Notes for dating app project

.net core 3 & angular

# Downloaded

.net core 3

Postman

Sqllitebrowser

Node.js

# VS EXTENSTIONS INSTALLED

C#

C# Extensions

# Section 1 intro

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. -n for name

# Section 2 Building a walking skeleton

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 e.g. a user class with properties of id and name

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 injected 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)

\*Run the above command after creating a new model, and adding that model to the DataContext file. This creates a new schema based on the model. Then to update the database with the new table type ‘dotnet ef database update’

Migration folder contains the migration classes which is based on the DataContext class (our derived DbContext class). How does it know what models to use in our migration file? It gets this information form our DataContext.cs file that derives from dbContext.

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. Methods with async keywords can use FirstOrDefaultAsync()

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.
4. Add following file types to the new gitignore file

* .vscode
* bin
* obj
* \*.db

1. Click on + sign to stage changes then cmd + enter to commit changes locally
2. If a file already exists in the repo but then added to gitignore you need to keep the cached version to remove it from source control

Typoe ‘git rm appsettings.json –cached’

# Section 3 Security

Don’t store passwords as clear text in the DB. Hash and salt the password. Because hashing the password will create the same encrypted text for the same password used by multiple different users, salting changes the encrypted text.

Salt is added to a password before hashing takes place.

#### Repository pattern

Contains a repository interface that allows different controllers access different methods that update the database. It removes replicate code as this one repository interface can be accessed by multiple controllers instead of multiple controllers access multiple repository interfaces that would contain the same code to access the db.

Means we know all our DB queries are in the same place

Decouples application from framework

Interfaces start with I in c#

Generics types are used for functions and classes. Properties use normal types

Implementing repository pattern.

Example for login

Create IAuthRepository interface and implement methods inside this interface. The methods with return a Task of type User.

create a new authorization class and inherit the interface created above. Inject the DataContext file into this class and assign the inijected data into a private field. Fields normally start with an underscore

The out keyword changes the value types passed to a function, into reference types e.g.

Number digit;

Sum(out digit)

Digit is of type number so value type. Out changes it to reference type in the function call.

var hmac = new System.Security.Cryptography.HMACSHA512()

above method is how we are going to incrypt the password as it returns a hash version of our data. This method is in a class that inherits IDisposable. Anything that inherits this class lets you dispose of any data when the call is finished. To start with you add the method call into the using method parentheses.

await \_context.Users.AddAsync(user); // add new data to datacontext

await \_context.SaveChangesAsync(); // save changes back to db

services.AddScoped<IAuthRepository, AuthRepository>();

This line of code that was added in the start up class is how you make a service make a service available to the entire app.

Yes you can inject by adding the service through a constructor but first you need to make the application aware of the service. This is what this line of code is doing. Then you can inject the service into a class

AddScoped() adds a scoped service. Alternatives are AddSingleton() and AddTransient()

Controllers that inherit ControllerBase are controllers without view support.

Controller that’s inherit Controller are controllers with view support. Controllers in this project inherit ControllerBase because the front end is created by angular.

DTO – Data transfer Object. – is used to map our domain models e.g. The user model class into simpler objects that get returned to the view. I created a dto class to container the username and password properties that are sent when creating a new user

Store DTO classes in its own dto folder

Postman is very powerful. You can send data to the database by postman if you have no UI so it’s a great tool. E.G I used postman to register a new user by directly sending data to the method in the auth controller. How did I do It?

1. Select HTTP method GET/post
2. Enter url to the controller
3. Select Body option in postman
4. Select raw
5. Change text type to JSON
6. Enter data the controller takes (or the properties the DTO takes if used as the parameter for the controller)in JSON format.

BLOB is a Binary large object

Add validation into the DTO class

The required attribute ‘[Required]’ can be added to properties to ensure they have a value.

The DataAnnotations namespace provides all attribute class available. System.ComponentModel.DataAnnotations

[StringLength(8,MinimumLength = 4, ErrorMessage = "You Must specify password between 4 and 8 characters")]

In this attribute the first argument ‘8’ is required by the constructor. The 2nd and 3rd arguments are properties of the string length class and are not required during initialization.

