Set-up API Dot Net Project in VS2022 with Dapper, Dependency injection, SQL and Layers

1. Set-Up Project
2. Folder Structure
3. Dependency Injection
4. Project Dependency/References
5. SQL Configuration
6. Dapper ORM Integration
7. JWT Token Implementation

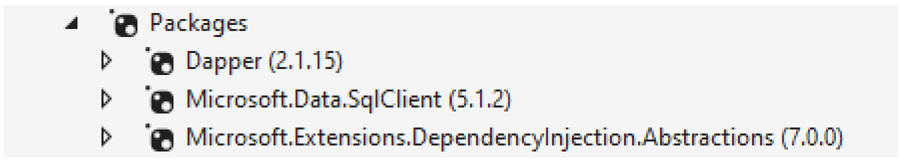
1. Set-Up Project

Step 1 Create a project in API project

Add 3 layers (Class Library Project)

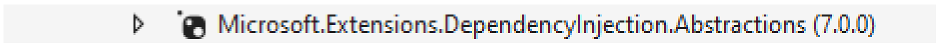
> **Service** – Validation and other purpose
> **Repository** – Dapper and Database connection only
> **Shared Layer** – Models

Step 2 Add SQL client and dapper to Repository layer



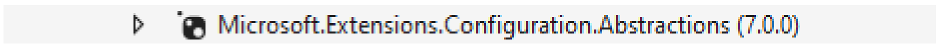
```
Packages
    Dapper (2.1.15)
    Microsoft.Data.SqlClient (5.1.2)
    Microsoft.Extensions.DependencyInjection.Abstractions (7.0.0)
```

Step 3 Add Dependency library to all projects



```
Microsoft.Extensions.DependencyInjection.Abstractions (7.0.0)
```

Step 4 Add Configuration package in Application layer for SQL configuration



```
Microsoft.Extensions.Configuration.Abstractions (7.0.0)
```

2. Folder Structure

1. Controller Project
    a. Controller Folder
        i. ***{ControllerName}***
    b. Injectable
        i. Dependency Injection.cs

2. Service Project
    a. ***{ControllerName}***Feature
        i. I***{ControllerName}***Interface

3. Repository Project
    a. ***{ControllerName}***Repository
        i. I***{ControllerName}***Interface
    b. DapperORM
        i. File..

3. Dependency Injection

Create a file name `InjectableServices.cs in Injectable folder in Controller APP`

`Add this code to InjectableServices.cs file`

```
public class InjectableServices
{
    public static void Services(WebApplicationBuilder builder)
    {
        builder.Services.AddHttpContextAccessor();
    }
}
```

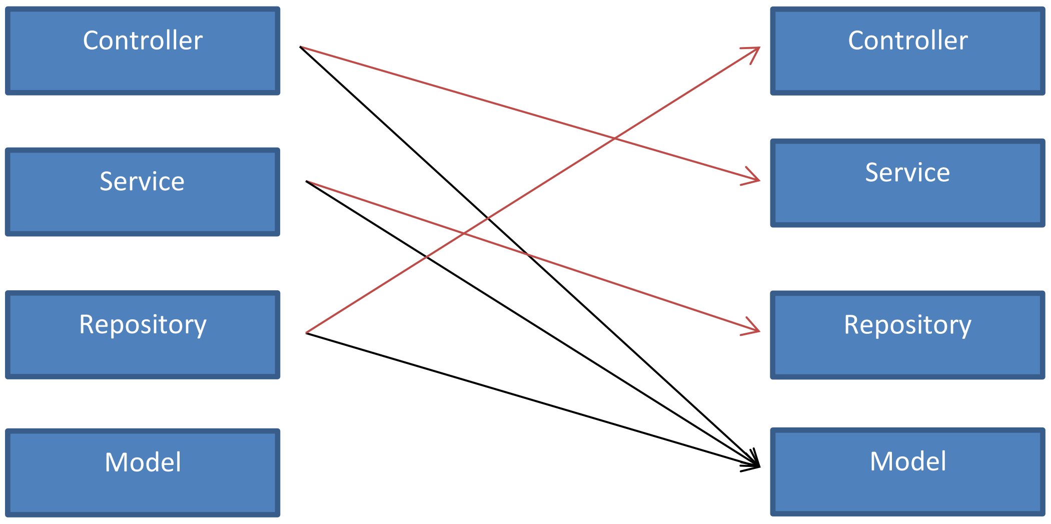
Add this code to Program.cs file before line `var app = builder.Build();`

```
#region Dependancy Injection
InjectableServices.Services(builder);
builder.Services.AddInfrastructureServices();
#endregion Dependancy Injection end
```

