Contents

[CMD command for creating dot net application 1](#_Toc27670857)

[Create Project (Login, Registration) 3](#_Toc27670858)

[Seeding Data 12](#_Toc27670859)

[Basic angular setup 14](#_Toc27670860)

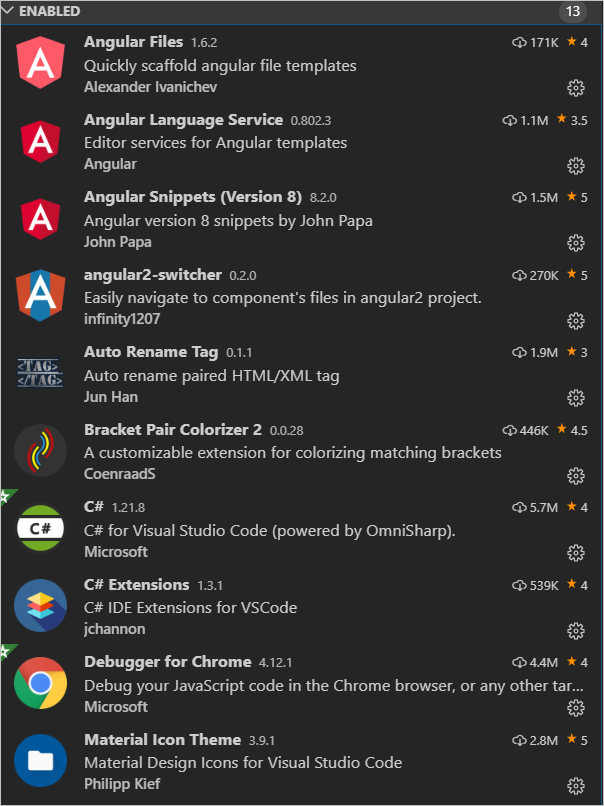
[Error handling 17](#_Toc27670861)

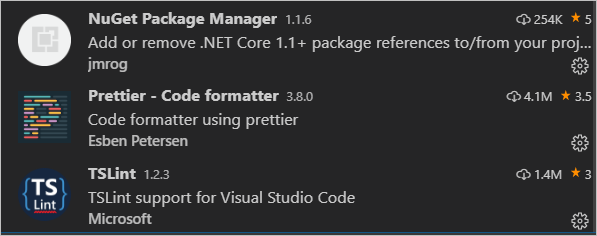
[Deployment 18](#_Toc27670862)

# CMD command for creating dot net application

* **dotnet -info** -> provides information about dot net sdk.
* **dotnet -h**
* **dotnet new -h**
* **dotnet new webapi -h**
* **dotnet new webapi -n <udApp.API>** -> creates an webApi project <name>
* **dotnet watch run** -> to run dot net core with watcher in vs code.
* **dotnet run** -> for normally running app

Some vs code extensions

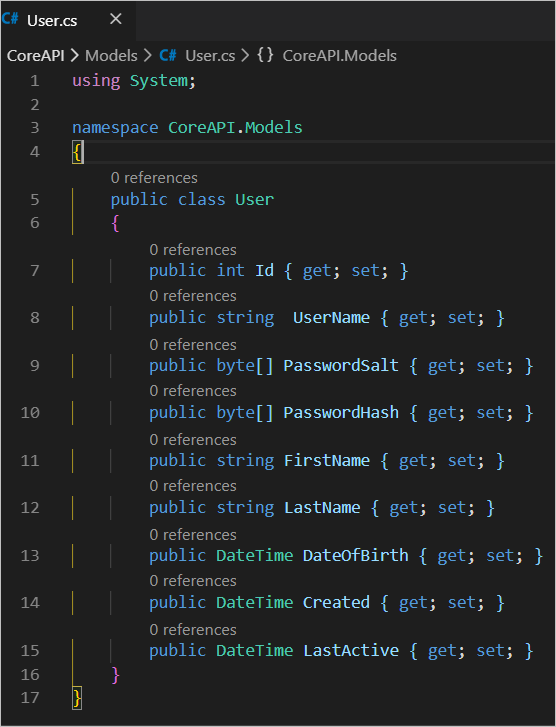




# Create Project (Login, Registration)

Create a project, code first approach as core supports only code first approach (**dotnet new webapi -n <udApp.API> )**

1. Add a folder Model and a class **User**.



1. Add a folder Repositories and class Dbcontext **(Install Microsoft.EntityFrameworkCore)**

using CoreAPI.Models;

using Microsoft.EntityFrameworkCore;

namespace CoreAPI.Repositories

{

public class DataContext: DbContext

    {

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

       public DbSet<User> Users { get; set; }

    }

}

1. **appSettings.Json Add DB connection.**

{

  "ConnectionStrings": {

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

  },

 "AppSettings": {

    "Token": "super secret key"

  },

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft": "Warning",

      "Microsoft.Hosting.Lifetime": "Information"

    }

  },

  "AllowedHosts": "\*"

}

Make sure Connection string is pluralized i.e. **ConnectionStrings**.

1. In Startup.cs in **ConfigureServices**  add connection string (**Microsoft.EntityFrameworkCore.Sqlite/ Microsoft.EntityFrameworkCore.SqlServer**), **Microsoft.EntityFrameworkCore.Design**

  public void ConfigureServices(IServiceCollection services)

        {

            // local

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

            // server

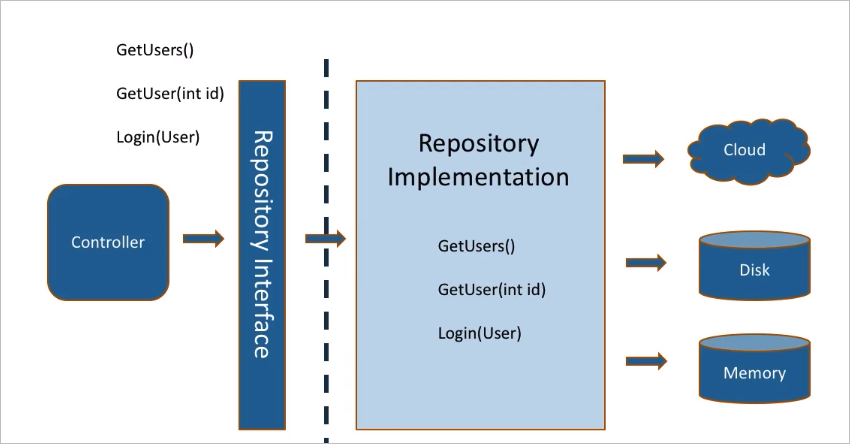
     services.AddDbContext<DataContext>(x => x.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

