Prerequisites

1. Create a WebAPI project
2. Install the CLI tool - **dotnet-ef**
   https://learn.microsoft.com/pl-pl/ef/core/cli/dotnet
3. Install Nuget packages:
   Microsoft.EntityFrameworkCore.Design
   Microsoft.EntityFrameworkCore.SqlServer

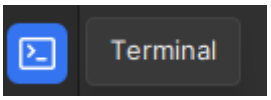
Scaffolding (Reverse Engineering)

Before performing scaffolding, create a database and execute the **cw7_create.sql** script, which will create the appropriate tables and will seed the data.

Add ConnectionString to the **appsettings.json** file. Example contents of this file:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "Default": "Data Source=(localdb)\\MSSQLLocalDB;Initial
Catalog=scaffold;Integrated Security=True;"
  }
}
```

After saving the application configuration, start the terminal in Rider. Below is a sample terminal icon:

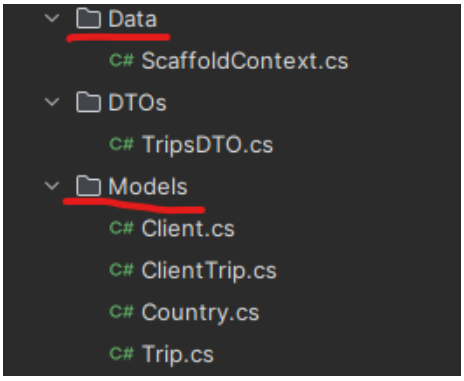


Using the command line interface, change the directory to the folder with the project. By default, the terminal will start in the solution directory, so go to the project by executing such a command:

```
cd NameOfTheProject
```

Being in the directory, we can perform database scaffolding with the **dotnet ef dbcontext scaffold** command. Two parameters ConnectionString and the database driver are required. In addition, you can add the **`--context-dir`** parameter to define where the database context will be placed. It is also worth using the optional **`--output-dir`** parameter, which will allow you to place the database models in a single specified folder. Below is the entire command to be executed:

```
dotnet ef dbcontext scaffold "Name=ConnectionStrings:Default" --context-dir
Data --output-dir Models Microsoft.EntityFrameworkCore.SqlServer
```



The database context (above ScaffoldContext.cs) contains the configuration of the connection to the database. The following line of code should be removed, as the configuration will be done during dependency injection.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    => optionsBuilder.UseSqlServer("Name=ConnectionStrings:Default");
```

In the Program.cs class, we inject a dependency with a context:

```
builder.Services.AddDbContext<ScaffoldContext>(
    options => options.UseSqlServer("Name=ConnectionStrings:Default"));
```

Performing queries

EntityFramework has several options for retrieving data. We will use the EagerLoading technique for this.

At first, pass the context as an argument to the constructor:

```
[Route("api/[controller]")]
[ApiController]
public class TripsController : ControllerBase
{
    private readonly ScaffoldContext _context;

    public TripsController(ScaffoldContext context)
    {
        _context = context;
    }
}
```

To create a query, use the LINQ syntax that was in the previous exercises. Below is an example of downloading trips (data is incomplete):

```
var trips = await _context.Trips.Select(e => new TripDTO()
{
    Name = e.Name,
    DateFrom = e.DateFrom,
    MaxPeople = e.MaxPeople,
    Clients = e.ClientTrips.Select(e => new ClientDTO()
    {
        FirstName = e.IdClientNavigation.FirstName,
        LastName = e.IdClientNavigation.LastName
    })
})
    .OrderBy(e => e.DateFrom)
    .ToListAsync();
```

Skip and take methods can come in handy for pagination:

```
.Skip((page-1) * pageSize)
.Take(pageSize)
```

For insertions and deletions, please watch the lecture :)