# Dating App Notes:

Section 1: Creating Project Directory and Solution file.

1. Create dotnet API using Command CLI within Visual Studio Code.
2. Create Directory ‘DatingApp’ and use command ‘dotnet new sln’, this will create .sln which will be same name as your directory ‘DatingApp’.
3. Create dotnet webapi in subfolder ‘API’ by using command ‘dotnet new webapi -o API’ .
4. Add ‘API’ project to the solution .sln by using command ‘dontet sln add API’.
5. Install following extensions in Visual Studio Code.
6. C# for Visual Studio Code by Microsoft
7. C# Extensions
8. Install material icons
9. go to setting and type exclude and hide obj to hide in the list.
10. go to setting and type folder then uncheck compact folder.
11. In Visual code go to Terminal and type command ‘dotnet run’, this will build and run the application. It will show you the web address where the application is listening.
12. In order to use https, trust the certificate by using the following command

‘dotnet devcerts https --trust’ , run it as admin if error occurred.

1. Edit “appsettings.Devlopment.json” and change “Default” to “Information” from “Warning” to see more information in terminal, restart the app by usin “dotnet run”.
2. Create Entities Folder and create new class named “AppUser.cs”.
3. Install Nuget Gallery extension. Open it after installation from command pallets and install “Microsoft Entity Framework SQLLite” or “Microsoft Entity Framework SQLServer”.
4. Add new folder and name it “Data”. Create new class “DataContext.cs” and create a constructor and properties(method) to set the data. Review the class for more info.
5. Open startup.cs class and add the following code in ConfigureServices method to add DataContext with options. Note: Connection string to be updated later.

services.AddDbContext<DataContext>(options =>

           {

               options.UseSqlite(\_config.GetConnectionString("DefaultConnection"));

            });

1. Add the following lines to “appsettings.Development.json” to setup the database connection string:

"ConnectionStrings" : {

    "DefaultConnection":  "Data source=datingApp.db"

  },

1. Go to startup.cs class now and remove “public IConfiguration Configuration{get;}” method from it. Rename configuration parameter name to config, use quickfix to initialize the field from parameter. Go to setting and search for “private” specify “\_” as a prefix for private variable.

Search for “this” as well under “C# Extenstion” properties and uncheck Use “this” assignment.

1. Replace the ConnectionString in Configuration service method in startup.cs class with the following default connection string we created:

options.UseSqlite(\_config.GetConnectionString("DefaultConnection"));

1. Got to Nugget.org and search for “dotnet –ef”, download the version of .net that you installed and copy the command and paste it in the vscode terminal to install dotnet –ef.
2. In terminal window type the command “dotnet ef migration –o /Data/Migrations”, if error occurred then go to Nugget gallery and search for “Microsoft Entity Framework core design” and install the version that matches dotnet version. This is going to create migration which is Schema to create the database from class properties automatically.
3. Go to terminal and type “dotnet ef create”, this will create the database user with two fields

“ID” and “UserName” with “ID” set as a primary key with Auto increment properties.

1. Install the “SQLite” extension, go to command pallet and search for SQLite to open the database and select “datingapp.db”. You should be able to see database and table in the explorer.
2. Right click on User Table in explorer and select New Query insert to add data into the table.
3. Go to Controller folder in API folder and create “UsersController.cs”, edit the controller and drive it from “ControllerBase” class, quick fix the error by Using “Microsoft.ASPNetCore.Mvc”,

add the attribute [Controller] and route option [Route(“api/[controller]”)], use quick fix to Generate Constructor in controller class. Pass on the “DataContext” parameter in constructor to initialize it. Add two End points, one to get single user and other to get all users.

