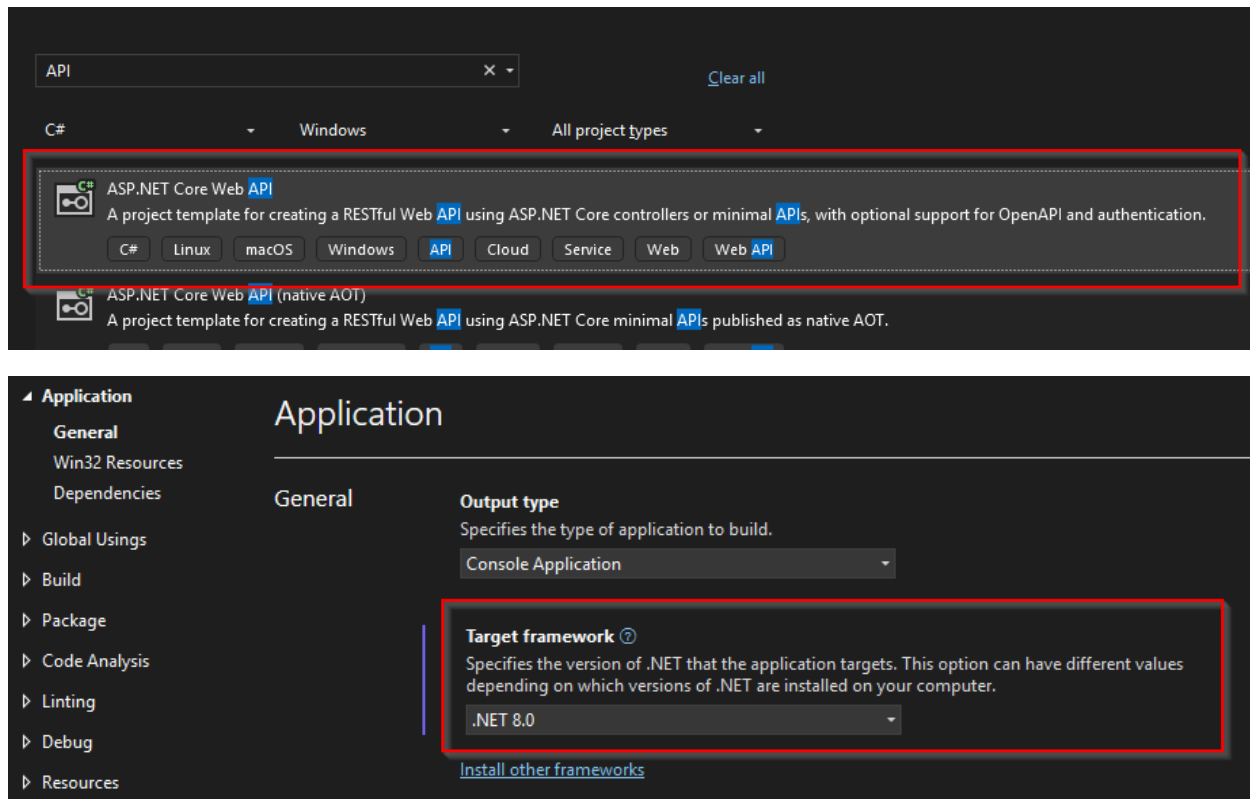


# C# ASP.NET CORE API HOSTING GUIDE

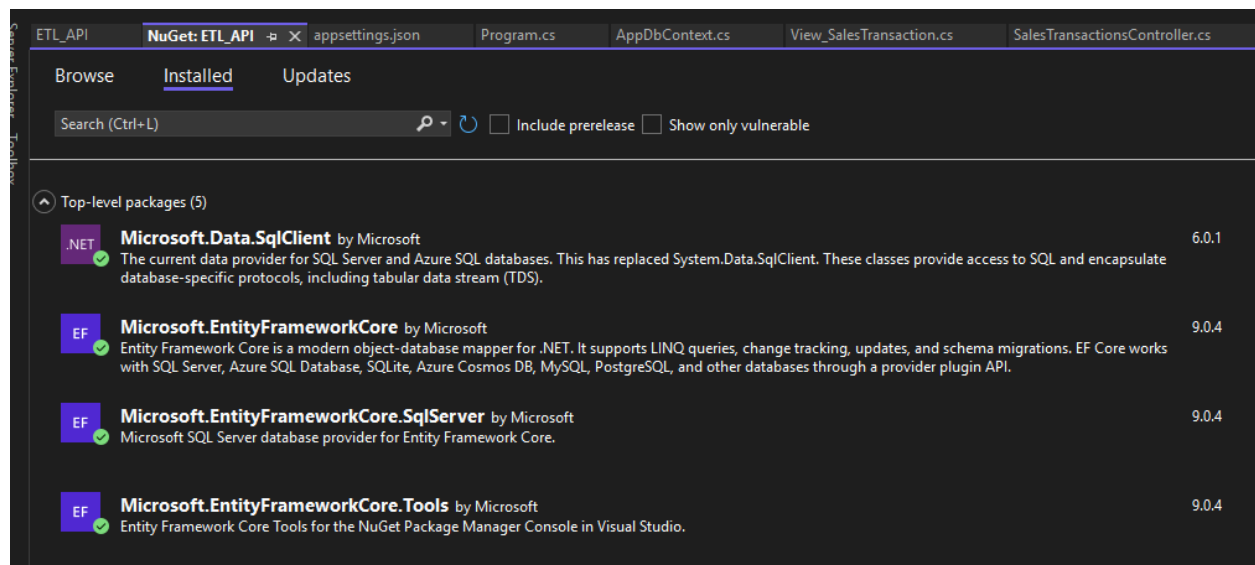
This documentation explain on how to develop an easy Web API Project in C# ASP.NET Core and host it on server or local IIS Service. Web API is a very common topic in software engineering, it is commonly used in both frontend and backend development on various types of projects. To set up a Web API as the service provider, we need to host our project online so others can access through our API urls like “https://domian.com/api/endpoint”.

## 1.0 Project Base Start Up



**1.1 Project Start Up:** First, create a new project in visual studio, choose **ASP .NET Core Web API** and choose **.NET 8** for the project base as it is one of the fastest web frameworks available. Besides, it is ideal for microservices or modular backend architecture and having a strong built-in support for authentication, authorization, and data protection. In short, high performance and high security.

## 2.0 Project Dependencies (NuGet Packages)



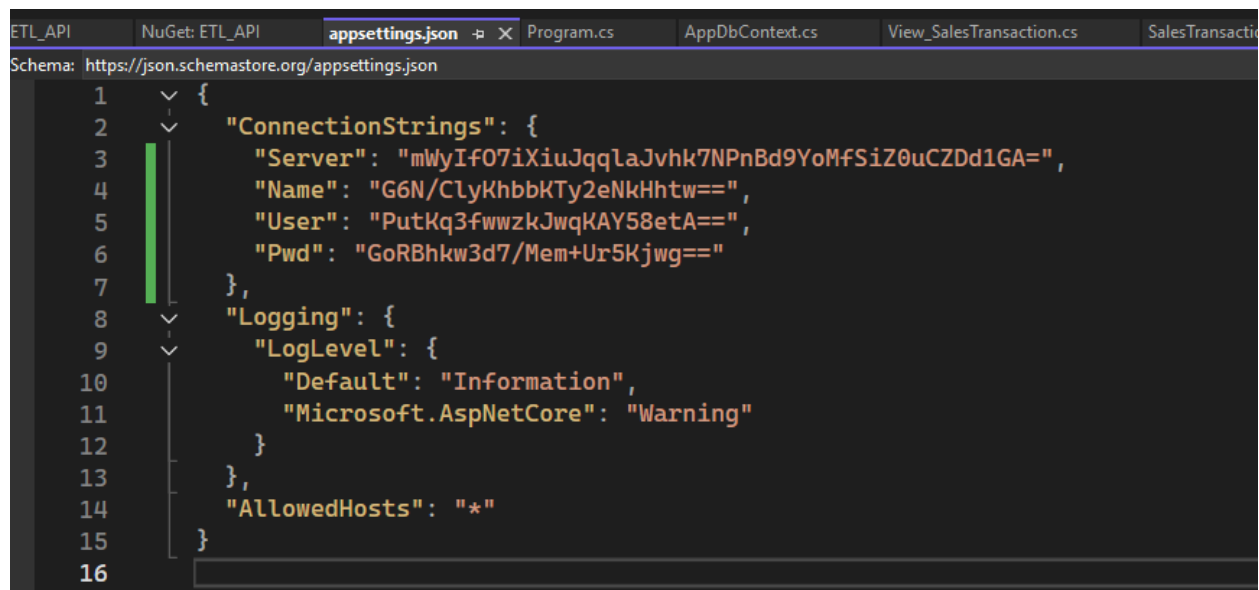
**2.1 Microsoft.Data.SqlClient:** The official .NET provider for SQL Server and Azure SQL Database. It supports async operations, secure connections, connection pooling, and advanced features like Always Encrypted and AAD authentication. It's cross-platform and optimized for modern .NET applications, replacing the older System.Data.SqlClient.

