12 VISUAL STUDIO ИНТЕГРАЦИЯЛАНҒАН ОРТАСЫНДА WEB-КЛИЕНТТІК ҚОСЫМШАЛАР ҚҰРУ

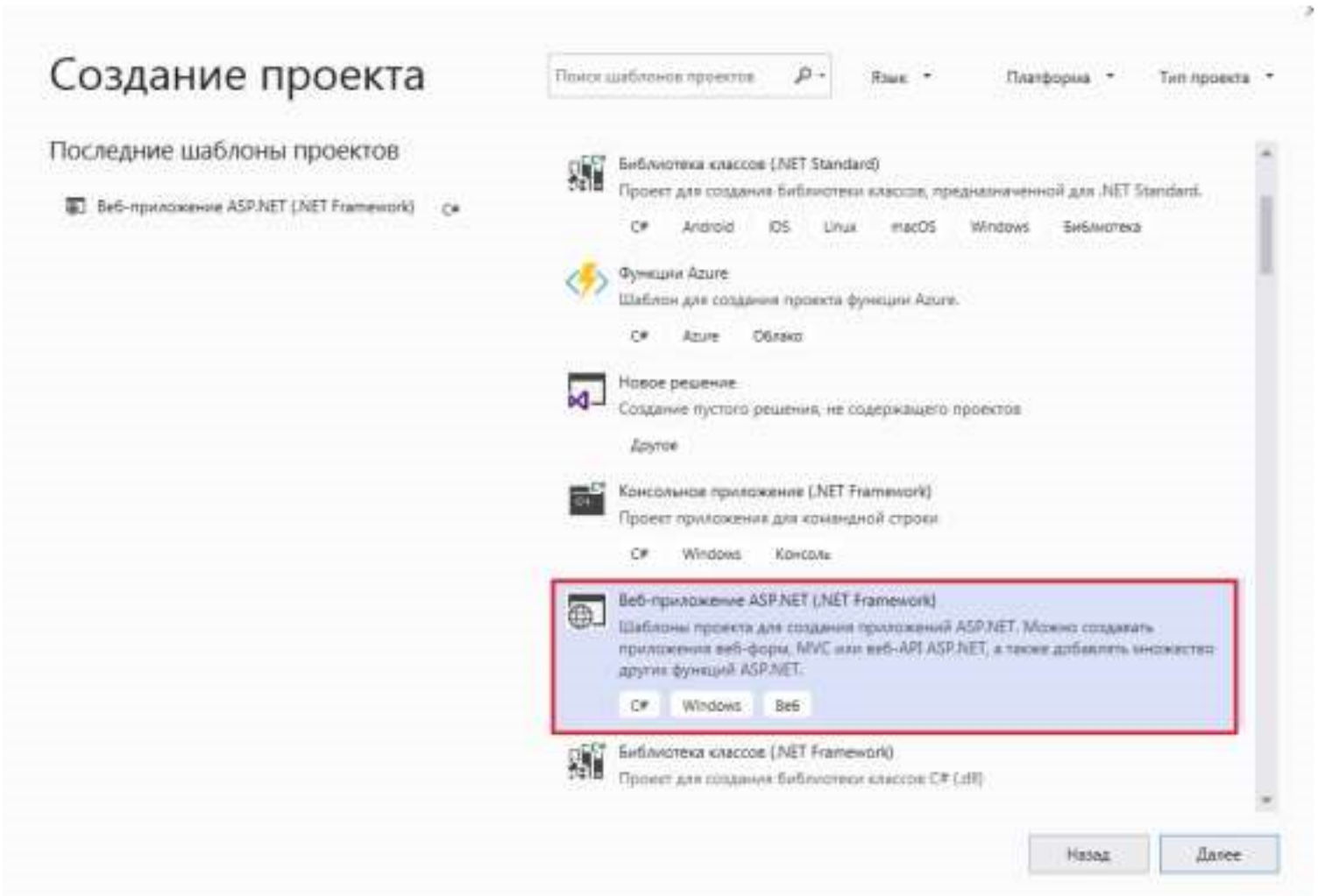
12. Visual Studio интеграцияланған ортасы

Visual Studio - бағдарламалық жасақтаманы әзірлеу қызметіне арналған және басқа да бірқатар құралдарды қамтитын Microsoft өнімдерінің желісі болып табылады. Бұл өнімдер көмегімен консольді және графикалық интерфейс арқылы Windows формалар, web-сайттар, web-қосымшалар және web-қызметтер түрлерін жасақтауға болады. Бағдарлама кодын құру және басқаруда Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone, .NET Compact Framework және Silverlight платформаларын қолдайды. Бұл орта бағдарлама кодын құру, күйге келтіру және редакторлау жұмыстарына арналған шығармашылық алаң. Интеграцияланған жасақтамалар ортасы (IDE) – бағдарламалар құрудың көптеген аспектісінде қолдануға арналған көп функционалды бағдарлама болып табылады. IDE–де қамтылған стандартты редакторға қоса Visual Studio бағдарламалық қамтаманы жасақтау үрдісін жеңілдетуге арналған компиляторлар, кодты аяқтау аспаптары, графикалық дизайнерлер және тағы да басқа көптеген функцияларды қамтиды [18]. Бұл ортаға қажетті нысандарды Visual Studio Installer көмегімен қосымша орнатуға болады.

Visual Studio 2019 интерфейсімен танысу қадамдарына көшейік. Бағдарламаның алғашқы беті келесі түрде ашылады. Жаңа жоба құру үшін *Создание проекта* батырмасын таңдаймыз.

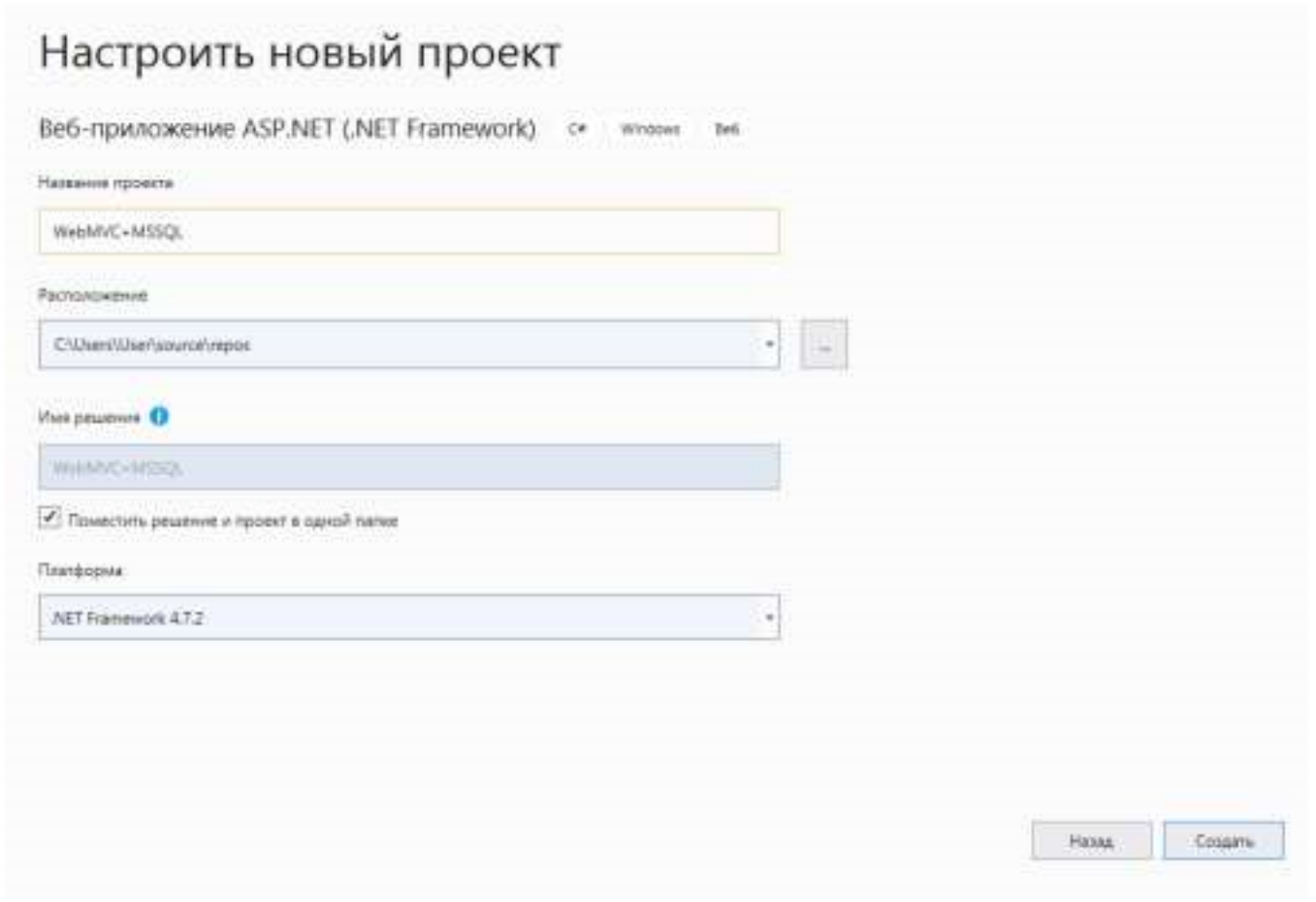
Сурет 8.1 - Visual Studio 2019 ортасында жаңа жоба құру

Келесі қадамда жоба шаблонының түрін таңдаймыз. Біздің жағдайда *Web-приложение ASP.NET (.NET Framework)* арқылы жоба құру түрі таңдалады.



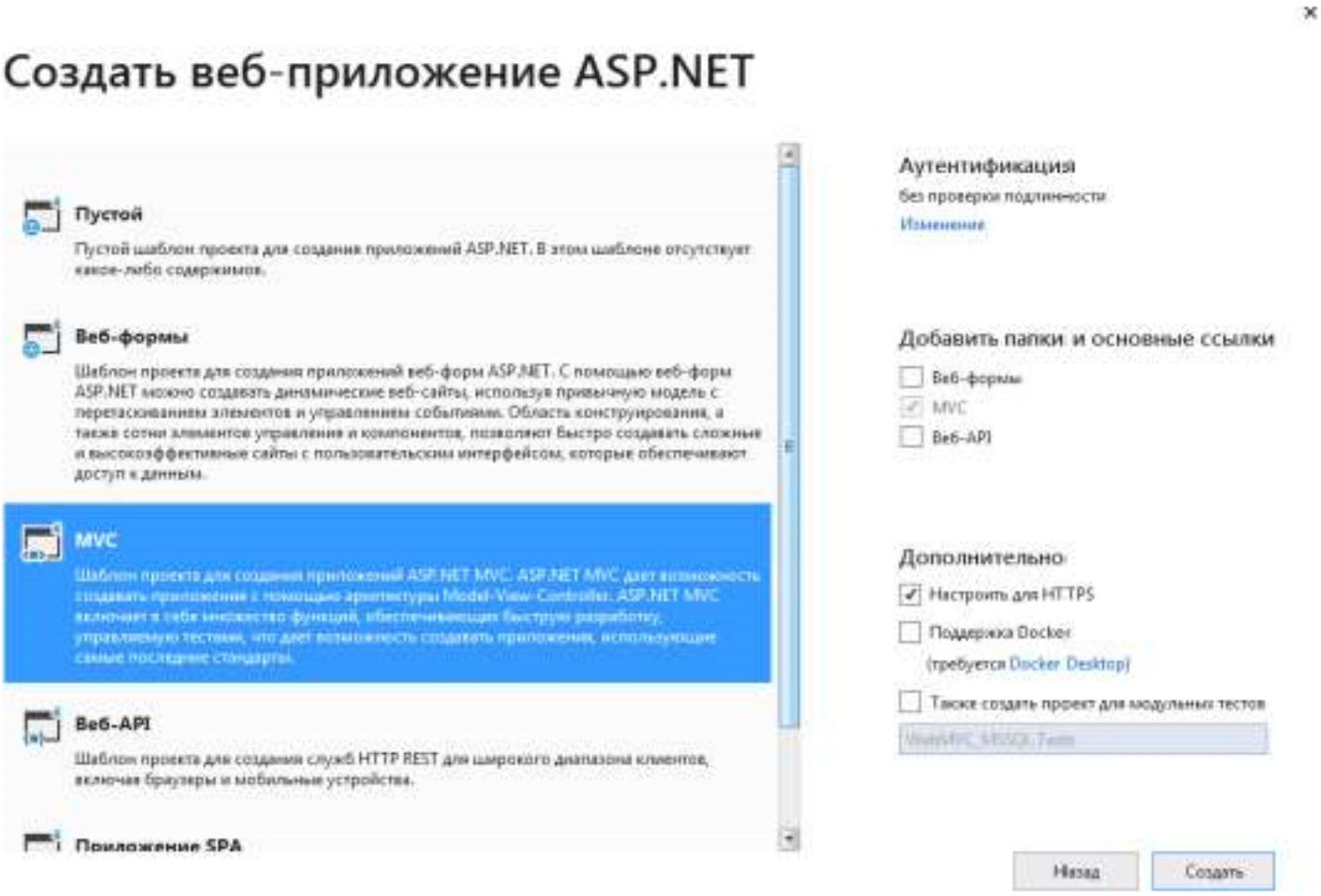
Сурет 8.2 – Жоба құру шаблонын таңдау

Далее батырмасы арқылы жоба құру терезесіне өтеміз. Ашылған терезеде жаңа жобаның параметрлері, яғни, жоба атауы, сақталатын орны, жүзеге асатын платформасы түріндегі деректер енгізіледі.