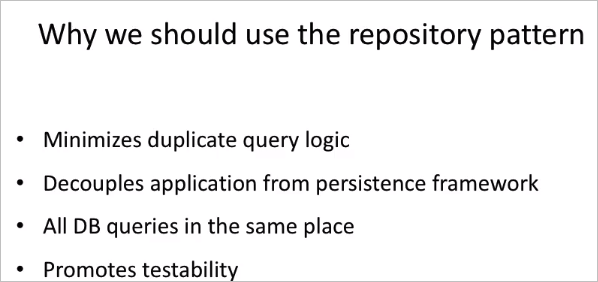
}

**DataMigration commands:-**

* dotnet tool install --global dotnet-ef --version 3.0
* dotnet ef migrations -h
* dotnet ef migrations add <InitialCreation> (install nuget Microsoft.EntityFrameworkCore.Design)
* dotnet ef database update (if we add **dotnet ef database update <InitialCreation**> then it will drop table)
* dotnet ef migrations add <UserEntityAdded> (whenever new class is added run to update DB)

**Repository Pattern**





**Working with Login and Registration: (**there are many ways for password authentication)

1. Create Interface and its implementation

 public interface IAuthRepository

    {

         Task<User> Register(User user, string password);

         Task<User> Login(string userName, string password);

         Task<bool> UserExist(string userName);

    }

public class AuthRepository : IAuthRepository

    {

        public DataContext \_context;

        public AuthRepository(DataContext context)

        {

            \_context = context;

        }

        public async Task<bool> IsUserExist(string userName)

        {

            if (await \_context.Users.AnyAsync(a => a.UserName == userName))

                return true;

            return false;

        }

        public async Task<User> Login(string userName, string password)

        {

            var user = await \_context.Users.FirstOrDefaultAsync(a => a.UserName == userName);

            if (user == null)

                return null;

            if (!VerifyPasswordHash(password, user.PasswordHash, user.PasswordSalt))

                return null;

             return user;

        }

        private bool VerifyPasswordHash(string password, byte[] passwordHash, byte[] passwordSalt)

        {

            using(var Hmac = new System.Security.Cryptography.HMACSHA512(passwordSalt))

           {

               var computedHash =  Hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));

                for (int i = 0; i < computedHash.Length; i++){

                    if (computedHash[i] != passwordHash[i])

                        return false;

                }

           }

           return true;

        }

       public async Task<User> Register(User user, string password)

        {

            byte[] passwordHash , passwordSalt;

            CreatePasswordHash(password,out passwordHash, out passwordSalt);

            user.PasswordHash = passwordHash;

            user.PasswordSalt = passwordSalt;

            await \_context.Users.AddAsync(user);

            await \_context.SaveChangesAsync();

            return user;

        }

private void CreatePasswordHash(string password,out byte[] passwordHash, out byte[] passwordSalt)

        {

           using(var Hmac = new System.Security.Cryptography.HMACSHA512())

           {

               passwordSalt = Hmac.Key;

               passwordHash = Hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));

           }

        }

    }

Dto Object for Registration

    public class UserForRegistrationDto

    {

        [Required]

        public string UserName { get; set; }

        [Required]

        [StringLength(8,MinimumLength = 4, ErrorMessage = "Password must be between 4 to 8 character")]

        public string Password { get; set; }

        public string FirstName { get; set; }

        public string LastName { get; set; }

    }

**Controller with dependency injection**

**Note:** Inherit from **ControllerBase** and [**ApiController**] filter

namespace CoreAPI.Controllers

{

    [ApiController]

    [Route("[api/controller]")]

    public class AuthController : ControllerBase

    {

        private IAuthRepository \_repo;

        private IConfiguration \_config;

        public AuthController(IAuthRepository repo, IConfiguration config)

        {

            \_repo = repo;

            \_config = config;

        }

        [HttpPost("register")]

        public async Task<IActionResult> Registration(UserForRegistrationDto userDto)

        {

            //check is user exist

           if(await \_repo.UserExist(userDto.UserName.ToLower()))

           {

               return BadRequest("User all ready exist");

           }

           var user = new User();

           user.UserName = userDto.UserName;

          var createdUser = await \_repo.Register(user, userDto.Password);

          return StatusCode(201);

        }

        [HttpPost("login")]

        public async Task<IActionResult> Login(UserForLoginDto obj)

        {

            var userFromRepo = await \_repo.Login(obj.UserName.ToLower(), obj.Password);

            if (userFromRepo == null) {

                return Unauthorized();

            }

            var claim = new[] {

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

            new Claim(ClaimTypes.Name,userFromRepo.UserName),

            };

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

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

            var tokenDesc = new SecurityTokenDescriptor

            {

                Subject = new ClaimsIdentity(claim),

                Expires = DateTime.Now.AddDays(1),

                SigningCredentials = cred,

            };

            var tokenHandler = new JwtSecurityTokenHandler();

            var token = tokenHandler.CreateJwtSecurityToken(tokenDesc);

            return Ok(new

            {

                token = tokenHandler.WriteToken(token)

            });

        }

    }

}

**JWT Token is used for Login (System.IdentityModel.Tokens.Jwt, Microsoft.AspNetCore.Authentication.JwtBearer)**

Used Configuration to read (\_config.GetSection("AppSettings:Token").Value) -> It defined secret Key.

Once User login send Token to client.

**JWT in startup.cs**

**Adding repository to application -> register it in startup class**

 public void ConfigureServices(IServiceCollection services)

        {

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

**services.AddScoped<IAuthRepository,AuthRepository>();**

            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,**

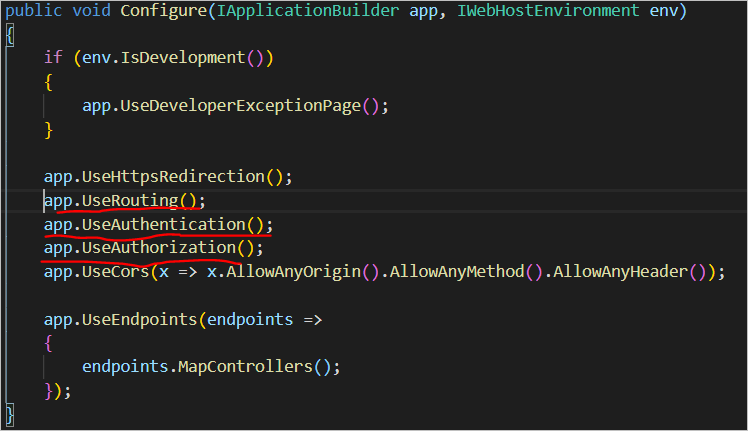
**ValidateAudience = false**

**};**

**});**

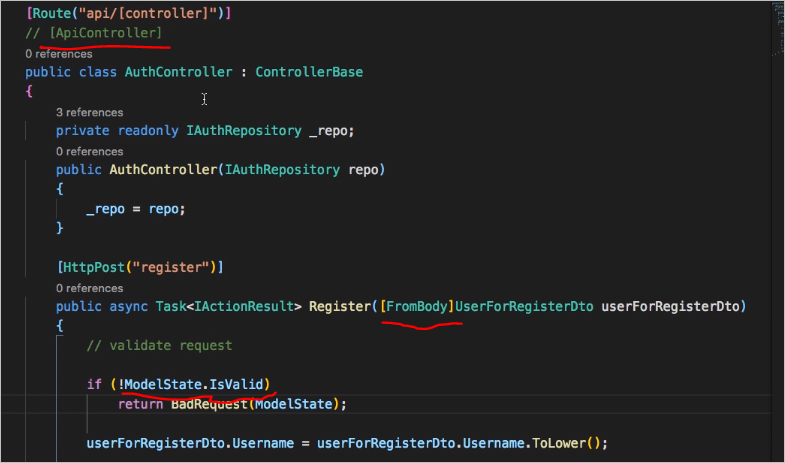
            services.AddControllers();

        }



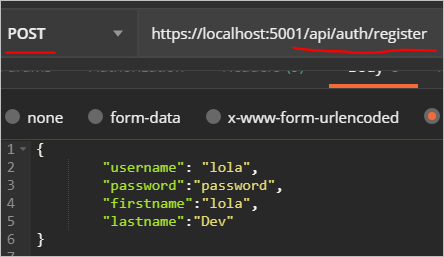
Add for authentication.

