Dotnet with VSCode Guide

A Comprehensive Guide to Setting Up and Using .NET with Visual Studio Code

LaTeX Document by abd, more @ github.com/abdxdev/notes-maker
Report errors: abdulrahman.abd.dev@gmail.com | +92 312 4996133
Compiled on: Wednesday 17th December, 2025 at 10:34pm

Contents

1 Setup SQL Server LocalDB with VS Code for .NET Development

1.1 Install Sql Server

- Download Microsoft® SQL Server® 2022 Express
- Install with **Download Media** option
- Select **SqlLocalDB**, install it
- Open folder and install the downloaded installer

1.2 Access LocalDB using VS Code (easy way)

1.2.1 Connect to LocalDB

1. Install **SQL Server (mssql)** extension from VS Code marketplace
2. From the left sidebar open **Sql Server** tab
3. Wait for the tools to downlaod. You can see the progress from output window on the bottom
4. Click + icon to add new connection
5. Fill the connection details as below:

    - Server name: (localdb)
    - Authentication Type: Windows Authentication
    - Database name: (leave blank to show all databases)

> **NOTE**
>
> This is the connection string format you can use in your application:
>
> ```
> Server=(localdb)\MSSQLLocalDB; Database=your_database_name;
> ↪  Trusted_Connection=True; TrustServerCertificate=True;
> ```
>
> `your_database_name` *should be replaced with your actual database name.*

1.2.2 Create and Manage Databases

1. After connecting, you can see the databases in the sidebar
2. Right click on **MSSQLLocalDB** and select **New Query**
3. To create a new database, run the following SQL command:

```sql
CREATE DATABASE your_database_name;
```

`your_database_name` *should be replaced with your actual database name.*

1.2.3 Create and manage tables

1. Expand the **MSSQLLocalDB** node in the sidebar
2. Expand **Databases** and right on database select **New Query**
3. Write your query and run it.

1.3 Access LocalDB using sqlcmd (hard way)

1.3.1 Setup sqlcmd

1. Install sqlcmd using `winget install sqlcmd`

2. Paste the following command in terminal to test the connection:

```powershell
sqllocaldb start MSSQLLocalDB ; sqlcmd -S "$(sqllocaldb info
↪  MSSQLLocalDB | Select-String 'Instance pipe name' |
↪  ForEach-Object { $_ -replace '.*:\s*', '' })" -Q "SELECT
↪  GETDATE()"
```

3. If the connection is successful, you will see the current date and time from the SQL Server.

1.3.2 Run sqlcmd

1. To connect to LocalDB, run the following command in terminal:

```powershell
sqllocaldb start MSSQLLocalDB
sqllocaldb info MSSQLLocalDB
```

2. You will see the instance pipe name, copy it.

1.3.3 Create Database

```powershell
sqlcmd -S "your_instance_pipe_name" -Q "CREATE DATABASE
↪  your_database_name;"
```

`your_instance_pipe_name` *should be replaced with the actual instance pipe name you copied earlier.* `your_database_name` *should be replaced with your actual database name.*

1.3.4 Create Tables and Insert Data

1. Now paste the query in a new file and save it as `script.sql`
2. To run the script, use the following command:

```powershell
sqlcmd -S "your_instance_pipe_name" -d your_database_name -i
↪  "script.sql"
```

`your_instance_pipe_name` *should be replaced with the actual instance pipe name you copied earlier.* `your_database_name` *should be replaced with your actual database name.*

2 Create .net Project

> **NOTE**
>
> Check if the `dotnet` CLI is installed by running `dotnet --version` in your terminal. If not installed, download and install the .NET SDK from here. For this guide, we will use .NET 8.0.

2.1 Create a new MVC project

```powershell
dotnet new mvc -n your_project_name -f net8.0
```

`your_project_name` *should be replaced with your actual project name.*

2.2 Create a new API project

```powershell
dotnet new webapi -n your_project_name -f net8.0
```

`your_project_name` *should be replaced with your actual project name.*

2.3 Connect to LocalDB from .NET Application

1. Open the project folder in VS Code
2. Install the required NuGet packages:

```powershell
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version
↪  8.*
dotnet add package Microsoft.EntityFrameworkCore.Tools --version 8.*
```

3. Update `appsettings.json` to add the connection string:

```json
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;
    ↪  Database=your_database_name; Trusted_Connection=True;
    ↪  TrustServerCertificate=True;"
  }
}
```

Replace `your_database_name` *with the actual database name you created earlier.*

2.4 Create Models

1. Install the EF Core Design package:

```powershell
dotnet add package Microsoft.EntityFrameworkCore.Design --version 8.*
```

2. Scaffold the database to create models and DbContext:

```powershell
dotnet ef dbcontext scaffold "Name=DefaultConnection"
↪  Microsoft.EntityFrameworkCore.SqlServer -o Models -c AppDbContext
```

> **NOTE**
> If this doesnt work, try using the full connection string instead of ”Name=DefaultConnection”.

