SQL е езикът, който е специално създаден максимално да се доближава до английския.

HTML (HyperText Transfer Language) е първият език на интернет, който е специално създаден за четене на текст, както показва името му, а Hyper идва от хиперлинковете, т.е. от един текстов документ да може да се прехвърляме на друг. За целта, се използват протоколи http (HyperText Transfer Protocol) и https = http + Secured (за логвания с по-голяма сигурност).

Всичко в HTML се представя с тагове, като вътре в тага са невидимите атрибути за свойствата, а между отварящия и затварящ таг е това, което потребителят вижда.

Например: <a href=www.abv.bg> Кликни тук </a>

Потребителят вижда „Кликни тук“.

Поради спецификата на създаване, HTML се доближава изключително много до текстообработващите програми като MS Word, например, тагове <h1> … <h6> съответстват аналогично на heading 1 (най-голямото заглавие) до h6, като те имат и своето скрито значение за търсещите машини, а не са просто форматиране за големина, фон, цвят, т.е. имат и скрит смисъл.

HTML е с по-съкратен синтаксис. Удобно е да използваме Snippet за вкарване на основна скелетна конструкция като ! + Enter

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

<h1> = heading
<p> = paragraph
<img src="" width="" heigh=""> = image source
вътре в тага са атрибутите за свойствата
<a href=""> = anchor hyperlink reference
<table>
<th> = table heading
<tr> = table row
<td> = table data
<ul> = Unordered list
<ol> = Ordered list
<li> = list item

 = division block – групира в блок съдържанието

 = break line – празен ред

<nav> = navigation bar
<button>
<footer>

```
<iframe src="https://www.omg-bg.com" name="omg" width="100%"
height="1000"></iframe>
```

alt="" = alternative – алтернативен текст, ако картинката не се зареди или информира за линка.

Обикновения <title> Начало <title> в C# често се заменя с динамичния

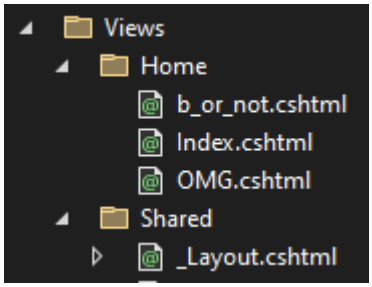
```
@{
    ViewData["Title"] = "Начало";
}
```

Където @{ } означават начало и край на C# код.

В site.css задаваме оформлението. Дори може сами да си измислим таг и да му зададем персонални свойства.

```
ler.cs    site.css    OMG.cshtml    Index.cshtml*

body {
    background-color: antiquewhite;
    margin-bottom: 60px;
}

h1 {
    font-family: 'Times New Roman', Times, serif;
    text-decoration-style: solid;
    color: maroon;
}

ul {
    list-style-type: armenian;
    font-family: 'Times New Roman', Times, serif;
    text-decoration-style: solid;
    color: maroon;
}
```

Какво разбрахме до момента от MVC?



Имаме 3 странички .cshtml в папка Views – това, което потребителят вижда. Там ги оформяме. Техният скелет от код – навигационната заглавна лента navbar се намира във Views/Shared/_Layout.cshtml . Долната черта показва, че макар да е със същото разширение .cshtm, това е рамка, скелет, а не самостоятелна страница. Намира се в папка Shared, за да бъде достъпна и взета от другите страници.

```
site.css   b_or_not.cshtml   OMG.cshtml   Index.cshtml*   _Layout.cshtml*
Shakespeare
13  <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
14      <div class="container-fluid ">
15          <a class="navbar-brand text-light " asp-area="" asp-controller="Home" asp-action="Index">Shakespeare</a>
16          <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls=
17                  aria-expanded="false" aria-label="Toggle navigation">
18              <span class="navbar-toggler-icon"></span>
19          </button>
20          <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
21              <ul class="navbar-nav flex-grow-1">
22                  <li class="nav-item">
23                      <a class="nav-link text-light" asp-area="" asp-controller="Home" asp-action="OMG">ОМГ "Акад. Н. Попов"</a>
24                  </li>
25                  <li class="nav-item">
26                      <a class="nav-link text-light " asp-area="" asp-controller="Home" asp-action="Index">Начало</a>
27                  </li>
28                  <li class="nav-item">
29                      <a class="nav-link text-light" asp-area="" asp-controller="Home" asp-action="b_or_not">2 b || ! 2 b</a>
30                  </li>
31              </ul>
32              <partial name="_LoginPartial" />
33          </div>
34      </div>
35  </nav>
```

В него виждаме нашите 3 страници, които биват извикани от единствения засега HomeController и действието в него action – съответно със същите имена Index (отваря се по подразбиране в началото), OMG и b_or_not.

Нека погледнем и какво се случва в самия HomeController – единственото, което той прави е да връща View – нашите 3 .cshtml страници.

```
ml   HomeController.cs   OMG.cs
Shakesp
0 references
public IActionResult OMG()
{
    return View();
}
0 references
public IActionResult Index()
{
    return View();
}

0 references
public IActionResult b_or_not()
{
    return View();
}
```

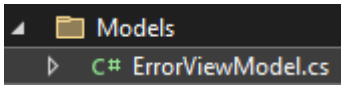
Видяхме модела: View <-> Controller.

Нека си добавим 2 нови страници в _Layout.cshtml:

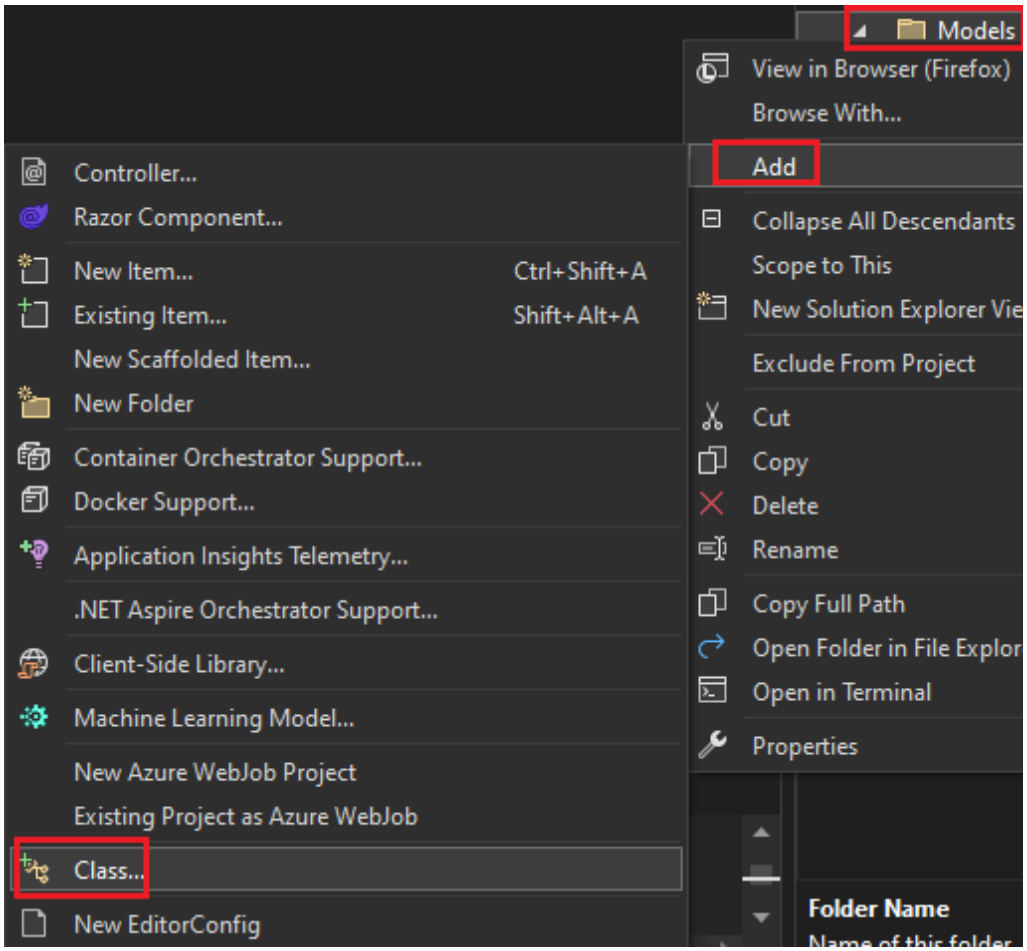
```
_Layout.cshtml*
Shakespeare
21  <ul class="navbar-nav flex-grow-1">
22      <li class="nav-item">
23          <a class="nav-link text-light" asp-area="" asp-controller="Home" asp-action="OMG">ОМГ "Акад. К. Попов"</a>
24      </li>
25      <li class="nav-item">
26          <a class="nav-link text-light " asp-area="" asp-controller="Home" asp-action="Index">Начало</a>
27      </li>
28      <li class="nav-item">
29          <a class="nav-link text-light" asp-area="" asp-controller="Home" asp-action="b_or_not">2 b || ! 2 b</a>
30      </li>
31      <li class="nav-item">
32          <a class="nav-link text-light" asp-area="" asp-controller="Quotes" asp-action="Index">Цитати</a>
33      </li>
34      <li class="nav-item">
35          <a class="nav-link text-light" asp-area="" asp-controller="Quotes" asp-action="Search">Търси</a>
36      </li>
37  </ul>
```

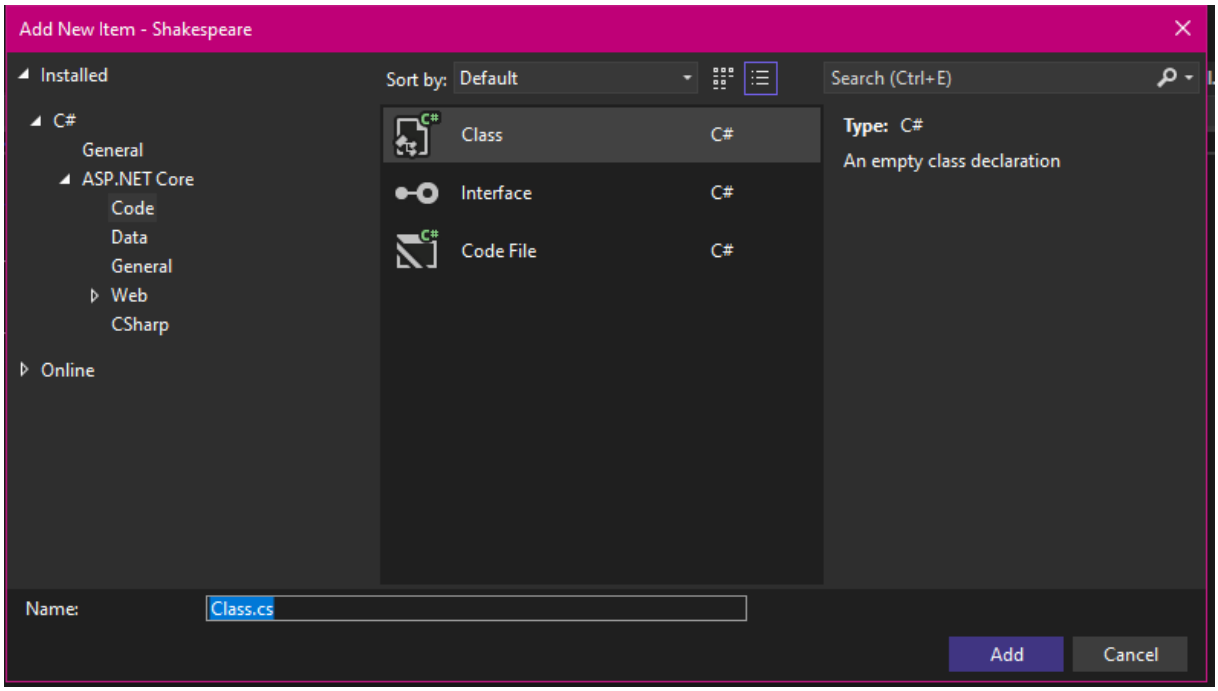
Вече може да ги видим, но ако кликнем върху тях, нищо няма да се зареди, защото не са готови. Трябва да си създадем QuotesController.

Имаме MVC = Model-View-Controller:



И до момента имаме само 1 модел за грешки. Нека създадем още един:





По подразбиране изписва Class.cs, но ние ще запишем Quote – в единствено число без “S” и дори без разширение, VS знае, че това е .cs файл.

```
Quote.cs
Shakespeare
1  namespace Shakespeare.Models
2  {
       0 references
3      public class Quote
4      {
5      }
6  }
```

Това е автоматично генерирания код за клас, съответно в него пишем “prop” + използваме snippet за още генериране на автокод с 2 последователни натискания на бутон Tab:

```
0 references
public class Quote
{
    prop
}
    prop        Tab 2x        property        property
    propa                     Code snippet for 'property'
    propdp
    propfull
    propg                     get-only property
    propi                     init-only property
    ServiceProviderOptions
```

Това много ускорява нещата и ето го резултата: дописа ни свойството:

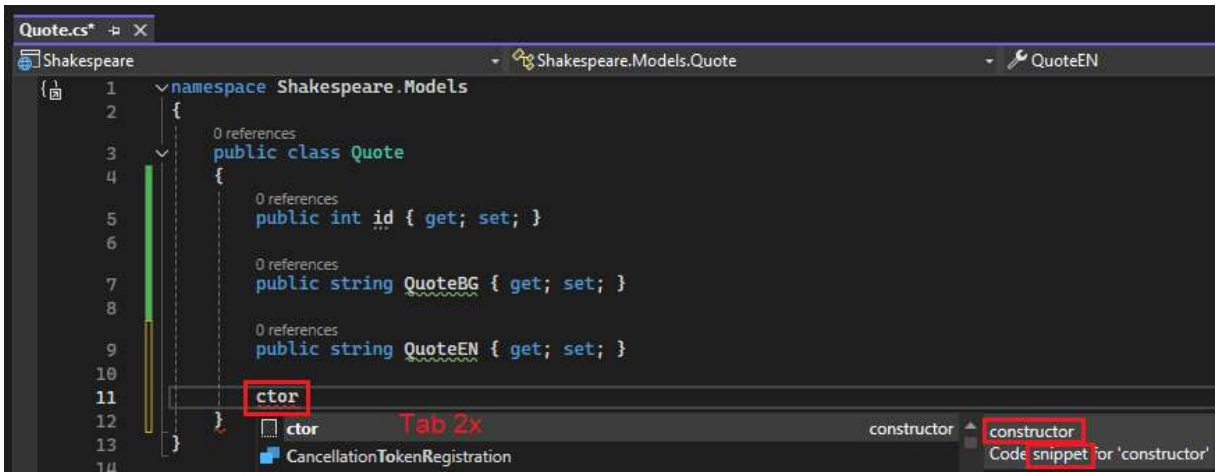
```
0 references
public class Quote
{
    0 references
    public int MyProperty { get; set; }
}
```

В случая, ще генерираме ORM модел на БД и затова ще въведем id:

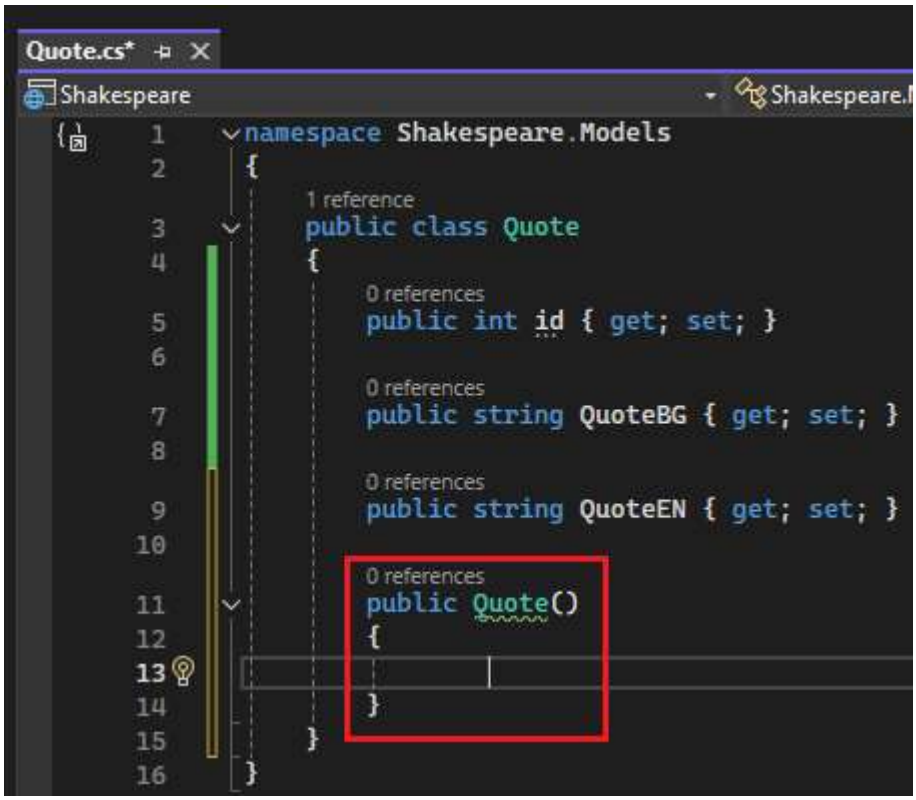
```
public class Quote
{
    0 references
    public int id { get; set; }   Tab

    public string name { get; set; }
}
```

IntelliSense се включва и може да бъде потвърден с единичен Tab.

Ще искаме да има цитат в превод на български и в оригинал.

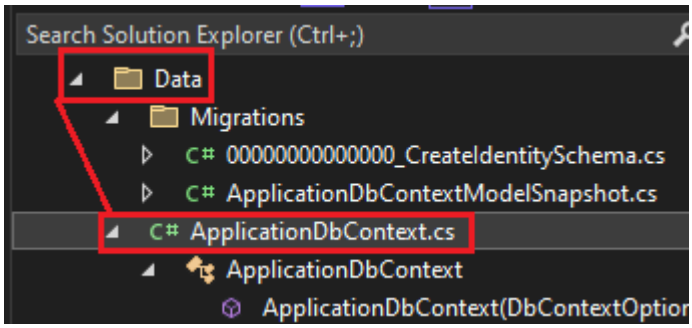


Създаваме празен конструктор със snippet „ctor“ +Tab 2x.

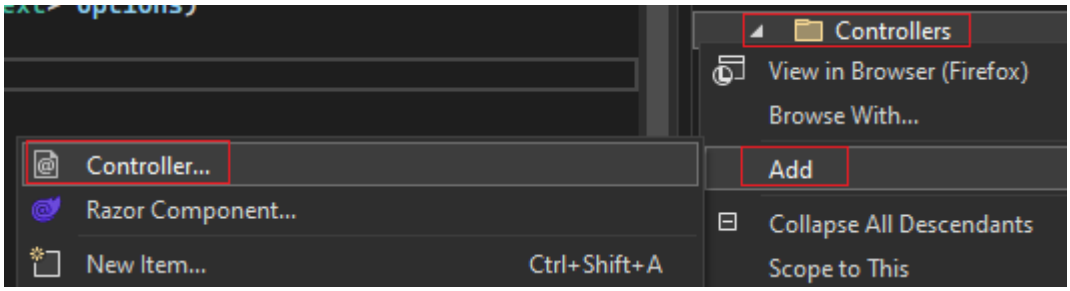


Оставяме го засега празен, защото ще бъде използван от други парчета от програмата. Трябва да го има.

