Complete guide to building an app with net core and react

.dotnet core sdk 3.x

dotnet –info

dotnet –version

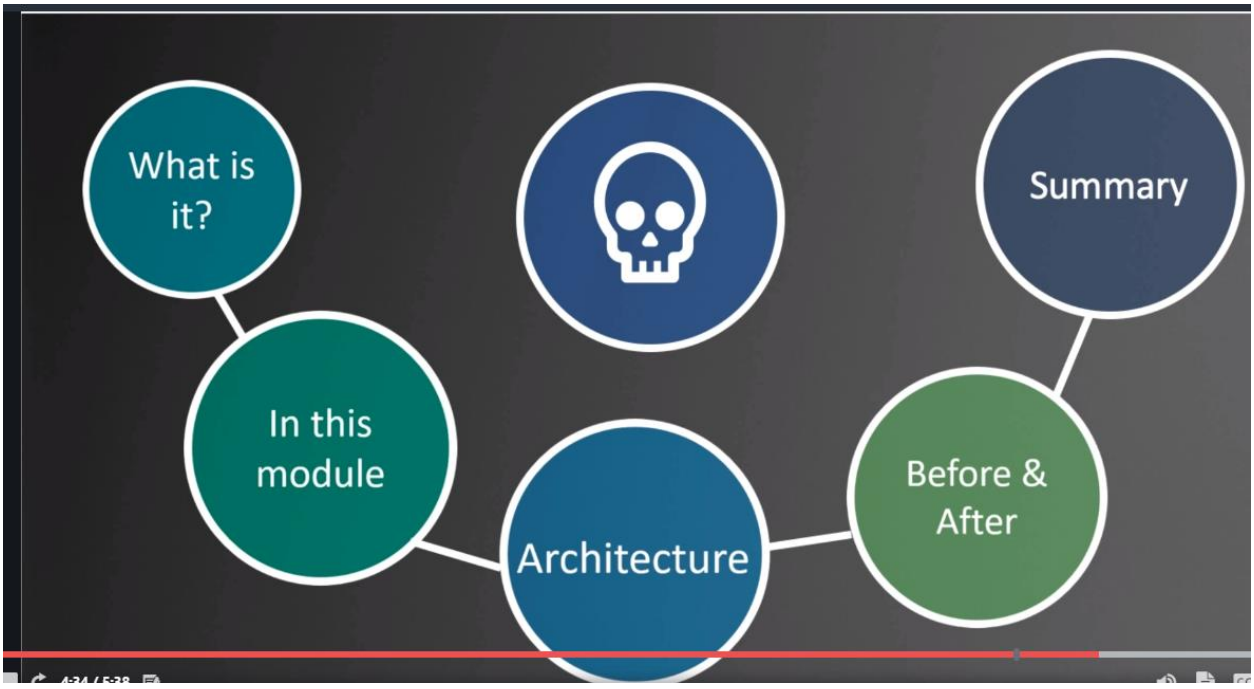
- node.js (https://nodejs.org/en/)
- Installing node.js with NVM (https://medium.com/@Joachim8675309/installing-node-js-with-nvm-4dc469c977d9)