3. You should now see the generated models in the `Models` folder.

2.5 Setup AppDbContext

1. Install the EF Core Tools globally (if not already installed):

```powershell
dotnet tool install --global dotnet-ef --version 8.*
```

1. Open `Program.cs` and add the following code to configure the DbContext:

```csharp
using Microsoft.EntityFrameworkCore;
using your_project_name.Models; // Update with your actual namespace

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

// Configure DbContext
builder.Services.AddDbContext<AppDbContext>(options =>

    ↪ options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConn

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

`your_project_name` *should be replaced with your actual project namespace.*

3 Admin Panel Generation

Install the required package for scaffolding:

```powershell
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design
↪  --version 8.*
```

Install the code generator tool:

```powershell
dotnet tool install -g dotnet-aspnet-codegenerator --version 8.*
```

3.1 Scaffold Controllers and Views (for MVC projects)

1. Paste the following code to the `generate.py` file:

```python
import os

for item in [
    item.replace(".cs", "")
    for item in os.listdir("Models")
    if item.endswith(".cs")
    and item
    not in [
        "AppDbContext.cs",
        "ErrorViewModel.cs",
    ]
]:
    print(f"Generating controller for model: {item}")
    os.system(f"dotnet aspnet-codegenerator controller
    ↪  --controllerName {item}Controller --model
    ↪  your_project_name.Models.{item} --dataContext
    ↪  your_project_name.Models.AppDbContext --useAsyncActions
    ↪  --useDefaultLayout --force --relativeFolderPath Controllers")
```

`your_project_name` *should be replaced with your actual project namespace.*

2. Run python code:

```powershell
python generate.py
```

> **NOTE**
> Make sure python is installed and available in your PATH. If not, download and install Python from python.org.

3. You should now see the generated controllers in the `Controllers` folder and views in the `Views` folder.

3.2 Scaffold Controllers only (for API projects)

1. Paste the following code to the `generate.py` file:

```python
import os

for item in [
    item.replace(".cs", "")
    for item in os.listdir("Models")
    if item.endswith(".cs")
    and item
    not in [
        "AppDbContext.cs",
        "ErrorViewModel.cs",
    ]
]:
    print(f"Generating controller for model: {item}")
 os.system(f"dotnet aspnet-codegenerator controller --controllerName
  ↪  {item}Controller --model your_project_name.Models.{item}
  ↪  --dataContext your_project_name.Models.AppDbContext
  ↪  --restWithNoViews --force --relativeFolderPath Controllers")
```

`your_project_name` *should be replaced with your actual project namespace.*

2. Run python code:

```powershell
python generate.py
```

> **NOTE**
> Make sure python is installed and available in your PATH. If not, download and install Python from python.org.

3. You should now see the generated controllers in the `Controllers` folder.

3.3 Add Swagger for API Documentation (for API projects)

1. Install the Swashbuckle package:

```powershell
dotnet add package Swashbuckle.AspNetCore --version 8.*
```

2. Update `Program.cs` to add Swagger services and middleware:

```csharp
using Microsoft.EntityFrameworkCore;
using your_project_name.Models; // Update with your actual namespace

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

// Configure DbContext
builder.Services.AddDbContext<AppDbContext>(options =>
```

```csharp
        ↪ options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConn
    // SWAGGER
    builder.Services.AddControllersWithViews();
    builder.Services.AddEndpointsApiExplorer();

    var app = builder.Build();

    // SWAGGER
    app.UseSwagger();
    app.UseSwaggerUI();


    // Configure the HTTP request pipeline.
    if (!app.Environment.IsDevelopment())
    {
        app.UseExceptionHandler("/Home/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");

    app.Run();
```

`your_project_name` *should be replaced with your actual project namespace.*

3. You can now access the Swagger UI at `https://localhost:5001/swagger` to explore your API endpoints.

4 Build and run the Application

4.1 Run the application

1. Run the application using the following command:

```powershell
dotnet run
```

2. Open your browser and navigate to `https://localhost:5001` to see your application in action.

4.2 Run the application with hot reload

1. Run the application with hot reload using the following command:

```powershell
dotnet watch
```

2. Open your browser and navigate to `https://localhost:5001` to see your application in action. Any code changes will be reflected immediately without restarting the application.