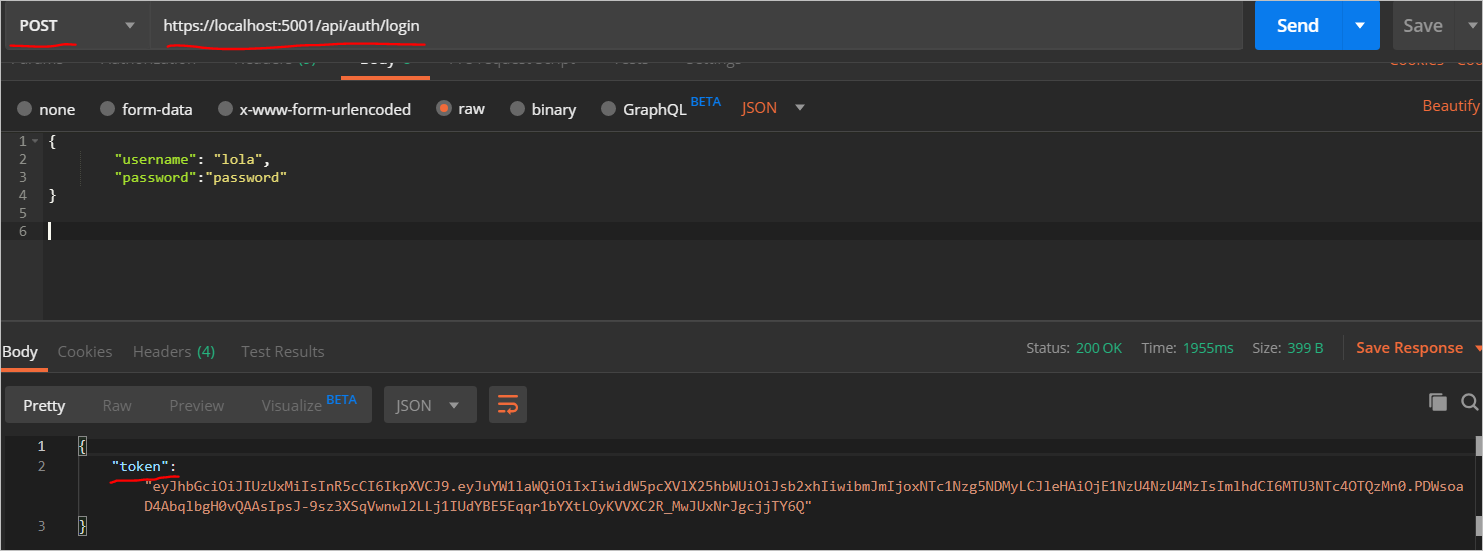
**Note**: if we remove [ApiController] we need to validate model state and provide input source like formbody



Now Run your App -> dotnet run

Test using Postman.

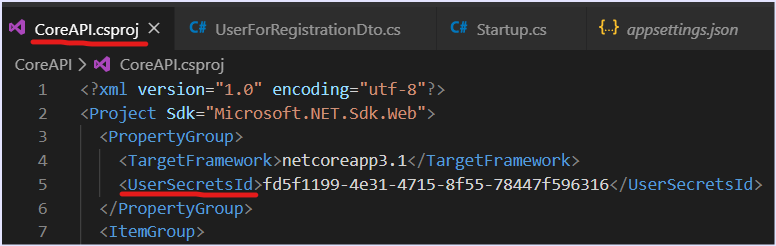




**Storing Secrets:**

* AppSettings
* Environment Variable
* Dot net user secrets(stored on development machine only) : **dotnet user-secrets init**

Will add a line in csproj (Guid)



dotnet user-secrets set "AppSettings:Token" "super secret key"

now instead of dot net reading appsettings token value from appSettings it will read from dotnet secret.

# Seeding Data

Before running make, sure DB is created (**dotnet ef migrations add InitialCreation** , **dotnet ef database update**)

Create a static class which reads json file and call that class in Program.cs before Run to seed the DB.

public class Seed

    {

     //Create a static method as we will never use it more than once, hence we also keep it sync.

        public static void SeedUser(DataContext dataContext)

        {

            if (!dataContext.Users.Any())

            {

             var userData = System.IO.File.ReadAllText("Repositories/UserSeedData.json");

                var users = JsonConvert.DeserializeObject<List<User>>(userData);

                foreach (var user in users)

                {

                    //Modify or add those values which are not part of json file but we need to have that in our DB like password salt and password hash

                    byte[] passwordHash, passwordSalt;

                    CreatePasswordHash("password", out passwordHash, out passwordSalt);

                    user.PasswordHash = passwordHash;

                    user.PasswordSalt = passwordSalt;

                    user.UserName = user.UserName.ToLower();

                    dataContext.Users.Add(user);

                }

                dataContext.SaveChanges();

            }

        }

         private static void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)

        {

           using(var Hmac = new System.Security.Cryptography.HMACSHA512())

           {

               passwordSalt = Hmac.Key;

               passwordHash = Hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));

           }

        }

}

public static void Main(string[] args)

        {

          var host = CreateHostBuilder(args).Build();

            using (var scope = host.Services.CreateScope())

            {

                var services = scope.ServiceProvider;

                //Add try catch block to handle error

                try

                {

                    var context = services.GetRequiredService<DataContext>();

                    context.Database.Migrate();

                    Seed.SeedUser(context); //Seed class calling SeedUser with context.

                }

                catch (Exception ex)

                {

                    var logger = services.GetRequiredService<ILogger<Program>>();

                    logger.LogError(ex, "An Error occured during Data Seed");

                }

            }

            host.Run();

        }

# Basic angular setup

* **npm install -g @angular/cli**
* **ng new my-dream-app**
* **cd my-dream-app**
* **ng serve**

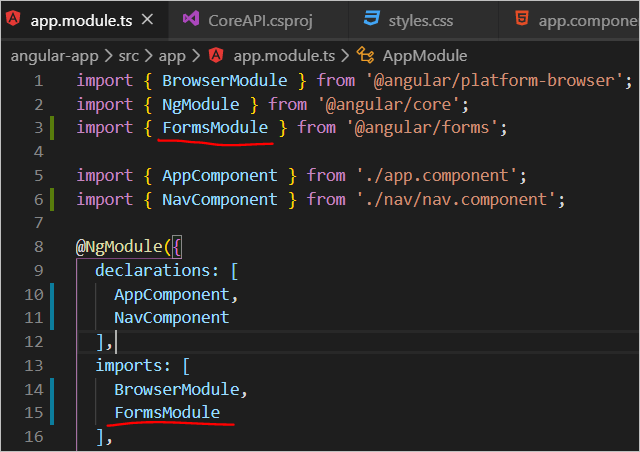
adding bootstrap and font-awesome (npm i bootstrap font-awesome)

We can add styles reference in angular.json file, as styles in angular.json will not render as required.

