Notes for dating app project

.net core & angular

# Downloaded

.net core 3

Postman

Sqllitebrowser

Node.js

# VS EXTENSTIONS INSTALLED

C#

C# Extensions

# Section 1

To install a dotnet web api application type the following in the terminal: dotnet new webapi -n datingapp

-h for help e.g. dotnet new -h

# Section 2

To exclude certain folders in vscode e.g. bin folder go to

Vscode > preferences > settings

Once there search exclude and click on the files section. Add relevant path to exclude e.g \*\*/bin

in the terminal in vscode type ‘dotnet run’ to run application

To prevent https in production, remove settings in the launchsettings.json file (this is set in the application url key) and comment out trigger in the Startup class.

To see a data returned from the controller add the controller into the url e.g.

Localhost:5000/weatherforecast

How does the application run?

every dotnet core application has a program.cs file. The application looks for the program class and runs the main method

ConfgureServices() method inside Startup class is the method used to add a service

Middleware is software we use to interact with http request as it journeys through the pipeline

The Configure method inside the startup class contains middleware.

Csproj file identifies which framework we are targeting

DatingApp.API in the launchSettings.json file identifies the launch settings when you type ‘dotnet run’ in the terminal

In development mode, appsettings.Development.json will exist. This file contains extra information that will be injected into the logs e.g. logging

To switch between development and production mode, open the launchSettings.json, locate the “DatingApp.API” object and change “APNETCORE\_ENVIRONMENT” to “Production”

In devtools chrome, under the network tab check option ‘Preserve log’ to help debug exception errors in production mode.

‘Dotnet watch run’ – type this into the terminal to save you from closing then running dotnet everytime you make a change.

.NetCore 3.0 removes Microsoft.AspNetCore.App meta package. This package includes various assemblies including entity framework core (used for DbContext)..need to add manually

If you have an error hover over error and press ( cmd . ) – shortcut fix

### Process to add DbContext into project for seeding

1 ) create a model file

2) Create a file (DataContext)that derives from DbContext. DbContext is only available from the entityframework package so you need to install this from nugget. Search the following ‘’ Microsoft.EntityFrameworkCore” in the nugget extension. Once installed the file that derives from DbContext will look like this

using DatingApp.API.models;

using Microsoft.EntityFrameworkCore;

namespace DatingApp.API.Data

{

public class DataContext : DbContext

{

public DataContext(DbContextOptions<DataContext> options) : base(options)

{

}

public DbSet<Value> Values { get; set; }

}

}

This file uses the model file also. In this case ‘Value’ is the model file Value.cs which is located in the Model folder.

3) The third step are included in the notes below. Provides info on how to inject the DataContext into the startup file and link to a SQl string e.g.

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<DataContext>(x => x.UseSqlite(Configuration.GetConnectionString("DefaultConnection")));

}

4) To get data from DB and add them to the ui, inject the service into the controller

public class ValuesController : ControllerBase

{

private readonly DataContext \_context;

public ValuesController(DataContext context)

{

this.\_context = context;

}

Adding a service to the DB is created in the startup.cs file.

The startup class includes a Configuration injection inside its constructor. This lets you access configuration from the appsettings.json file e.g.

Configuration.GetConnectionString() lets you get the ConnectionStrings config in the appsettings.json file.

NOTE ‘ConnectionStrings’ needs to be typed as that in the json file because GetConnectionString() looks for ConnectionStrings

Use to install entity framework tools. Version depends. For latest version remove version command

dotnet tool install --global dotnet-ef --version 3.0.0

Use dotnet ef to scaffold the database.

Creates a new migration file

dotnet ef migrations add InitialCreate

(InitialCreate will be the name of the file. Can be whatever you want)

Migration folder contains the migration classes which is based on the DataContext class (our derived DbContext class)

Entity framework automatically identifies the Id property in our model is going to be the primary key.

