**Setting up Identity Server 4**

Steps:

1. Create a new dotnet core web application (Web API) using the ‘**dotnet new webapi Identity.API’**.
2. Open the project folder in VS Code
3. Add the following packages to the project

***dotnet add package IdentityServer4 --version 2.1.2***

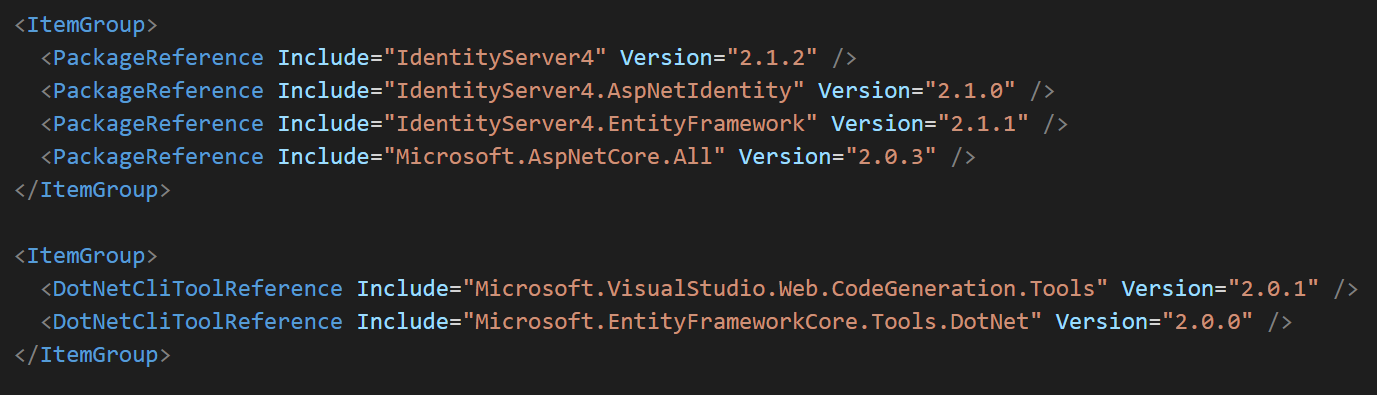
***dotnet add package IdentityServer4.AspNetIdentity --version 2.1.0***

***dotnet add package IdentityServer4.EntityFramework --version 2.1.1***

1. Add the following package as DotNetCLIReferenceTool

***Microsoft.EntityFrameworkCore.Tools.DotNet***

1. You can verify the packages list in the **Identity.API.csproj** file



1. Create a Configuration class to store the list of API resources, Identity resources and Clients lists. For this create a new file in the project root folder with the name Config.cs. Add the following code to that file.

public static class Config

{

public static IEnumerable<ApiResource> GetApiResources()

{

return new List<ApiResource>

{

new ApiResource("orders", "Orders Service"),

new ApiResource("basket", "Basket Service"),

new ApiResource("marketing", "Marketing Service"),

new ApiResource("locations", "Locations Service")

};

}

public static IEnumerable<IdentityResource> GetIdentityResources()

{

return new List<IdentityResource>

{

new IdentityResources.OpenId(),

new IdentityResources.Profile()

};

}

public static IEnumerable<Client> GetClients(Dictionary<string, string> clientsUrl)

{

return new List<Client>

{

new Client

{

ClientId="spa",

ClientName="eShop SPA OpenId Client",

AllowedGrantTypes=GrantTypes.Implicit,

RedirectUris={ $"{clientsUrl["Spa"]}/" },

PostLogoutRedirectUris={ $"{clientsUrl["Spa"]}/" },

AllowedCorsOrigins ={ $"{clientsUrl["Spa"]}" },

AllowAccessTokensViaBrowser = true,

RequireConsent=false,

ClientSecrets=new []{ new Secret("MySecret".Sha256())},

AllowedScopes=new List<string>

{

IdentityServerConstants.StandardScopes.OpenId,

IdentityServerConstants.StandardScopes.Profile,

"orders",

"basket",

"locations",

"marketing"

}

}

};

}

}

1. The Config.cs file provides the list of APIs you want to protect using identity server, list of client applications that can access the resources and the identity resource types that a client can access.
2. Now you need to create a model class to store the users credential information. You can use the built-in class **IdentityUser** or a custom class by extending the IdentityUser class to add more fields to the class. Here I am creating a class named **ApplicationUser** in the **Models** folder with the following fields.