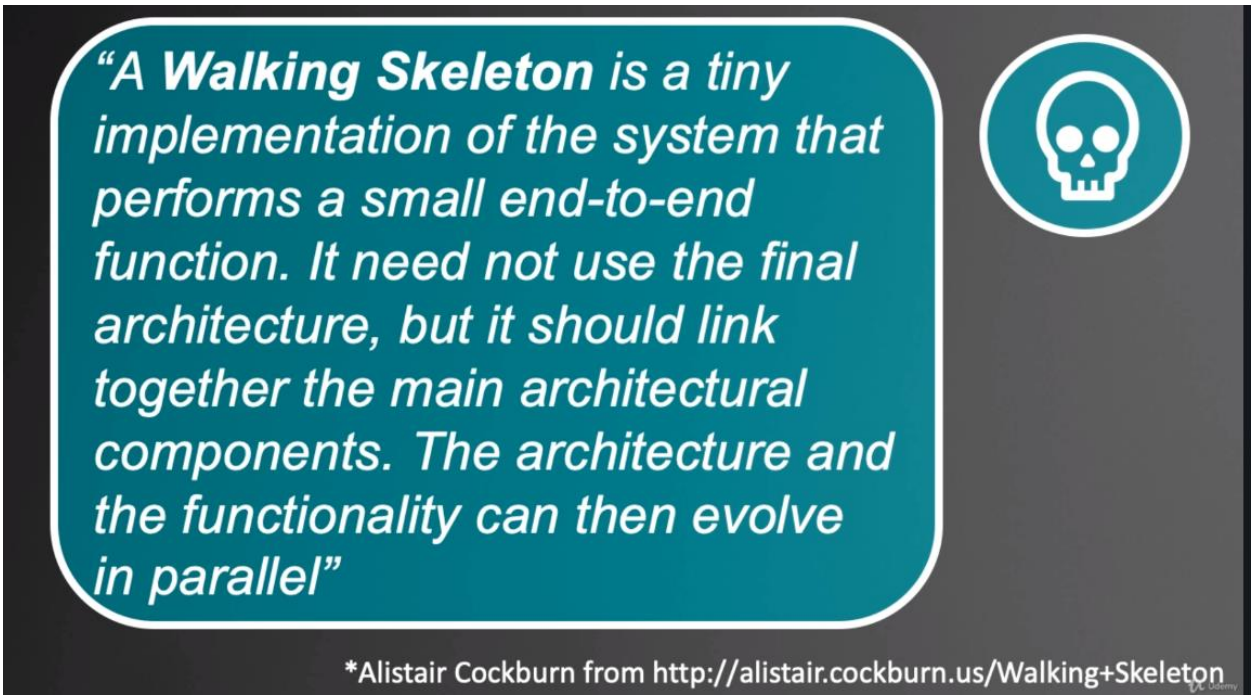
- Install git

- Vscode

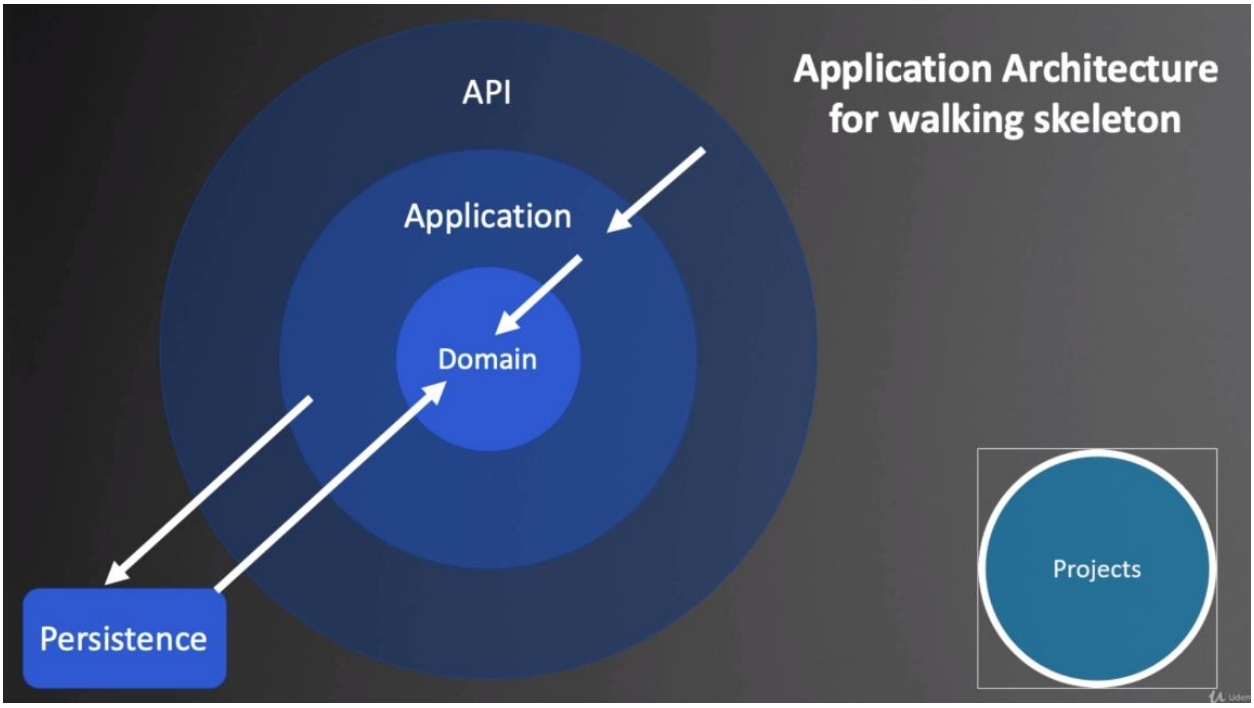
- Vscode Extensions

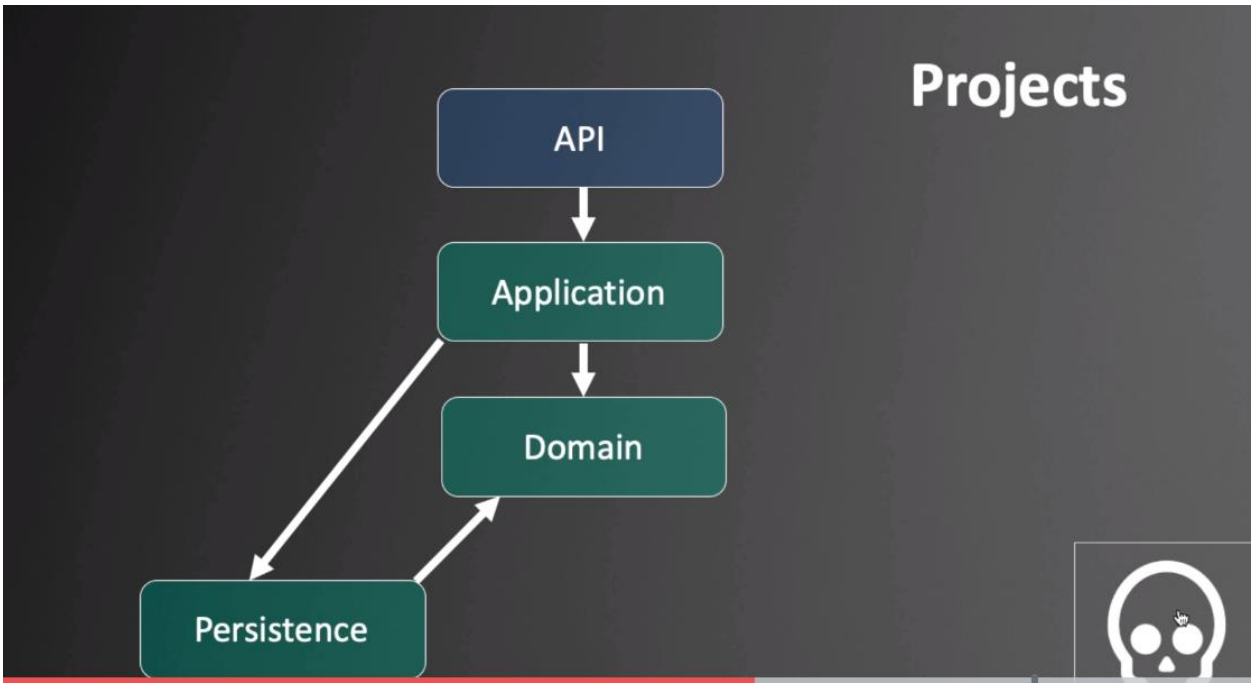
    - Auto close tag
    - Auto Rename Tag
    - Bracket pair colorizer2
    - C# power by omnisharp
    - C# extension (jchannon)
    - Es7 React/Redux/Graph …
    - Material icon theme
    - Nugget package manager
    - Prettier code formatter
    - SqLite




"A **Walking Skeleton** is a tiny implementation of the system that performs a small end-to-end function. It need not use the final architecture, but it should link together the main architectural components. The architecture and the functionality can then evolve in parallel"
*Alistair Cockburn from http://alistair.cockburn.us/Walking+Skeleton


In this module
- Intro to Clean Architecture
- Using the dotnet CLI
- Reviewing the project templates
- Running the app
- EF Migrations
- Seeding data
- Postman
- Using git for source control





dotnet –info

mkdir Reactivities

dotnet –h (list the command)

dotnet new –h

dotnet new sln (will use the name of containing server for solution name)

dotnet new classlib –n Domain

dotnet new classlib –n Persistence

dotnet new classlib - n Application

dotnet new webapi –n API

Adding dependencies:

dotnet sln –h

dotnet sln add Domain/

dotnet sln add Persistence

dotnet sln add application

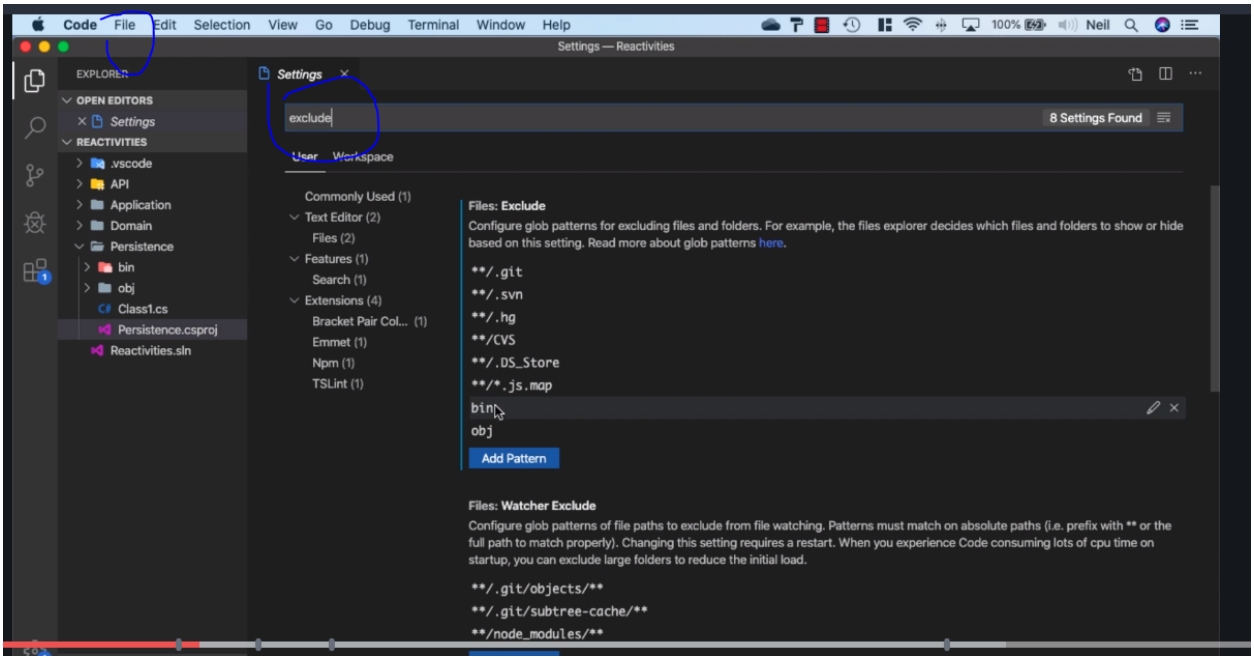
dotnet sln add API

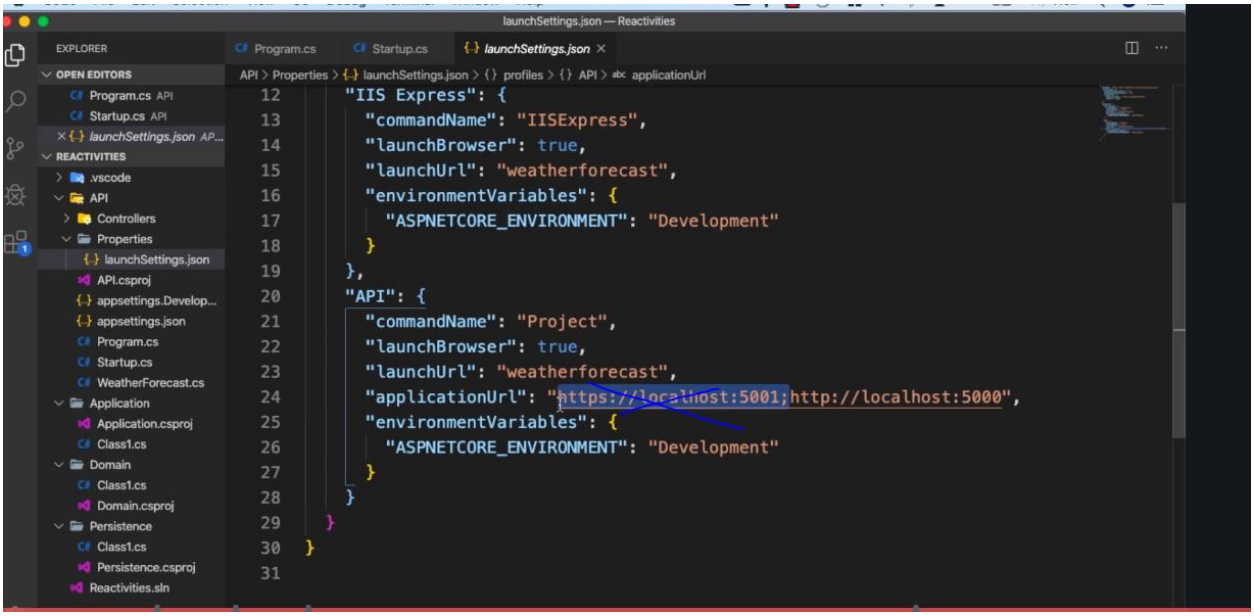
Cd to the Application project use dotnet add reference …

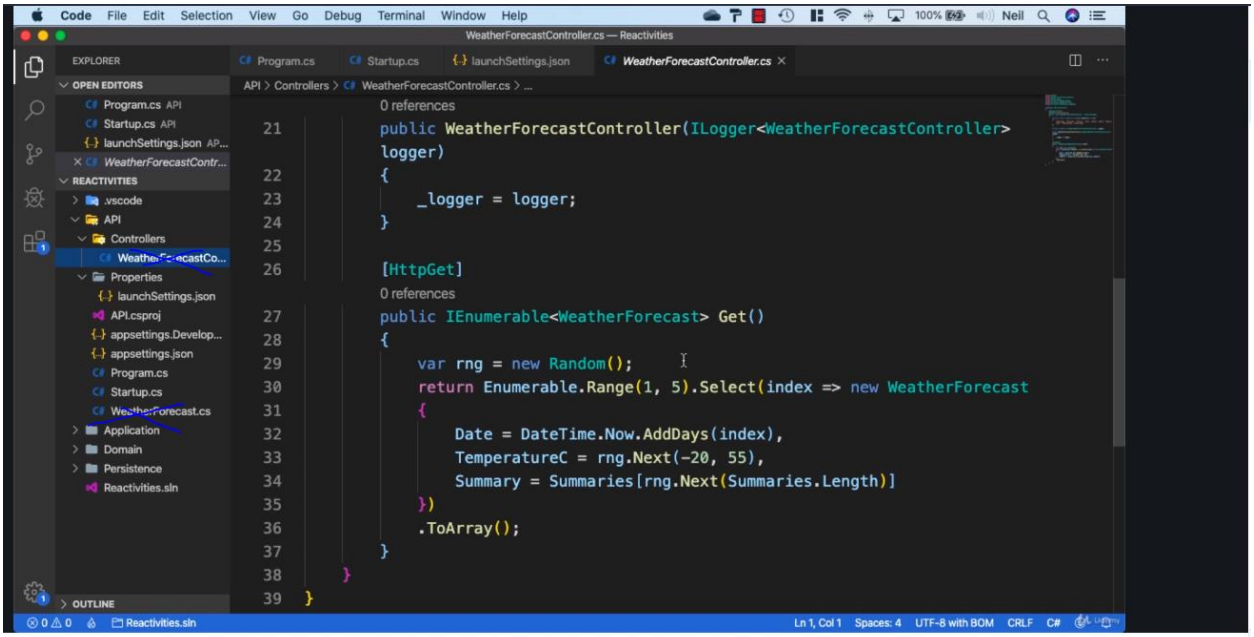
```
MbPro:Reactivities neil$ cd Application/
MbPro:Application neil$ dotnet add reference ../Domain/
Reference `..\Domain\Domain.csproj` added to the project.
MbPro:Application neil$ dotnet add reference ../Persistence/
Reference `..\Persistence\Persistence.csproj` added to the project.
MbPro:Application neil$ cd ..
MbPro:Reactivities neil$ cd API/
MbPro:API neil$ dotnet add reference ../Application/
Reference `..\Application\Application.csproj` added to the project.
MbPro:API neil$ cd ..
MbPro:Reactivities neil$ cd Persistence/
MbPro:Persistence neil$ dotnet add reference ../Domain/
Reference `..\Domain\Domain.csproj` added to the project.
```

