng g c name --flat`
- There are other options ...

# Angular Cli build command options

- We can serve our angular app, so it is built and run in memory with:
  - `ng serve -open`
  - or `ng serve -o`
  - `-o` is a shortcut
- We can build our angular with a development build with:
  - `ng build`
  - Or for production with :
    - `ng build --prod`

The mapping used to determine which environment file is used can be found in `.angular-cli.json`:

```
"environmentSource": "environments/environment.ts",
"environments": {
  "dev": "environments/environment.ts",
  "prod": "environments/environment.prod.ts"
}
```

These options also apply to the `serve` command. If you do not pass a value for `environment`, it will default to `dev` for `development` and `prod` for `production`.

```
# these are equivalent
ng build --target=production --environment=prod
ng build --prod --env=prod
ng build --prod
# and so are these
ng build --target=development --environment=dev
ng build --dev --e=dev
ng build --dev
ng build
```

# Angular Cli build options - highlight a few

- <https://github.com/angular/angular-cli/wiki/build>
- `--sourcemap` (aliases: `-sm`, `sourcemap`s)
- `--aot` (Build using Ahead of Time compilation)
- `--output-hashing` (aliases: `-oh`) Define the output filename cache-busting hashing mode.

## Bundling & Tree-Shaking

All builds make use of bundling and limited tree-shaking, while `--prod` builds also run limited dead code elimination via UglifyJS.

### `--dev` vs `--prod` builds

Both `--dev` / `--target=development` and `--prod` / `--target=production` are 'meta' flags, that set other flags. If you do not specify either you will get the `--dev` defaults.

Flag	<code>--dev</code>	<code>--prod</code>
<code>--aot</code>	false	true
<code>--environment</code>	dev	prod
<code>--output-hashing</code>	media	all
<code>--sourcemaps</code>	true	false
<code>--extract-css</code>	false	true
<code>--named-chunks</code>	true	false
<code>--build-optimizer</code>	false	true with AOT and Angular 5



# Break & Raffle

# Session 2

# Credit Card Processor Demo

- Run through some Angular Cli
- Demo Mock of a credit card machine – so the user enter a credit card, makes a transaction. We will mock a retailer's user setting to revokes the card machines ability to process transaction after 3 failed tries. i.e. to stop a customer locking their card. So we use all of the items below.
- Modules, Components, Directives, Services, Pipes
- One Way binding, two Way Binding [(ngModel)], events, @Input, @Output
- Directives @HostBinding, \*ngIf, \*ngFor
- Templates, external, inline, using template containers, ng-content.

In summary

- We looked at Modules, Components, Directives, Services, Pipes. Touched on the Demo – will expand upon that at a future meetup.
- We looked at the tooling, config files with AngularCli , typescript, Visual Studio Code, NodeJS and npm.
- Set down the basics for the next discussion covering Angular routing and back end services i.e. Nodejs and Rxjs in an upcoming meetup.