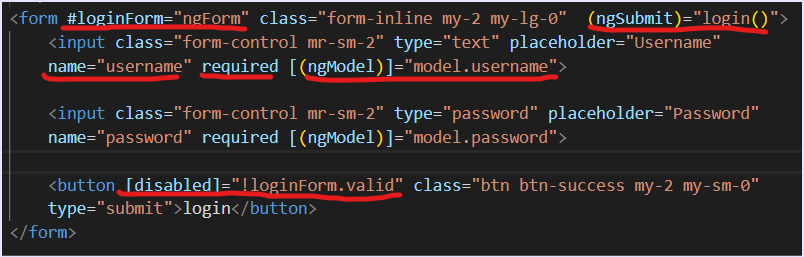


In style.css we define order of css file.

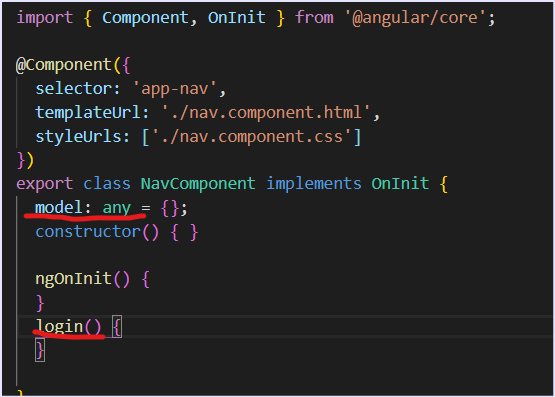
**Forms Module**



**Html**



**Js**

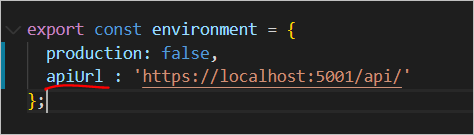


Service

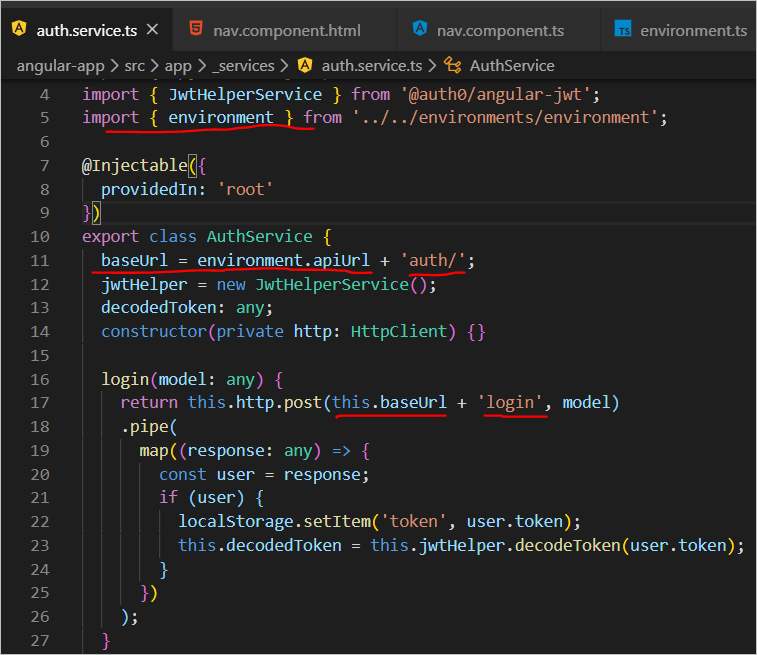


**Component ts(login)**

**Defining Api Url in Environment file.**



Add url in service



**AlertifyJs**:

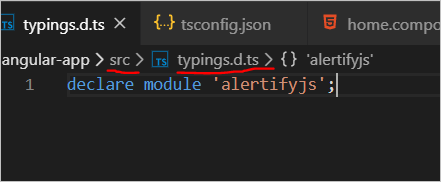
1. npm install alertifyjs
2. add css is style.css

@import '../node\_modules/alertifyjs/build/css/alertify.css';

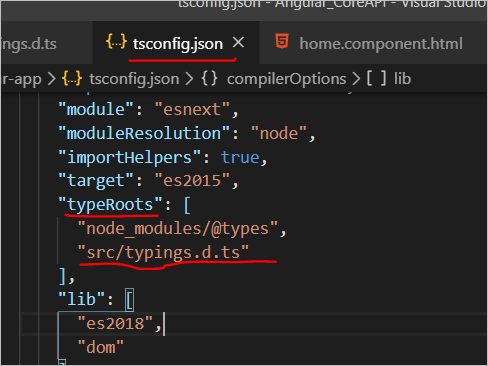
@import '../node\_modules/alertifyjs/build/css/themes/bootstrap.css';

* Create a type definition for alertify library .(It provides intellisense for 3rd party library which don’t have definition for typescript)
* **.d.ts** extension
* Create a wrapper service for alertify.

1. Add d.ts file inside src.



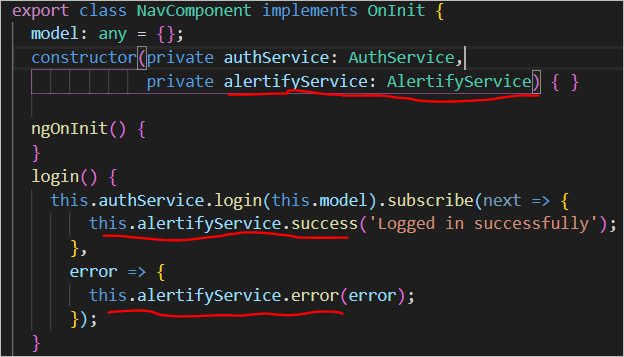
1. Add reference of this file. (tsconfig.json -> typeRoots)