Exclude **/obj , **/bin etc

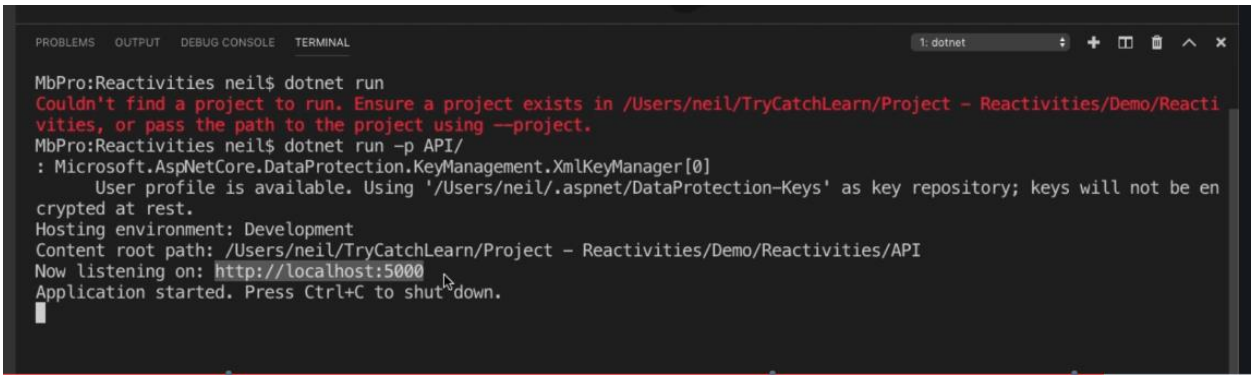
File->preferences search for the pattern


Code File Edit Selection View Go Debug Terminal Window Help
Settings — Reactivities
EXPLORER
Settings
OPEN EDITORS
Settings
REACTIVITIES
.vscode
API
Application
Domain
Persistence
bin
obj
Class1.cs
Persistence.csproj
Reactivities.sln
exclude
8 Settings Found
User
Workspace
Commonly Used (1)
Text Editor (2)
Files (2)
Features (1)
Search (1)
Extensions (4)
Bracket Pair Col... (1)
Emmet (1)
Npm (1)
TSLint (1)
Files: Exclude
Configure glob patterns for excluding files and folders. For example, the files explorer decides which files and folders to show or hide based on this setting. Read more about glob patterns here.
**/.git
**/.svn
**/.hg
**/CVS
**/.DS_Store
**/*.js.map
bin
obj
Add Pattern
Files: Watcher Exclude
Configure glob patterns of file paths to exclude from file watching. Patterns must match on absolute paths (i.e. prefix with ** or the full path to match properly). Changing this setting requires a restart. When you experience Code consuming lots of cpu time on startup, you can exclude large folders to reduce the initial load.
/.git/objects/
/.git/subtree-cache/
/node_modules/