**2.2 Microsoft.EntityFrameworkCore:** The core package for Entity Framework Core (EF Core) provides modern, lightweight, and cross-platform Object-Relational Mapper (ORM) for .NET. It lets you interact with databases using C# classes instead of raw SQL, supports LINQ queries, migrations, change tracking, and works with many database providers like SQL Server, PostgreSQL, and SQLite.

**2.3 Microsoft.EntityFrameworkCore.SqlServer:** The SQL Server database provider for Entity Framework Core. It enables EF Core to work specifically with Microsoft SQL Server, allowing you to run LINQ queries, apply migrations, and map your C# models to SQL Server tables.

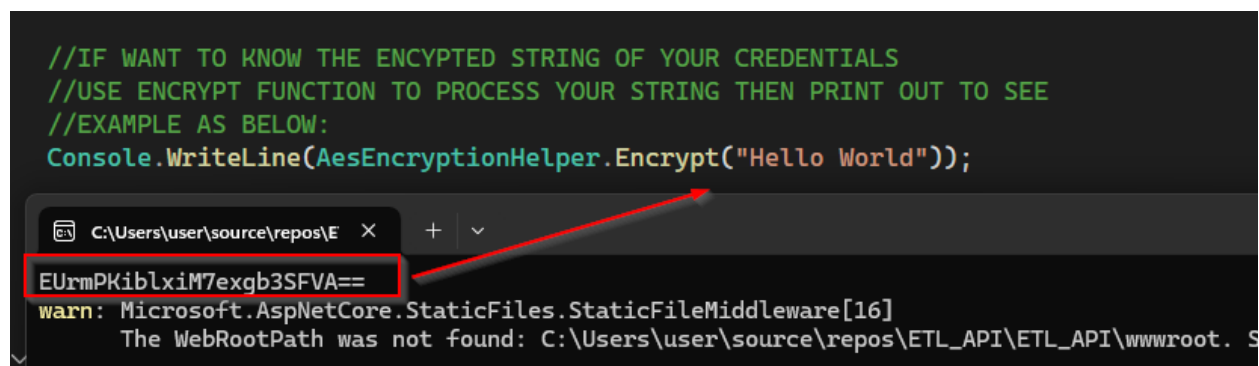
**2.4 Microsoft.EntityFrameworkCore.Tools:** Provides command-line tools for working with Entity Framework Core in .NET projects. It enables commands like Add-Migration, Update-Database, and Scaffold-DbContext, which help with managing database migrations, updating schemas, and reverse-engineering existing databases. These tools are essential for development-time tasks and integration with the Package Manager Console or dotnet ef.

## 3.0 Project Configuration Settings



```
1  {
2  }
3  "ConnectionStrings": {
4    "Server": "mWyIf07iXiuJqqlaJvhk7NPnBd9YoMfSiZ0uCZDd1GA=",
5    "Name": "G6N/ClyKhbbKTy2eNkHhtw==",
6    "User": "PutKq3fwwzkJwqKAY58etA==",
7    "Pwd": "GoRBhkw3d7/Mem+Ur5Kjwg=="
8  },
9  "Logging": {
10   "LogLevel": {
11     "Default": "Information",
12     "Microsoft.AspNetCore": "Warning"
13   }
14 },
15 "AllowedHosts": "*"
16 }
```

**3.1 Configuration Settings:** Inside the appsetting.json file is where we set the project configuration settings such as project connection string, logging settings and the project hostnames. In this project, we only set up the connection strings credentials for connecting to the project database. The credentials are also purposely encrypted in **AES (Advanced Encryption Standard)**, it's a symmetric encryption algorithm used to securely encrypt and decrypt data. It's fast, secure, and used worldwide including by governments, banks, and tech companies. Others can just leave them as default.



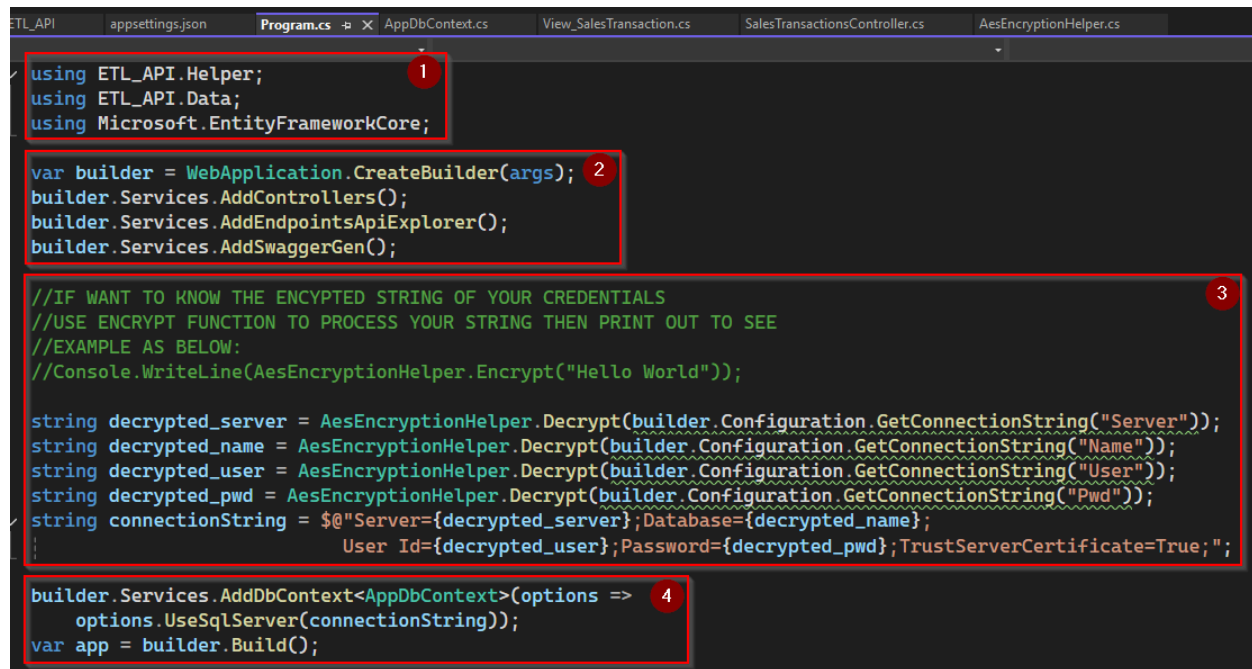
```
//IF WANT TO KNOW THE ENCRYPTED STRING OF YOUR CREDENTIALS
//USE ENCRYPT FUNCTION TO PROCESS YOUR STRING THEN PRINT OUT TO SEE
//EXAMPLE AS BELOW:
Console.WriteLine(AesEncryptionHelper.Encrypt("Hello World"));
```

EUrmPKiblxiM7exgb3SFVA==

warn: Microsoft.AspNetCore.StaticFiles.StaticFileMiddleware[16]  
The WebRootPath was not found: C:\Users\user\source\repos\ETL\_API\ETL\_API\wwwroot. S

**3.2 Credential Encryption:** This is the example on how to retrieve the encrypted string using the AES Encryption Helper Function then copy and paste into the appsettings file. The example is provided inside the **Program.cs** file (Refer 4.3).

## 4.0 Program Build



```
using ETL_API.Helper;
using ETL_API.Data;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

//IF WANT TO KNOW THE ENCRYPTED STRING OF YOUR CREDENTIALS
//USE ENCRYPT FUNCTION TO PROCESS YOUR STRING THEN PRINT OUT TO SEE
//EXAMPLE AS BELOW:
//Console.WriteLine(AesEncryptionHelper.Encrypt("Hello World"));

string decrypted_server = AesEncryptionHelper.Decrypt(builder.Configuration.GetConnectionString("Server"));
string decrypted_name = AesEncryptionHelper.Decrypt(builder.Configuration.GetConnectionString("Name"));
string decrypted_user = AesEncryptionHelper.Decrypt(builder.Configuration.GetConnectionString("User"));
string decrypted_pwd = AesEncryptionHelper.Decrypt(builder.Configuration.GetConnectionString("Pwd"));
string connectionString = $"Server={decrypted_server};Database={decrypted_name};
User Id={decrypted_user};Password={decrypted_pwd};TrustServerCertificate=True;";

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(connectionString));
var app = builder.Build();
```

**4.1 Dependencies:** These are the required dependencies for setting up the Program.cs such as the Helper folder is for the AES Encryption, then the Data folder is to get the AppDbContext, and Microsoft.EntityFrameworkCore for building the actual database connection.