Сурет 8.3 – Жаңа жоба параметрлерін тағайындау

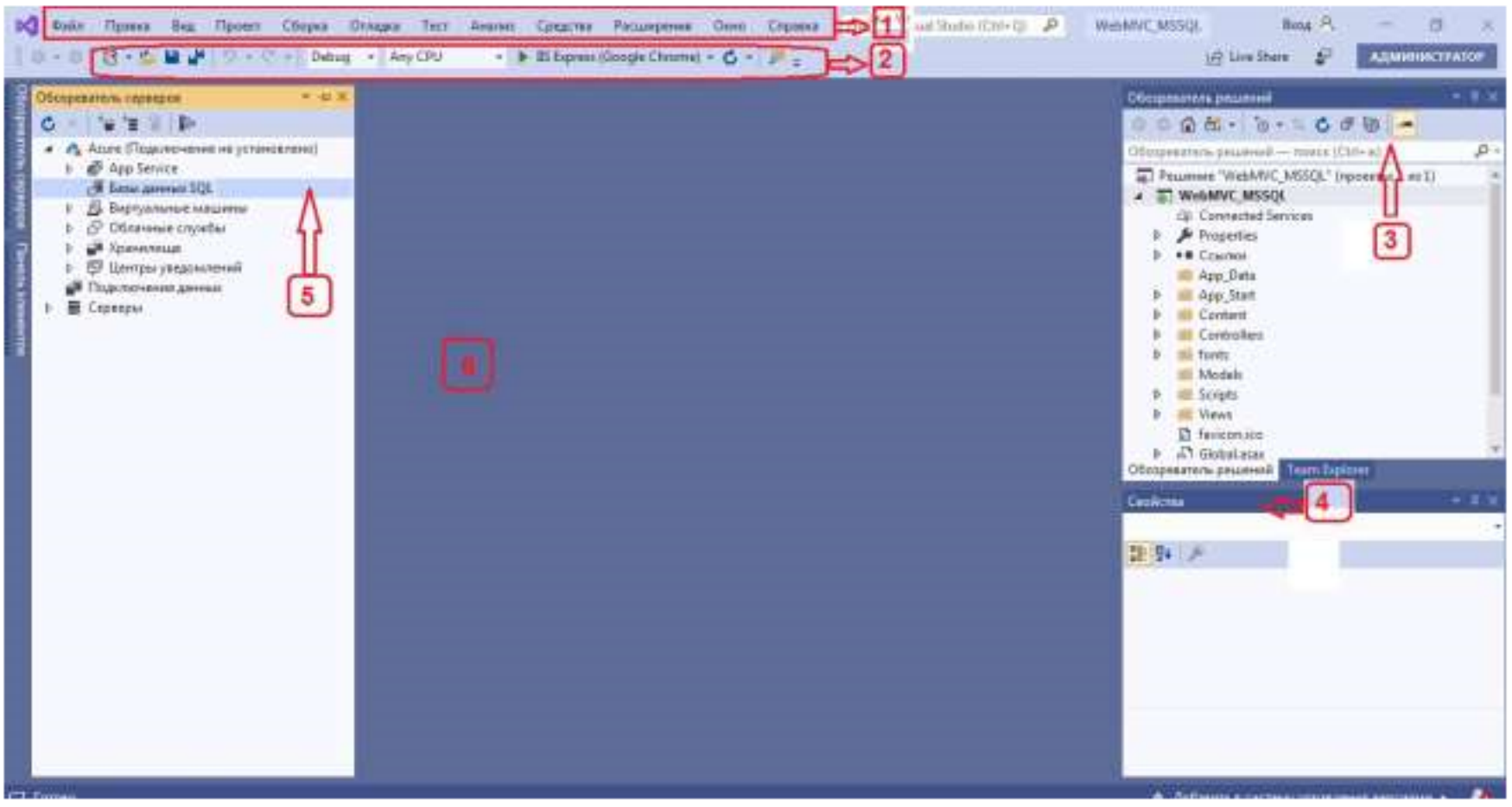
Біздің жағдайда жобаға WebMVC_MSSQL атауын береміз. Үнсіз келісім бойынша таңдалған жобаны сақтау орнын өзгертпейміз. Шешімдер атауы жобаға сәйкес автоматты түрде шығады. Сонымен бірге, бұл терезеде *Поместить решение и проект в одной папке* жолына жалауша белгісін қоямыз. Платформа ретінде .NETFramework 4.7.2 таңдалады. Қажетті деректер енгізілгеннен кейін *Создать* батырмасына шерту арқылы жоба құру терезесіне өтеміз. Келесі терезеде ASP.NET web-қосымшасын құру стилі таңдалады. Біздің жағдайда MVC жоба шаблонын құру стилі таңдалады (MVC шаблоны туралы келесі тақырыпта кеңірек тоқталатын боламыз).



Сурет 8.4 - ASP.NET web-қосымшасы үшін MVC жоба шаблонын таңдау

Создать батырмасы арқылы жоба құру терезесіне кіреміз.

Visual Studio бағдарламасының графикалық интерфейсі басқа бағдарламалардағыдай бас мәзір және құрал-саймандар панелінен тұрады. Деректер қорымен байланысқан жобалар құру кезінде IIS Express сервері қолданылады. Бұл ортада құрылатын жоба Шешім (Решение) түрінде құрылады. Шешімдер шолушысының құрылымы өзіңіз таңдаған жоба құру шаблонына сәйкес жасақталады. Мысалы, біздің жағдайда MVC паттерніне сәйкес жоба құру түрі таңдалғаннан кейін, жоба құрылымы Models-Views- Controllers бумаларынан тұратын құрылымды қамтиды. Бұл ортада деректер қоры орналасқан жергілікті және бұлттағы серверлермен байланыс орнатуға болады. Ол үшін терезенің сол жақ бөлігіндегі *Обозреватель серверов* батырмасына шертеміз.



Сурет 8.5 – Visual Studio бағдарламасының интерфейсі

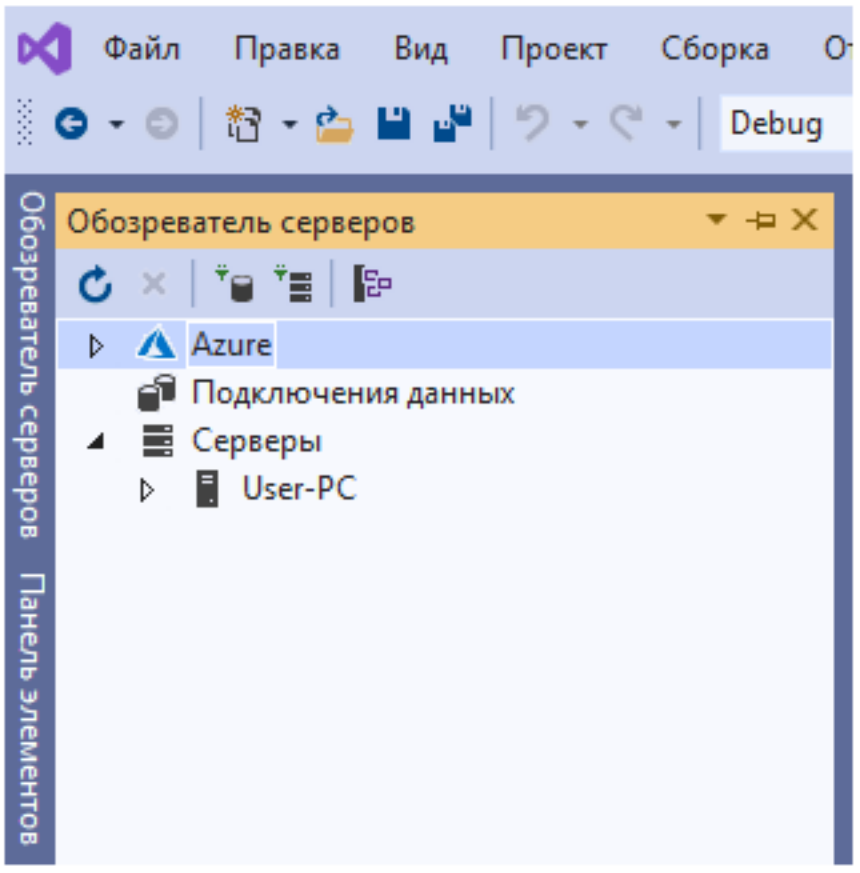
Visual Studio бағдарламасының интерфейсі келесі құрылымнан тұрады (сурет 8.5):

1 – Бас мәзір жолы;
2 – Құрал-саймандар панелі;
3 – Шешімдер шолушысы;
4 – Қасиеттер терезесі;
5 – Серверлер шолушысы;
6 – Жұмыс алаңы.

7.1 Visual Studio ортасында деректер қорының серверлерімен байланыс орнату

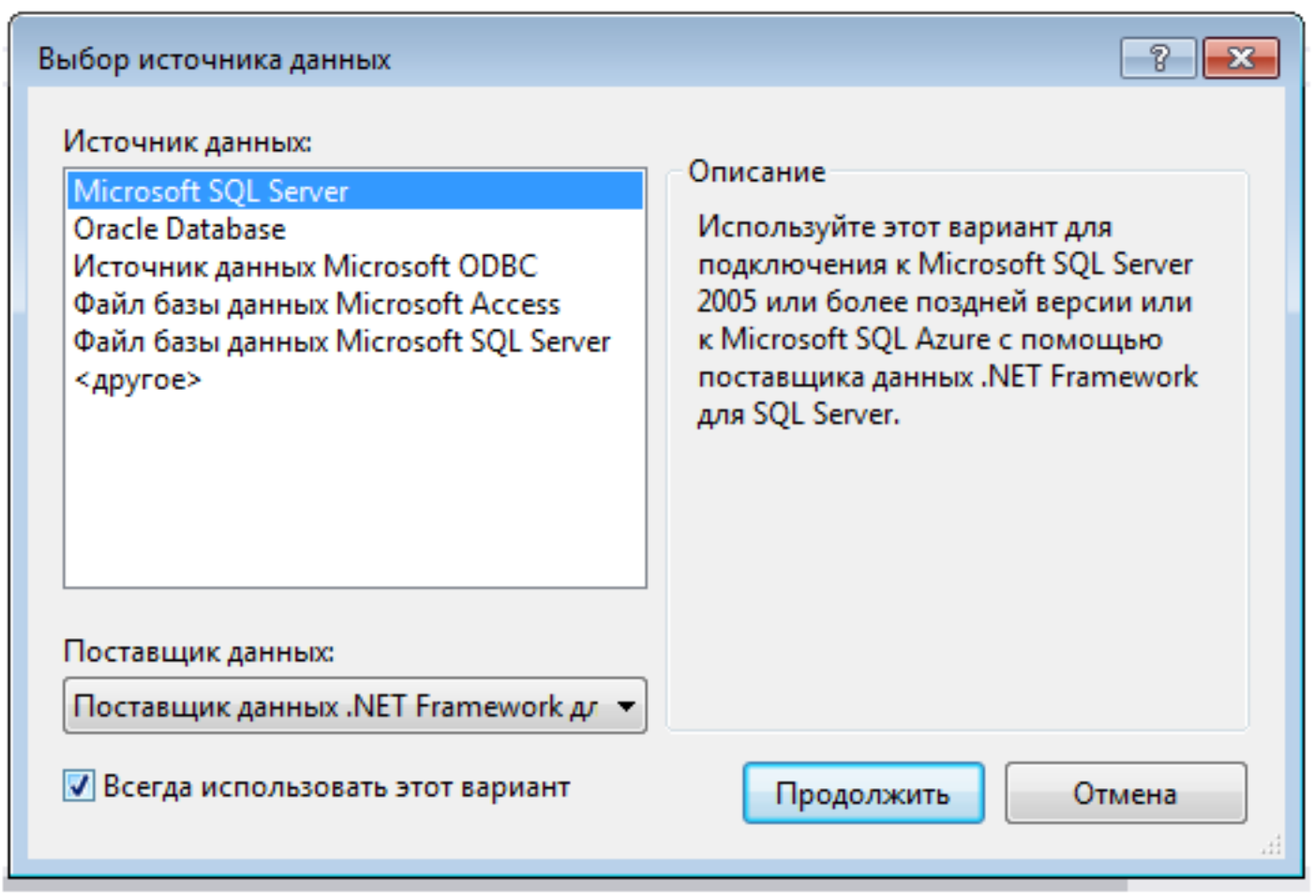
Visual Studio ортасында деректер қоры қосымшаларын құру барысында үнемі MS SQL Server немесе Oracle типті серверлерге қатынау қажеттілігі туындайды. Кестелер мазмұнын қарау, сұраныстар құру тағы да басқа көптеген жұмыстарды SSMS түріндегі басқа ортаға өтпей бір жүйеде орындаған ыңғайлы. Visual Studio-да серверлер шолушысының мүмкіндігі кіріктірілген. Ол арқылы серверде орналасқан деректер қорына Visual Studio ортасынан қалаған уақытта қосыла аламыз.

Обозреватель серверов арқылы жұмысты бастамас бұрын, байланысқа қажетті кейбір баптауларды орындау қажет. *Обозреватель серверов* терезесі Visual Studio ортасының сол жақ жоғарғы бөлігінде орналасқан.



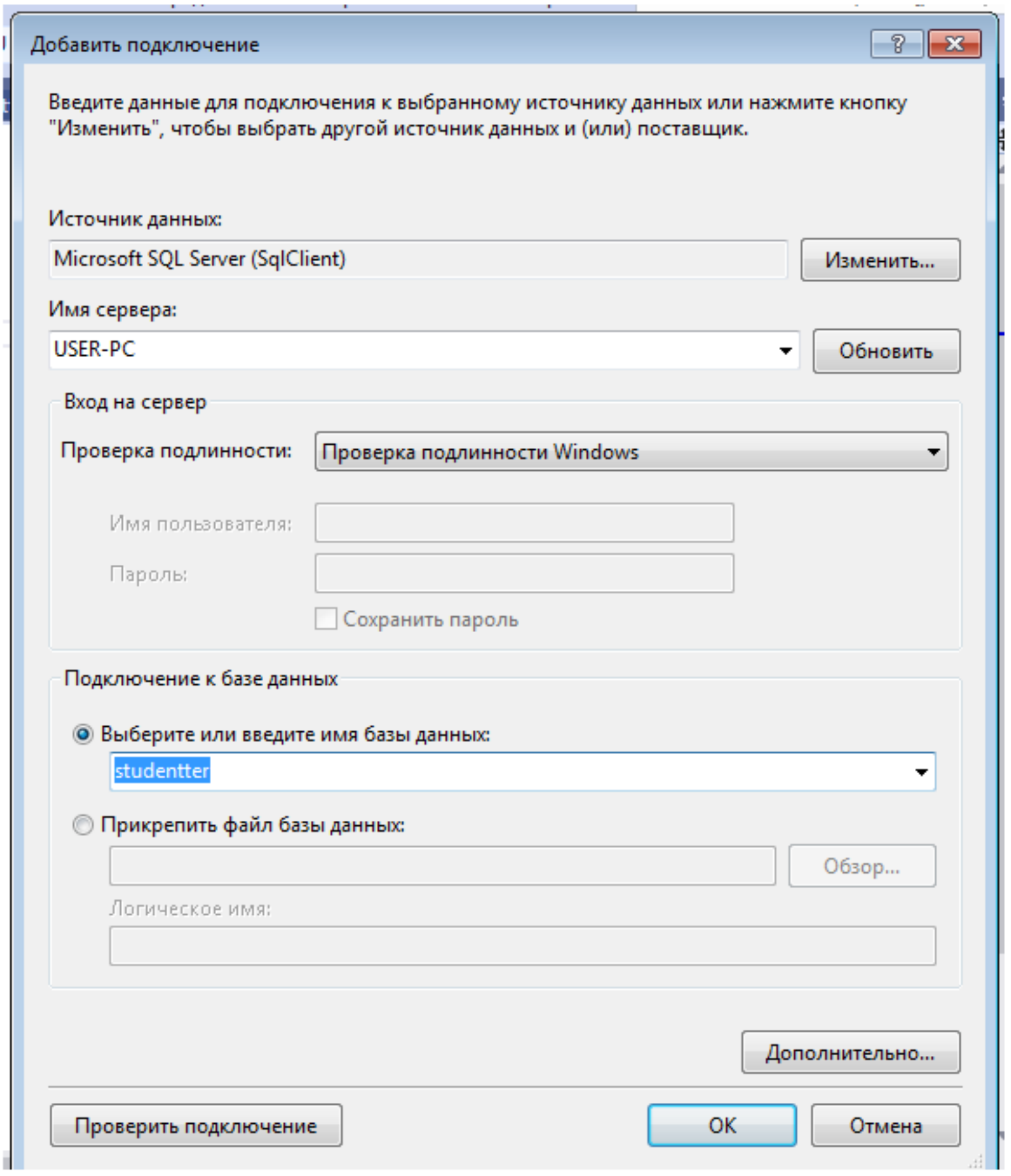
Сурет 8.6 – Серверлер шолушысы терезесін ашу

Серверге қосылу үшін  *Подключиться к базе данных* батырмасына шертіп, қажетті баптауларды жүргіземіз. *Источник данных* - Misrosoft SQL Server, сонымен бірге *Поставщик данных .NET Framework для SQL Server* жолын таңдаймыз.