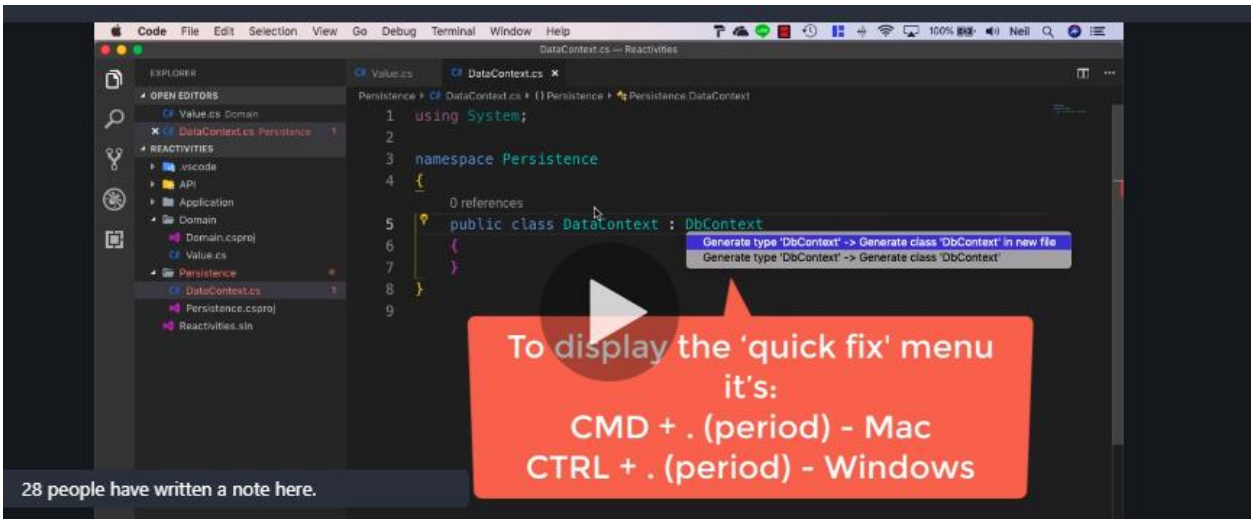

launchSettings.json — Reactivities
EXPLORER
Program.cs
Startup.cs
launchSettings.json
API > Properties > launchSettings.json > profiles > API > applicationUrl
OPEN EDITORS
Program.cs API
Startup.cs API
launchSettings.json AP...
REACTIVITIES
.vscode
API
Controllers
Properties
launchSettings.json
API.csproj
appsettings.Develop...
appsettings.json
Program.cs
Startup.cs
WeatherForecast.cs
Application
Application.csproj
Class1.cs
Domain
Class1.cs
Domain.csproj
Persistence
Class1.cs
Persistence.csproj
Reactivities.sln
"IIS Express": {
"commandName": "IISExpress",
"launchBrowser": true,
"launchUrl": "weatherforecast",
"environmentVariables": {
"ASPNETCORE_ENVIRONMENT": "Development"
},
"API": {
"commandName": "Project",
"launchBrowser": true,
"launchUrl": "weatherforecast",
"applicationUrl": "https://localhost:5001;http://localhost:5000",
"environmentVariables": {
"ASPNETCORE_ENVIRONMENT": "Development"



Run dotnet run at the startup level



DBContext



Add the

Microsoft.EntityFrameWorkCore

Microsoft.EntityFrameWorkCore.Sqlite

Microsoft.EntityFrameWorkCore.Design

Add entityframework tools

dotnet tool install --global dotnet-ef

dotnet ef

Got to the Root of the project

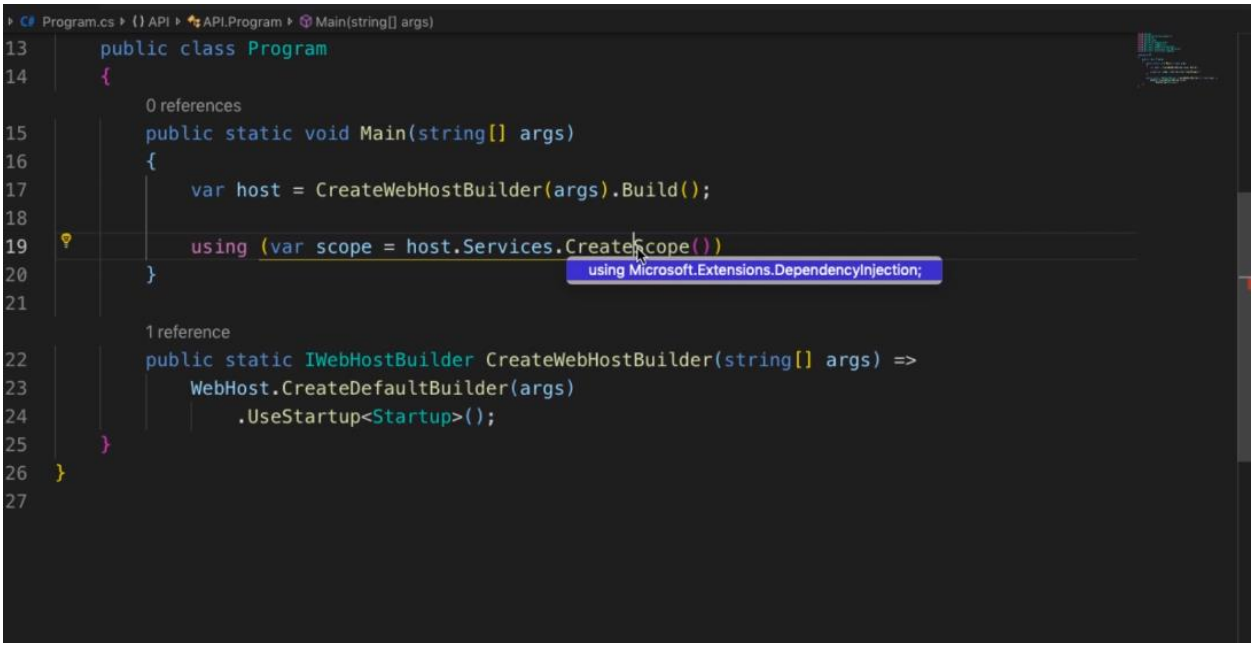
dotnet ef migrations add InitialCreate --project Persistence/ --stratup-project API/ (-p project – s startup project)