To Initialize A dependency from interface to class
Use this code

```
builder.Services.AddScoped<IDemoRepository, DemoRepository>();
```

4. Project Dependency/References



5. SQL Configuration

Add this code

Add Connection string to Appsettings.JSON

```
"ConnectionStrings": {
"DBConnectionString": "Server=100.0.0.27; User=backup; initial
catalog=inventory_db; Password=password"
}
```

Add public class DbContext.cs in Repository Layer

```
public class DbContext
{
    private readonly SqlConnection connection;
    public DbContext(IConfiguration configuration)
    {
        this.connection = new
(configuration.GetConnectionString("DBConnectionString"));
    }
    public SqlConnection GetConnection()
    {
        return connection;
    }
}
```

Add New public class in Application layer InfrastructureServiceRegistration in Repository

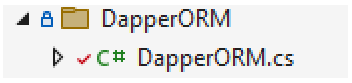
Add this code to InfrastructureServiceRegistration.cs

```
public static class InfrastructureServiceRegistration
{
    public static IServiceCollection AddInfrastructureServices(this
IServiceCollection services)
    {
        services.AddScoped<DbContext>();

        return services;
    }
}
```

6. Dapper ORM Integration

1. Create Folder Name DapperORM in Repository
2. Create a File Name DapperORM.cs in DapperORM



Inside DapperORM.cs Write this code…

```
public class DapperORM

{
        private readonly DbContext dbContext;
        public DapperORM(DbContext dbContext)
        {
                this. dbContext = dbContext;
        }
}
```

Then Create Generic Methods For getting Data For example :-

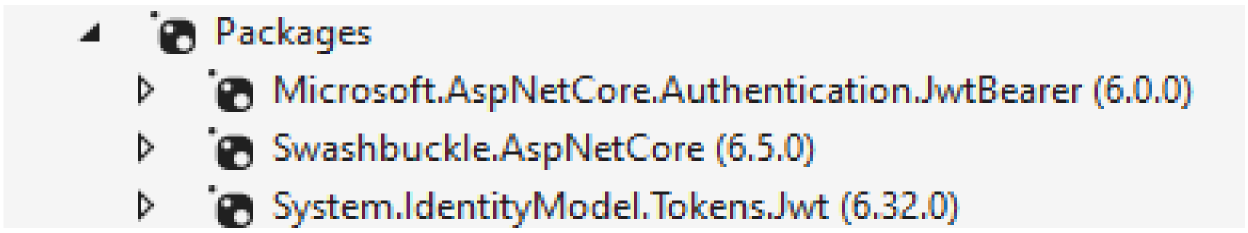
```
public List<T> GetAll<T>(string query)
{
    using (var db = dbContext.GetConnection())
    {
       return db.Query<T>(query, commandType:
CommandType.StoredProcedure).ToList();
    }
}
```

7. JWT Token

1. Inside `appsettings.json` Write this code...

```
"JwtConfig": {
  "SecretKey": "KEY_OF_ANY_LENGTH",
  "Issuer": "https://localhost",
  "Audience": "http://localhost",
  "ExpireInMinutes": "3000"
}
```

2. Install these package In Controller Layer.



```
Packages
  Microsoft.AspNetCore.Authentication.JwtBearer (6.0.0)
  Swashbuckle.AspNetCore (6.5.0)
  System.IdentityModel.Tokens.Jwt (6.32.0)
```

In Controller Create Model

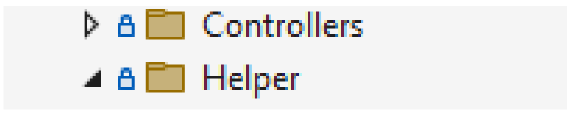
```
public class LoginResponse
{
    public string name { get; set; }
    public string roleId { get; set; } = "ABC456";
    public string email { get; set; }
    public string mobile { get; set; }
    public string designation { get; set; }
    public string level { get; set; }
    public string permission { get; set; }
    public string officeCode { get; set; } = "ABC123";
}

public class JwtTokenResponse
{
    public string? Token { get; set; }
    public bool IsRefreshToken { get; set; }
}

public class UserRequest
{
    public string roleId { get; set; }
    public string officeCode { get; set; }
}

public class JWTSettings
{
    public string Issuer { get; set; } = "";
    public string Audience { get; set; } = "";
    public string SecretKey { get; set; } = "";
    public int ExpireInMinutes { get; set; } = 0;
}
```