1. Add service for alertify



Use alertifyservice in component



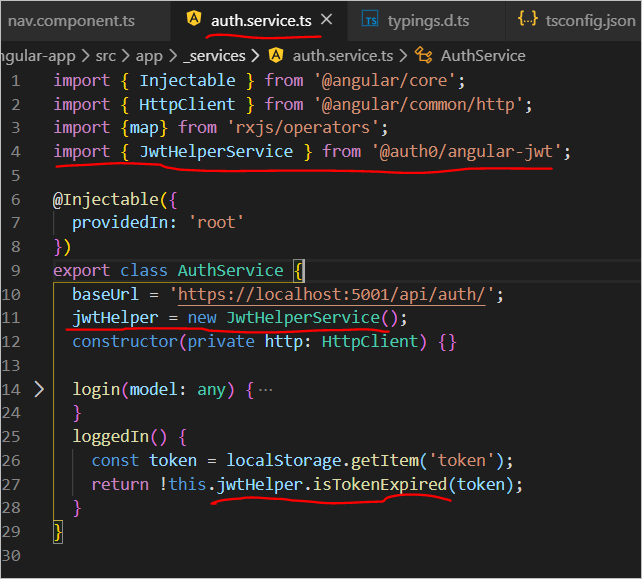
Better way to manage token:

**JWT token**

<https://github.com/auth0/angular2-jwt> (Check this url it has all implementation detail)

* npm install @auth0/angular-jwt

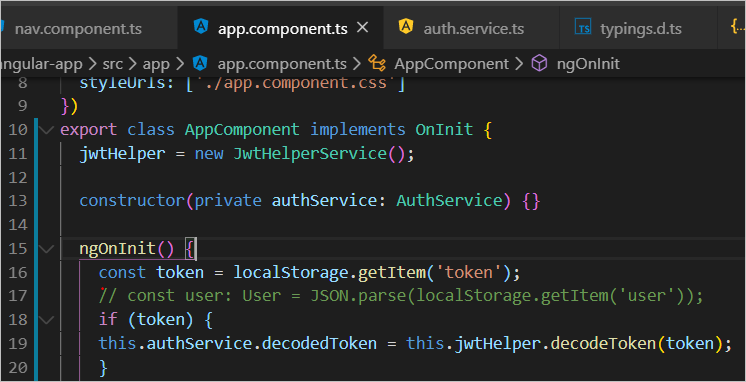
Adding below line in service to check if token is expired.



**Decode token to get username.**



On page reload to persist token value



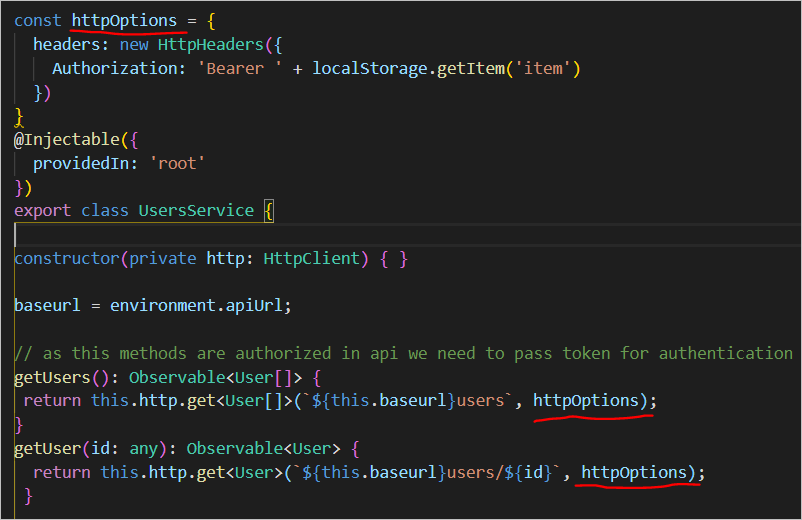
This displays logged in user name

 <a class="dropdown-toggle text-light" dropdownToggle>

        Welcome {{ **authService.decodedToken?.unique\_name** | titlecase }}

      </a>

Here we are passing httpheaders for each request



Instead we can use JWT module to automatically pass token for us. (httpheaders are not needed)

Thus we can remove httpOptions from above.



**Routing & Route Guard.**

import {Routes} from '@angular/router';

import { HomeComponent } from './app/home/home.component';

import { MemberListComponent } from './app/member-list/member-list.component';

import { MessagesComponent } from './app/messages/messages.component';

import { ListsComponent } from './app/lists/lists.component';

import { AuthGuard } from './app/\_guards/auth.guard';

// this one is for restricting router with AuthGurad

// export const appRoutes: Routes = [

//     { path: 'home', component: HomeComponent },

//     { path: 'members', component: MemberListComponent, canActivate: [AuthGuard] },

//     { path: 'messages', component: MessagesComponent },

//     { path: 'lists', component: ListsComponent },

//     { path: '\*\*', redirectTo: 'home', pathMatch: 'full' }

// ];

// this is for adding authguard to all routes at once using childern

export const appRoutes: Routes = [

    { path: '', component: HomeComponent },

    {

        path: '',

        runGuardsAndResolvers : 'always',

        canActivate: [AuthGuard],

        children: [

            { path: 'members', component: MemberListComponent},

            { path: 'messages', component: MessagesComponent },

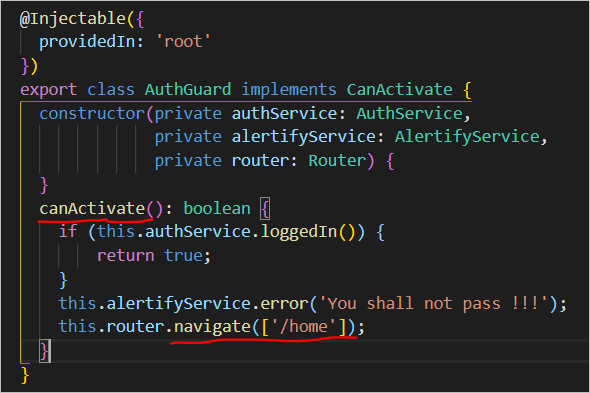
            { path: 'lists', component: ListsComponent },

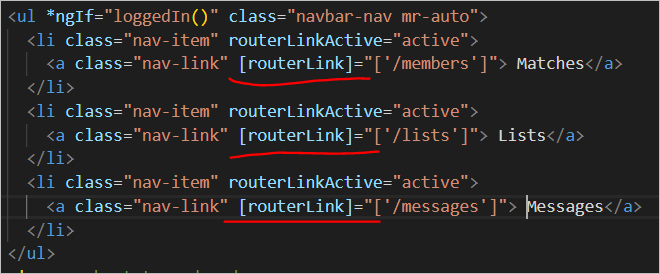
        ]

    },

    { path: '\*\*', redirectTo: '', pathMatch: 'full' }

];





**Route Resolver.**

@Injectable()

export class MemberDetailResolver implements Resolve<User> {

  constructor(

    private userService: UsersService,

    private router: Router,

    private alertify: AlertifyService

  ) {}

  resolve(route: ActivatedRouteSnapshot): Observable<User> {

      // resolver automatically subscribe to method unlike we do in component.

    return this.userService.getUser(route.params[`id`]).pipe(

      catchError(error => {

        this.alertify.error('Problem retrieving data' + error);

        this.router.navigate(['/members']);

        return of(null); // rxjs 6

      })

    );

  }

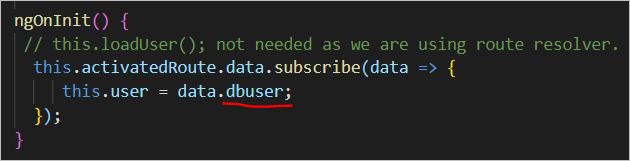
}

* Declare resolver in app.module (in provider)