public class ApplicationUser : IdentityUser

{

[Required]

public string CardNumber { get; set; }

[Required]

public string SecurityNumber { get; set; }

[Required]

[RegularExpression(@"(0[1-9]|1[0-2])\/[0-9]{2}", ErrorMessage = "Expiration should match a valid MM/YY value")]

public string Expiration { get; set; }

[Required]

public string CardHolderName { get; set; }

public int CardType { get; set; }

[Required]

public string Street { get; set; }

[Required]

public string City { get; set; }

[Required]

public string State { get; set; }

[Required]

public string Country { get; set; }

[Required]

public string ZipCode { get; set; }

[Required]

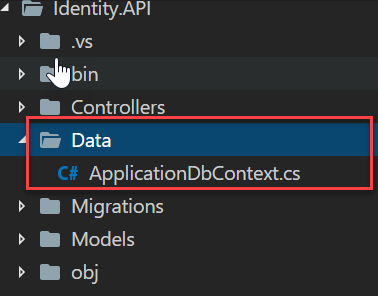
public string Name { get; set; }

[Required]

public string LastName { get; set; }

}

1. You also need to create a Database context class by which you can access the credentials of the users. Add a new folder called ‘**Data’** and create a new file called ‘**ApplicationDbContext.cs**’ inside it.



1. Add the following code in the ApplciationDbContext.cs file.

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using Synergetics.Services.Identity.API.Models;

namespace Synergetics.Services.Identity.API.Data

{

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

}

}

}

1. Next, we need to configure the IdentitySever in the Startup file. Add the following lines of code in the ConfigureServices method of the Startup file. With this code you are configuring the storage for credentials, APIs, Users. Here, I am using Sqlite as the database storage, so you can set the connection string in the configuration file.

services.AddDbContext<ApplicationDbContext>(options => options.UseSqlite(Configuration.GetValue<string>("ConnectionString"),

sqlOptions =>

{

sqlOptions.MigrationsAssembly(typeof(Startup).GetTypeInfo().Assembly.GetName().Name);

}));

services.AddIdentity<ApplicationUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

services.AddIdentityServer()

.AddDeveloperSigningCredential()

.AddAspNetIdentity<ApplicationUser>()

.AddConfigurationStore(options =>

{

options.ConfigureDbContext = builder => builder.UseSqlite(Configuration.GetValue<string>("ConnectionString"),

sqlOptions =>

{

sqlOptions.MigrationsAssembly(typeof(Startup).GetTypeInfo().Assembly.GetName().Name);

});

})

.AddOperationalStore(options =>

{

options.ConfigureDbContext = builder => builder.UseSqlite(Configuration.GetValue<string>("ConnectionString"),

sqlOptions =>

{

sqlOptions.MigrationsAssembly(typeof(Startup).GetTypeInfo().Assembly.GetName().Name);

});

});

1. To add the identity service middleware, add the following line in the Configure method of the Startup class.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseStaticFiles();

***app.UseIdentityServer();***

app.UseMvcWithDefaultRoute();

}

1. We almost done with the configuration. To store the details of Api resources, Identity resources and Clients information we need to seed the data to the database table. Create a private method “**InitializeDatabase**()” in the start up class with the following code.

private void InitializeDatabase(IApplicationBuilder app)

{

//callbacks urls from config:

var clientUrls = new Dictionary<string, string>();

clientUrls.Add("Spa", Configuration.GetValue<string>("SpaClient"));

clientUrls.Add("MarketingApi", Configuration.GetValue<string>("MarketingApiClient"));

clientUrls.Add("BasketApi", Configuration.GetValue<string>("BasketApiClient"));

clientUrls.Add("OrderingApi", Configuration.GetValue<string>("OrderingApiClient"));

using (var serviceScope = app.ApplicationServices.GetService<IServiceScopeFactory>().CreateScope())

{

serviceScope.ServiceProvider.GetRequiredService<PersistedGrantDbContext>().Database.Migrate();

var context = serviceScope.ServiceProvider.GetRequiredService<ConfigurationDbContext>();

context.Database.Migrate();

if (!context.Clients.Any())

{

foreach (var client in Config.GetClients(clientUrls))

{

context.Clients.Add(client.ToEntity());

}

context.SaveChanges();

}

if (!context.IdentityResources.Any())

{

foreach (var resource in Config.GetIdentityResources())

{

context.IdentityResources.Add(resource.ToEntity());

}

context.SaveChanges();

}

if (!context.ApiResources.Any())