[ApiController] attribute helps returns helpful errors. You place this attribute just before the controller class

Errorwithout ApiController

System.NullReferenceException: Object reference not set to an instance of an object.

at DatingApp.API.Controllers.AuthController.Register(UserForRegisterDto userForRegisterDto) in /Users/mila/Documents/Dating App/DatingApp.API/Controllers/AuthController.cs:line 29

at Microsoft.AspNetCore.Mvc.Infrastructure.ActionMethodExecutor.TaskOfIActionResultExecutor.Execute(IActionResultTypeMapper mapper, ObjectMethodExecutor executor, Object controller, Object[] arguments)

at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.<InvokeActionMethodAsync>g\_\_Awaited|12\_0(ControllerActionInvoker invoker, ValueTask`1 actionResultValueTask)

at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.<InvokeNextActionFilterAsync>g\_\_Awaited|10\_0(ControllerActionInvoker invoker, Task lastTask, State next, Scope scope, Object state, Boolean isCompleted)

at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Rethrow(ActionExecutedContextSealed context)

at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(State& next, Scope& scope, Object& state, Boolean& isCompleted)

at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeInnerFilterAsync()

--- End of stack trace from previous location where exception was thrown ---

at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeFilterPipelineAsync>g\_\_Awaited|19\_0(ResourceInvoker invoker, Task lastTask, State next, Scope scope, Object state, Boolean isCompleted)

at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeAsync>g\_\_Logged|17\_1(ResourceInvoker invoker)

at Microsoft.AspNetCore.Routing.EndpointMiddleware.<Invoke>g\_\_AwaitRequestTask|6\_0(Endpoint endpoint, Task requestTask, ILogger logger)

at Microsoft.AspNetCore.Authorization.AuthorizationMiddleware.Invoke(HttpContext context)

at Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware.Invoke(HttpContext context)

Error with ApiController attribute

{

    "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",

    "title": "One or more validation errors occurred.",

    "status": 400,

    "traceId": "|21ab6403-406b6e76590f42ea.",

    "errors": {

        "Password": [

            "The Password field is required.",

            "You Must specify password between 4 and 8 characters"

        ],

        "Username": [

            "The Username field is required."

        ]

    }

}

[FromBody] bounds the data passed to the original properties in the type specified as the argument

public async Task<IActionResult> Register([FromBody]UserForRegisterDto userForRegisterDto)

In this instance UserForRegisterDto has 2 string properties for username and password. If you was to pass EMPTY data to this controller method without FromBody attribute, the data will be null but if you was to pass empty data to this controller method WITH FromBody, the data will be an empty string – so binds the data type to the type used in the class

If im not using the ApiController attribute I need to use the model state to validate, in order to get the same error I get when using the ApiController.

ModelState is a class that identifies if the data passed to an Action has successfully binding to the model used. In this case the model is UserForRegisterDto

public async Task<IActionResult> Register([FromBody]UserForRegisterDto userForRegisterDto)

{

// validate request

if(!ModelState.IsValid)

{

return BadRequest(ModelState);

}

Token authentication used in this project

* JSON Web Tokens
* Industry Standard for tokens (RFC 7519)
* Self contained and can contain:
  + Credentials
  + Claims
  + Other information

JWT structure (Json wen tokens)

Header – specifies the typee e.g. ‘typ’ : ‘jwt’ ‘alg’ : ‘HS512; alg = algorithm’

Payload – contains information stored inside token. Payload and header info can be decoded by anyone. Be careful with data stored here

Secret – used to encode / hash payload and header. The secret is saved in the server.

Token Authentication steps

1 – User sends username and password to server

2 – server validates and sends back JWT to client. JWT saved locally

3 – client sends jwt for further requests to server

4 – Server validates JWT and sends back response

To start with jwt in .net 3.0 you need to install some packages.