**4.2 Services:** These are the standard services added by default for building the web application base. AddControllers() to include the use of Controller files, AddEndpointsAPIExporer() to allow setting up the API endpoints, then AddSwaggerGen() to include Swagger services such as the swagger UI page for API overviews when we started up the project.

**4.3 Connection String:** This section showing how to get the connection string for connecting to the database, the project included an encryption feature to protect the actual connection string credentials prepared in the appsetting.json file. By retrieving the encrypted values and decrypting them using the same keys (*refer 1.1*), the actual values will then be used to build the complete connection string.

**4.4 Build App:** After the connection string is retrieved successfully, it will be used to connect to the database then map to the AppDbContext where the views or tables are constructed. Then, the app is built if the database connection succeeded.

```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment() || app.Environment.IsStaging()
    || app.Environment.IsProduction())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

```
app.MapGet("/", context =>
{
    context.Response.Redirect("/swagger", permanent: false);
    return Task.CompletedTask;
});
```

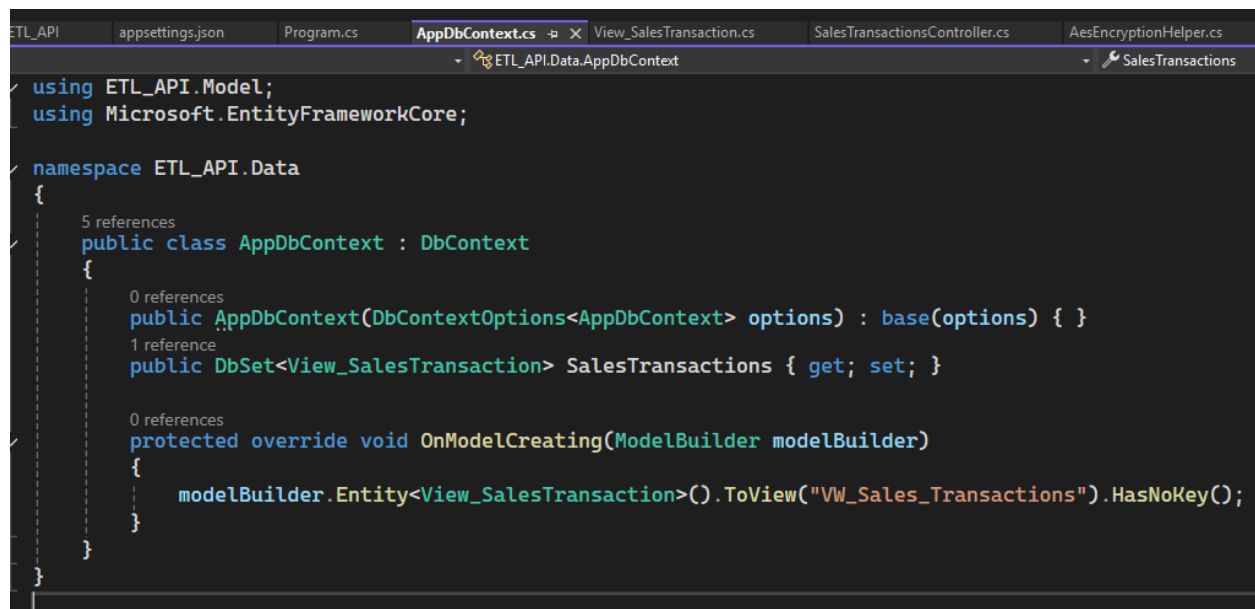
```
app.UseHttpsRedirection();
app.UseAuthorization();
app.UseStaticFiles();
app.UseDefaultFiles();
app.MapControllers();
app.Run();
```

**4.5 Pipeline:** The request pipeline controls which UI or page the project will be pointing to as default in different stages. In this project, there is no UI provided except the default Swagger API Overview Page. Therefore in both development and production stages, the project will be targeting the swagger UI page as default.

**4.6 Url Mapping:** The url mapping allows us to customize the redirect url according to the url pattern. In this case, when the url pattern is by default like "https://domain.com/", we will be redirected to "https://domain.com/swagger" then triggers inside the swagger url mapping which also pointing to "https://domain.com/swagger/index.html" which brings us to the Swagger API Overview Page.

**4.7 Run App:** These are the standard functions included in the app for higher accessibility of the web application. Each function provides different services accordingly where commonly used during accessing the web application. Finally the app is built after finishing adding the functions.

## 5.0 Data Mapping

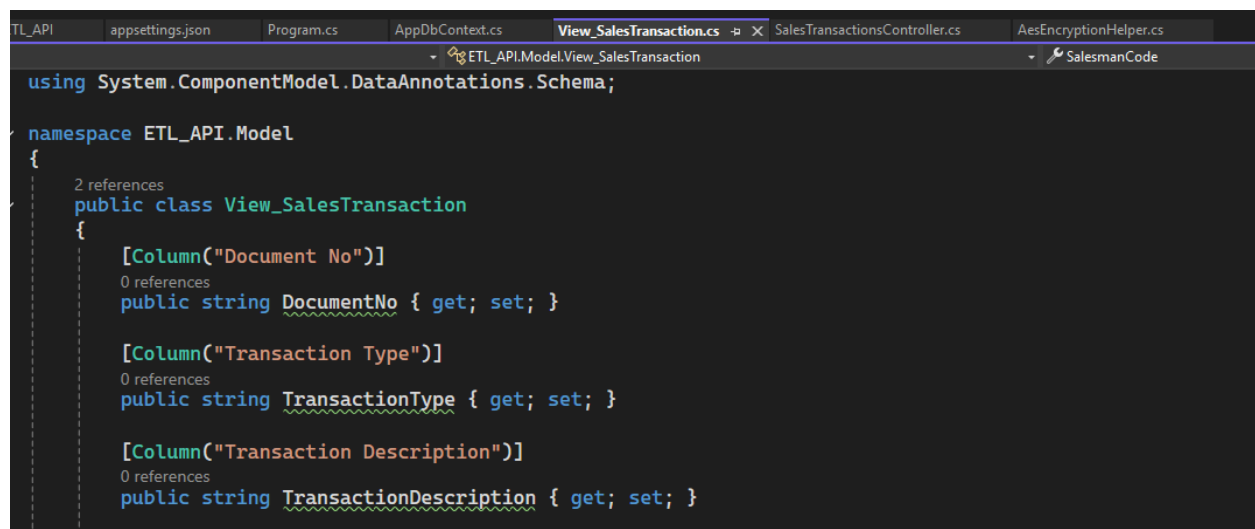


```
using ETL_API.Model;
using Microsoft.EntityFrameworkCore;

namespace ETL_API.Data
{
    5 references
    public class AppDbContext : DbContext
    {
        0 references
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }
        1 reference
        public DbSet<View_SalesTransaction> SalesTransactions { get; set; }

        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<View_SalesTransaction>().ToView("VW_Sales_Transactions").HasNoKey();
        }
    }
}
```

**5.1 Data.AppDbContext:** In this project, there is only 1 view used for the sales transaction details API, the view is purposely created in the database for the API used and mapped to the program using Microsoft.EntityFrameworkCore component.



```
using System.ComponentModel.DataAnnotations.Schema;

namespace ETL_API.Model
{
    2 references
    public class View_SalesTransaction
    {
        [Column("Document No")]
        0 references
        public string DocumentNo { get; set; }

        [Column("Transaction Type")]
        0 references
        public string TransactionType { get; set; }

        [Column("Transaction Description")]
        0 references
        public string TransactionDescription { get; set; }
    }
}
```

**5.2 Model.View\_SalesTransaction:** In this View\_SalesTransaction class file, we need to create the data structure exactly like the view we created previously. Then, by specifying the column names referring to the view columns, the data is automatically mapped to the correct attributes.

