Certainly! Below, I'll walk you through creating a simple ASP.NET Core Minimal API project with Swashbuckle for Swagger documentation and a basic connection to a SQL database. We'll set up an API that can read data from a database.

Step 1: Setting up the project

First, ensure you have the .NET SDK installed on your system. You can check by running `dotnet --version` in the terminal. If it's not installed, you can download it from the official .NET website.

Create a new folder for your project and navigate into it:

```
mkdir MyMinimalApi
cd MyMinimalApi
```

Create a new ASP.NET Core project:

```
dotnet new web -n MyMinimalApi
```

Step 2: Adding necessary packages

You'll need to add packages for Swashbuckle (Swagger) and Entity Framework Core:

```
cd MyMinimalApi
dotnet add package Swashbuckle.AspNetCore
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

Step 3: Configure Swagger

Open the `Program.cs` file and configure Swashbuckle to add Swagger generation:

```
using Microsoft.OpenApi.Models;
using Swashbuckle.AspNetCore.SwaggerUI;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(options =>
{
    options.SwaggerDoc("v1", new OpenApiInfo { Title = "My Minimal API", Version =
"v1" });
});

var app = builder.Build();
```

```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "My
Minimal API v1"));
}

app.MapGet("/", () => "Hello World!");

app.Run();
```

Step 4: Configuring the Database Connection

You'll need to define a connection string and configure Entity Framework:

1. **Add Connection String**: In `appsettings.json`, add your database connection string:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=
(localdb)\\mssqllocaldb;Database=MyDatabase;Trusted_Connection=True;"
  }
}
```

2. **Configure DbContext**: Create a new folder called `Data` and inside that folder, create a file named `MyDbContext.cs`:

```
using Microsoft.EntityFrameworkCore;

public class MyDbContext : DbContext
{
    public MyDbContext(DbContextOptions<MyDbContext> options)
        : base(options)
    {
    }

    public DbSet<Item> Items { get; set; }
}

public class Item
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

3. **Register DbContext**: In `Program.cs`, add:

```
builder.Services.AddDbContext<MyDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"
)));
```

Step 5: Creating a Simple API to Access Data

Define a simple API endpoint to get items from the database:

```
app.MapGet("/items", async (MyDbContext dbContext) =>
{
    return await dbContext.Items.ToListAsync();
});
```

Step 6: Run and Test Your Application

Run your application with:

```
dotnet run
```

Navigate to `http://localhost:5000/swagger` to see your Swagger UI, where you can test the `/items` endpoint.

Final Note

This setup provides a basic structure. For a real-world application, you should handle migrations, add error handling, authentication, logging, and more robust configuration and security practices.