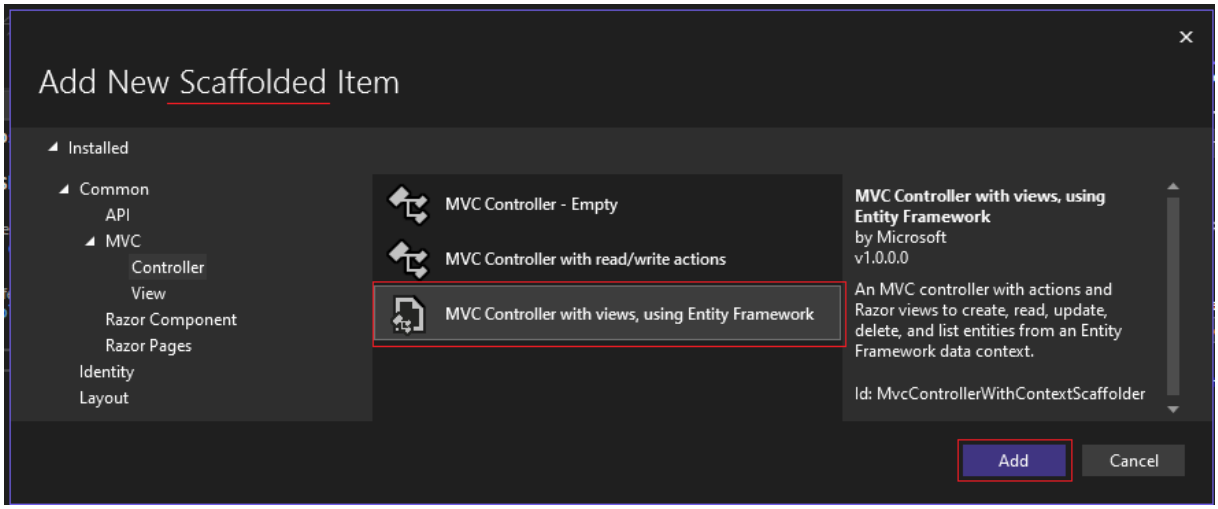
Хвърляме бърз поглед в папка Data -> автоматично генериран файл ApplicationDbContext.cs. Трябва да се уверим, че е там, защото ще го поиска в един момент:



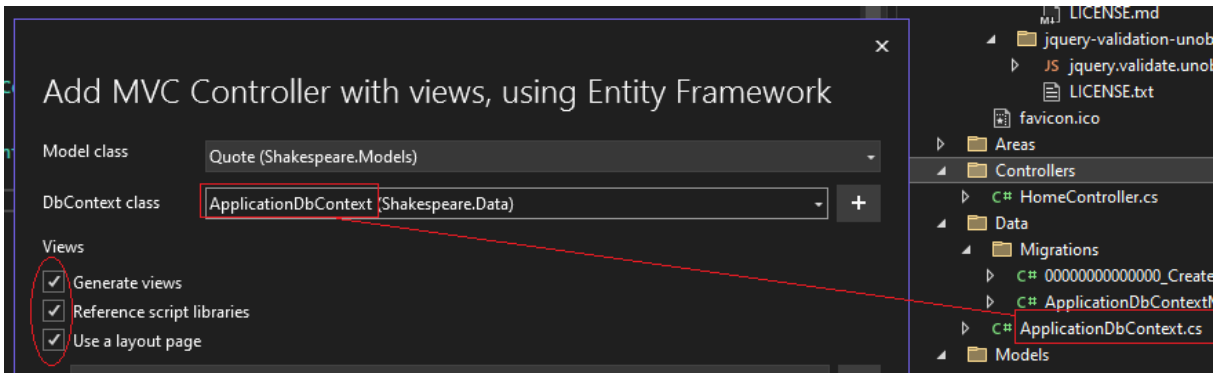
Ще създадем нашия QuotesController.cs с възможно най-много генериране на автокод за CRUD операциите (create, read, update, delete).



Controllers -> Add -> Controller.

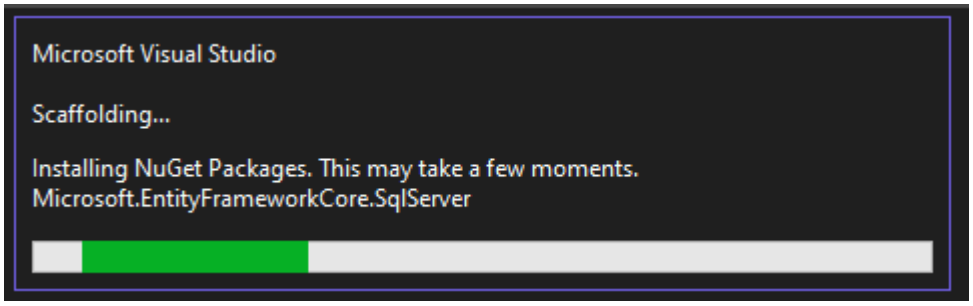


Добавяме готово Scaffold-ване със Views, използвайки Entity – т.е. таблици за БД.



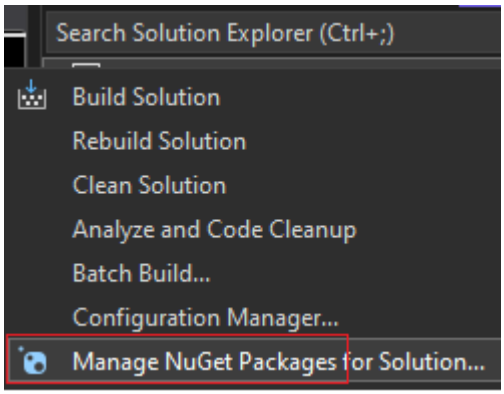
Проверяваме дали имената съвпадат и дали всички отметки са маркирани за максимално генериране на автокод. Не успях да направя напвреме скрийншот, но автоматично ни дописва и името на контролера QuotesController -> OK

Автогенерирането е бавно, но това спестява страшно много писане от страна на разработчика.