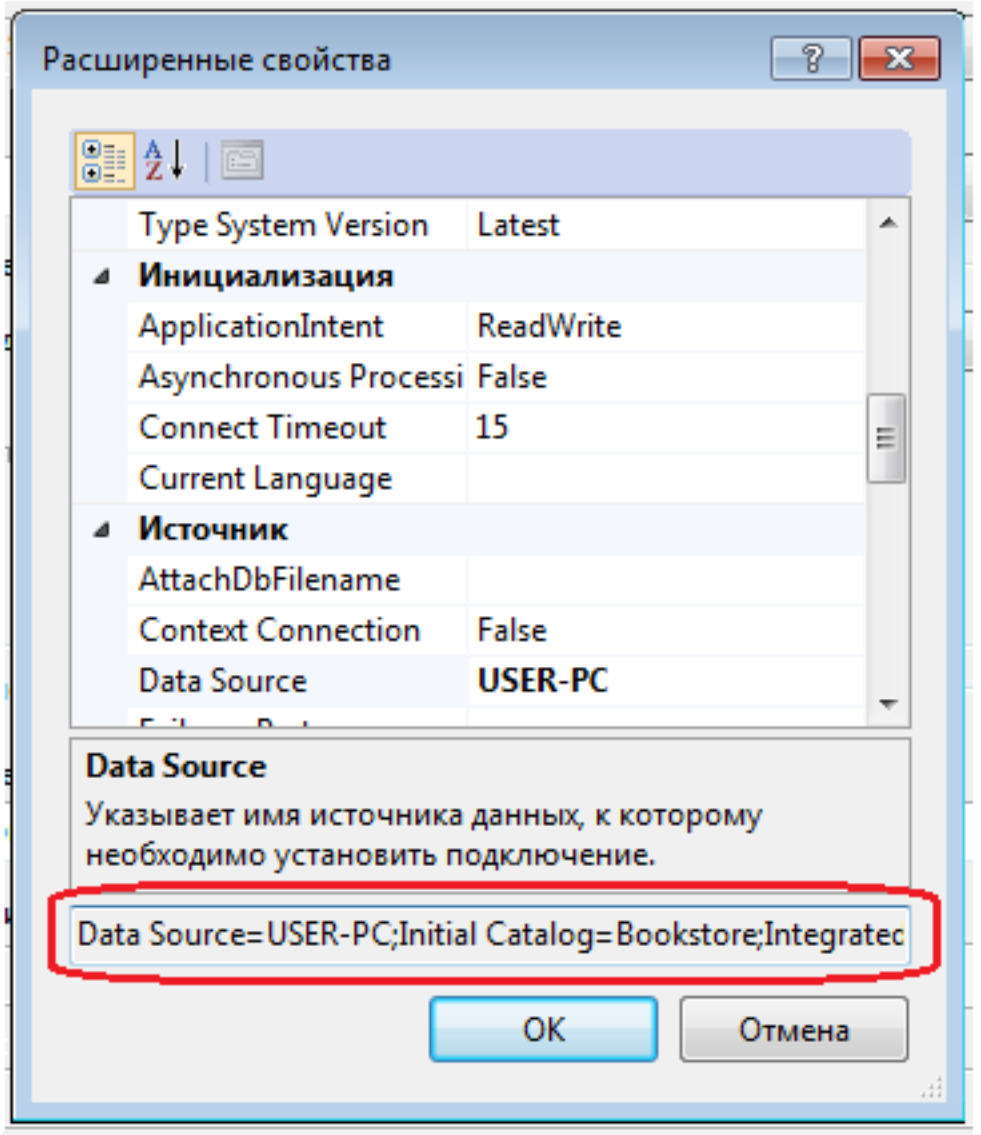
Сурет 8.7 – Деректерге қосылу көзін таңдау

Ашылған терезеде *Имя сервера* жолынан орнатылған серверлер тізімінен қажетті серверіңіздің және қажетті деректер қорын таңдаймыз. Мысалы, SSMS арқылы studentter деректер қорын қосамыз.



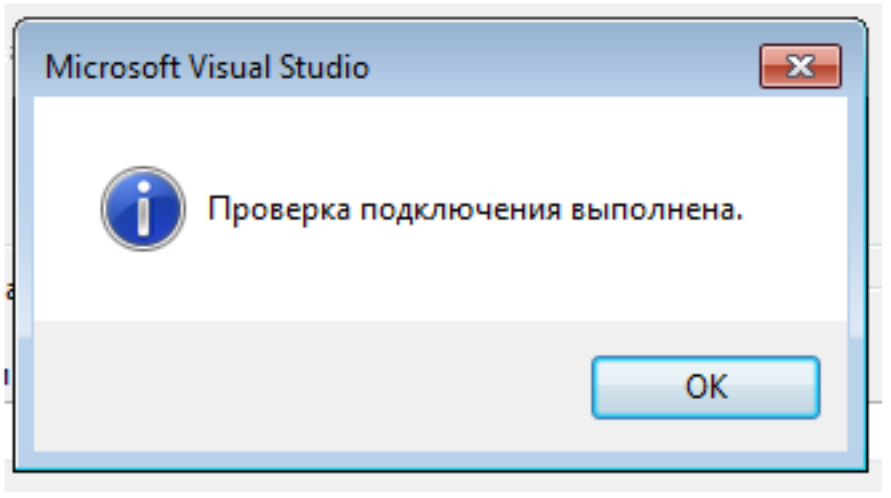
Сурет 8.8 – Қажетті сервер мен деректер қорын қосу

Дополнительно батырмасына шерту арқылы сервер мен деректер қорына қосылу жолын көре аламыз.



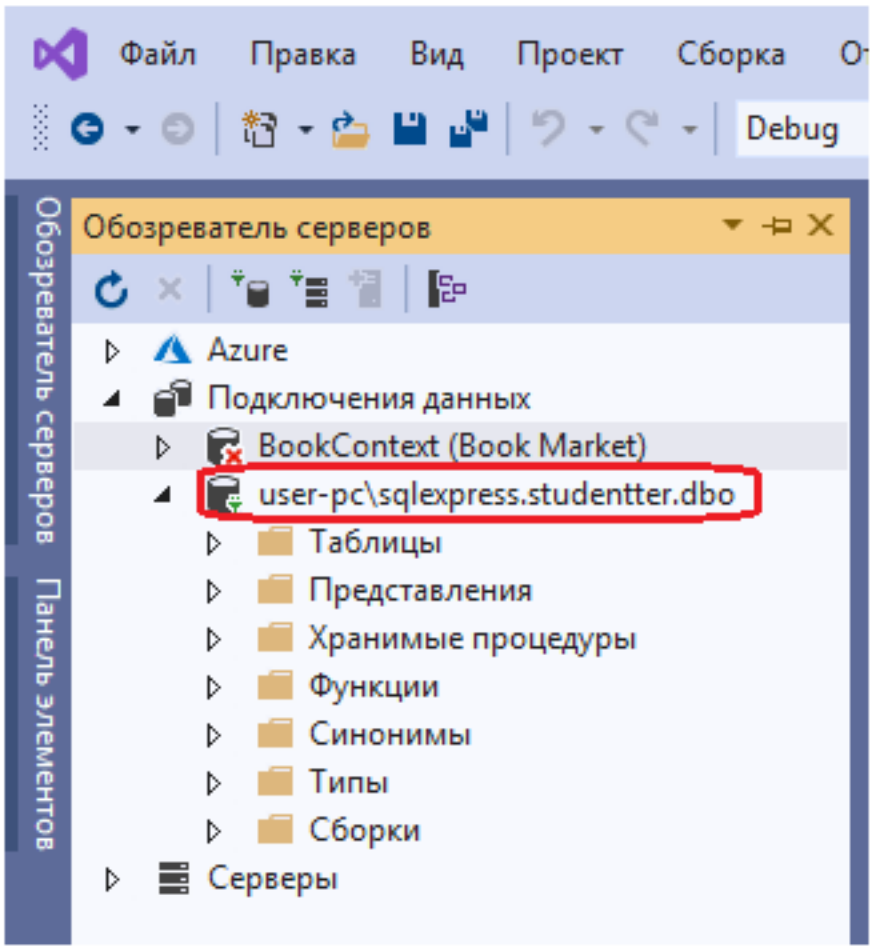
Сурет 8.9 – Деректер қорына қосылу жолы

Проверить подключение батырмасы арқылы серверге қосылу дұрыстығын тексереміз.



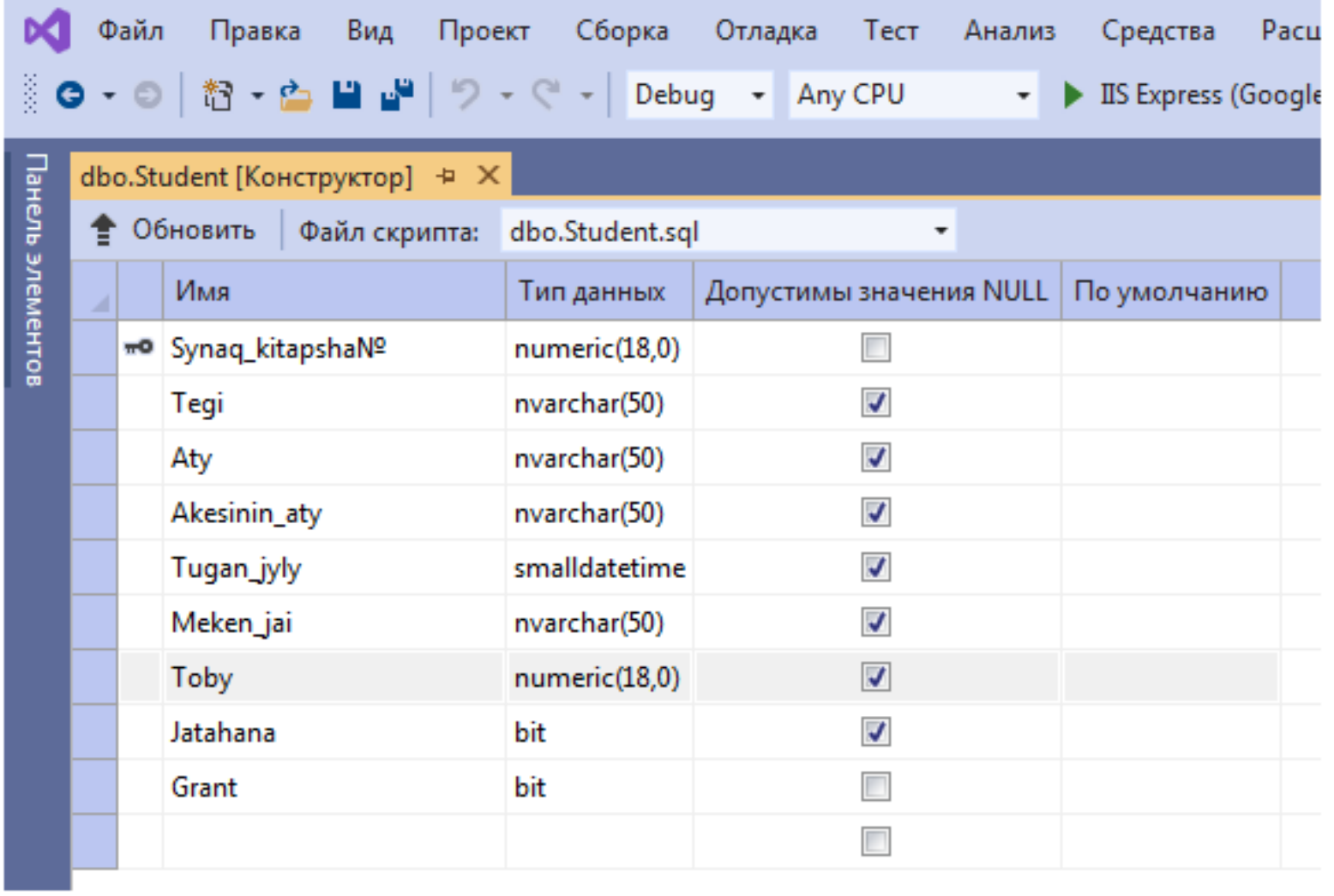
Сурет 8.10 – Деректер қорына қосылуды тексеру

Бұдан кейін *Обозреватель серверов* терезесі арқылы сервер нысандарымен жұмыс мүмкіндігін тексереміз.



Сурет 8.11 – Visual Studio ортасында сервердегі деректер қорын ашу

Бағдарлама ортасында деректер қорымен жұмыс әдеттегі күйінен ерекшеленбейді.



Сурет 8.12 – Visual Studio-да SSMS арқылы құрылған Student кестесін ашу (Studentter деректер қоры)

7.2 MVC паттерні

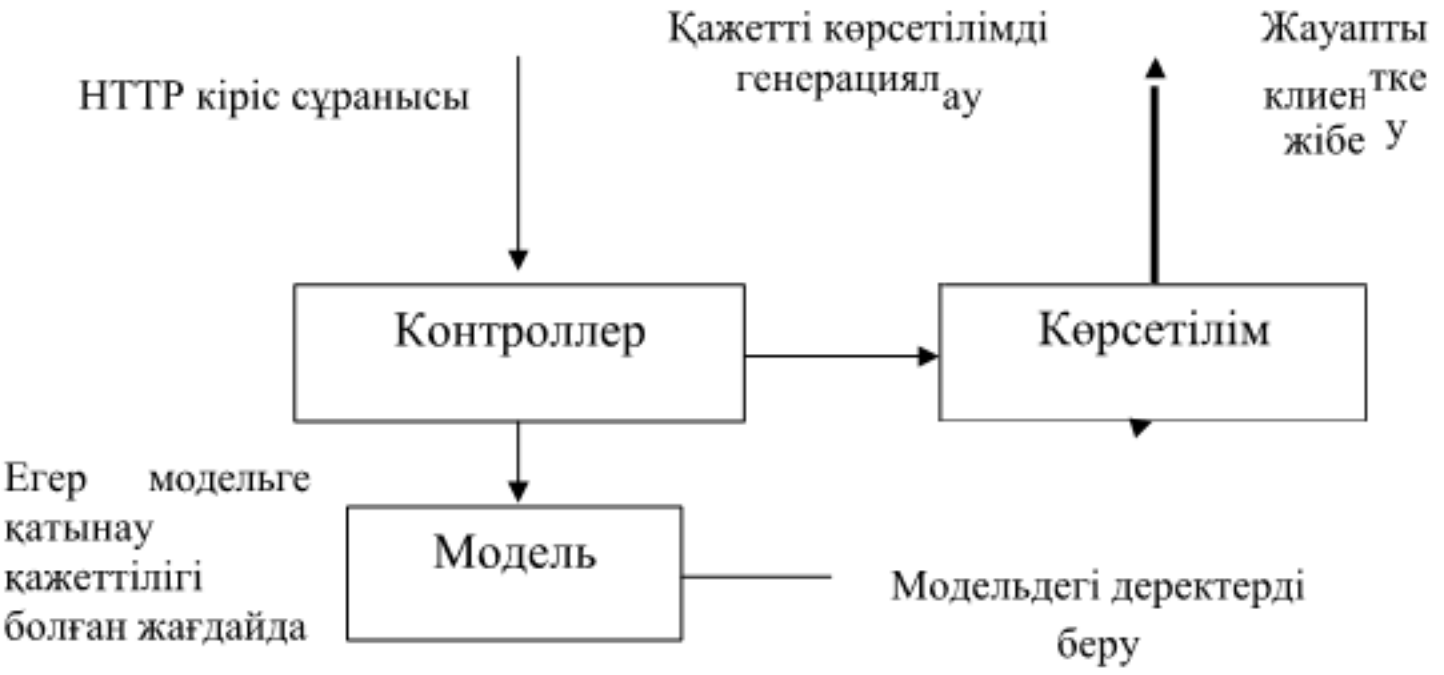
ASP.NET MVC платформасы сайттар мен web-қосымшалар құруға арналған фреймворк қызметін ұсынады. Ол MVC паттерні (шаблоны) арқылы жүзеге асырылады. MVC (model - view - controller) паттернінің концепциясына сәйкес жоба үш компонентке бөлінеді:

Контроллер (controller) – қолданушы мен жүйенің, көрсетілім мен деректер қоймасы арасындағы байланысты қамтамасыз ететін класс. Ол қолданушы енгізген деректерді қабылдап, өңдеу жұмыстарын орындайды. Өңдеу қорытындысына байланысты қолданушыға белгілі-бір деректерді жібереді, мысалы көрсетілім түріндегі деректер.

