Setup a mvc project

Webpack has typescript loader (awesome webpack)

Vs webpack react knockout

<http://blog.stevensanderson.com/2016/05/02/angular2-react-knockout-apps-on-aspnet-core/>

1. Reference
   1. From pluralsight course:
   2. + Angular + webpack + Aurelia (series) <https://www.youtube.com/watch?v=wQaAACHj7w8>
   3. list npm user-installed packages from <http://stackoverflow.com/questions/17937960/how-to-list-npm-user-installed-packages>
   4. install a nice console emulator <http://cmder.net/>
   5. skeleton navigation <https://github.com/aurelia/skeleton-navigation>
2. Setup
3. intall typescript by npm: npm install typescript@next -g
4. install typings tool (replaced tsd): npm install typings -g
5. install webpack for loader (main entry): npm install webpack -g
6. list all packages globally: npm list -g --depth=0
7. location of packages: npm packages installed npm root –g
8. tslint
9. Steps
10. create web app for angular js: yo aspnet (we can also run: bower install)
11. adding new controller: by using yo subgenerator: yo aspnet –-help
12. modify project.json:

“Microsoft.AspNetCore.Mvc":"1.0.1"

1. config DI (dependency injection) in startup.cs

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!");  
// });

1. add WebAPI GreetingsController to the project using yo and delete everything else except the Get method that returns a string “Hello World”
2. npm init
3. install at local: npm install typescript@next --save-dev
4. npm install typings --save-dev
5. it will be an error because we don't have package.json yet : npm init --save-dev // for development environment
6. now look at the folder structures: wwwroot : dotnet core Kestral will use to serve as client
7. let's create folder src/app then add greeter.ts file
8. 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` }

### }

1. 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"

### }

### }

1. we traspile at command lines: tsc -watch // everytime we save it will compile
2. 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>

1. add app.UseStaticFiles(); to startup.cs file
2. modify the project.json file "Microsoft.AspNetCore.StaticFiles":"1.0.0",
3. 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);

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

18. the typescript watch loader too and use webpack !!!!

19. explore webpack webpack --help

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

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

22. 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"

### }

### }

23. 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

24. run the program: npm install

25. now we can copy greeter

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

27. use Emmet template> div> h1> input type text

28. use string interpolation with $bracket

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

30. publish code:

dotnet publish

--framework netcoreapp1.0

--output "./bin/Release/PublishOutput2"

--configuration Release

dotnet publish --framework netcoreapp1.0 --output ".\bin\Release\PublishOutput2" --configuration Release

31. use default page:

// default page

DefaultFilesOptions options = new DefaultFilesOptions();

options.DefaultFileNames.Clear();

options.DefaultFileNames.Add("index.html");

app.UseDefaultFiles(options);

from <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/static-files>

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)