Migration file ending with \_initialcreate.cs in the migration folder contains the information of our db schema. This will automatically include the properties in the model. CreateTable method inside the up method in this file shows us the name of the table, columns and constraints

If db is not created yet ‘dotnet ef database update’ will create the db based on the created migration file. So this needs to occur after creating the migration file. This will also create a new .db file you can see in the root directory. To view this file you need to open it in Db Browser for SQLite

Can use postman to display the values that are returned from the server. REAL EASY UI.

To search click new tab the select the http request using e.g. get or post.

204 response is a 200 response with no content so basically returning null. FirstOrDefault() does this. First() returns an exception

Think about scalability when creating an application. Try to use asynchronous code instead of synchronous code. Synchronous code blocks the thread/application until that request is completed e.g. synchronous code to get values from a db in the controller can take a few seconds depending on the scale of the db and method.

Synchronous method in the controller

public IActionResult GetValues()

{

var values = \_context.Values.ToList();

return Ok(values);

}

Asynchronous version of the above method

public async Task<IActionResult> GetValues()

{

var values = await \_context.Values.ToListAsync();

return Ok(values);

}

Cant include (-) when naming .net files.

When creating new angular project, create it outside the .DatingApp.API folder

**Understanding Angular Directory**

Node\_modules contains all the dependencies in the Angular project. What is inside the node\_modules folder depends on the package.json file. Saying this you don’t need to push the node\_modules folder into source control. You just need to push package.json

Angular injects all javascript files into the index.html via webpack. Angular.json locates whats files to inject too.

Extensions installed for angular

* Angular2 switcher – Lets you switch between related component files quicker. On mac (shift + alt + (i, u , o))
* Angular files
* Angular Language service
* Angular Snippets
* Auto Rename tag
* Bracket Pair color
* Debugger for chrome
* Material Icon theme
* Prettier – code formatter
* TSLint

Observables are a stream of data you get from the api but in order to get data from an observable to need to subscribe to an observable.

To create an API request from angular to the backend you need to

1. Add HTTPClientModule into the imports array in the app.module file . Import

import { HttpClientModule } from '@angular/common/http';

1. Inject the service to a component you want to retrieve the data from the api. The service should be of type HttpClient and import this from

import { HttpClient } from '@angular/common/http';

constructor(private http: HttpClient) { }

1. You then need to retrieve the data from the service

this.http.get('http://localhost:5000/api/values').subscribe(response => {

this.values = response;

}, error => {

console.log(error);

});

The http property has access to get() and post() methods. In this method input the string of the api you want to get data from. Both methods return an observable JSON file and in order to access data in a observable you need to subscribe to it hence the subscribe method. Saving data directly from an observable wont work.

After completing the above steps and running the Angular app, THIS WONT WORK WHICH IS NORMAL BEHAVIOUR. THIS WONT WORK BECAUSE YOU NEED A CORS POLICY (angular runs on port 4200 & .net runs on port 5000. Browsers think these are different domains)

To add CORS you need to add it as a service then add as middleware.

Ordering in the ConfigureServices() method is not important but ordering in the Configure() method is. Both methods are in the startup class. First add services.AddCors() into the services method

Add UserCors(), in the exact position as below inside Configure(). Reminder Confiugure() contains the middleware. This is basic implementation for development. Will tighten security in production.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

// app.UseHttpsRedirection();

app.UseRouting();

app.UseCors(x => x.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

You can add additional global styles to the application by adding the style sheet into angular.json or import additional style sheets to the global style.css file by using the @import keyword.

PACKAGE.JSON – INCLUDES PROJECT DEPENDENCIES. ‘Npm install’ will install all dependencies located in this file

ANGULAR.JSON – INCLUDES FILES THAT ARE INCLUDED WHEN BOOTING APPLICATION

### Adding both angular and .net to the same GIT repo

When creating a new Angular project, it auto creates GIT aswell. .gitignore file lists all the files git will ignore

1. Locate hidden git folder in angular application and remove. Don’t delete .gitignore and .editorconfig
2. Type ‘Git init’ into the terminal, whilst in parent repo of both projects.
3. Create a new .gitignore file inside the DatingApp.API project.