Көрсетілім (view) – қосымшаның визуалды бөлігі немесе қолданушы интерфейсі. Әдетте, қолданушы сайтқа кіргенде көретін html-бет.

Модель (model) – қолданылатын деректердің логикасын сипаттайтын класс.

Бұл компоненттердің байланысу схемасын келесі түрде суреттеуге болады:



Сурет 8.13 - MVC паттерніндегі компоненттердің байланысу схемасы

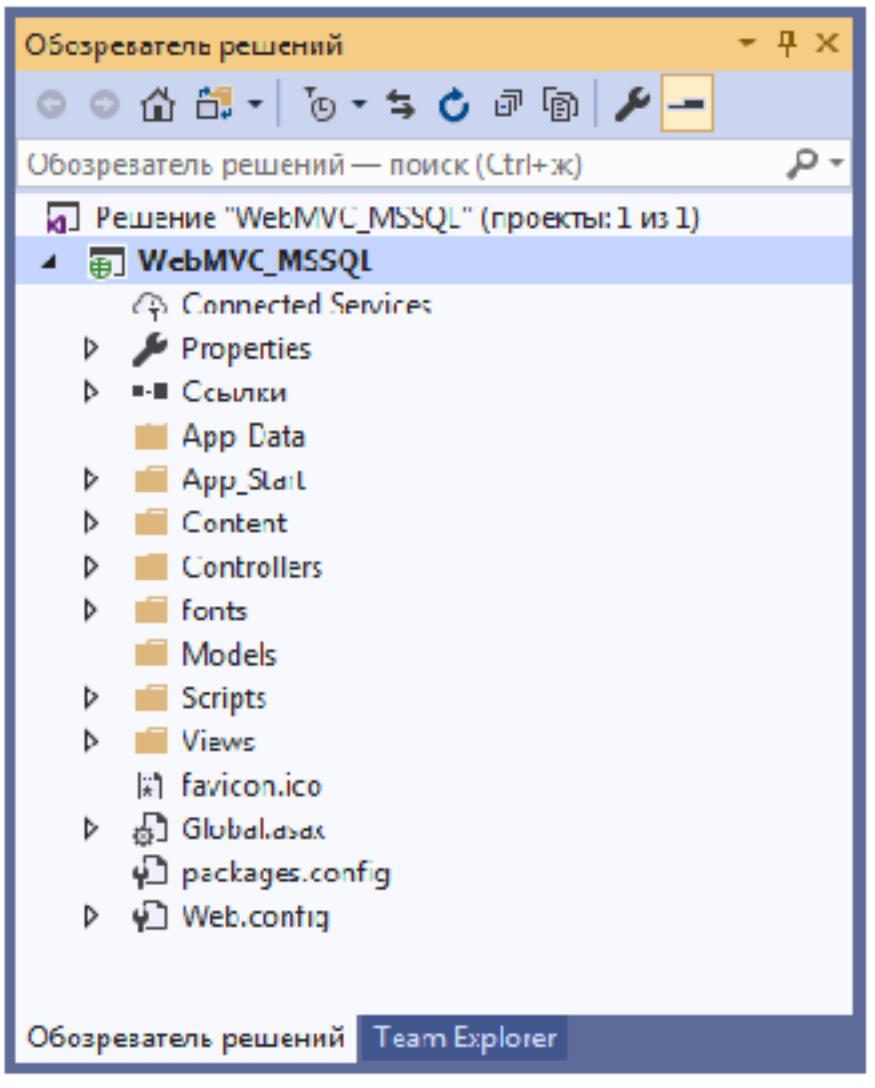
Сурет 8.13 – те көрсетілген сұлбадан модельдің тәуелсіз компонент екенін көруге болады, яғни контроллер мен көрсетілімде жүргізілген өзгерістер модельге әсер етпейді. Контроллер мен көрсетілім салыстырмалы түрде тәуелсіз компоненттер болып табылады және оларды бір-біріне тәуелсіз өзгерте аласыз. Сол арқылы жауапкершілікті бөлу тұжырымдамасы жүзеге асырылады, сондықтан жұмысты жеке компоненттер бойынша құру оңайырақ болады. Сонымен бірге бұл қосымша нәтижесін жоғары тестілеу қасиетіне ие. Егер

бізге қосымшаның визуалды бөлігі немесе фронтэнд маңызды болса, көрсетілімді контроллерге тәуелсіз, ал бэкэнд бөлігі үшін контроллерді тестілей аламыз.

Соңғы релиз ретінде ASP.NET MVC платформасының MVC 5 нұсқасы қолданылуда. MVC 5 нұсқасының көптеген аспектілері MVC 4 релизінен көп өзгешелігі жоқ. Бір нұсқаға арналған көптеген тәсілдерді келесісіне қиындықсыз қолдана береміз. Бірақ, өзіндік ерекшеліктері де бар:

- MVC 5 нұсқасында аутентификация және авторизация концепциясы өзгерді. SimpleMembershipProvider орнына ASP.NET Identity жүйесі енгізілді;
- Адаптивті және кеңейтілген интерфейс құру үшін Bootstrap css – фреймворк қолданылады;
- Аутентификация фильтры қосылған, сонымен бірге фильтрларды алдын-ала анықтау қызметімен толықтырылған;
- Бұған қоса MVC 5 паттерніне маршрутизация атрибуттары қосылған [19].

Бұл айтылғандар MVC 5 нұсқасындағы маңызды толықтырулар болып табылады. Бұдан өзге маңызды жаңалықтар ретінде үнсіз келісім бойынша Entity Framework 6 нұсқасының қолданылуы, жоба құру барысындағы кейбір өзгешеліктер (One ASP.NET концепциясы), қосымша компоненттер тағы да басқаларды айтуға болады.



Сурет 8.14 – MVC жобасының құрылымы

MVC жобасының құрылымына кіретін бумалар мен файлдардың қызметтеріне тоқталайық.

- App_Data: қосымшада қолданылатын файлдар, ресурстар және деректер қорынан тұрады.

- App_Start: қосымшаны іске қосуға қажетті инициализация логикасынан тұратын статикалық файлдарды сақтайды.

- Content: C# немесе javascript кодына қосылмайтын қосымшамен бірге орналастырылған көмекші файлдардан тұрады, мысалы, css стильдер файлдары.

- Controllers: контроллерлер кластарының файлдарынан тұрады. Үнсіз келісім бойынша бұл бумаға HomeController және AccountController файлдары қосылады.

- fonts: қосымшада қолданылатын қосымша шрифтілер файлдарын сақтайды.

- Models: модельдер файлдарынан тұрады.

- Scripts: javascript тіліндегі скриптілер мен кітапханалар каталогы.

- Views: мұнда көрсетілімдер сақталады. Барлық көрсетілімдер бумаларға топтастырылады. Бумалардың әрқайсысы бір контроллерге сәйкес келеді. Сұраныс өңделгеннен кейін контроллер бұл көрсетілімдердің біреуін клиентке жібереді. Сондай-ақ, мұнда Shared каталогы бар, ол барлық көрсетілімдерге ортақ қасиетттерді сақтайды.

- Global.asax: қосымша іске қосылған кезде орындалатын бастапқы инициализацияны орындайтын файл. Мұнда, App_Start бумасында анықталған класс әдістері жұмыс жасайды.

- packages.config - жобада орнатылған Nuget пакеттерін қамтитын файл.

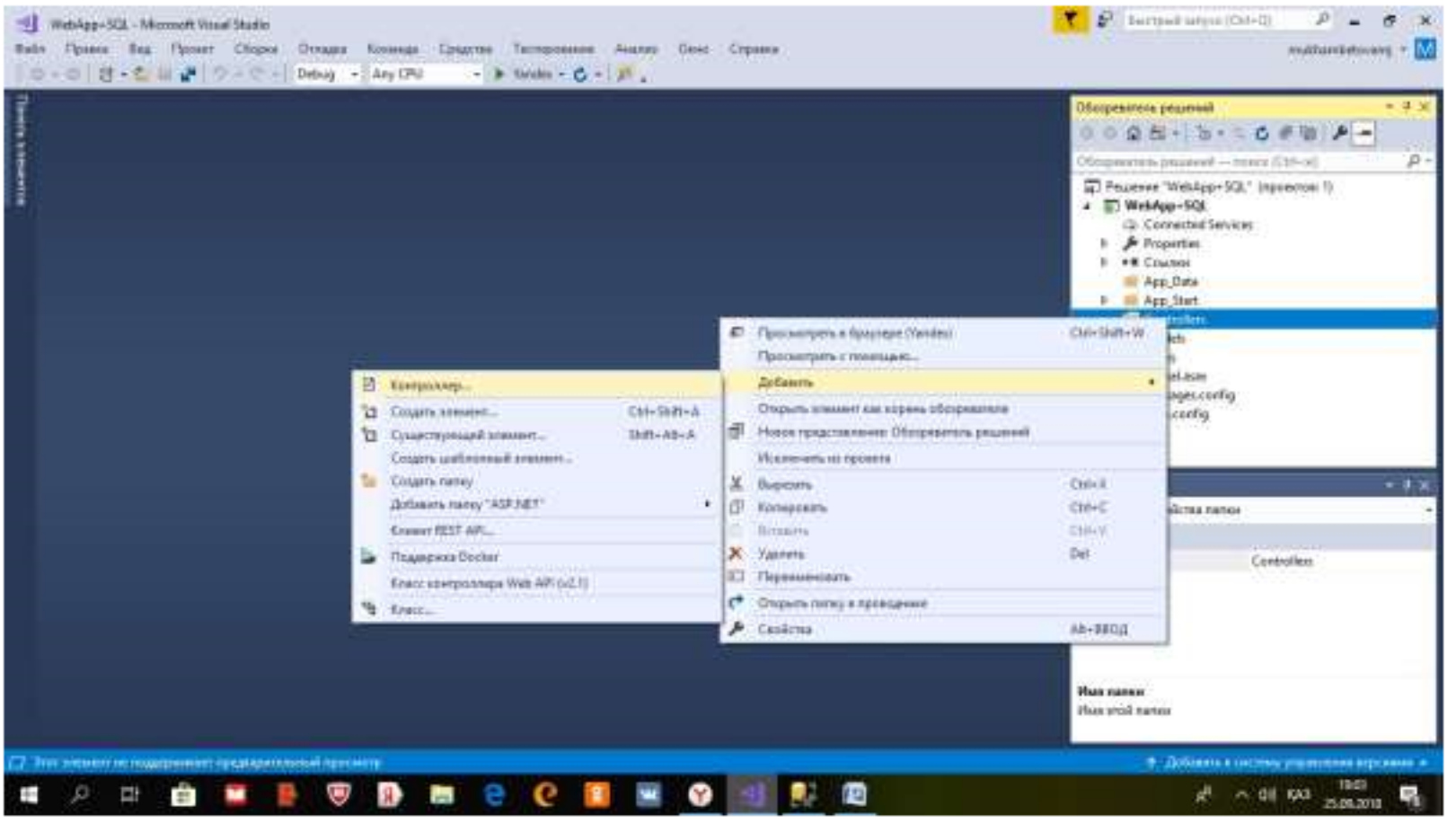
- Web.config – қосымша конфигурациясының файлы [20].

7.3 MS SQL серверінде құрылған деректер қорын MVC паттерні арқылы web-қосымша түрінде жариялау

Жаңа жоба ашу қадамдары алдыңғы тақырыптарда қарастырылған болатын. Келесі кезекте деректер қоры серверімен байланыс орнатып, деректер қорынан алынған ақпаратты web-клиенттік қосымша түрінде жариялау жұмысымен айналысамыз. MVC 5 шаблонына сәйкес жоба құру ісін контроллер нысанын құрудан бастаймыз.

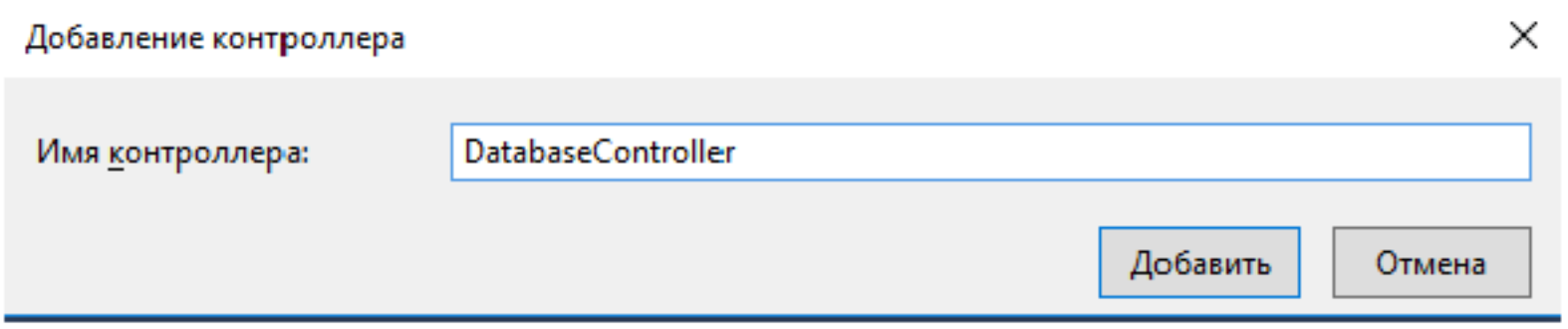
7.3.1 Controller қызметімен танысу

Вид-Обозреватель решений командасы арқылы жоба құрылымының терезесін ашамыз. Жоба MVC – ға негізделгендіктен жаңа Controller құру жұмысынан бастаймыз. Ол үшін Controllers - *Добавить* - *Контроллер* командасын орындап, бос контроллер құрамыз.



Сурет 8.15 – Жаңа контроллер құру

Келесі қадамда контроллерге ат беру кезінде Controller сөзін өшірмей, тек префиксті ғана өзгертеміз. Негізгі атауды өшірмеу қабылданған.



Сурет 8.16 – Контроллерге ат беру

Бұдан кейін MS SQL деректер қорымен байланыс орнататын контроллер құрылады.