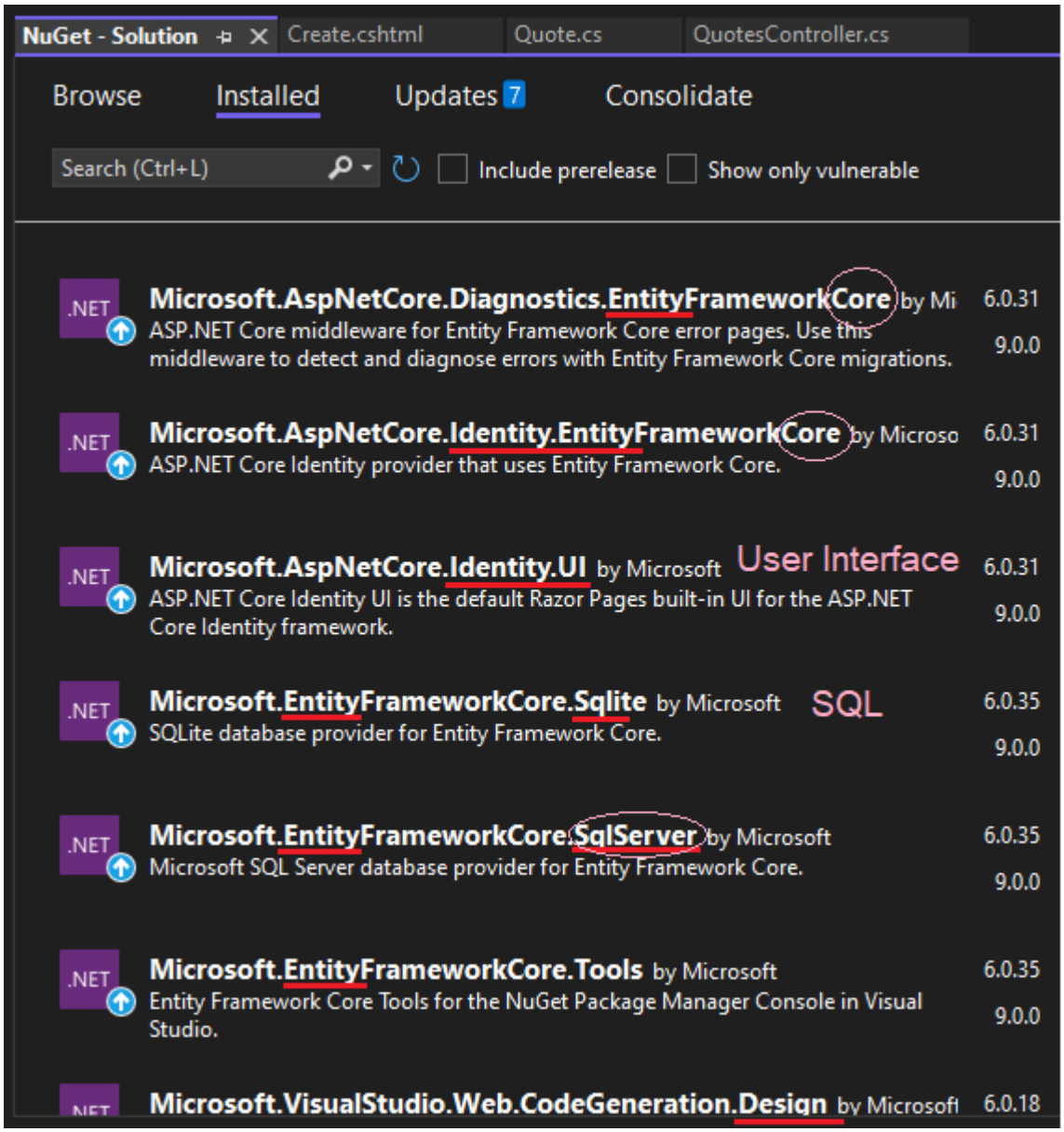


Прочитаме, че инсталира необходимите NuGet пакети, Updating dependency information, Building project, Generation code…

Нека проверим кои NuGet пакети е инсталирал -> д. б. върху Solution Exploerer -> NuGet :



Отваря страницата Installed:

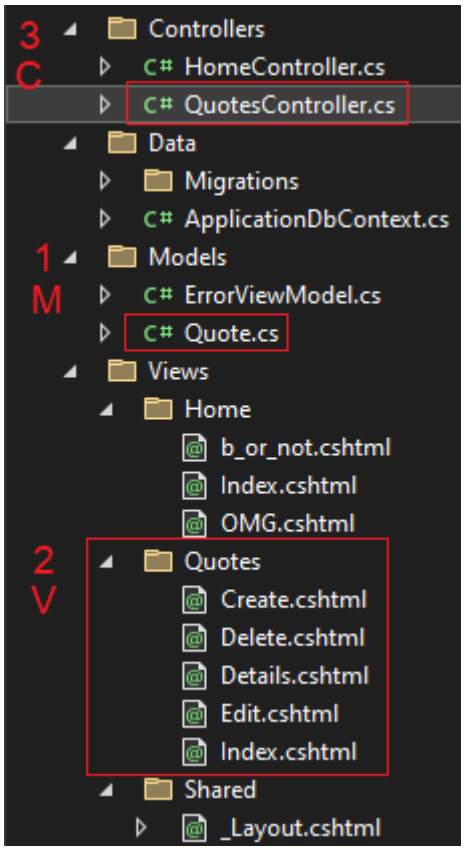


Имаме много повтарящи Entity – т.е. таблица на БД.

Ето част от кода за CRUD операциите, който трябва да ни бъде познат:

```
QuotesController.cs
Shakespeare.Controllers.QuotesController        _context

        // GET: Quotes/Create
        0 references
        public IActionResult Create()
        {
            return View();
        }

        // POST: Quotes/Create
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> Create([Bind("id,QuoteBG,QuoteEN")] Quote quote)
        {
            if (ModelState.IsValid)
            {
                _context.Add(quote);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(quote);
        }
```

Може да го видим в Solution Explorer, както и всички 3 промени в MVC, с цели 5 нови страници във Views в новата папка Quotes:



Нека видим и какво има в първата страница Quotes/Create.cshtml:

```
Create.cshtml   Quote.cs   QuotesController.cs
Shakespeare
 7    <h1>Create</h1>
 8
 9    <h4>Quote</h4>
10    <hr />
11    <div class="row">
12        <div class="col-md-4">
13            <form asp-action="Create">
14                <div asp-validation-summary="ModelOnly" class="text-danger"></div>
15                <div class="form-group">
16                    <label asp-for="QuoteBG" class="control-label"></label>
17                    <input asp-for="QuoteBG" class="form-control" />
18                    <span asp-validation-for="QuoteBG" class="text-danger"></span>
19                </div>
20                <div class="form-group">
21                    <label asp-for="QuoteEN" class="control-label"></label>
22                    <input asp-for="QuoteEN" class="form-control" />
23                    <span asp-validation-for="QuoteEN" class="text-danger"></span>
24                </div>
25                <div class="form-group">
26                    <input type="submit" value="Create" class="btn btn-primary" />
27                </div>
28            </form>
```

Имаме познатия label за надпис и input за въвеждане на текст.

Когато стартираме проекта и отворим нашата страница Quotes:

https://localhost:44367/Quotes/

A database operation failed while processing the request.