In router table add route resolver



In component



**Ngx gallery.**

* npm i ngx-gallery

<https://www.npmjs.com/package/ngx-gallery>

**CanDeactivate route guard.**

Help to alert once use moves away from current route.

  export class **PreventUnsavedChanges** implements CanDeactivate<MemberEditComponent> {

    canDeactivate(component: MemberEditComponent) {

        if (component.memberEditForm.dirty) {

            return confirm ('are u sure u want to continue! unsaved changes will e lost');

        }

        return true;

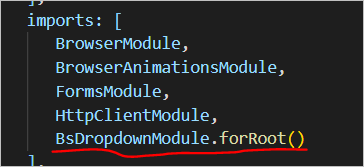
    }



**Ngx bootstrap**

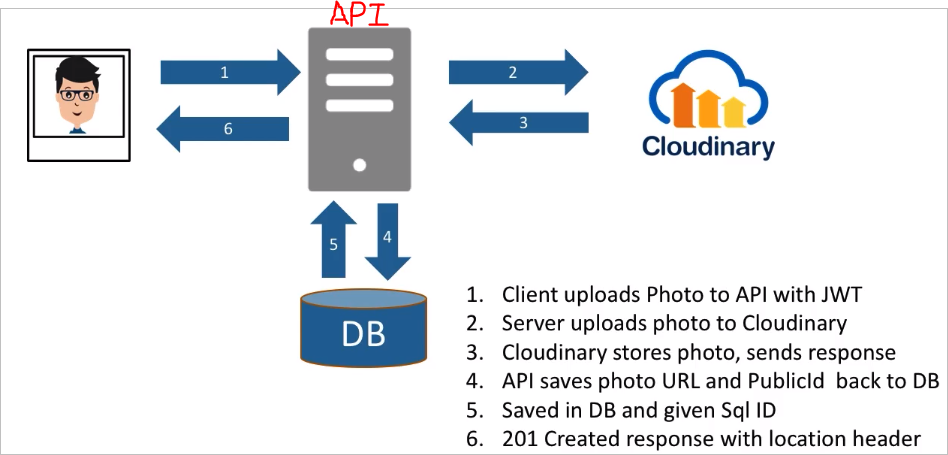
* npm install ngx-bootstrap --save

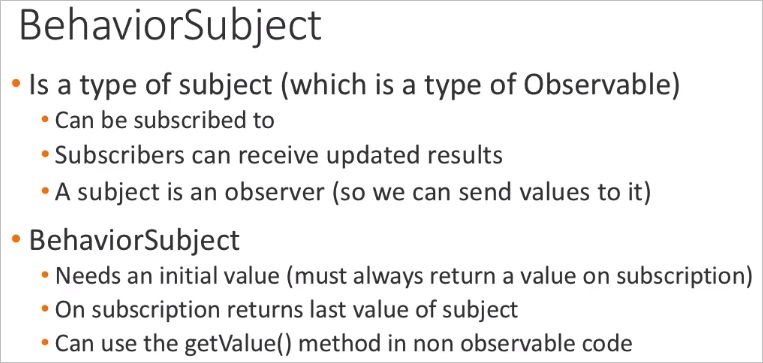
import { BsDropdownModule } from 'ngx-bootstrap';





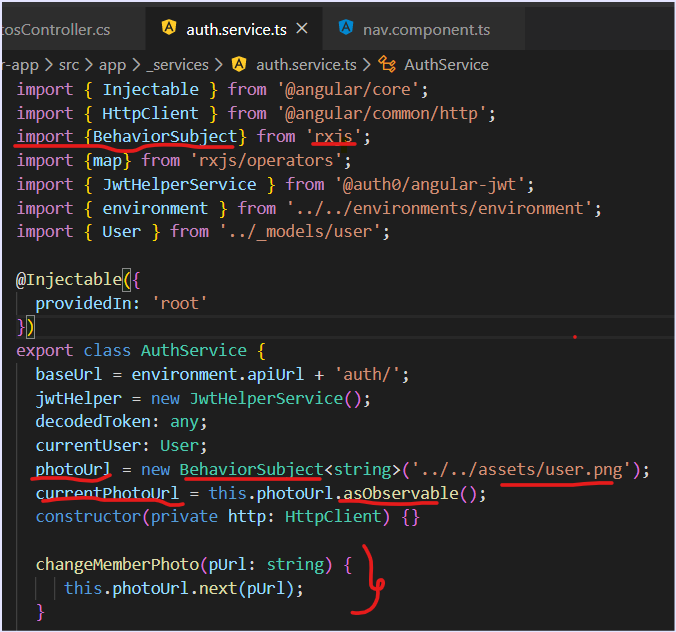
**Cloudinary to save photo on server**





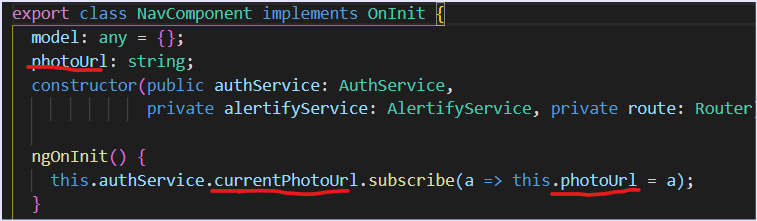
We are using Behaviour subject for communication from one to many component.

Here when button is clicked as ‘mainPhoto’ in child component we are changing parent as well as Nav component photo

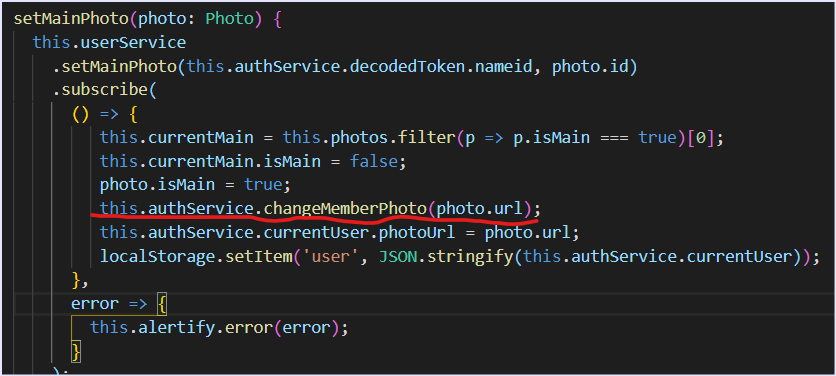


* Here photoUrl is a behavior subject which should have default value.
* currentPhotoUrl is from where we read the value via subscribing it.
* When we need to change photoUrl use ChangeMemberPhoto and via next() method we are changing value of photoUrl.
* Any component can subscribe to currentPhotoUrl to read it.
* Any component can call ChangeMemberPhoto() and update PhotoUrl.

Here we are reading currentPhotoUrl and assigning to local variable



Here we are updating photo URL via ChangeMemberPhoto().