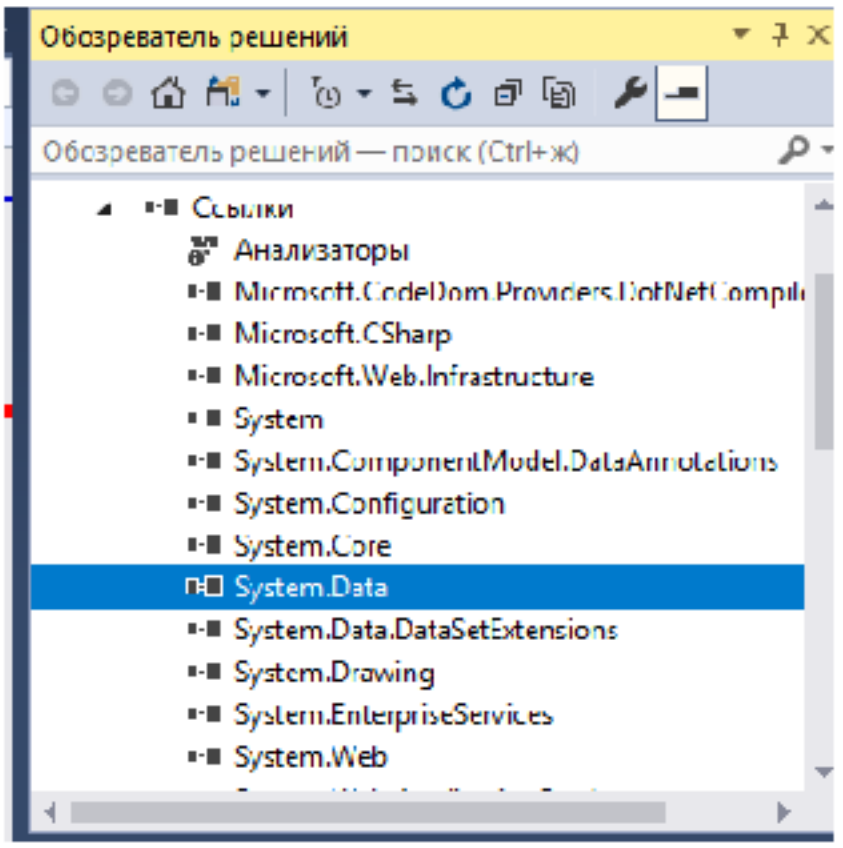
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
```

```
namespace WebApp_SQL.Controllers
{
   public class DatabaseController : Controller
   {
      // GET: Database
      public ActionResult Index()
      {
         return View();
      } } }
```

Мұндағы Index() әдісі ActionResult типін қайтарады. Index() әдісі String типті, яғни жолды қайтаратын етіп, бағдарламаны өзгертеміз, ал return View() жолын әзірге жоямыз.

```
.....
using System.Data.SqlClient;

using System.Data
namespace WebApp_SQL.Controllers
{
   public class DatabaseController : Controller
   {
      // GET: Database
      public string Index()
      {
SqlConnection sqlConnection1 = new SqlConnection("");
         SqlCommand cmd = new SqlCommand();
         SqlDataReader reader;
         cmd.CommandText = "SELECT * FROM Student";
         cmd.CommandType = System.Data.CommandType.Text;
         cmd.Connection = sqlConnection1;
         sqlConnection1.Open();
         reader = cmd.ExecuteReader();
         sqlConnection1.Close();
    } } }
```

SqlConnection *Ссылки* бөліміндегі System.Data файлында орналасады. Оны атаулар кеңістігінде using System.Data.SqlClient және using System.Data жолдарын қосу арқылы бағдарламаға белсенді етеміз.



Сурет 8.17 – Деректер қорымен жұмыс жасауға арналған System.Data файлының орналасуы

Бұдан кейін new SqlConnection жолына өз серверіңізге байланыс жолын көрсетесіз.

....

SqlConnection sqlConnection1 = new SqlConnection("Data Source=User-PC;Initial Catalog=studentter;Integrated Security=True");

...

Data Source – компьютер атауын білдіреді.

Initial Catalog – деректер қорының атауы.

Integrated Security – Windows арқылы немесе қолданушы арқылы қосылу командасын білдіреді.

Бағдарлама құрылымын түгелдей ұсынсақ келесі түрде болады.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Data.SqlClient;
using System.Data;
namespace WebApp_SQL.Controllers
{
   public class DatabaseController : Controller
   {
      // GET: Database
```

```
        public ActionResult Index()
        {
    SqlConnection sqlConnection1 = new SqlConnection("Data Source=User-
PC;Initial      Catalog=studentter;Integrated Security=True");
            SqlCommand cmd = new SqlCommand();
            SqlDataReader reader;
            cmd.CommandText = "SELECT * FROM Student";
            cmd.CommandType = CommandType.Text;
            cmd.Connection = sqlConnection1;
            sqlConnection1.Open();
            reader = cmd.ExecuteReader();
            sqlConnection1.Close();
            return View();
        } } }
```

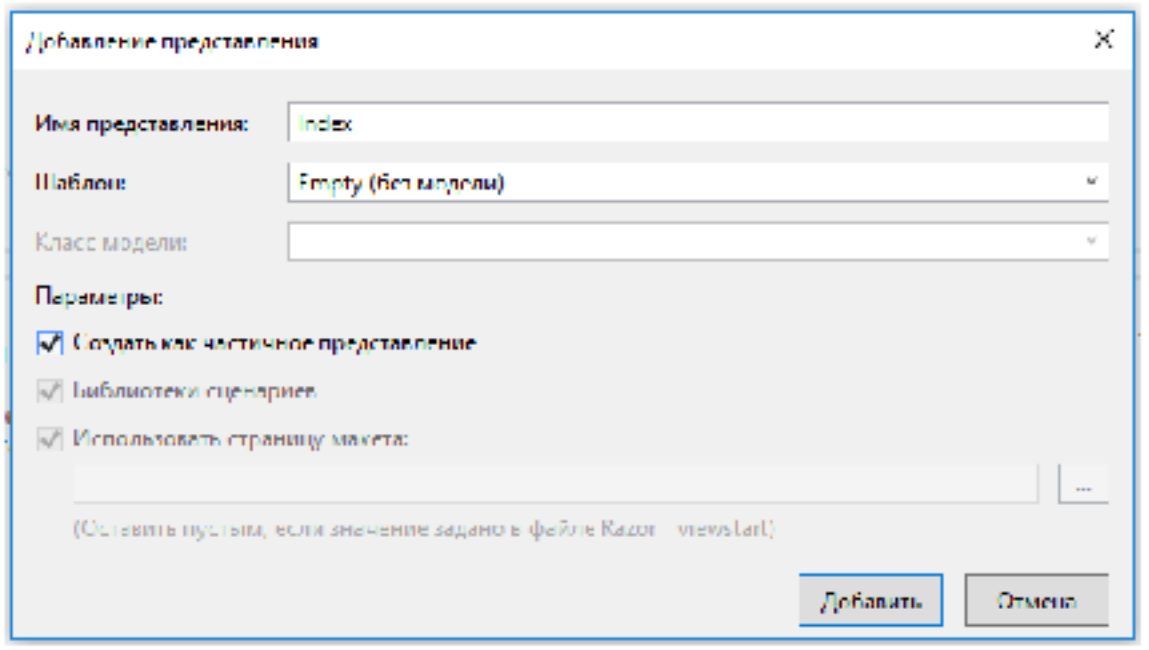
Нысанға-бағытталған бағдарламалау тілімен айтқанда sqlConnection1 жаңа нысан болып табылады. SqlConnection серверге қатынау жолын көрсетеді. Бұл кезде сервер және деректер қоры атауы, қолданушы параметрлері түріндегі деректер сақталады. Қашықтан қатынау қажет болған кезде, сервер орналасқан компьютер атауының орнына IP-адрес жазылады. SqlCommand типі cmd нысанын құрады. Мұнда деректер қорымен қандай да бір операция орындағымыз келгенде жазылады.

SqlDataReader типті reader нысаны деректерді оқу үшін қажет болады.

- cmd.CommandText = "SELECT * FROM Student"; //деректер қоры кестесінен деректерді оқиды.
- cmd.CommandType = CommandType.Text; // келесі жол біздің команданың мәтіндік тип екендігін білдіреді.
- cmd.Connection = sqlConnection1; // Connection қасиетін жоғарыда құрылған sqlConnection1 командасына береміз.
- sqlConnection1.Open(); // бұдан соң sqlConnection1 – ді ашамыз.
- reader = cmd.ExecuteReader();//Reader арқылы команданы орындаймыз.
- sqlConnection1.Close(); // sqlConnection1 – ді жабамыз.
- return View(); //Көрсетілімді қайтарамыз.

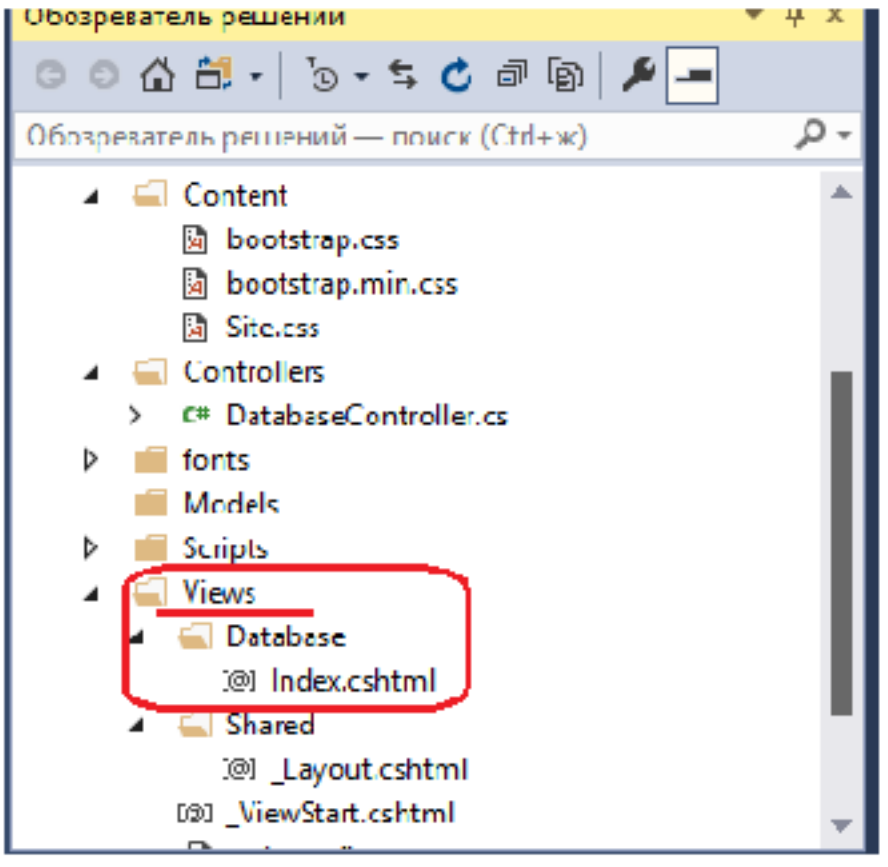
7.3.2 Views нысанын ішінара қолдану

Көрсетілім құру деректерді визуалды көрсету арналған бөлік болып табылады. Жоғарыдағы код қатесіз жүктелу үшін көрсетілім құру қажет болады. Ол үшін Index() әдісіне контексті мәзір шақырып *Добавить представление* командасын орындаймыз.



Сурет 8.18 – Көрсетілім құру терезесі

Ашылған терезеде *Создать как частичное представление* параметрін таңдаймыз. Бұдан кейін Views бумасында Database бумасы және Index файлы пайда болады.



Сурет 8.19 – Көрсетілім құрылымын қарау

Құрылған Index файлына келесі кодты жазамыз.

```
@{
    ViewBag.Title = "Index";
    }
<h2> Index </h2>
```

Әріқарай AppStart бумасында routeConfig.cs файлының кодына Controller-ге Database атауын көрсетеміз.

```
namespace WebApp_SQL
    {
        public class RouteConfig
        {
            public static void RegisterRoutes(RouteCollection routes)
            {
                routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

                routes.MapRoute(
                    name: "Default",
                    url: "{controller}/{action}/{id}",
                    defaults: new { controller = "Database", action = "Index", id =
UrlParameter.Optional } );
            }
        }
    }
```

Бағдарлама интерфейсін құру қадамдарына көшеміз. Ол үшін қосымша код енгізіледі. Мұнда өріс атауы жолдық массив ретінде алынады.

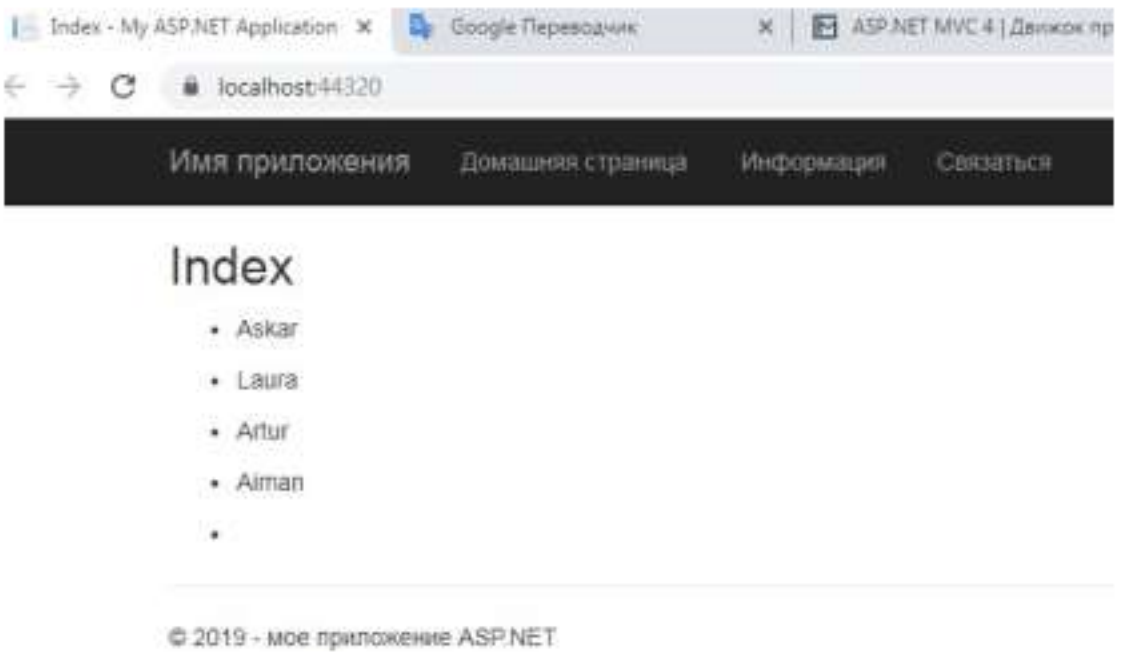
```
public class DatabaseController : Controller
    {
        // GET: Database
        public ActionResult Index()
        {
            int i = 0;
            string[] Aty = new string[5];

    SqlConnection sqlConnection1 = new SqlConnection("Data Source = User-PC;
Initial Catalog = studentter; Integrated Security = True;");
        SqlCommand cmd = new SqlCommand();
        SqlDataReader reader;
```

```
cmd.CommandText = "SELECT * FROM Student";
cmd.CommandType = System.Data.CommandType.Text;
cmd.Connection = sqlConnection1;
sqlConnection1.Open();
reader = cmd.ExecuteReader();
while (reader.Read())
   {
   IDataRecord record = (IDataRecord)reader;
         Aty[i] = record["Aty"].ToString();
         i++;
       }
       ViewBag.A = Aty;
      reader.Close();
      sqlConnection1.Close();
      return View(); }  }
}
```

Бұдан соң Көрсетілімдегі Index.cshtml файлына барып код құрылымын өзгертеміз.

```
@{
   ViewBag.Title = "Index";
}
<h2>Index</h2>
@foreach (var Aty in ViewBag.A)
{
   <ul><li>@Aty</li></ul>
   }
```

Razor – серверлік кодты web-беттерге ендіру синтаксисінен тұратын белгілеу.@ белгісі Razor визуализация механизмін білдіреді. Бұл арқылы браузерге жіберілетін динамикалық мазмұндағы нәтижеден тұратын ASP.NET контенті мен нұсқаулықтар өңделеді. Жоғарыдағы синтаксисте Razor белгісімен айқындалған код C# тілінің foreach циклы арқылы массив түрінде берілген кесте атрибуттарына қол жеткізуге мүмкіндік береді [21]. Бағдарлама нәтижесі келесі түрде болады.