SqlException: Cannot open database "aspnet-Shakespeare-826e384d-921b-4bb6-84c3-dddfb1a6af3b" req

Applying existing migrations may resolve this issue

There are migrations that have not been applied to the following database(s):

ApplicationDbContext

- 00000000000000_CreateIdentitySchema

Apply Migrations

In Visual Studio, you can use the Package Manager Console to apply pending migrations to the database:

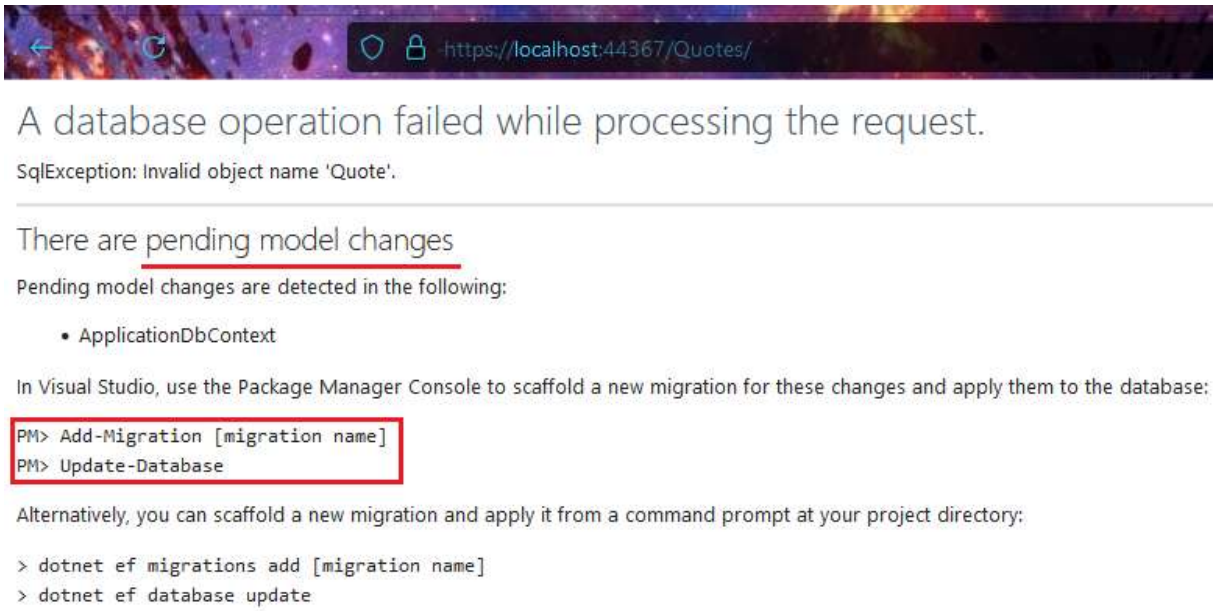
```
PM> Update-Database
```

Alternatively, you can apply pending migrations from a command prompt at your project directory:

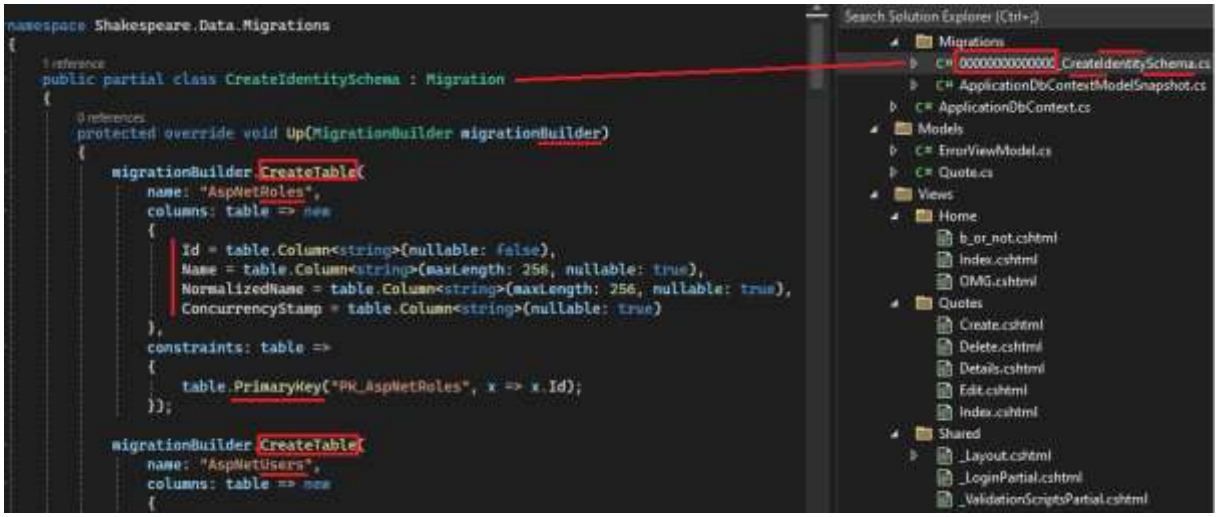
```
> dotnet ef database update
```

Ни изписва, че не сме дигнали базата данни и ни подсказва как да го направим: като приложим миграции в - Package Manager - PM>

Ако натисна синия бутон за Apply Migrations, ни подсказва още:

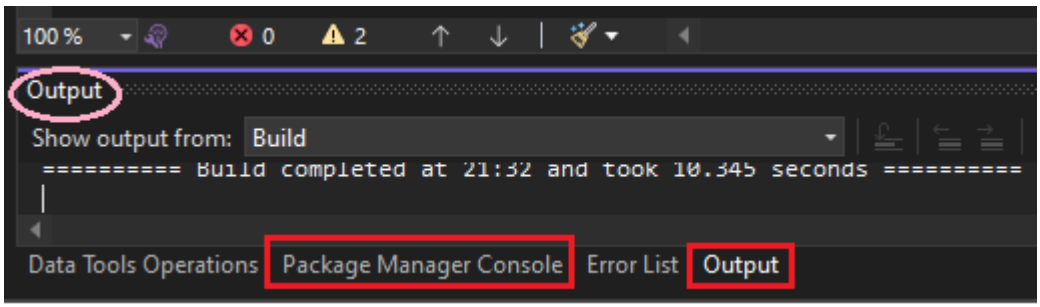


Трябва да използваме команда в Package Manager PM> Add-Migration и името на миграцията., а понеже всичко в БД става само след Update и Refresh, за да я потвърдим, трябва след създаването на нова миграция, да я ъпдейтнем/приложим.