```
VW_Sales_Transacti...master (green (53))
CREATE VIEW [dbo].[VW_Sales_Transactions] AS
SELECT
    [Document No],
    [Transaction Type],
    'SALES' AS [Transaction Description],
    [Stock] AS [Stock Code],
    [Quantity],
    [Invoice Amount] AS [Total Amount],
    [Unit Cost],
    ar.[Location],
    [Document Date] AS [Issue Date],
    [Customer] AS [Customer Code],
    [Customer Name],
    [Salesman] AS [Salesman Code],
    sm.[Salesman Name] AS [Salesman Name],
    [Nett Qty]
FROM view_AR_Invoice_Tran ar WITH(NOLOCK)
INNER JOIN Stock st WITH(NOLOCK)
ON ar.[Stock] = st.[Stock Code]
LEFT JOIN Salesman_tbl sm
ON ar.[Salesman] = sm.[Salesman Code]

UNION ALL
```

```
SELECT
    [Document No],
    [Transaction Type],
    'RETURN' AS [Transaction Description],
    [Stock] AS [Stock Code],
    [Quantity],
    [CN Amount] AS [Total Amount],
    [Unit Cost],
    ar.[Location],
    [Document Date],
    [Customer] AS [Customer Code],
    [Customer Name],
    [Salesman] AS [Salesman Code],
    sm.[Salesman Name] AS [Salesman Name],
    [Nett Qty]
FROM view_AR_CN_Tran ar WITH(NOLOCK)
INNER JOIN Stock st WITH(NOLOCK)
ON ar.[Stock] = st.[Stock Code]
LEFT JOIN Salesman_tbl sm
ON ar.[Salesman] = sm.[Salesman Code];
```

**5.3 VW\_Sales\_Transactions:** Remember to create the View table inside the database, and make sure the column names are matched with the Model.View\_SalesTransaction (Refer 5.2).

## 6.0 API Controller

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ETL_API.Data;

namespace ETL_API.Controllers
{
    [Route("api/sales")]
    [ApiController]
    1 reference
    public class SalesTransactionsController : ControllerBase
    {
```

```
        [HttpPost("transactions")]
        0 references
        public async Task<IActionResult> GetTransactions([FromBody] SalesQueryRequest request)
        {
            if (request.Limit > 50)
                return BadRequest("Limit cannot exceed 50.");

            if (request.Page < 1 || request.Limit < 1)
                return BadRequest("Page and limit must be greater than 0.");

            var filteredData = await _context.SalesTransactions
                .Where(t => t.IssueDate >= request.StartDate && t.IssueDate <= request.EndDate)
                .ToListAsync();

            var totalCount = filteredData.Count;

            var pagedData = filteredData
                .OrderBy(t => t.IssueDate)
                .Skip((request.Page - 1) * request.Limit)
                .Take(request.Limit)
                .ToList();

            return Ok(new
            {
                totalCount,
                request.Page,
                request.Limit,
                results = pagedData
            });
        }
    }
}
```

```
public class SalesQueryRequest
{
    public DateTime StartDate { get; set; }

    public DateTime EndDate { get; set; }

    public int Page { get; set; }
    5 references
    public int Limit { get; set; }
}
```



**6.1 Dependencies:** Microsoft.AspNetCore.Mvc is for model view controls, but this project does not involve using views. Microsoft.EntityFrameworkCore for using sql data functions while Data folder include the ApplicationDbContext for the data process.

**6.2 GetTransactions():** This is the POST API function where users can get sales transactions with the url endpoint `"/api/sales/transactions"`. In this function, the API has set the max limit to 50 transactions per request and users can select which page to view. Other than that, users are also required to provide the start date and end date for retrieving the transactions by date.

## SalesTransactions

POST

/api/sales/transactions

Parameters

No parameters

Request body

```
{  "startDate": "2020-01-01",  "endDate": "2020-12-31",  "page": 1,  "limit": 20}
```

Request URL

https://localhost:7097/api/sales/transactions

Server response

Code

Details

200

Response body

```
{  "totalCount": 26965,  "page": 1,  "limit": 20,  "results": [    {      "documentNo": "CS0029468",      "transactionType": "INV",      "transactionDescription": "SALES",      "stockCode": "OIL-TQ18",      "quantity": 1,      "totalAmount": 160,      "unitCost": 0,      "location": "L1",      "issueDate": "2020-01-02T00:00:00",      "customerCode": "3000/ZZZ",      "customerName": "CASH SALES",      "salesmanCode": "C0",      "salesmanName": "COMPANY",      "nettQty": 0    }  ],  }
```

## 7.0 Helper.AesEncryption

```
using System.Security.Cryptography;
using System.Text;

namespace ETL_API.Helper
{
    4 references
    public class AesEncryptionHelper
    {
        //Can use this link to test your secret string - https://mothereff.in/byte-counter
        private static readonly byte[] Key = Encoding.UTF8.GetBytes("GreenstemBusinessSoftwareAPI2383"); //Must Exactly 32 Bytes
        private static readonly byte[] IV = Encoding.UTF8.GetBytes("Gb$b62633933@##"); //Must Exactly 16 Bytes

        0 references
        public static string Encrypt(string plainText)...

        4 references
        public static string Decrypt(string encryptedText)...
    }
}
```