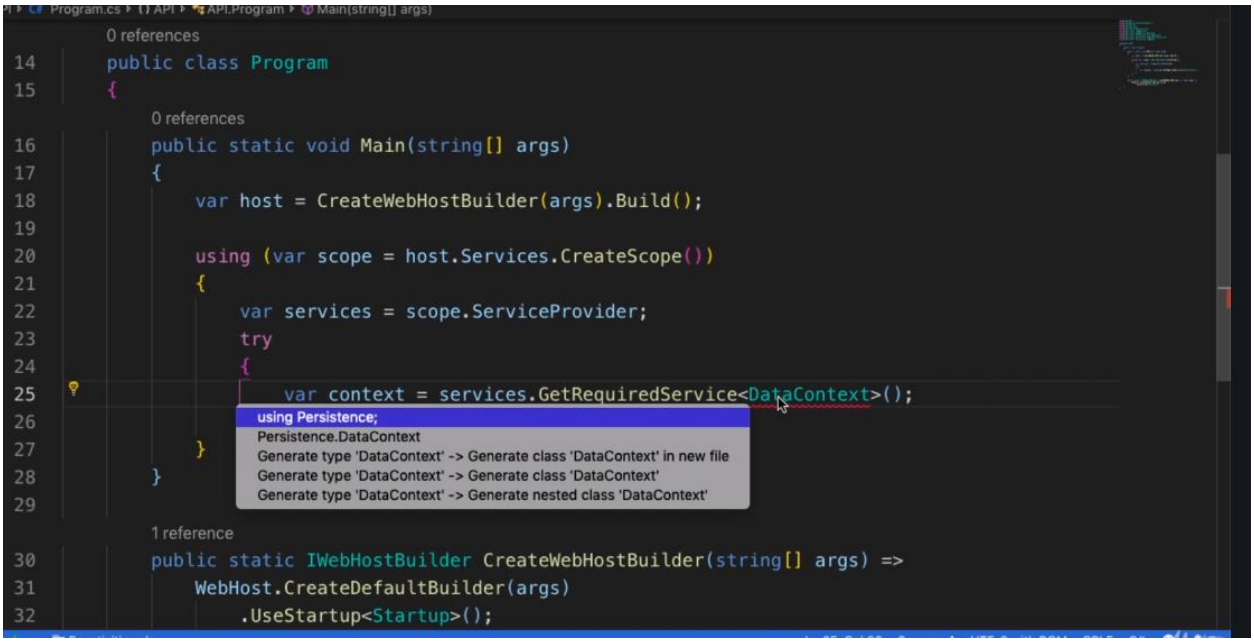
Install any reference and install them

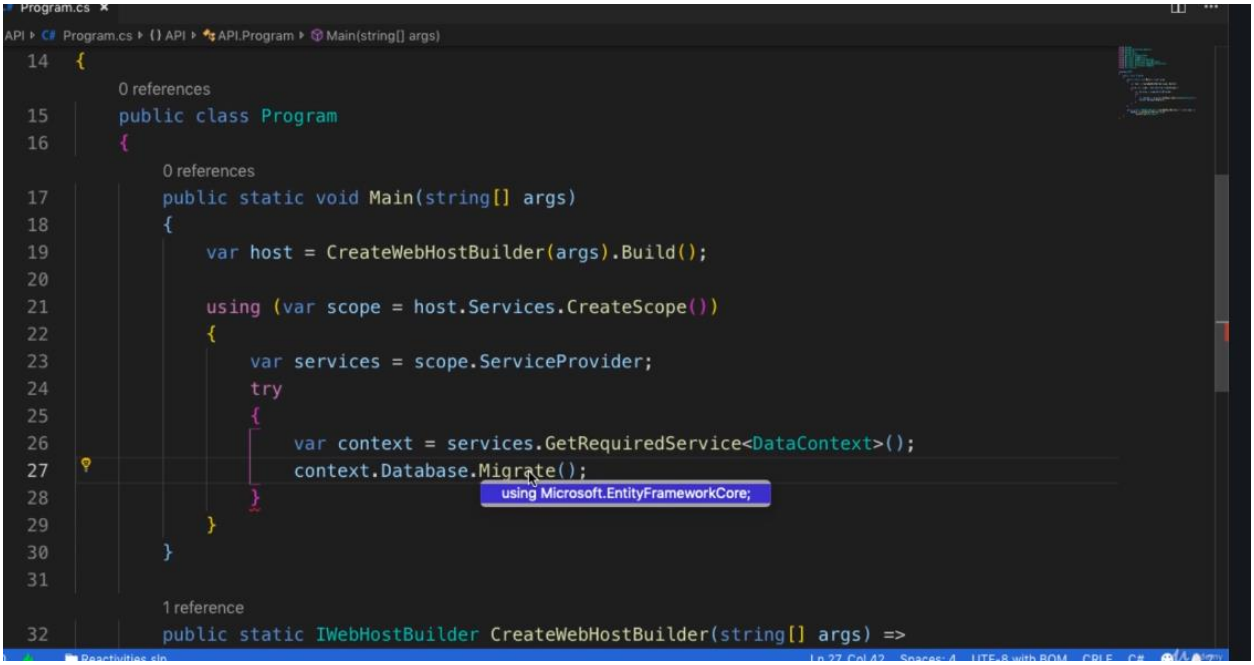
Update Database manually or every time the program runs