Add Helper Folder in Controller Layer



In Helper Folder Create `JwtMiddleware.cs` File

ADD This Code

```
public class JwtMiddleware
{
   private readonly RequestDelegate _next;
   private readonly JWTSettings _jwtSettings;

   public JwtMiddleware(RequestDelegate next, IOptions<JWTSettings> jwtSettings)
   {
      _next = next;
      _jwtSettings = jwtSettings.Value;
   }

   public async Task Invoke(HttpContext context)
   {
      var token = context.Request.Headers["Authorization"].FirstOrDefault()?.Split(" ").Last();
      if (token != null)
      {
         attachUserToContext(context, token);
      }

      await _next(context);
   }

   private async Task attachUserToContext(HttpContext context, string token)
   {
      try
      {
         var tokenHandler = new JwtSecurityTokenHandler();
         var key = Encoding.ASCII.GetBytes(_jwtSettings.SecretKey);
         tokenHandler.ValidateToken(token, new TokenValidationParameters
         {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(key),
            ValidateIssuer = false,
            ValidateAudience = false,
            ClockSkew = TimeSpan.Zero
         }, out SecurityToken validatedToken);


         var jwtToken = (JwtSecurityToken)validatedToken;
         //UserConfiguration config = new UserConfiguration();
         UserRequest config = new UserRequest();
         config.roleId = jwtToken.Claims.First(x => x.Type == "RoleId").Value != null ? Convert.ToString(jwtToken.Claims.First(x => x.Type ==
"RoleId").Value) : "";
         config.officeCode = jwtToken.Claims.First(x => x.Type == "OfficeCode").Value != null ? Convert.ToString(jwtToken.Claims.First(x =>
x.Type == "OfficeCode").Value) : "";

         if (config.roleId != "" || config.officeCode != "")
         {
            context.Items["UserConfig"] = config;
         }

      }
      catch (Exception ex)
      {
         throw new Exception(ex.Message);
      }
   }
```

In Helper Folder Create TokenAuthorizeAttribute.cs File

ADD This Code

```
public class TokenAuthorizeAttribute : Attribute, IAuthorizationFilter
{
    public void OnAuthorization(AuthorizationFilterContext context)
    {
        UserRequest userConfig =
(UserRequest)context.HttpContext.Items["UserConfig"];
        if (userConfig == null)
        {
            context.Result = new JsonResult(new { message = "Unauthorized" }) {
StatusCode = StatusCodes.Status401Unauthorized };
        }
    }
}
```

In Helper Folder Create TokenUtil.cs File

ADD This Code

```
public class TokenUtil
{
    private readonly JWTSettings _jwtSettings;
    public TokenUtil(IOptions<JWTSettings> jwtSettings)
    {
        _jwtSettings = jwtSettings.Value;
    }

    public string GenerateJwtToken(LoginResponse response)
    {
        var tokenHandler = new JwtSecurityTokenHandler();
        var key = Encoding.ASCII.GetBytes(_jwtSettings.SecretKey);
        var expiresTime = _jwtSettings.ExpireInMinutes;
        var tokenDescriptor = new SecurityTokenDescriptor
        {

            Subject = new ClaimsIdentity(new[] {
                    new Claim("RoleId", response.roleId.ToString()),
                    new Claim("OfficeCode", response.officeCode.ToString())
            }),
            Expires = DateTime.UtcNow.AddMinutes(expiresTime),
            Issuer = _jwtSettings.Issuer,
            Audience = _jwtSettings.Audience,
            SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
        };
        var token = tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
    }
}
```

INSIDE Program.cs File

ADD

```
builder.Services.AddSwaggerGen(setup =>
{
    // Include 'SecurityScheme' to use JWT Authentication
    var jwtSecurityScheme = new OpenApiSecurityScheme
    {
        Scheme = "bearer",
        BearerFormat = "JWT",
        Name = "JWT Authentication",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.Http,
        Description = "Put **_ONLY_** your JWT Bearer token on textbox below!",
        Reference = new OpenApiReference
        {
            Id = JwtBearerDefaults.AuthenticationScheme,
            Type = ReferenceType.SecurityScheme
        }
    };
    setup.AddSecurityDefinition(jwtSecurityScheme.Reference.Id, jwtSecurityScheme);
    setup.AddSecurityRequirement(new OpenApiSecurityRequirement
 {
 { jwtSecurityScheme, Array.Empty<string>() }
 });
});

#region JWT
builder.Services.Configure<JWTSettings>(builder.Configuration.GetSection("JwtConfig"));
#endregion JWT
```