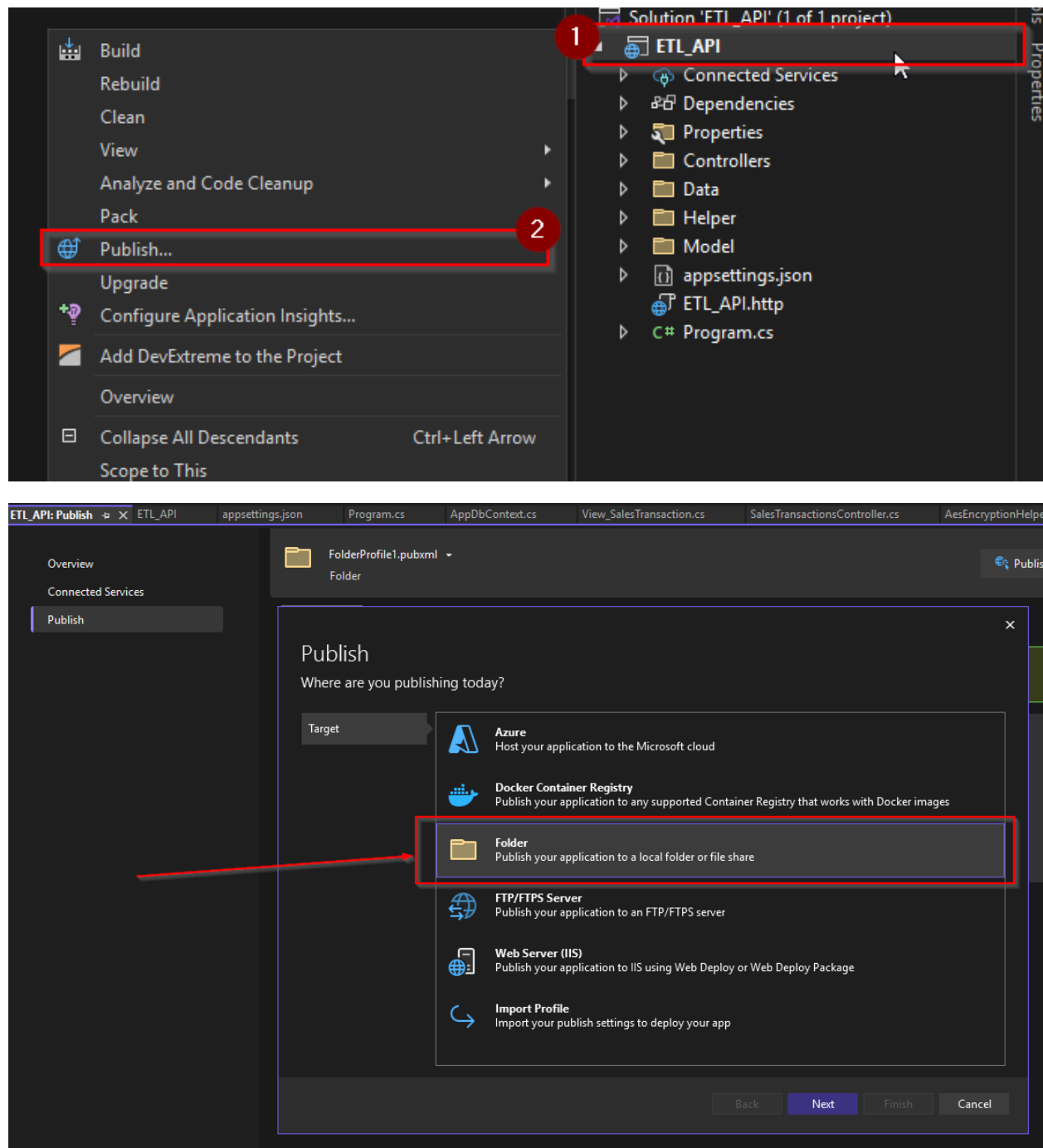
```
//IF WANT TO KNOW THE ENCRYPTED STRING OF YOUR CREDENTIALS
//USE ENCRYPT FUNCTION TO PROCESS YOUR STRING THEN PRINT OUT TO SEE
//EXAMPLE AS BELOW:
//Console.WriteLine(AesEncryptionHelper.Encrypt("Hello World"));

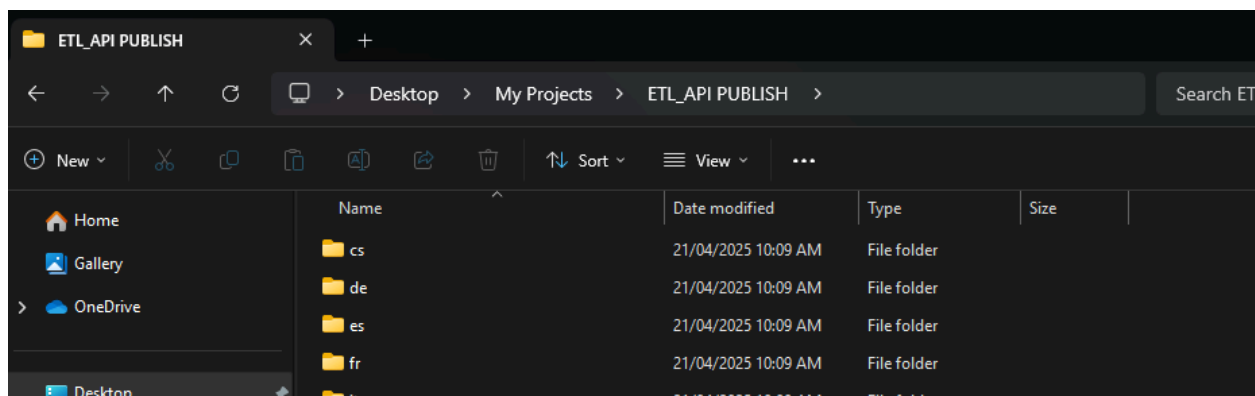
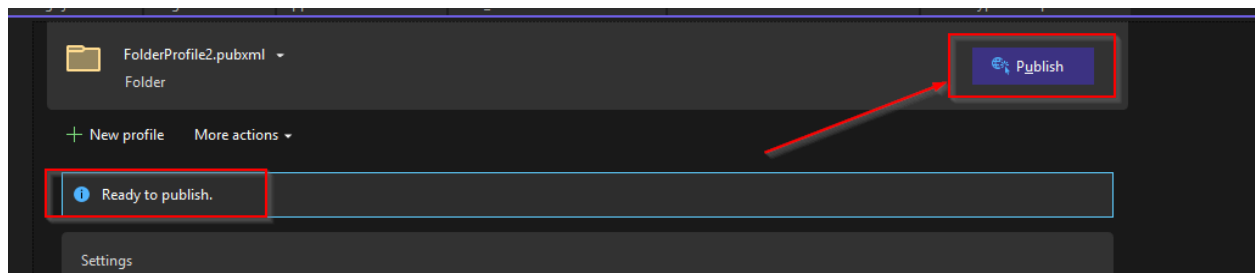
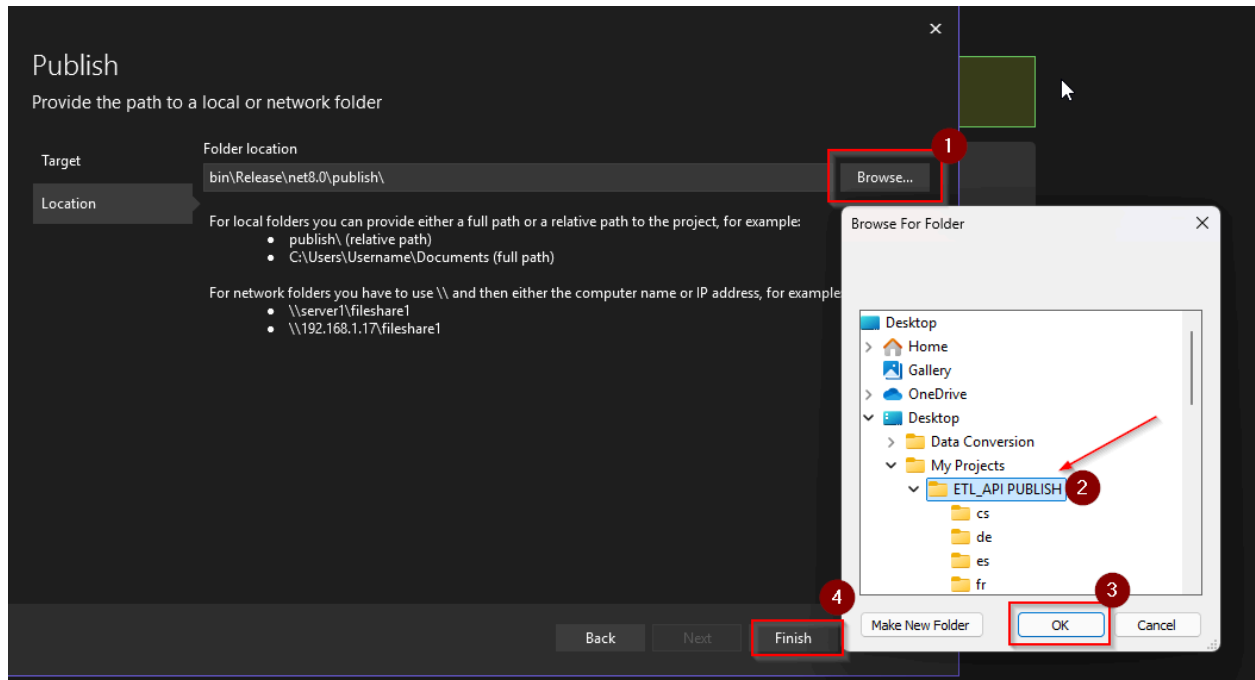
string decrypted_server = AesEncryptionHelper.Decrypt(builder.Configuration.GetConnectionString("Server"));
string decrypted_name = AesEncryptionHelper.Decrypt(builder.Configuration.GetConnectionString("Name"));
string decrypted_user = AesEncryptionHelper.Decrypt(builder.Configuration.GetConnectionString("User"));
string decrypted_pwd = AesEncryptionHelper.Decrypt(builder.Configuration.GetConnectionString("Pwd"));
string connectionString = $"Server={decrypted_server};Database={decrypted_name};
User Id={decrypted_user};Password={decrypted_pwd};TrustServerCertificate=True;"
```

```
appsettings.json
{
  "ConnectionStrings": {
    "Server": "mWyIf07iXiuJqqlaJvhk7NPnBd9YoMfSiZ0uCZDd1GA=",
    "Name": "G6N/ClyKhbbKTy2eNkHhtw==",
    "User": "PutKq3fwwzkJwqKAY58etA==",
    "Pwd": "GoRBhkw3d7/Mem+Ur5Kjwg=="
  }
}
```