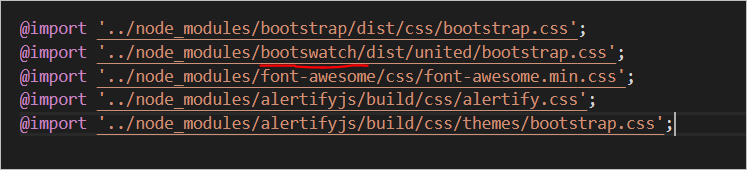
**Theme from Bootswatch**

<https://bootswatch.com/>

* npm install bootswatch

we will be using united theme.

In style.css after bootstrap add bootswatch



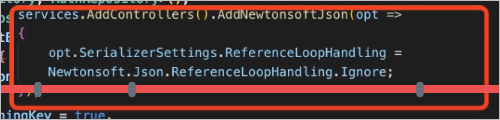
**User Controller , DatingRepository, IDatingRepository**

Install-Package Microsoft.AspNetCore.Mvc.NewtonsoftJson -Version 3.1.0

 services.AddControllers().AddNewtonsoftJson();

new in dot net core 3.0 for json response

use below code when there is loop reference error due to data schema



**AutoMapper**

* AutoMapper.Extensions.Microsoft.DependencyInjection (**install**)
* services.AddAutoMapper(typeof(DatingRepository).Assembly); (add in services)

 [HttpGet]

        public async Task<IActionResult> GetUsers(){

            var users = await \_repo.GetUsers();

            var usersToReturn = \_mapper.Map<IEnumerable<UserForListDto>>(users);

            return Ok(usersToReturn);

        }

        [HttpGet("{id}")]

         public async Task<IActionResult> GetUser(int id){

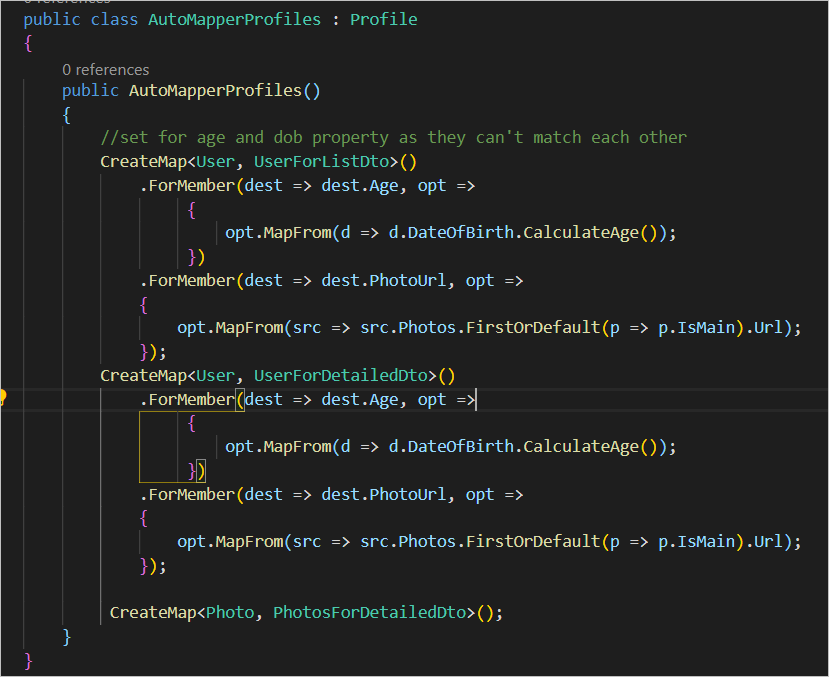
            var user = await \_repo.GetUser(id);

            var userToReturn = \_mapper.Map<UserForDetailedDto>(user);

            return Ok(userToReturn);

        }

Create an autoMapper profile class.



# Error handling

Global exception handler:

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

        {

            if (env.IsDevelopment())

            {

                app.UseDeveloperExceptionPage();

            }

            else

            {

**app.UseExceptionHandler(builder =>**

**{**

**builder.Run(async context =>**

**{**

**context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;**

**var error = context.Features.Get<IExceptionHandlerFeature>();**

**if (error != null)**

**{**

**context.Response.AddApplicationError(error.Error.Message);**

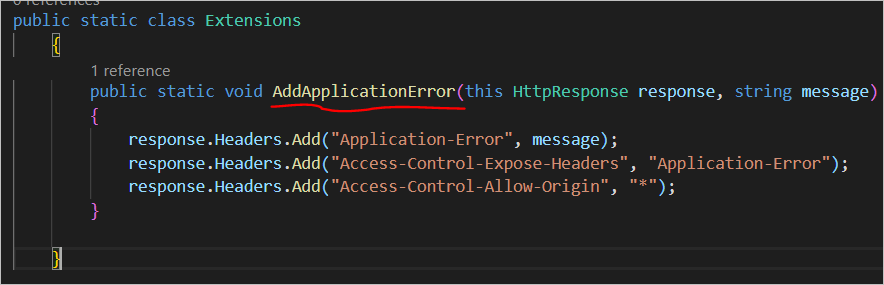
**await context.Response.WriteAsync(error.Error.Message);**

**}**

**});**

**});**

            }

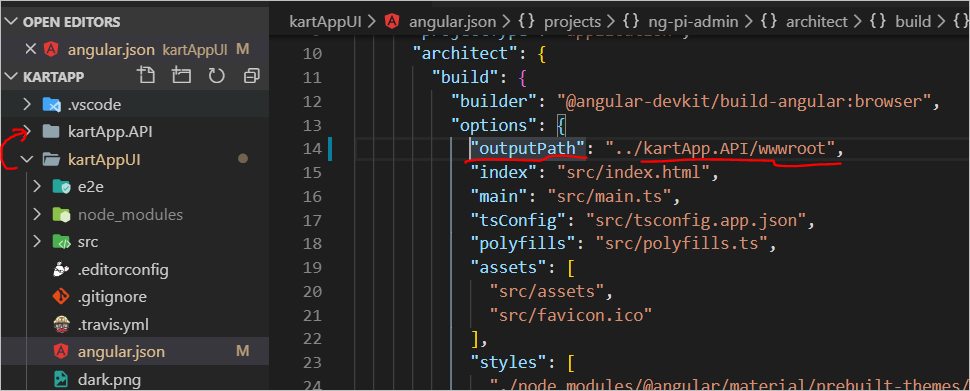


# Deployment



NG build

Change path for build output path for Angular App ( Path is inside of our core API App).