{

foreach (var resource in Config.GetApiResources())

{

context.ApiResources.Add(resource.ToEntity());

}

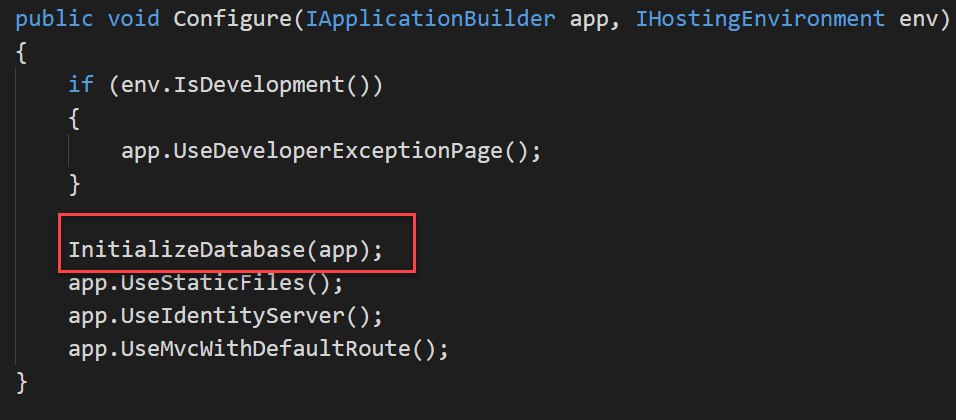
context.SaveChanges();

}

}

}

1. Update the Configure method of the Startup class to call the above function.



1. Now, you need to enable migrations for the Database. Because you are using Identity Server you need to enable migration for **ConfigurationDbContext**, **PersistedGrantDbContext** and **AppliationDbContext**. *ConfigurationDbContext* and *PersistedGrantDbContext* are built-in context classes which is provided by **IdentityServer4.EntityFramework.DbContexts** namespace. To enable migrations run the following codes.

***dotnet ef migrations add “Initial” -c ApplicationDbContext***

***dotnet ef migrations add “Initial” -c PersistedGrantDbContext***

***dotnet ef migrations add “initial” -c ConfigurationDbContext***

1. After enabling migrations run the following code to create the necessary tables on the database.

***dotnet ef database update -c ApplicationDbContext***

**Setting up the Web API client with Identity Server**

1. Open the client API project where you need to enable the Authentication using Identity Server.
2. Add the following method in the startup class to enable the Identity server authentication service.

private void ConfigureAuthService(IServiceCollection services)

{

JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();

services.AddAuthentication()

.AddJwtBearer(options =>

{

options.Authority = "http://localhost:5001";

options.RequireHttpsMetadata=false;

options.Audience = "basket";

options.TokenValidationParameters.NameClaimType = "client\_id";

});

services.AddAuthorization(Options =>

{

Options.DefaultPolicy = new AuthorizationPolicyBuilder(JwtBearerDefaults.AuthenticationScheme)

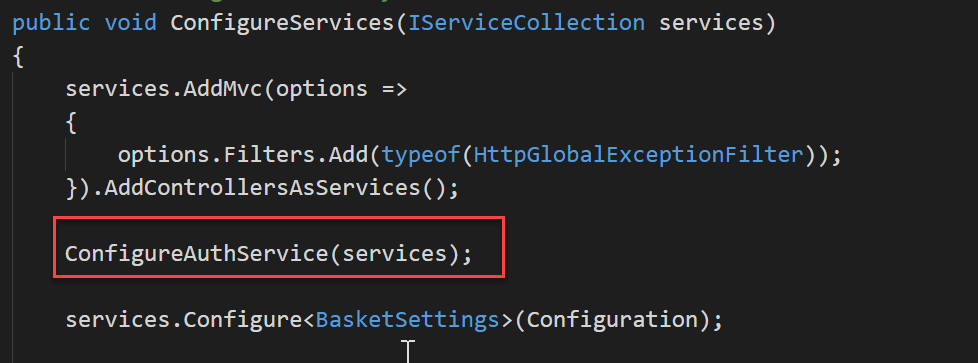
.RequireAuthenticatedUser()

.Build();

});

}

1. The audience name should match with the Api resource name which you have configured in the previous project.
2. Now, you can call the **ConfigureAuthService** method from the **ConfigureServices** method of the **Startup** class.



1. You also need to add the Authentication middleware to the configure method.



1. You have done with the configuration. Client applications who wants to access the API need to get authenticated with Identity server and use the access token to call the API.