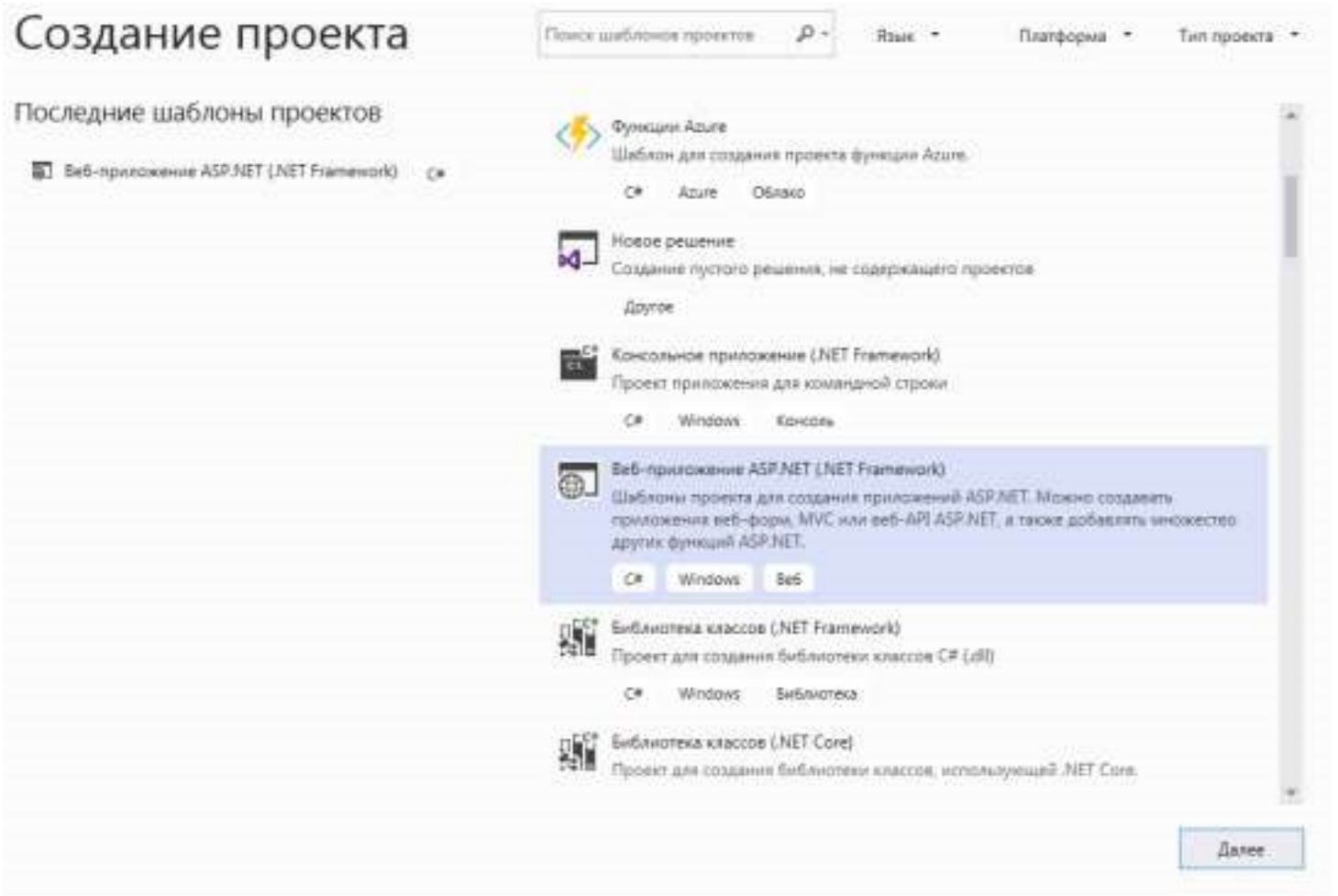
Сурет 8.20 – Web-қосымша түріндегі бағдаарлама нәтижесі

7.4 MVC жобасындағы модель және деректер қоры

Алдыңғы қадамда MS SQL ортасында құрылған Student кестесін бағдарламалық ортамен байланыстыру арқылы деректерді web-қосымша түрінде жарияланған болатынбыз.

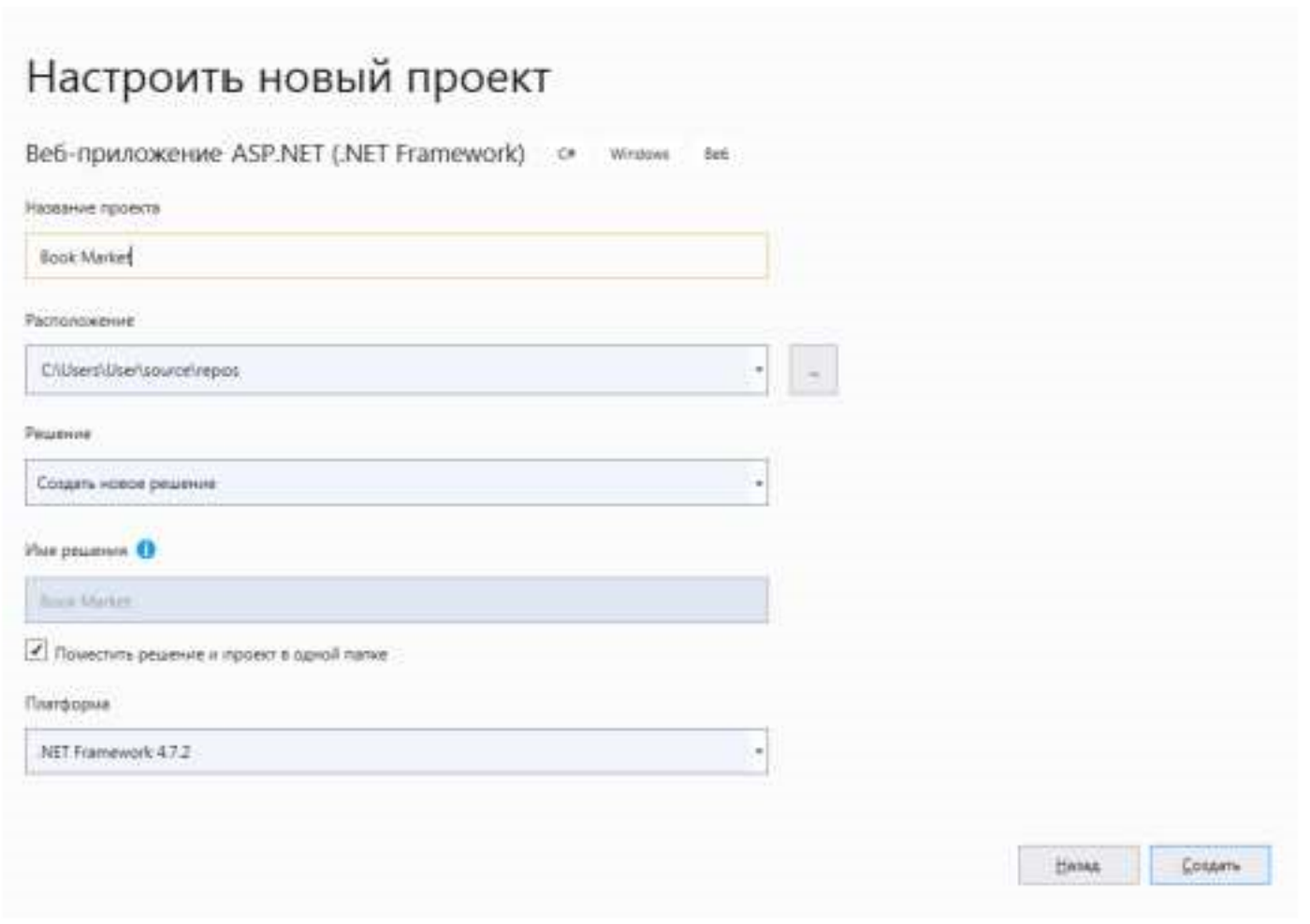
Бұл тақырыпта деректер қорының құрылымын модель нысаны арқылы бағдарламалық жолмен құруды жүзеге асырамыз. Мысал ретінде интернет дүкеннің жобасы қарастырылады.

ASP.NET MVC 5 жобасын құру үшін алдыңғы тапсырмадағыдай Visual Studio ортасында *Создание проекта* командасын орындап, ашылған терезеде *Web-приложение ASP.NET (.NET Framework)* жолын таңдаймыз.



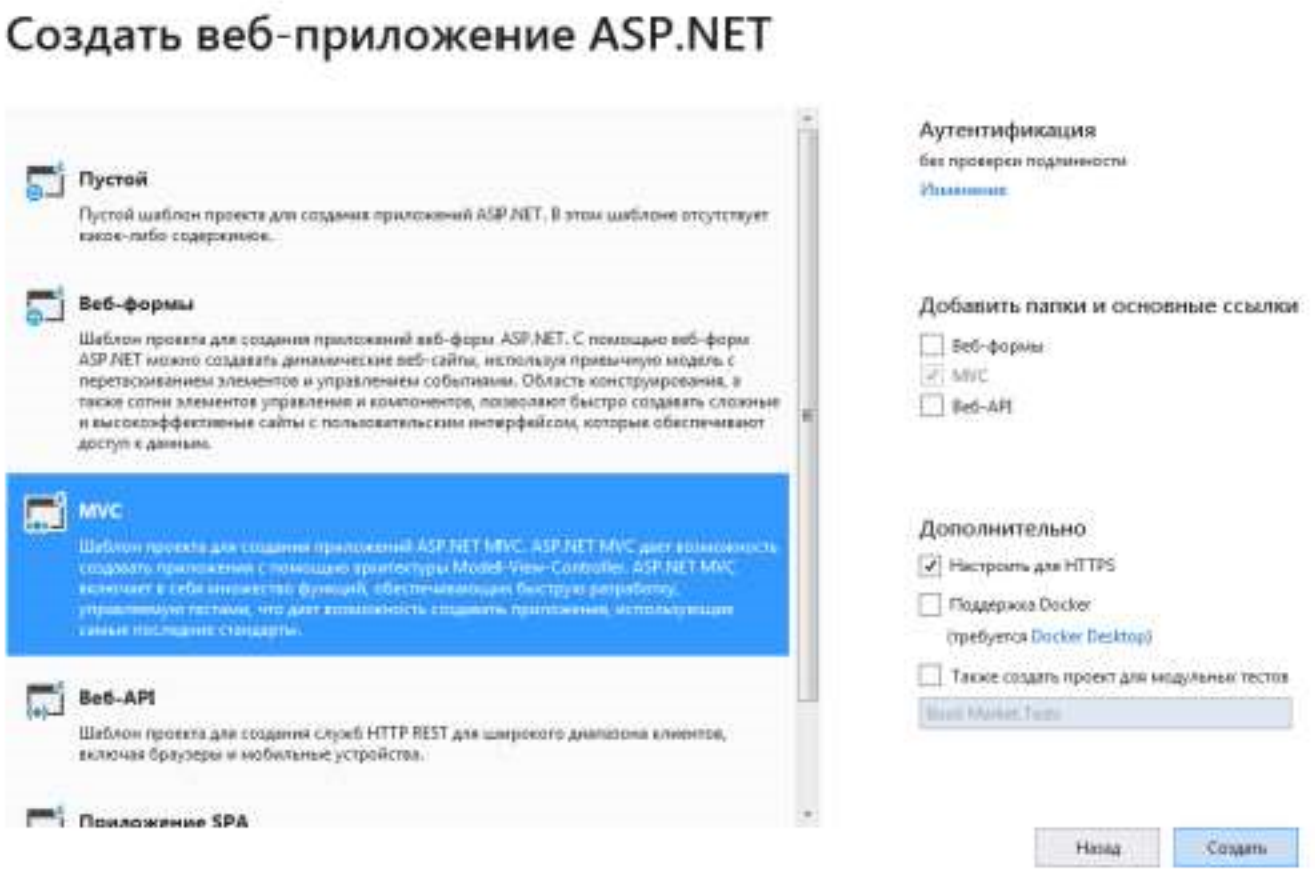
Сурет 8.21 – Интернет дүкен жобасын құру

Жобаның жаңа атауын және сақталатын буманың жолын көрсетеміз. Жаңа жобаны «Book Market» атауымен сақтаймыз.



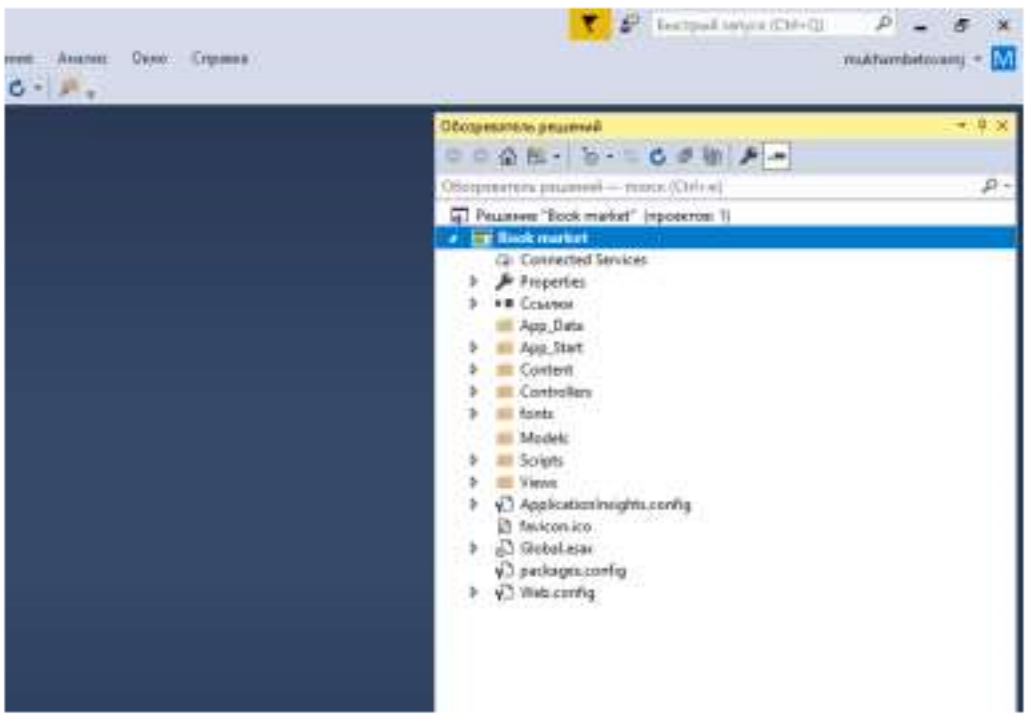
Сурет 8.22 – Book Market жобасының параметрлері

Жаңа ашылған терезеден MVC паттернінің белгісін таңдаймыз.