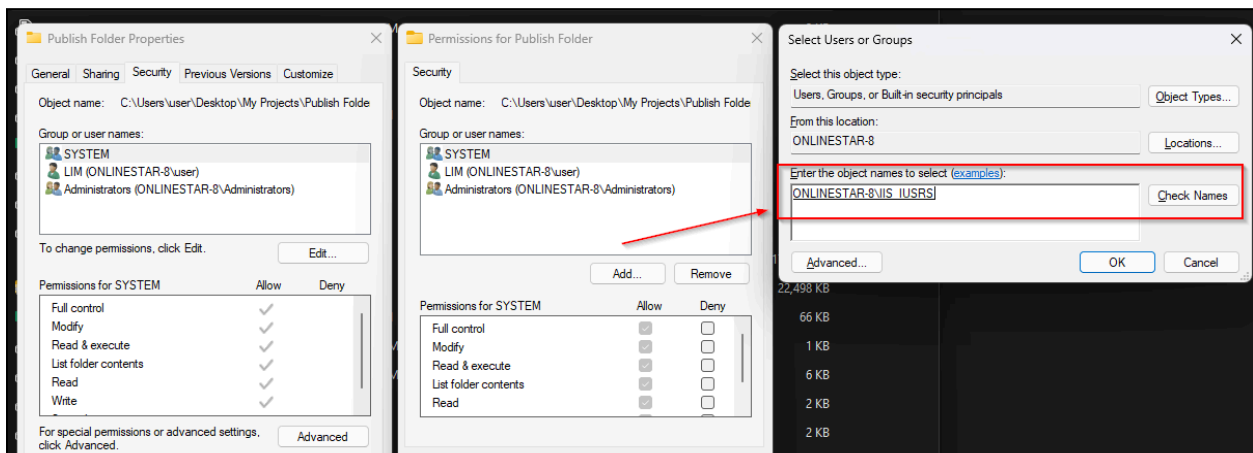
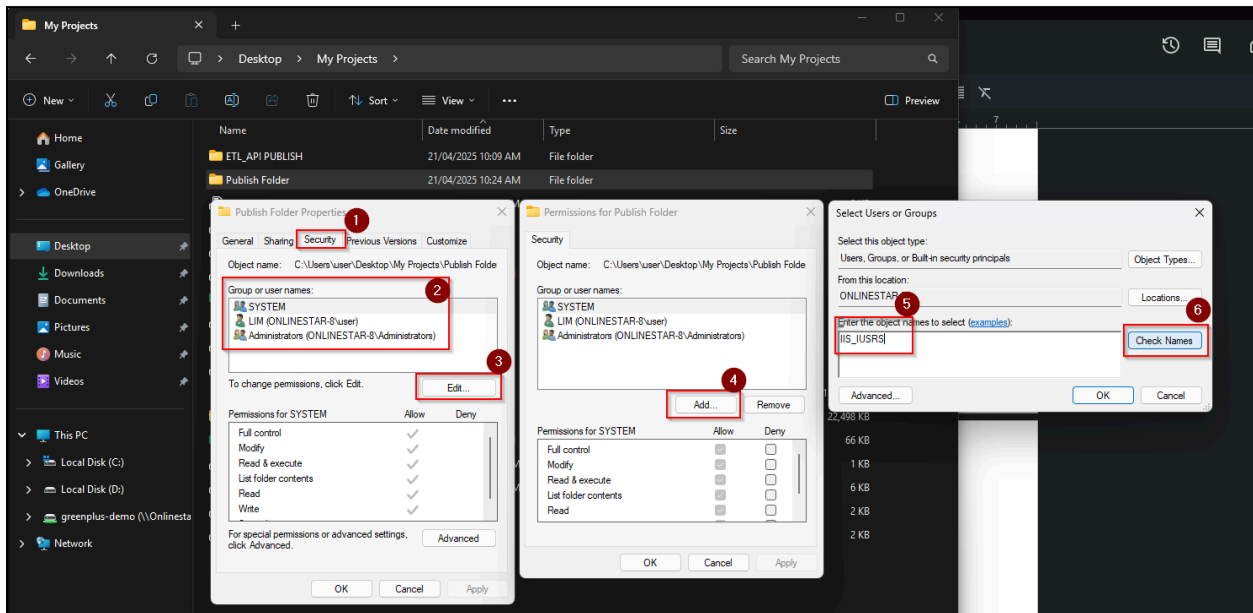
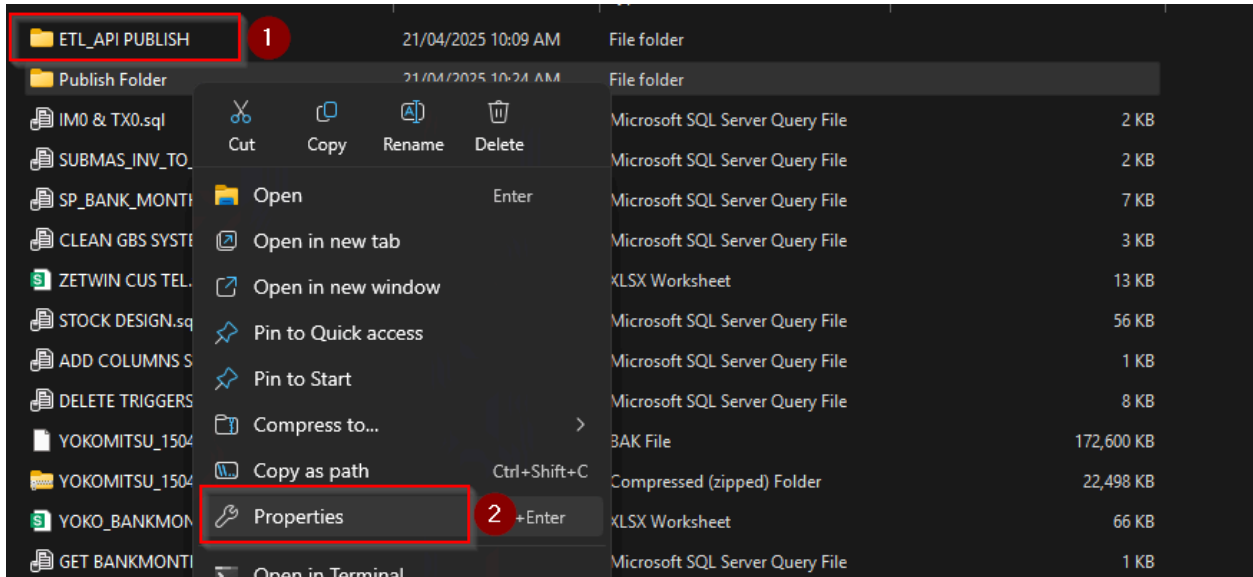
**7.1 AES Encryption:** This is a helper function used for encryption, the credential information inside appsettings are vulnerable and can be viewed easily, therefore the encryption function is needed for the credential encryption. Apart from that, in future the encryption can also be applied in other functions for data protection purposes.

