# June 2nd - Introduction

I have prepared for the HTML, CSS, and JS and C#, but the client is already aware of those technologies. As she is already working on asp.net web forms, need to prepare advanced topics in the upcoming sessions.

Questions asked in the session:

1. How the entity is transformed to JSON
2. The settings file
3. **Expectations** – security, authentication, difference between EF and EF core

Topics Planned for tomorrow – All the four HTTP methods and answer the questions

## Homework:

Adding XML formatters in Startup.cs:

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers()

.AddXmlSerializerFormatters();

}

In ASP.NET Core 3.0 or later, the default JSON formatters are based on System.Text.Json. Support for Newtonsoft.Json based formatters and features is available by installing the Microsoft.AspNetCore.Mvc.NewtonsoftJson NuGet package and configuring it in **Startup.ConfigureServices**.

.AddNewtonsoftJson();

# June 3rd – basic web API method using static list

Today, we implemented the basic CRUD operation on the same webapi core project using the person entity. All went well.

**Plan for tomorrow:** implement the same crud operations using the EF Core with local database file

**Expectations:** executing proc

# June 4th

Taken off due to personal reasons ☺

# June 5th – Code first approach and migrations

Today, we improved the same CRUD operations using ef core and saved the data in local database.

**Installer packages:**

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

**DB context class Constructor:**

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

**Startup.cs class configuration:**

services.AddDbContext<MyDBContext>(options => options.UseSqlServer(Configuration.GetConnectionString("MyDbConnection")));

**Connection string in appsettings.json file:**

"ConnectionStrings": {

"MyDbConnection": "Server=(LocalDB)\\MSSQLLocalDB; Database=MyDb; Integrated Security=true"

},

**EF Core migration commands:**

Add-migration migration\_name

Update-database

Remove-migration

Enable-Migrations -EnableAutomaticMigrations –Force (use when the tool throw errors)

**Database commands to force delete:**

USE master;

ALTER DATABASE MyDb SET SINGLE\_USER WITH ROLLBACK IMMEDIATE;

DROP DATABASE MyDb

**Plan for tomorrow:** implement the same crud operations using database approach and practice adding new entities with relations

# June 6th – Database first approach

Today, we discussed the database first approach with simple database attached here and worked fine.



Create database context and entities using this command: **Scaffold-DbContext** "Server=(LocalDB)\MSSQLLocalDB; Database=SchoolDB; Integrated Security=true" Microsoft.EntityFrameworkCore.SqlServer -OutputDir SchoolDBModels

Deleted the OnConfiguring() method in generated SchoolDBContext class and configured through Startup.ConfigureServices() method.

**Questions asked:** how to execute a procedure through EF core? How to create only selected entities (or except excluded entities) through nuget command?

**Plan for tomorrow:** Log the SQL command executed and lazy loading concept and executing the stored procedures using EF core.

# June 8th – Executing a stored procedure and limitations

Today, we have practiced executing a stored procedure using ef core and its limitations.

1. Result must be an entity type. This means that a stored procedure must return all the columns of the corresponding table of an entity.
2. Result cannot contain related data. This means that a stored procedure cannot perform JOINs to formulate the result.
3. Insert, Update and Delete procedures cannot be mapped with the entity, so the SaveChanges method cannot call stored procedures for CUD operations.

**Plan for tomorrow:** Executing any stored procedure using ado.net core and forming relations between entities in code first approach using data attributes

MAKE AN .MD FILE FOR EVERY DAY AND TRACK IT IN THE PROJECT

# June 9th

Discussed about data attributes like table, column, key, required, min, maxlength and foreignkey with virtual complex type. NotMapped left behind.

Executing any stored procedure using ado.net SqlDataAdapter

# 10th taken leave

# June 11th

Discussed about one to one, one to many relationship and tried many to many relationships

# June 12th

Forming many to many relationship using bridge table, inheritance in single table, eager loading and explicit loading

Eager loading: var person = dbContext.Persons.Include("PersonAddress").First();

Explicit loading: dbContext.Entry<Person>(person).Reference(p => p.PersonAddress).Load();

Plan for tomorrow: logging, and explaining lazy loading, eager loading and explicit loading and TPT inheritance and

<https://docs.microsoft.com/en-us/ef/core/querying/related-data#lazy-loading>

Cross check TPT and TPH

<https://www.thinktecture.com/en/entity-framework-core/table-per-type-inheritance-support-part-1-code-first/>