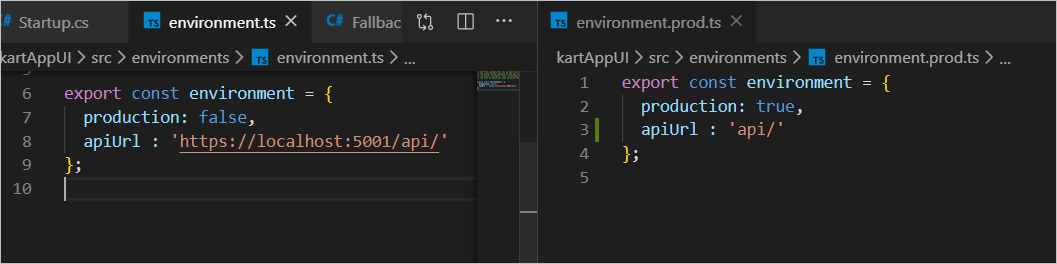
Normal build



This will create a folder wwwroot with angular code

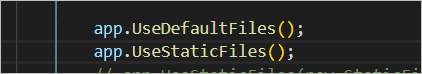
**Angular AOT Build**:

Change apiUrl in environment.prod.ts



PS C:\Users\v-sesin\UPS\_KART\_Project\kartApp\kartAppUI> **ng build --prod**

**Core** : Make changes to API to serve these static files.



Hit url of core api it will serve our angular pages.

But if we refresh any other url rather than base it will fail as API doesn’t know about routing of angular.

Whenever a url is request always redirect to angular index.html and then angular will take care of routing.

Add a controller in our API (make sure it inherits Controller not ControllerBase).

using System.IO;

using Microsoft.AspNetCore.Mvc;

namespace kartApp.API.Controllers

{

   public class FallbackController : Controller

    {

      public IActionResult Index(){

         return PhysicalFile(Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "index.html"),"text/HTML");

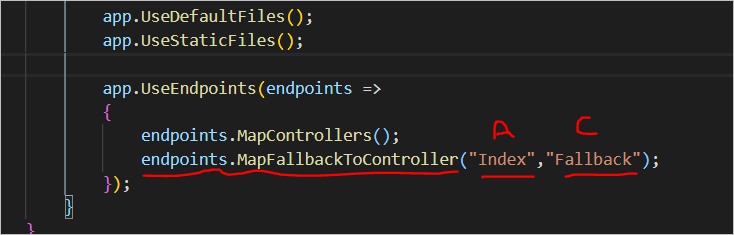
        }

    }

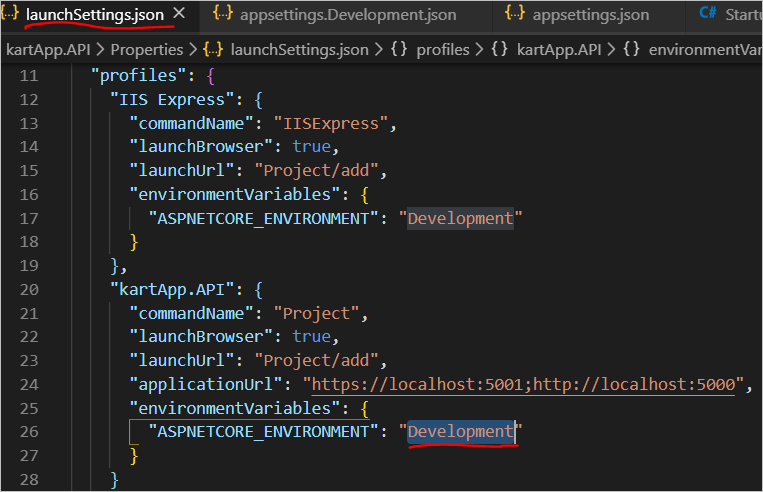
}

In startup.cs file add code to redirect to above controller.

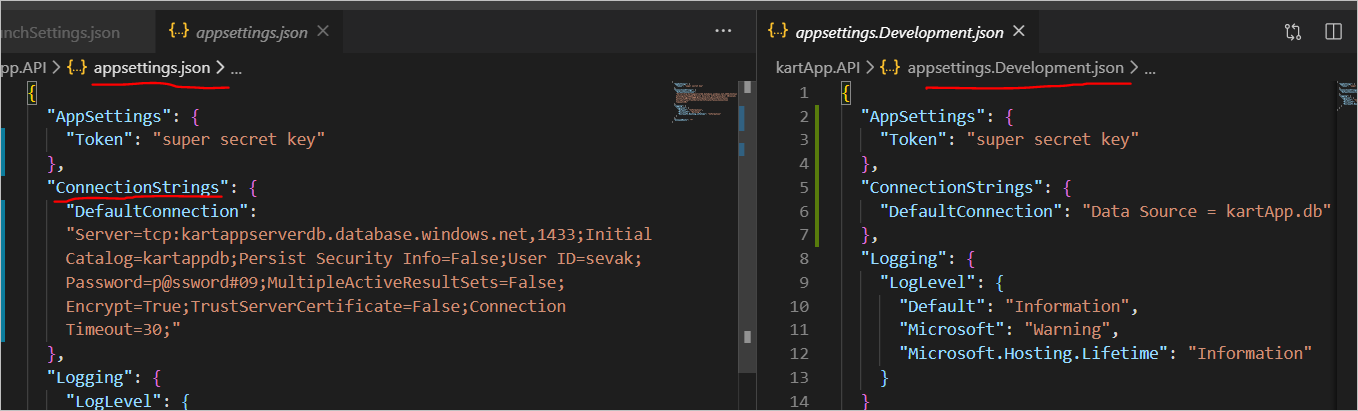
**MapFallbackToController** option redirects to Action and Controller.



In Launch setting change to **Production**



In Appsetting change DB Connection (Local and Production)



Create two methods **ConfigureDevelopmentServices**, **ConfigureProductionServices (**for different environment connection string)

 public class Startup

    {

        public Startup(IConfiguration configuration)

        {

            Configuration = configuration;

        }

        public IConfiguration Configuration { get; }

        public void ConfigureDevelopmentServices(IServiceCollection services)

        {

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

            ConfigureServices(services);

        }

        public void ConfigureProductionServices(IServiceCollection services)

        {

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

            ConfigureServices(services);

        }

        // This method gets called by the runtime. Use this method to add services to the container.

        public void ConfigureServices(IServiceCollection services)

        {

            // Add authentication for token.

            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

                };

            });

            // Add repository

            services.AddScoped<IAuthRepository,AuthRepository>();

            services.AddScoped<IProjectRespository,ProjectRespository>();

            services.AddControllers().AddNewtonsoftJson(opt =>

            {

                opt.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;

            });

            // Enable Cors.

            services.AddCors();

            services.Configure<Microsoft.AspNetCore.Http.Features.FormOptions>(o =>

            {

                o.ValueLengthLimit = int.MaxValue;

                o.MultipartBodyLengthLimit = int.MaxValue;

                o.MemoryBufferThreshold = int.MaxValue;

            });

        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

        {

           if (env.IsDevelopment())

           {

                app.UseDeveloperExceptionPage();

           }

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseAuthentication();

            app.UseAuthorization();

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

            //app.UseCors(x => x.WithOrigins("http://localhost:4200").AllowAnyMethod().AllowAnyHeader());

            app.UseDefaultFiles();

            app.UseStaticFiles();

            app.UseEndpoints(endpoints =>

            {

                endpoints.MapControllers();

                endpoints.MapFallbackToController("Index","Fallback");

            });

        }

    }