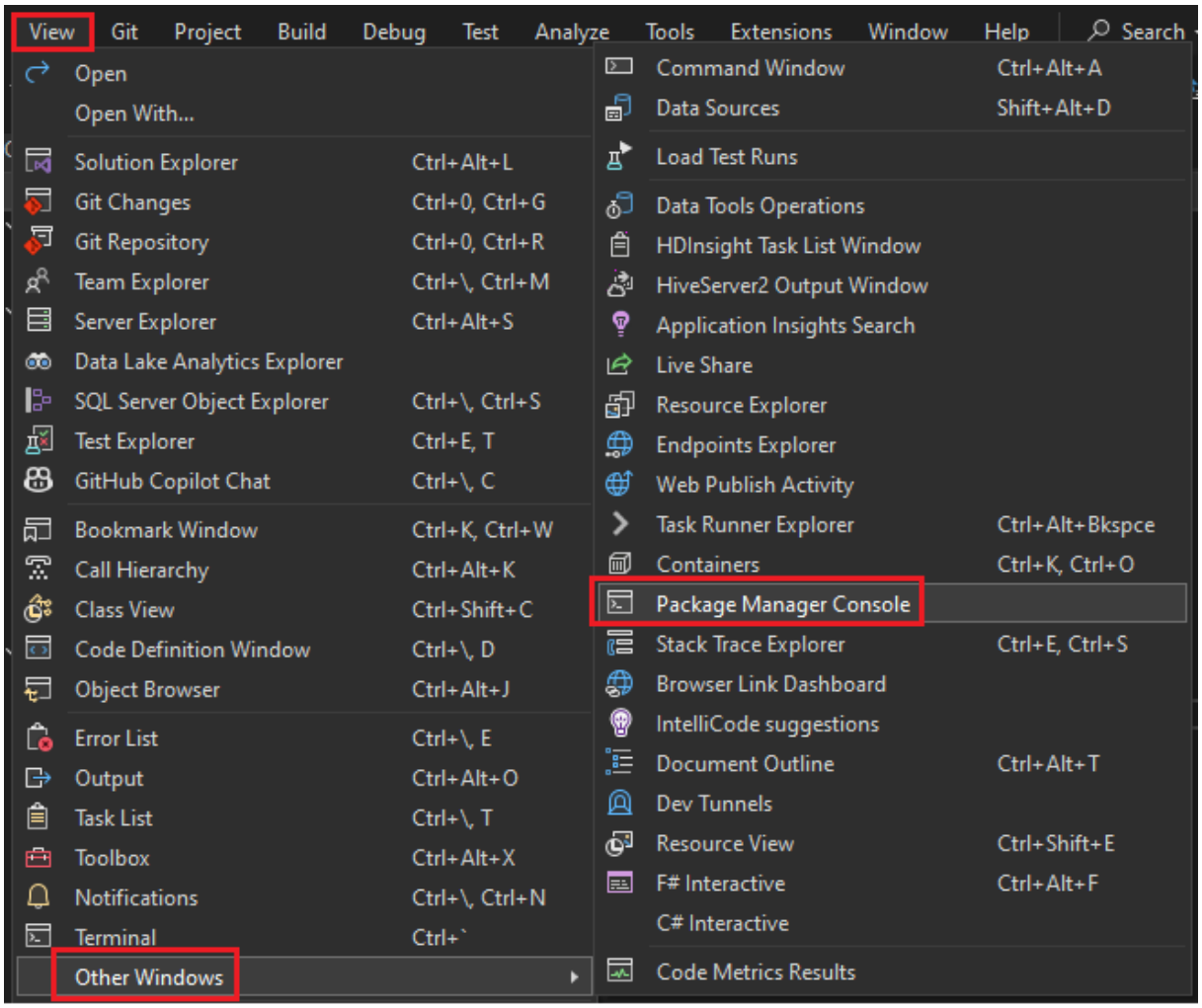


Миграциите са езикът, с който VS създава таблици за БД.

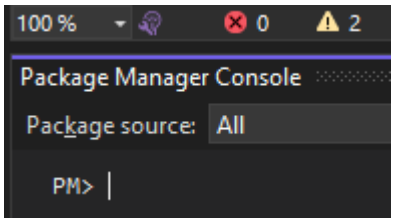
Долната лента е Output и трябва да я сменим с Package Manager:



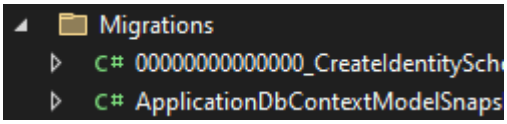
Ако не го виждаме, можем да го изведем от View-> Other Windows -> PM или в търсачката:



Прилича на терминала и е в готовност за въвеждане:



Хвърляме поглед към миграциите до момента:



Спазваме инструкцията и въвеждаме Add-Migradion с описателно име, обикновено първата винаги е initialmigration, за да се знае, че именно тя е началната:

Add-Migration initialmigration

```
100 %    No issues found
Package Manager Console
Package source: All    Default project:

PM> Add-Migration initialmigration
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 6.0.35 initialized 'Applica
options: None
To undo this action, use Remove-Migration.
PM>
100 %
Data Tools Operations | Package Manager Console | Error List | Output
```

Изписва, че създаването е готово и продължава да подсказва, че можем да отменим действието с Remove-Migration. Нека проверим. Ето я новата миграция.

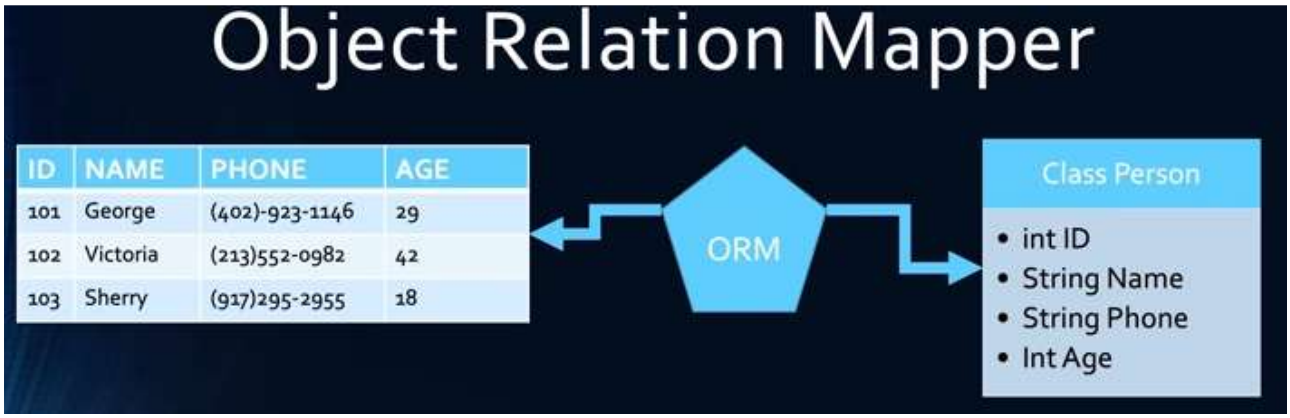
```
Migrations
  C# 00000000000000_CreateIdentitySchem
  C# 20241201203454_initialmigration.cs
  C# ApplicationDbContextModelSnapsho
```

Ето нейния автокод:

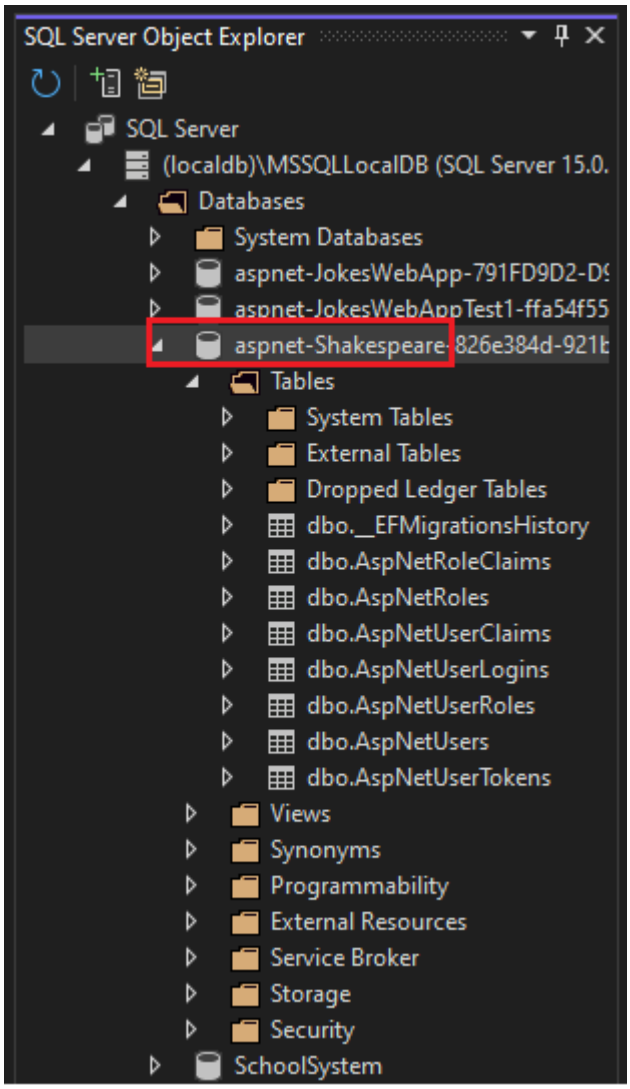
```
...lmigration.cs   Quote.cs   QuotesController.cs
Shakespeare.Data.Migrations.initialmigration   Up(MigrationBuilder m

namespace Shakespeare.Data.Migrations
{
    1 reference
    public partial class initialmigration : Migration
    {
        0 references
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Quote",
                columns: table => new
                {
                    id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    QuoteBG = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    QuoteEN = table.Column<string>(type: "nvarchar(max)", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Quote", x => x.id);
                });
        }

        0 references
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Quote");
        }
    }
}
```

Има два начина за създаване на БД за система като тази DAO и ORM:
DAO (data access object):
*традиционен, старомоден с ръчно въвеждане и все още предпочитан от компаниите и програмистите, заради по-голямата видимост какво се случва в БД и съответно по-големия контрол;
ORM (object relation mapper):
* предпочитан от програмистите, когато искат да спестят време; автоматично генерира таблици без да се налага да пишат никакви SQL-твърдения/отчети; автоматично генерира таблици въз основа на дефинирани класове в приложенията; БД се обновява чрез миграциите; подходящи за прости приложения.

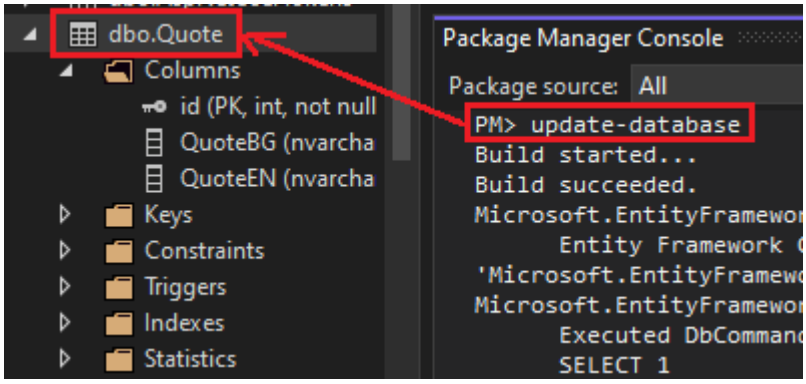
Entity – името на ORM на Майкрософт, който използваме в момента;



Нека видим какво сме генерирали в SQL Server Object Explorer:



Прилагаме миграцията с Update-DataBase и веднага се генерира нова таблица към всички останали с име dbo.Quote, където dbo = database object. Има точно колоните, които искаме.



dbo.Quote [Design] | site.css | Quote.cs | QuotesController.cs

Update | Script File: dbo.Quote.sql

Name	Data Type	Allow Nulls	Default
id	int	☐	
QuoteBG	nvarchar(MAX)	☑	
QuoteEN	nvarchar(MAX)	☑	
		☐	

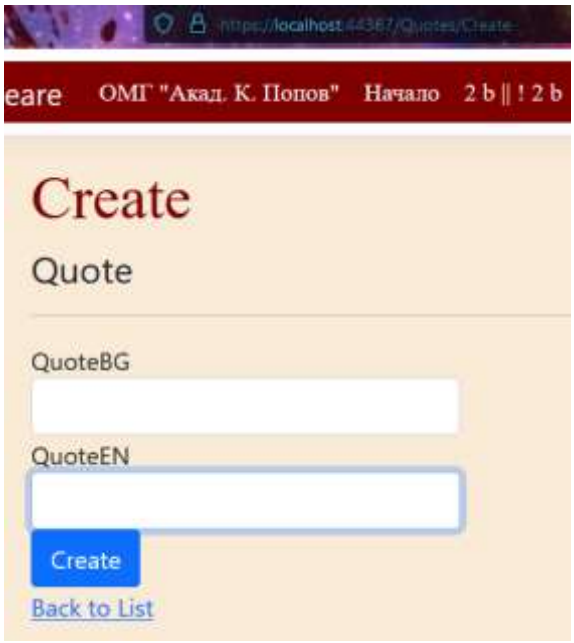
Keys (1)
PK_Quote (Primary Key, Clustered: id)
Check Constraints (0)
Indexes (0)
Foreign Keys (0)
Triggers (0)

Design | T-SQL

```
CREATE TABLE [dbo].[Quote] (
    [id]      INT            IDENTITY (1, 1) NOT NULL,
    [QuoteBG] NVARCHAR (MAX) NULL,
    [QuoteEN] NVARCHAR (MAX) NULL,
    CONSTRAINT [PK_Quote] PRIMARY KEY CLUSTERED ([id] ASC)
);
```

Резултат:

Ако дадем на Create:


https://localhost:44367/Quotes/Create
eare
ОМГ "Акад. К. Попов"
Начало
2 b || ! 2 b
Create
Quote
QuoteBG
QuoteEN
Create
Back to List