1-  [HttpGet]

        public async Task<ActionResult<IEnumerable<AppUser>>> GetUsers()

        {

           return await \_context.Users.ToListAsync();

        }

2- [HttpGet("{id}")]

         public async Task<ActionResult<AppUser>> GetUser(int id)

         {

            return await \_context.Users.FindAsync(id);

        }

# Creating Web Token for Login and Register

Starup.cs

namespace API

{

    public class Startup

    {

        private readonly IConfiguration \_config;

        public Startup(IConfiguration config)

        {

            \_config = config;

        }

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

        public void ConfigureServices(IServiceCollection services)

        {

            services.AddScoped<ITokenService, TokenService>();

            services.AddDbContext<DataContext>(options =>

            {

                options.UseSqlite(\_config.GetConnectionString("DefaultConnection"));

            });

            /\*services.AddCors(options =>{

            options.AddPolicy("CorsPolicy", x => x.AllowAnyOrigins()

                               .AllowAnyHeader()

                              .AllowAnyMethod());

                              });\*/

            services.AddCors();

            services.AddControllers();

            services.AddSwaggerGen(c =>

            {

                c.SwaggerDoc("v1", new OpenApiInfo { Title = "API", Version = "v1" });

            });

            services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

            .AddJwtBearer(options =>

            {

                options.TokenValidationParameters = new TokenValidationParameters

                {

                    ValidateIssuerSigningKey = true,

                    IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config["TokenKey"])),

                    ValidateIssuer = false,

                    ValidateAudience = false,

                };

            });

        }

        // 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.UseSwagger();

                app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "API v1"));

            }

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseCors(x=>x.AllowAnyHeader()

                            .AllowAnyMethod()

                            .WithOrigins("https://localhost:4200"));

            //app.UseCors("CorsPolicy");

            app.UseAuthentication();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>

            {

                endpoints.MapControllers();

            });

        }

    }

}

AccountController.cs

 public async Task<ActionResult<UserDto>> Register([FromBody] RegisterDto registerDto)

        {

            // Console.WriteLine(registerDto.Password);

            if (!ModelState.IsValid) return BadRequest(ModelState);

            if (registerDto.Username == null) return BadRequest("Null Value in username");

            if (await UserExists(registerDto.Username)) return BadRequest("user is taken");

            using var hmac = new HMACSHA512();

            var user = new AppUser

            {

                UserName = registerDto.Username.ToLower(),

                PasswordHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(registerDto.Password)),

                PasswordSalt = hmac.Key

            };

            \_context.Users.Add(user);

            await \_context.SaveChangesAsync();

            return new UserDto

            {

                Username = user.UserName,

                Token = \_tokenService.CreateToken(user)

            };

        }

        private async Task<bool> UserExists(string username)

        {

            return await \_context.Users.AnyAsync(x => x.UserName == username.ToLower());

        }

        [HttpPost("login")]

        public async Task<ActionResult<UserDto>> Login([FromBody] LoginDto logindto)

        {

            var user = await \_context.Users.SingleOrDefaultAsync(x => x.UserName == logindto.Username);

            if (user == null) return Unauthorized("Invalid user name");

            using var hmac = new HMACSHA512(user.PasswordSalt);

            var computedHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(logindto.Password));

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

            {

                if (computedHash[i] != user.PasswordHash[i]) return Unauthorized("Password is invalid");

            }

            return new UserDto

            {

                Username = user.UserName,

                Token = \_tokenService.CreateToken(user)

            };

        }

    }

}

TokenService.cs

namespace API.Services

{

    public class TokenService : ITokenService

    {

        private readonly SymmetricSecurityKey \_key;

        public TokenService(IConfiguration config)

        {

            \_key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(config["TokenKey"]));

        }

        public string CreateToken(AppUser user)

        {

            var claims = new List<Claim>

            {

                new Claim(JwtRegisteredClaimNames.NameId, user.UserName)

            };

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

            var tokenDescriptor = new SecurityTokenDescriptor

            {

                Subject = new ClaimsIdentity(claims),

                Expires = DateTime.Now.AddDays(7),

                SigningCredentials = creds

            };

            var tokenHandler = new JwtSecurityTokenHandler();

            var token = tokenHandler.CreateToken(tokenDescriptor);

            return tokenHandler.WriteToken(token);

        }

    }

}