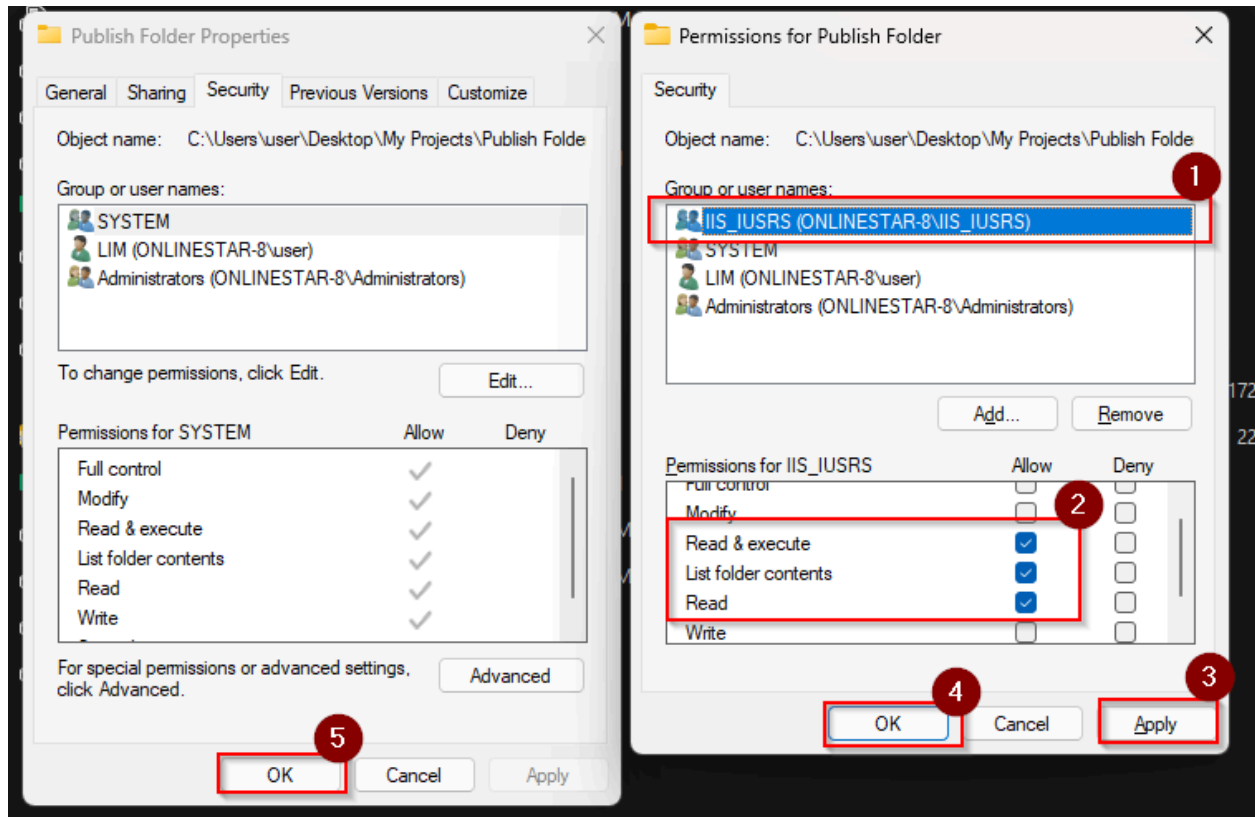
## 8.0 Project Publish





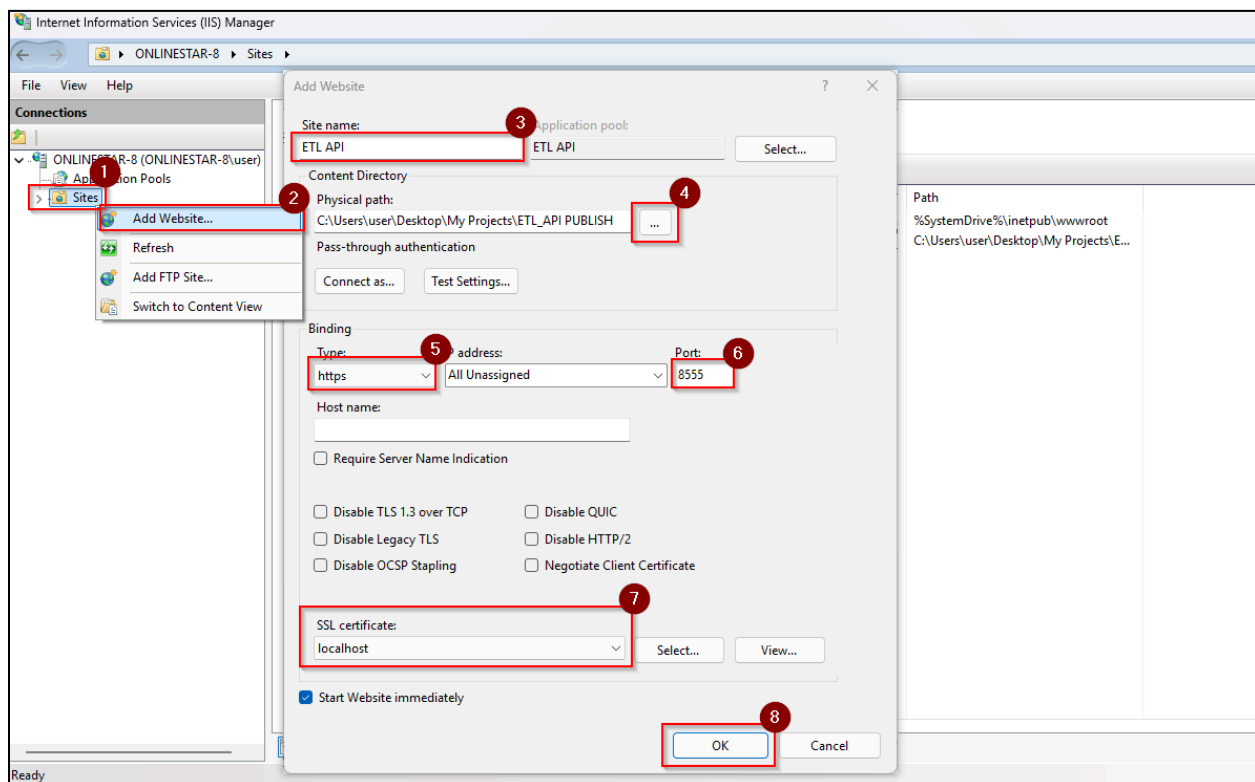
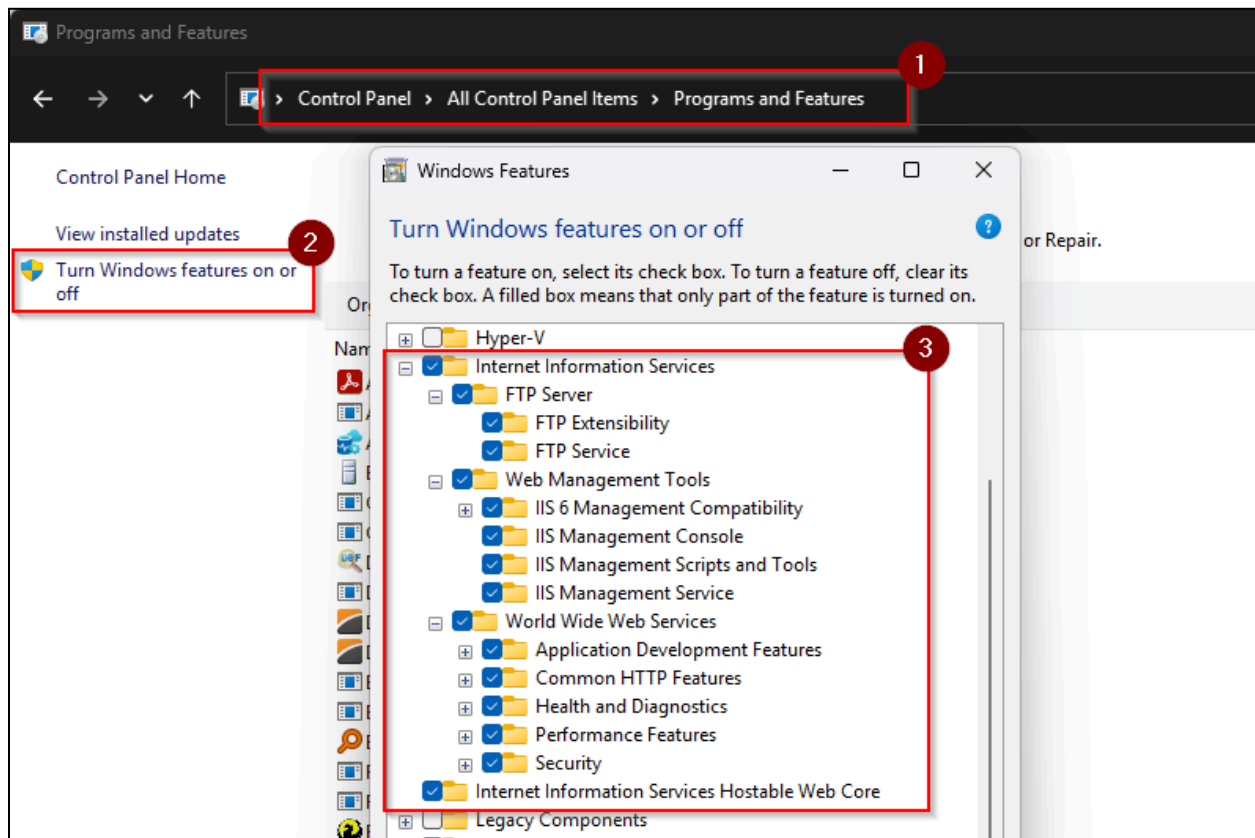
**8.1 Publish:** Right click your project, select Publish. Then you will see the publish panel, select Folder then next. After that, choose your folder directory for the publish, you may create a new folder, then ok, finish and close. You will see the project is ready to publish, now click the Publish button on top right. You should now see the published components and dependencies are created inside the folder.





**8.2 Published Folder:** Now, we need to make sure the folder has the IIS\_IUSRS user and be allowed to read this folder before we deploy into the IIS Manager. Right click and select Properties, go to the Security tab, check for IIS\_IUSRS, if there's none, select edit, add, and type IIS\_IUSRS then Check Names, it will automatically select the IIS user (*If Not, Try to install IIS First. Refer 9.1*). Then, make sure that IIS\_IUSRS has permissions on Read & Execute, List folder contents, and Read then click Apply and OK.

## 9.0 Install IIS



✂ If no certificate exists:

- Run this in PowerShell to generate a dev cert:

```
powershell
New-SelfSignedCertificate -DnsName "localhost" -CertStoreLocation "cert:\LocalMachine\My"
```

Then, bind that cert in IIS (it will appear in the list).

Run apps - Runtime ⓘ

## ASP.NET Core Runtime 8.0.15

The ASP.NET Core Runtime enables you to run existing web/server applications. On Windows, we recommend installing the Hosting Bundle, which includes the .NET Runtime and IIS support.

**IIS runtime support (ASP.NET Core Module v2)**

18.0.25074.15

OS	Installers	Binaries
Linux	<a href="#">Package manager instructions</a>	<a href="#">Arm32</a>   <a href="#">Arm32 Alpine</a>   <a href="#">Arm64</a>   <a href="#">Arm64 Alpine</a>   <a href="#">x64</a>   <a href="#">x64 Alpine</a>
macOS		<a href="#">Arm64</a>   <a href="#">x64</a>
Windows	<a href="#">x64</a>   <a href="#">x86</a>   <a href="#">Arm64</a>   <a href="#">Hosting Bundle</a>   <a href="#">winget instructions</a>	<a href="#">x64</a>   <a href="#">x86</a>   <a href="#">Arm64</a>