Manually: dotnet ef database –update

Startup => program.cs


Program.cs ▸ {} API ▸ API.Program ▸ Main(string[] args)
public class Program
{
0 references
public static void Main(string[] args)
{
var host = CreateWebHostBuilder(args).Build();
using (var scope = host.Services.CreateScope())
using Microsoft.Extensions.DependencyInjection;
}
1 reference
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
WebHost.CreateDefaultBuilder(args)
.UseStartup<Startup>();
}
}

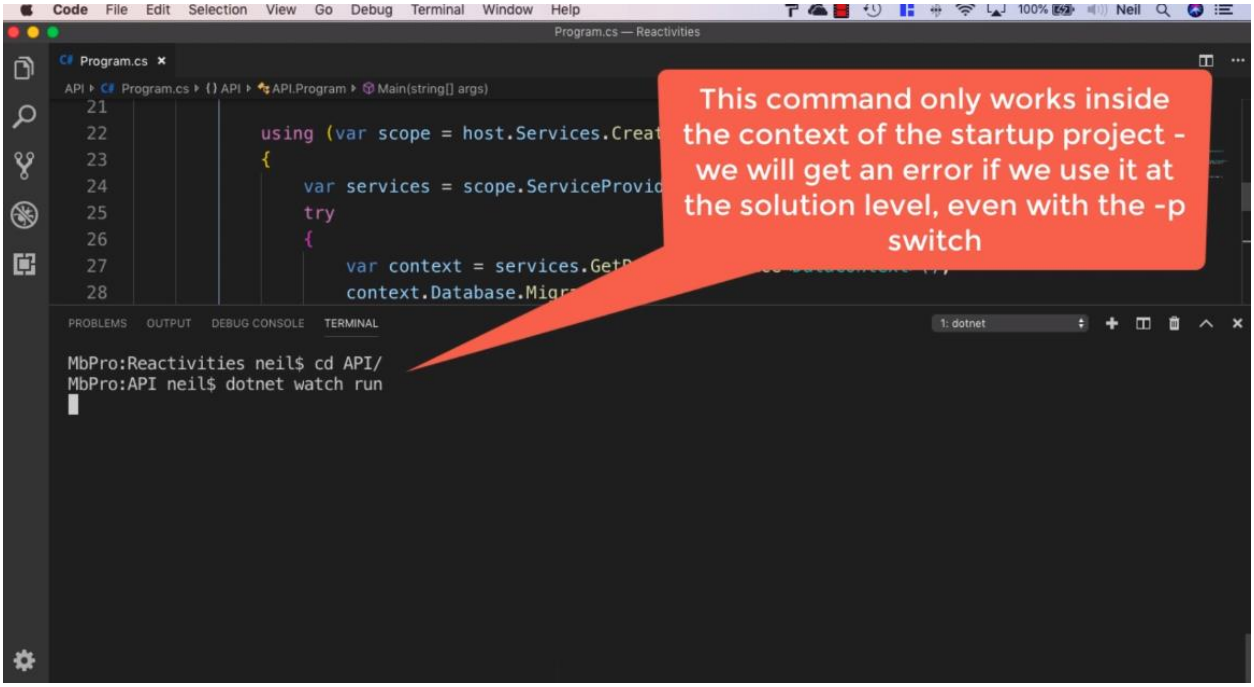

Program.cs ▸ {} API ▸ API.Program ▸ Main(string[] args)
0 references
public class Program
{
0 references
public static void Main(string[] args)
{
var host = CreateWebHostBuilder(args).Build();
using (var scope = host.Services.CreateScope())
{
var services = scope.ServiceProvider;
try
{
var context = services.GetRequiredService<DataContext>();
using Persistence;
Persistence.DataContext
Generate type 'DataContext' -> Generate class 'DataContext' in new file
Generate type 'DataContext' -> Generate class 'DataContext'
Generate type 'DataContext' -> Generate nested class 'DataContext'
}
}
1 reference
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
WebHost.CreateDefaultBuilder(args)
.UseStartup<Startup>();




```
public static void Main(string[] args)
{
    var host = CreateWebHostBuilder(args).Build();

    using (var scope = host.Services.CreateScope())
    {
        var services = scope.ServiceProvider;
        try
        {
            var context = services.GetRequiredService<DataContext>();
            context.Database.Migrate();
        }
        catch (Exception ex)
        {
            var logger = services.GetRequiredService<ILogger<Program>>();
            logger.LogError(ex, "An error occured during migration");
        }
    }

    host.Run();
}
```



Adding SeedValue

```
namespace Persistence
{
    4 references
    public class DataContext : DbContext
    {
        0 references
        public DataContext(DbContextOptions options) : base(options)
        {

        }
        0 references
        public DbSet<Value> Values {get; set;}

        0 references
        protected override void OnModelCreating(ModelBuilder builder){
            builder.Entity<Value>()
                .HasData(
                    new Value {Id = 1, Name = "Value 101"},
                    new Value {Id = 2, Name = "Value 102"},
                    new Value {Id = 3, Name = "Value 103"},
                    new Value {Id = 4, Name = "Value 104"}
                );
        }
    }
}
```

dotnet ef migrations add SeedValues --project Persistence/ --startup-project API/

dotnet ef migrations remove --project Persistence/ --startup-project API/ --force