setup angular js with mvc

## create scaffoler for angular js yo aspnet

## empty out the layout @{ Layout = null; }

## add angular js to aspnet project from here https://www.pluralsight.com/blog/software-development/angularjs-for-asp-net-applications use

## add angulart to bower.json and then run bower install

## 8 powerful extensions from here: http://www.hongkiat.com/blog/visual-studio-code-extensions/

## Bundle together https://www.youtube.com/watch?v=wQaAACHj7w8

# Steps

## create a empty web app using yo asp

## fix warning with "pdb" file: by modify the project.json with buildoption add "debugType": "portable" to it

## adding new controller: by using yo subgenerator: yo aspnet --help

add a Controllers, Views/Home folder add HomeController.cs, Index.cshtml using yo modify the project.json to have MVC in dependencies "Microsoft.AspNetCore.Mvc":"1.0.1", modify Startup.cs in configureservice services.AddMvc(); and then add

app.UseMvc(routes => {  
 routes.MapRoute(  
 name:"default",  
 template:"{controller=Home}/{action=Index}/{id?}"  
 );  
});  
  
// app.Run(async (context) =>  
// {  
// await context.Response.WriteAsync("Hello World!");  
// });

and then add WebAPI GreetingsController to the project using yo and delete everything else except the Get method that returns a string

## intall typescript by npm npm install typescript@next -g

install typings tool (replaced tsd) npm install typings -g

## install webpack for loader (main entry) npm install webpack -g

## list npm user-installed packages from http://stackoverflow.com/questions/17937960/how-to-list-npm-user-installed-packages I do npm list -g --depth=0

## where npm packages installed npm root -g

## then install at local: npm install typescript@next --save-dev

it will be an error because we don't have package.json yet so we need to do npm init

--save-dev // for development environment

Now look at the folder structures: wwwroot : dotnet core Kestral will use to serve as client

let's create folder src/app then add greeter.ts file

add these lines:

/\*\* \* Greeter \*/ class Greeter { constructor(private message: string) {

}  
sayHello(){  
 console.log(`Hello ${this.message} from typescript`);  
}  
get greetingMessage(): string {  
 return `Hello ${this.message} from typescript`  
}

}

add tsconfig.json file to the root and these lines { "compilerOptions": { "target": "es5", "module": "commonjs", "inlineSourceMap": true, //for debug "inlineSources": true, "sourceRoot": "src/app", "outDir": "wwwroot/app"

}

}

and then we traspile at command lines

tsc -watch // everytime we save it will compile

and then we create a index.html under wwwroot folder

<meta charset="UTF-8">  
<title>Welcome to ASPNET Core 1.0</title>  
<script src="app/greeter.js"></script>

<div id="greeting">  
</div>  
<script>  
 var greeterObj = new Greeter("World");  
 document.getElementByID("greeting").innerHTML = "<h1>" + greeterObj.greetingMessage() + "/</h1>";  
</script>

add app.UseStaticFiles(); to startup.cs file

and modify the project.json file "Microsoft.AspNetCore.StaticFiles":"1.0.0",

before working on webpack, export keyword export is for other to use symbols (class, variable, function) export is to access scope

bootstrap == main == entry point in other C language appjs, aurelia for bootstrap we usually rely on systemjs, requirejs implementation so we use webpack to have the loader for the entry point!

### add main.ts

import { Greeter } from "./greeter" export /\*\* \* Main \*/ class Main { private greeter: Greeter; constructor() { this.greeter = new Greeter("world!"); } sayHello() { this.greeter.sayHello(); document.getElementById("greeting").innerHTML = "

" + this.greeter.greetingMessage + "/

"; } get greetingMessage(): string { return this.greeter.greetingMessage; } }

var m = new Main(); m.sayHello(); console.log(m.greetingMessage);

## config webpack to have an entry point run this command webpack ./wwwroot/app/main.js ./wwwroot/app/bundle.js

## install a nice console emulator http://cmder.net/

after that clean up index.html and then modify in the script section

## we can stop the typescript watch loader too and use webpack !!!!

## explore webpack webpack --help

add new file: webpack.config.js on the root

and search in chrome: awesome typescript loader npm install awesome-typescript-loader --save-dev

then copy this to webpack.config.js

module.exports = { entry: "./src/app/main.ts", output: { path: \_\_dirname + "/wwwroot/app", filename: "bundle.js" },

// Currently we need to add '.ts' to the resolve.extensions array. resolve: { extensions: ['', '.ts', '.tsx', '.js', '.jsx'] },

// Source maps support ('inline-source-map' also works) devtool: 'source-map',

// Add the loader for .ts files. module: { loaders: [ { test: /.ts$/, loader: 'awesome-typescript-loader' } ] } };

then in the tsconfig.json, we need to change to this { "compilerOptions": { "target": "es5", "module": "commonjs", //"inlineSourceMap": true, //for debug "sourceMap": true, // "inlineSources": true, "sourceRoot": "src/app", "outDir": "wwwroot/app"

}

}

### Aurelia I "download the Skeletons" from aurelia.io website

in video https://www.youtube.com/watch?v=mfbVRKuGhxw part 4 says that copy aurelia Skeleton navigation https://github.com/aurelia/skeleton-navigation

copy typescript webpack (not with aspnet)

Downloading aurelia skeleton navigation After that copy four folders controllers properties views WWW route  
And docker file, Project json ,program ,read me ,start up And inside not VS code, launch, tasks Done and now we can run the program by command line Npm install Now we can copy greeter

Easy web pack If greater JS is a viewmodel, we need a view in html, so we create a new file

Use Emmet template div h1 input type text

Use string interpolation with $bracket

Bootstrap 1. Use declarative to body tag with aurelia-app 2. Programmatic api Modify main.ts

Aurelia fetch api for rest

They are running on different ports, domains so we use cors

Modify project.jaon Add "Microsoft.aspnet.cors"

And startup.cs with Services.addcors($

And app.usecors(builder=>builder.withorigins("localhost:9000") .allpwanyheader().allowanymethod());

Aurelia has di injector Insert these lines in greeter.js import {inject} from 'aurelia-framework '; Import {HttpCluent} from 'aurelia-fetch-client'}

And then decorate this @inject(HttpClient)