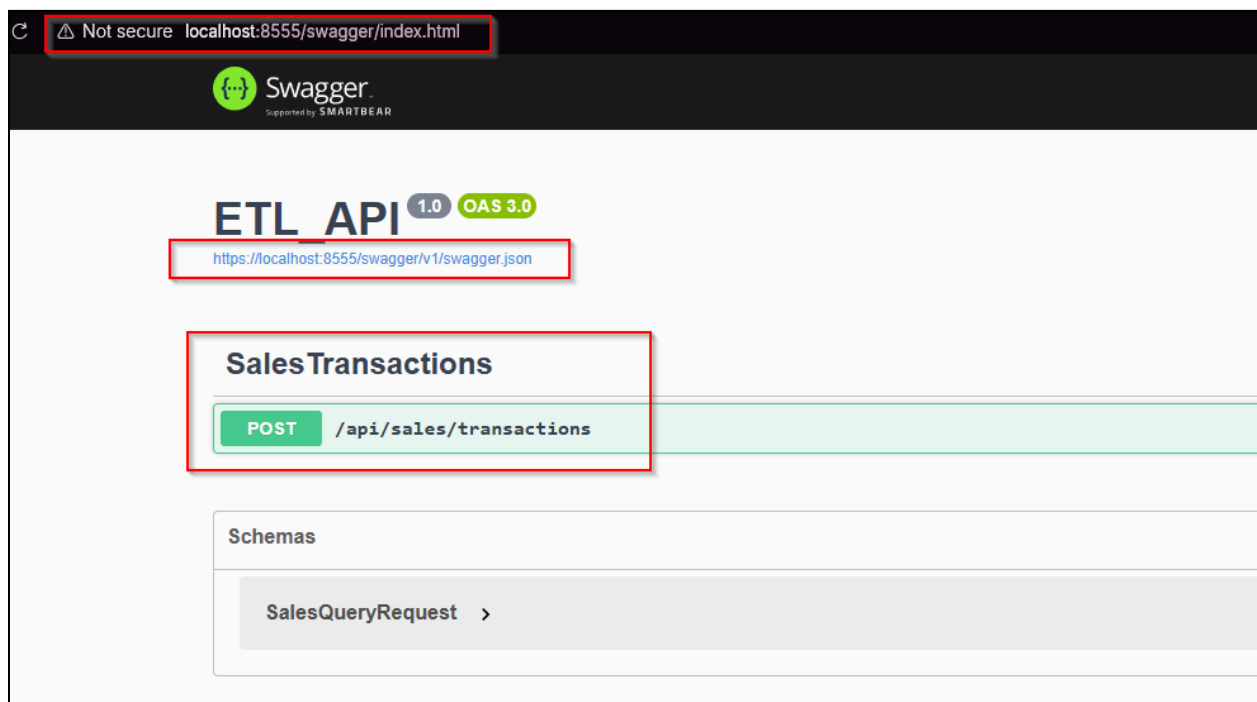
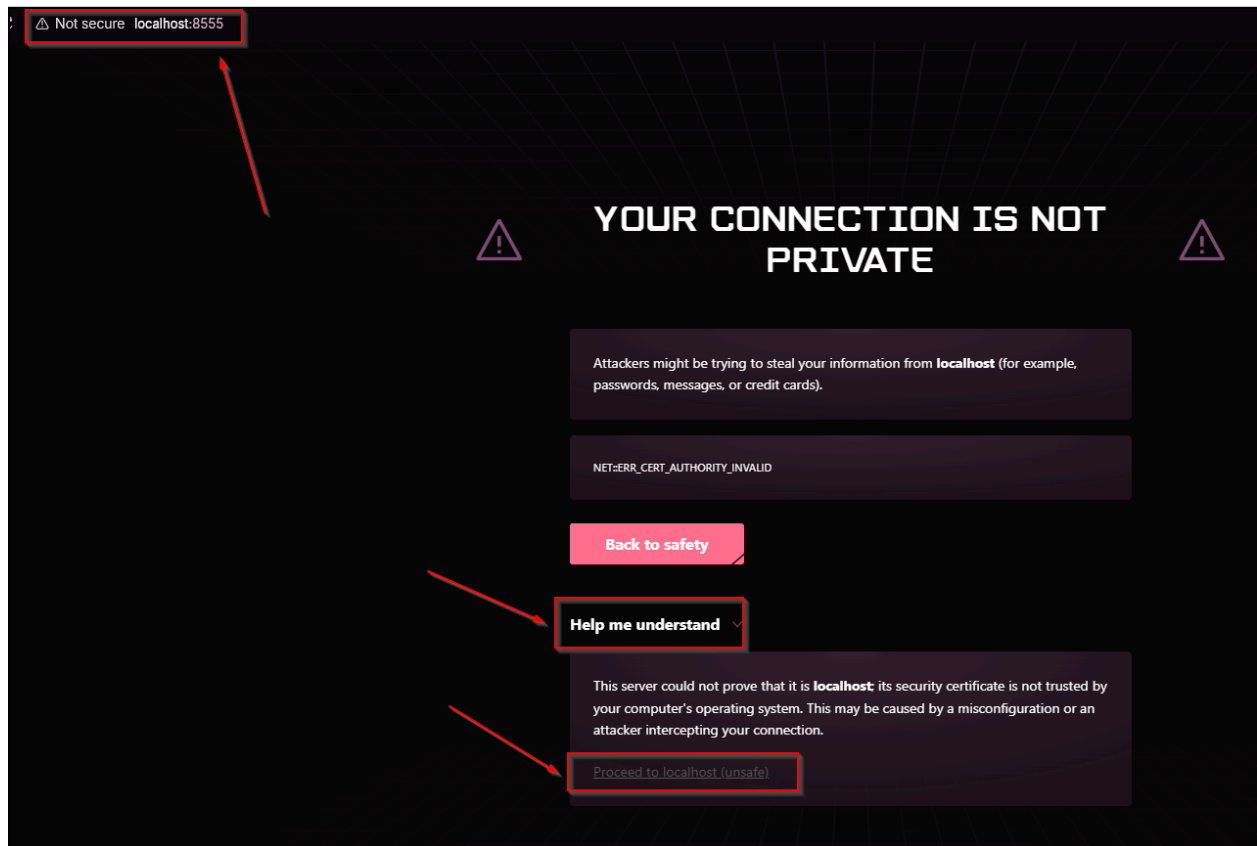
**9.1 IIS Set Up:** If your local PC does not have IIS Manager installed and ready, go to the Control Panel, select Programs and Features, on the left panel, select Turn Windows features on or off, then scroll down and find Internet Information Services and Internet Information Service Hostable Web Core. Select All services Then wait for the service to install on your PC.

**9.2 Host Website:** Open IIS Manager, go to Sites and right click, select Add Website. Fill in the Site Name, then browse the physical path to the publish folder created previously (*Refer 8.1*). Then the Binding selects https and changes the port to 8555 (customized port). For the SSL certificate, we could use a registered domain hostname or we can generate a localhost certificate. Method to create a localhost certificate provided above.

**9.3 .NET Hosting Bundle:** Remember to install the .NET Hosting Bundle, version depends on the project version. This must be installed for the IIS to read your web.config properly and host your project.



## 10.0 Test Project Deployment



POST

/api/sales/transactions

Parameters

No parameters

Request body

```
{  "startDate": "2020-01-01",  "endDate": "2020-12-31",  "page": 1,  "limit": 20}
```

Request URL

https://localhost:8555/api/sales/transactions

Server response

Code	Details
200	<div>Response body</div> <div><pre>{  "totalCount": 26965,  "page": 1,  "limit": 20,  "results": [    {      "documentNo": "CS0029468",      "transactionType": "INV",      "transactionDescription": "SALES",      "stockCode": "OIL-TQ18",      "quantity": 1,      "totalAmount": 160,      "unitCost": 0,      "location": "L1",      "issueDate": "2020-01-02T00:00:00",      "customerCode": "3000/ZZZ",      "customerName": "CASH SALES",      "salesmanCode": "CO",      "salesmanName": "COMPANY",      "nettQty": 0    }  ]}</pre></div>

POST

https://localhost:8555/apl/sales/transactions

Params • Authorization Headers (9) Body • Scripts Tests Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON

```
1 {2   "startDate": "2020-01-01",3   "endDate": "2020-12-31",4   "page": 1,5   "limit": 206 }
```

Body Cookies Headers (5) Test Results

{}

JSON

Preview

Visualize

```
1 {2   "totalCount": 26965,3   "page": 1,4   "limit": 20,5   "results": [6     {7       "documentNo": "CS0029468",8       "transactionType": "INV",9       "transactionDescription": "SALES",10      "stockCode": "OIL-TQ18",11      "quantity": 1.00,
```

**10.1 Access Website:** The browser might warn you that the connection is not safe as the SSL certificate is using a localhost SSL, but you can still proceed to the website. If successful, you will see the Swagger Overview Page and the project APIs are listed here. You can try them out using swagger or using Postman, then check the results.

## 11.0 Future Enhancements

```
CREATE TABLE API_TOKENS
(
    [ID] INT IDENTITY PRIMARY KEY,
    [User Name] VARCHAR(50),
    [Token] NVARCHAR(500),
    [Created Date Time] DATETIME,
    [Is Active] BIT DEFAULT(1)
)

SELECT * FROM API_TOKENS
```

156 %

Results Messages

	ID	User Name	Token	Created Date Time	Is Active
1	1	ABC Company	80372471c5e7067e18c79f9880424cfd6dcc0a9b730a14ff...	2025-04-21 12:22:50.993	1

POST https://localhost:8555/api/sales/transactions

Params • Authorization Headers (10) Body • Scripts Tests Settings

Headers 9 hidden

	Key	Value	Desc
<input checked="" type="checkbox"/>	Access-Token	80372471c5e7067e18c79f9880424cfd6dcc0a9b730a14ff90ce0e9f95cf5d96	
	Key	Value	Desc

**11.1 Access Token:** In future, for more secure API usage, we might need to add an additional table to keep track of the API authenticated user tokens. Only authenticated tokens are allowed to use the API. Use case example provided above.