MAKE AN .MD FILE FOR EVERY DAY AND TRACK IT IN THE PROJECT

MAKE A COMPLETE PLAN FOR EF CORE and identify remaining topics

Disconnected Scenario in Entity Framework Core & ChangeTracker in Entity Framework Core

<https://www.entityframeworktutorial.net/efcore/saving-data-in-disconnected-scenario-in-ef-core.aspx>

<https://www.entityframeworktutorial.net/efcore/working-with-disconnected-entity-graph-ef-core.aspx>

<https://www.entityframeworktutorial.net/efcore/changetracker-in-ef-core.aspx>

Shadow properties (this can be combined in web api authentication)

<https://www.entityframeworktutorial.net/efcore/shadow-property.aspx>

<https://www.entityframeworktutorial.net/efcore/cli-commands-for-ef-core-migration.aspx>

Preloading master data: initializing db with some data

<https://csharp-video-tutorials.blogspot.com/2019/05/entity-framework-core-seed-data.html>

And web api core

Asynchronous methods

Project and configuration files: web.config, appsettings.json, launch.json

And where are the references stored

Dotnet cli

Authentication

Repository pattern: <https://csharp-video-tutorials.blogspot.com/2019/04/repository-pattern-in-aspnet-core.html>

We could use the shadow properties along with it. Try making the DbSet<> as internal.

# June 13th

Today, we discussed about TPH and TPT and explained what the difference is. Observed SQL queries in eager loading, lazy loading and explicit loading scenarios by logging.

Plan for next session: disconnected scenario, change tracker, and if there is time seed data and shadow property

# June 15th

Today, we discussed about disconnected scenario, change tracker and the entity states. We also practiced seed data and loading data from a json file.

Plan for tomorrow: shadow properties and query filters and transactions

# June 16th

Today, we discussed about shadow properties of entities, global query filter, ignoring this filter and transactions.

We also discussed about how to modify these properties when they are added or modified by overriding SaveChanges() method.

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Blog>().Property<string>("\_tenantId").HasColumnName("TenantId");

modelBuilder.Entity<Blog>().HasQueryFilter(b => EF.Property<string>(b, "\_tenantId") == \_tenantId);

}

blogs = db.Blogs.Include(b => b.Posts)

.IgnoreQueryFilters()

.ToList();

Plan for tomorrow: Model Value Conversions, Inner Join and Left Join using Linq and Exceptions

# June 17th

Today, we discussed about value conversion using an entity, and practiced inner join and left join using linq syntax.

# June 18th

Today, we discussed about owned types and fluent API generated when scaffold-DbContext “connStr” provider –OutputDir used.

# June 20th

Today, we discussed IIS & kestrel server and the differences and different project settings file.

launchSetting.json: includes the profiles that specifies environment

The environment variable ASPNETCORE\_ENVIRONMENT will have the value when published on servers where launchSettings.json file is not required.

Appsettings.json and appsettings.{environment}.json: the environment decides which file the app should use.

app.UseDefaultFiles(); /// default.htm, default.html, index.htm, index.html

app.UseStaticFiles();

app.UseStaticFiles(new StaticFileOptions

{

FileProvider = new PhysicalFileProvider(

Path.Combine(Directory.GetCurrentDirectory(), "MyStaticFiles")),

RequestPath = "/StaticFiles"

});

# July 22th

Today, we discussed clearly on middle ware available in framework.

app.UseDefaultFiles();

app.UseStaticFiles();

app.UseDirectoryBrowser();

And

app.UseFileServer(new FileServerOptions

{

EnableDirectoryBrowsing = true

});

var options = new FileServerOptions

{

EnableDirectoryBrowsing = true,

FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), "MyStaticFiles")),

RequestPath = "/StaticFiles"

};

options.DefaultFilesOptions.DefaultFileNames = new List<string> { "home.html" };

app.UseFileServer(options);

Also discussed the basics of middleware methods and class.

The plan for tomorrow is to discuss in detail about middleware.

# June 23rd

Today, we discussed about middleware that can re-write the controller output text.

Also implemented a simple authorization middleware using header value.

# June 24th

Today, we discussed about dependency injection and the three ways to do it: Singleton, scoped and transient.

Plan for tomorrow: Authentication and authorization in dotnet core

# June 25th

Today, we discussed about basic authentication technique to secure the web api using filter and attribute classes.

June 26th and 27th – Taken off due to office workload

# June 28th

Today, we discussed about JWT authentication by generating a token on passing username and password. We can access the protected api end points by passing the token in authentication header.