Сурет 8.23 – MVC паттернін таңдау

Жоба құрылымындағы нысандарды басқару үшін Вид - Обозреватель решений командасы арқылы аталған шешімдер шолушысын шақырамыз.

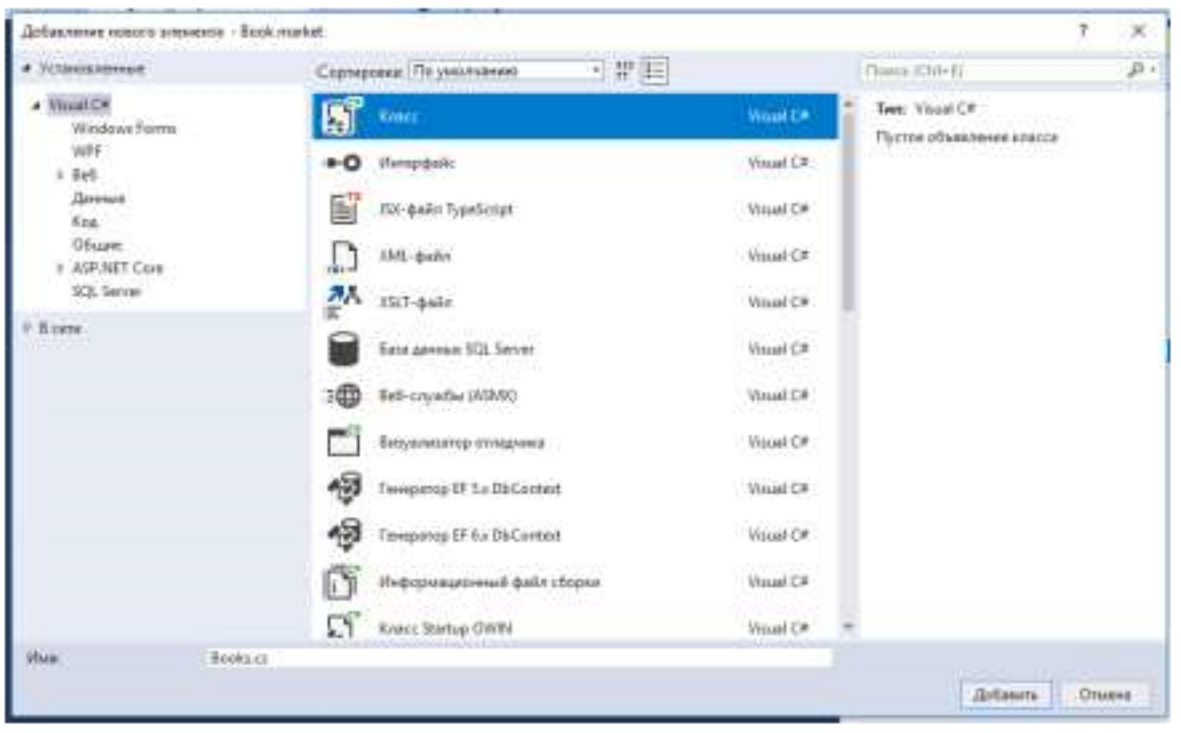


Сурет 8.24 – Жоба құрылымы

7.4.1 Models нысандарын құру

Жұмысымызды модель нысанын құрудан бастаймыз. Біздің жобада екі түрлі модель құрылады. Олар, тауарлар туралы деректерді сақтайтын (біздің жағдайда кітап тауарлары) Books және сатып алушыларға арналған Buy моделі.

Модель – бұл, класс болып табылады. Сол себепті аталған модельдері құру үшін Models бөлімінен контексті мәзірді шақыру арқылы Добавить – Класс командасын таңдаймыз. Алғашқы класс Books атауымен сақталады.



Сурет 8.25 - Модель нысанын құру үшін жаңа класс тағайындау

Жаңадан құрылған класс құрылымы:

```
namespace Book_market.Models /*Book_market шешіміндегі Models-те сақталған класс
    {
       public class Books
       {
       }
    }
```

Модель (model) - нысанның нақты қасиеттерін сипаттайды. Барлық модельдерге ортақ қасиет ретінде қолданылатын идентификатор болады. Оны модель атының Id қасиеті немесе жәй Id ретінде атауға болады.

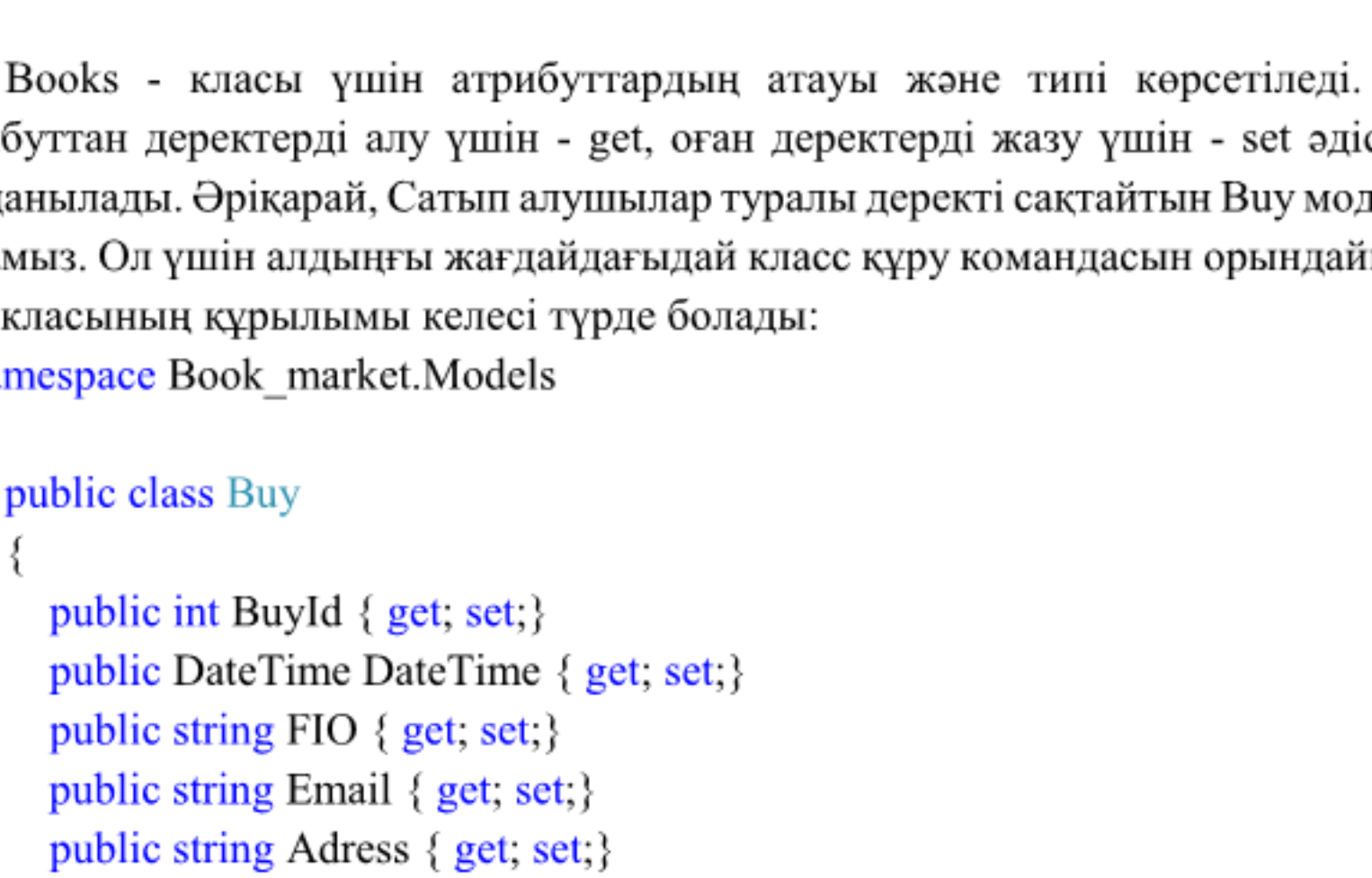
```
namespace Book_market.Models
    {
       public class Books
       {
          public int Id { get; set; }
          public string Name { get; set; }
          public string Publisher { get; set; }
          public int Year { get; set;}     } }
```

Books - класы үшін атрибуттардың атауы және типі көрсетіледі. Ал, атрибуттан деректерді алу үшін - get, оған деректерді жазу үшін - set әдістері қолданылады. Әріқарай, Сатып алушылар туралы деректі сақтайтын Buy моделін құрамыз. Ол үшін алдыңғы жағдайдағыдай класс құру командасын орындаймыз. Buy класының құрылымы келесі түрде болады:

```
namespace Book_market.Models
{
    public class Buy
    {
        public int BuyId { get; set;}
        public DateTime DateTime { get; set;}
        public string FIO { get; set;}
        public string Email { get; set;}
        public string Adress { get; set;}
    }
}
```

Жоба құруға қажетті тағы бір жаңа ұғым «деректер контексті» деп аталады. Деректер контексті – деректер қорында сақталатын деректерге рұқсат алу үшін қолданылатын делдал қызметін атқаратын класс. Бұл арқылы деректер қорына тікелей қатынамай – ақ, онымен жұмыс жасау мүмкіндігіне ие боламыз.

Деректер контексті үшін BookContext атауымен жаңа класс құрамыз.

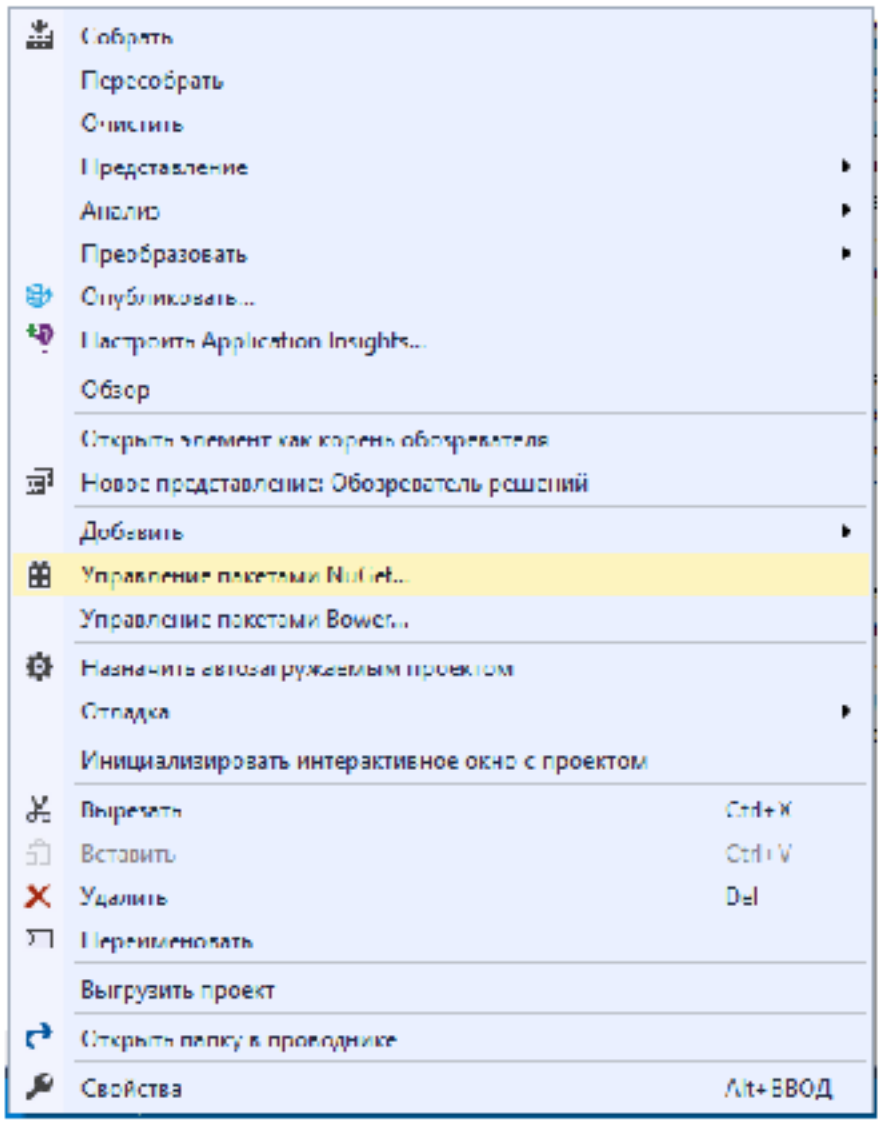


Сурет 8.26 – Деректер контекстін құру

Ескертпе. Деректер контексті үшін файл атауы DBContext мұрагерлік принципі бойынша сақталуы қажет. Яғни, қандай деректер қорын құрсаңыз, сол атауға Context сөзін тіркеп файлды сақтаймыз.