<PackageReference Include="Microsoft.IdentityModel.Tokens" Version="6.6.0"/>

<PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="6.6.0"/>

The claim class is used pass a clients claim of an existing username.

var claims = new[]

{

new Claim(ClaimTypes.NameIdentifier, userFromRepo.Id.ToString()),

new Claim(ClaimTypes.Name, userFromRepo.UserName)

};

IConfiguration interface lets you get data from config files. E.g.

Appsettings.json file below

{

"AppSettings" :{

"Token" : "super secret key"

},

"ConnectionStrings": {

"DefaultConnection" : "Data Source=datingapp.db"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

NOTE when committing to your repo, you don’t want to include the appsettings token which is sensitive information. A way to do this

1. Add appsettings to gitignore file.
2. Environment Variables – used more on production mode
3. User secrets – dotnet core lets you save information as a super secret key in a secret store on your local machine. It will be saved in memory. This is for use in development mode only. You wont access to this super secret key in production mode

In the line of code below we are getting the Appsettings token key’s value above

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config.GetSection("AppSettings:Token").Value));

To use IConfiguration inject into controller

public AuthController(IAuthRepository repo, IConfiguration config)

{

this.\_config = config;

this.\_repo = repo;

// In this controller we are injecting

}

Token should be a min of 12 characters randomly generated. If a hacker gets this info they can pretend to be users.

WHOLE LOGIN METHOD

public async Task<ActionResult> Login(UserForLoginDto userForLoginDto)

{

// check username and password is stored in db

var userFromRepo = await \_repo.Login(userForLoginDto.Username.ToLower(), userForLoginDto.Password);

if (userFromRepo == null)

{

return Unauthorized();

}

// building token

// 2 claims for token

var claims = new[]

{

new Claim(ClaimTypes.NameIdentifier, userFromRepo.Id.ToString()),

new Claim(ClaimTypes.Name, userFromRepo.UserName)

};

// to make sure tokens are valid when they come back the sever needs to sign this token. thats what key and creds is doing

// store key in appsettings because we'll use it in a number of places

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config.GetSection("AppSettings:Token").Value));

// create signing credientials key - takes key created above and algo we use to hash the key

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);

// create a security token descriptor. Create the token. contain our claims, expiring date for token and signing credentials

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(claims),

Expires = DateTime.Now.AddDays(1),

SigningCredentials = creds

};

// create jwt

var tokenHandler = new JwtSecurityTokenHandler();

var token = tokenHandler.CreateToken(tokenDescriptor);

return Ok(new {

token = tokenHandler.WriteToken(token)

});

}

‘Jwt.io’ is a useful site you can use to decode a token sent back from the server

Payload in jwt meaning

{

"nameid": "1",

"unique\_name": "john",

"nbf": 1591182177, - Not Before

"exp": 1591268577, - Expiration date

"iat": 1591182177 – Issued At Date

}

To test go to postman and send a request to the login action. obviously with the data the login action require seg. Username and password in json format. You should receive a token as a response. Paste this token into jwt.io to view the jwt.

Now we get a token back from the server what do we do with it? We need to authenticate it

Install package via nuget. Needed to authenticate tokens

<PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="3.0.0"/>

Can start with adding the [Authorize] attribute to a controllers class. Actions with the [AllowAnonymous] inside this controller will bypass the authorize attribute.

This is our Authentication scheme our application will use

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(

options => {

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.

GetBytes(Configuration.GetSection("AppSettings: Token").Value)),

ValidateIssuer = false,

ValidateAudience = false

};

}

);

Now to add it to the HTTP request Pipeline. Add ‘add.Authentication()’ just after Cors in the startup.cs file

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

app.UseAuthentication();

app.UseAuthorization();

Now test authentication with postman

1. Access controller action with AllowAnonymous attribute – should pass
2. Access controller action without AllowAnonymous attribute – should get 401 error. failed and require authorization
3. On postman go to login url and login with a valid user. Enter login credentials as JSON. A token will be sent back. Copy this token not including double quotes
4. Go back to login url on postman and you need to add the token into the header with the key value ‘Authorization’. The value is of type Bearer so in the Value column you enter (without the quotes and angle brackets) ‘Bearer <token>’. Token is the token you copied.
5. You will then get access to actions that don’t have the AllowAnonymous attribute