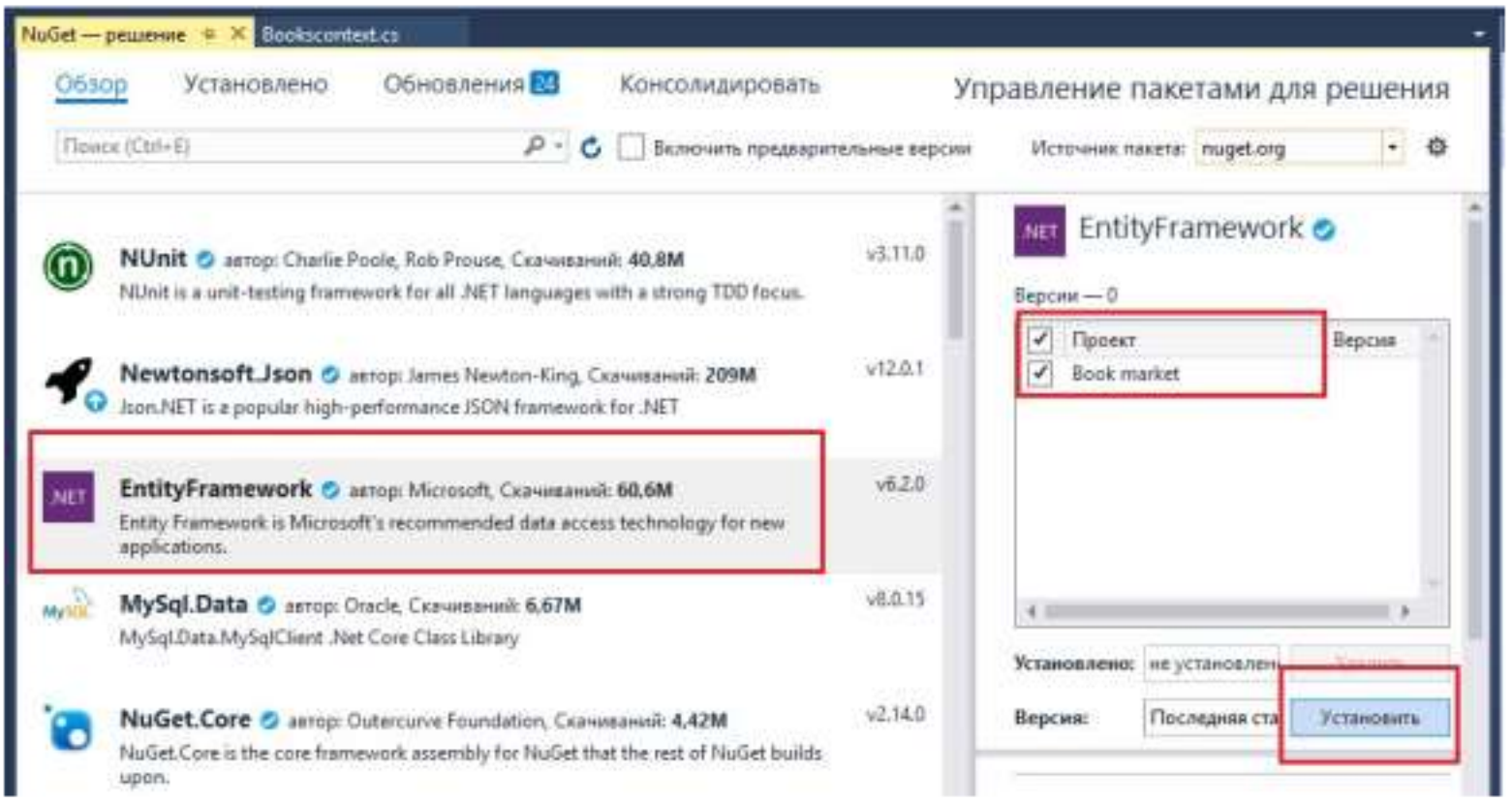
DBContext класының қызметі жүзеге асу үшін жобада Entity Framework-ты орнату қажет.

Entity Framework-ты ашу үшін Book Market шешімінен контексті мәзірді шақырып, Управление пакетами NuGet командасын таңдаймыз.



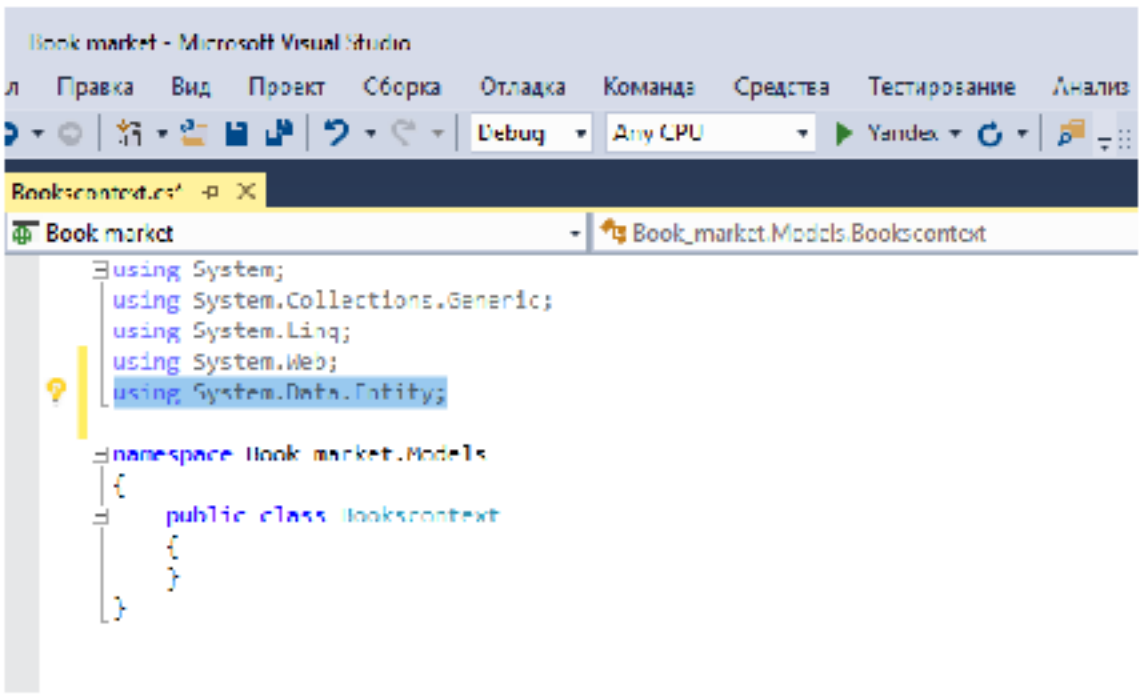
Сурет 8.27 – NuGet пакетін ашу

Обзор бөліміне өтіп, тізімнен Entity Framework-ты таңдап, терезенің оң жақ бөлігінен жоба атауына белгі қойып, Установить батырмасына шертеміз.



Сурет 8.28 – Entity Framework-ты жобаға орнату

Шарттарға келісу жауабын таңдап, фреймворкты орнатамыз. Фреймворк орнатылғаннан кейін, деректер қоры нысандарымен жұмыс жасау үшін атаулар кеңістігіне using System.Data.Entity жолын қосамыз.



Сурет 8.29 - Атаулар кеңістігіндегі Entity Framework

DbContext-ке мұрагерлік принципі бойынша BookContext нысанын меншіктейміз, ол екі ашық өрістен тұрады. Мұндағы, DbSet - деректер жиыны мағынасын білдіреді.

```
namespace Book_market.Models
{
```

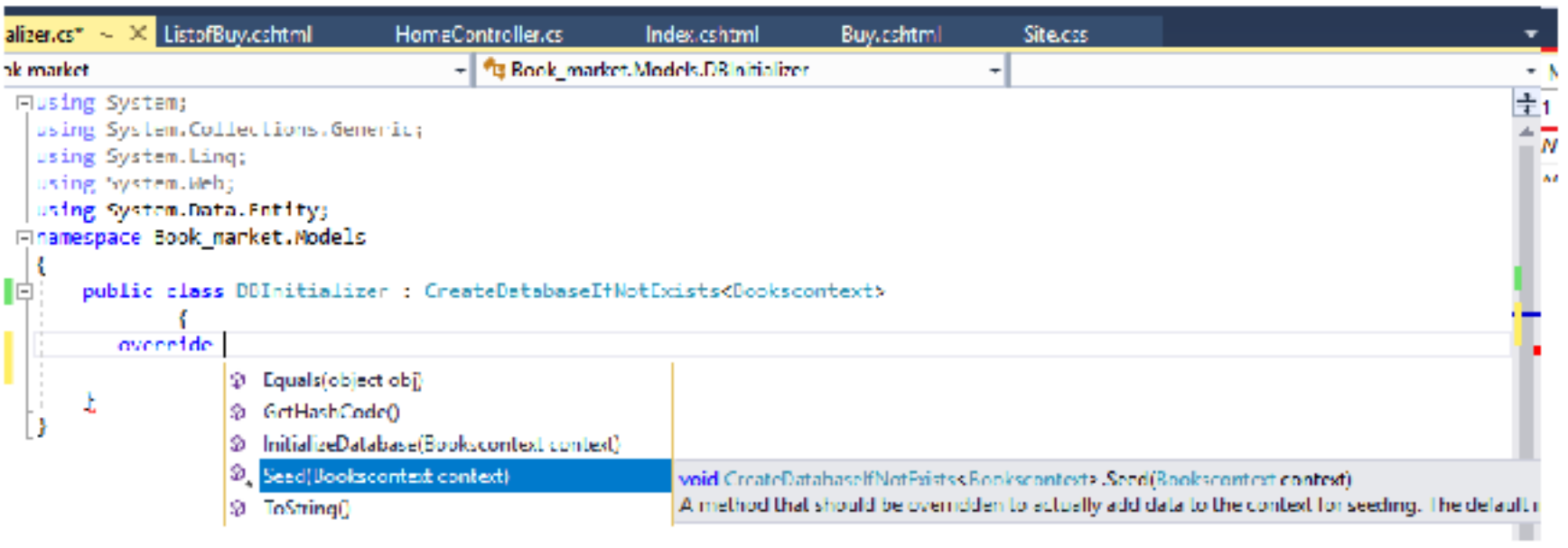
```
public class BookContext : DbContext
{ public DbSet<Books> Books { get; set; }
   public DbSet<Buy> Buys { get; set; }     }  }
```

Деректер контекстіне арналған класс құрылымын талдайтын болсақ, BookContext –ті деректер қорын, ал Books және Buys осы деректер қорының кестелерін білдіреді. Әр кесте үшін құрған класс, осы кестелердің өрістері түрінде сипатталған. Бұдан, контекст - деректер класы, деректер қорын сипаттайтын класс деп атай аламыз.

Деректер қорымен байланысты ұйымдастыру үшін, DBInitializer атауымен жаңа класс құрамыз. Бұл класты CreateDatabaseIfNotExists класына мұрагерлік қасиеті бойынша тағайындаймыз. CreateDatabaseIfNotExists – егер, деректер қоры болмаса оны жаңадан құру қызметін орындайды.

```
…
public class DBInitializer : CreateDatabaseIfNotExists<Bookscontext>
…
```

Осы класта override әдісін шақырып, осы жолда *Ctrl+Space (бос жол)* пернелер комбинациясына шертіп, Seed қасиетін таңдаймыз.



Сурет 8.30 – Деректер қорымен жұмыс жасаушы әдісті шақыру

Бұл кезде келесі түрдегі бағдарлама коды автоматты түрде генерацияланады:

```
.....
using System.Data.Entity;
namespace Book_market.Models
{
   public class DBInitializer : DropCreateDatabaseAlways<Bookscontext>
```

```
        {
    protected override void Seed(Bookscontext context)
    {
        base.Seed(context);
    }
}}
```

Бұл код бізге context типті деректер қорымен жұмыс жасауға мүмкіндік береді.

Бұдан кейін Web.config файлын ашып, деректер қорына қосылу жолын көрсетеміз.

Деректер қорына қосылу коды келесідей:

```
<connectionStrings>
    <add name="BookContext" connectionString="Data
Source=.\SQLEXPRESS;Initial     Catalog=Bookstore;Integrated Security=True"
    providerName="System.Data.SqlClient"/>
    </connectionStrings>
```

Бағдарлама коды </appSettings> жолына дейін орналасады.

Сурет 8.31 – Web.config файлында деректер қорына қосылу кодын енгізу

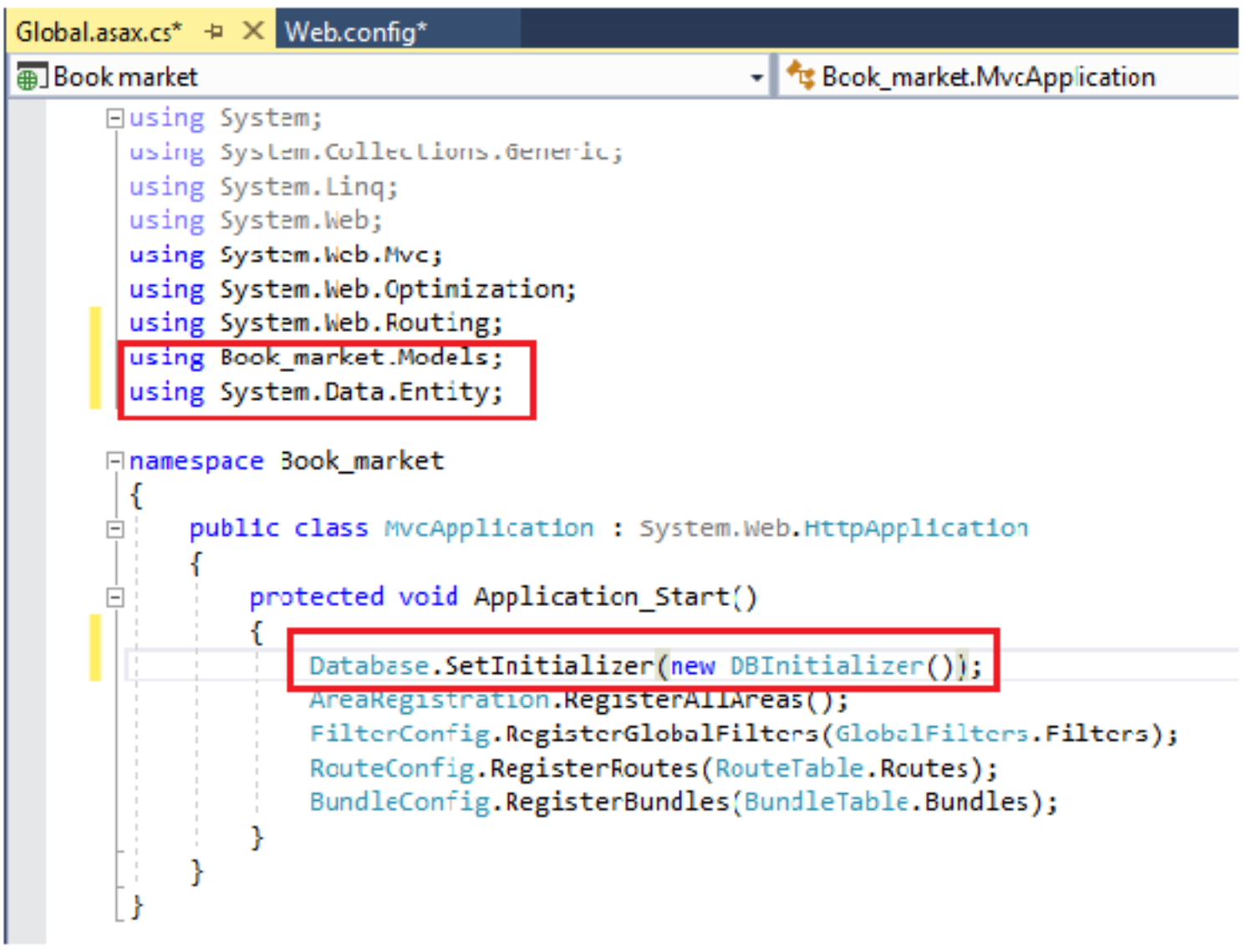
.\SQLEXPRESS – түрінде жазу арқылы сервердің нақты атын көрсетпей осы компьютердегі SQLEXPRESS-тің кез-келген экземплярымен байланыс жасай аламыз.

Әріқарай, осы деректерді клиенттік қосымшада шығару үшін қажетті кодты жазамыз. Ол үшін Global.asax файлын ашып, келесі атаулар кеңістігін бағдарлама кодына қосамыз:

```
...
using Book_market.Models;
using System.Data.Entity;
...
```

Бағдарлама денесінде DBInitializer() класын шақыруға арналған програма кодын жазамыз:

```
Database.SetInitializer(new DBInitializer());
```



Сурет 8.32 – Global.asax файлына деректер қорына жұмысқа қажетті жолдарды қосу

Орындалған қадамдар арқылы клиенттік қосымшада көрсетілетін деректер қорын даярладық.

Әріқарай, клиентті қосымша жасауға қажетті жұмыстарды орындауға көшеміз. Яғни, Controllers және Views нысандарын құрумен айналысамыз.

7.4.2 Деректер қорының миграциясы

Алдыңғы тақырыпта деректер қорының қызметін атқаратын модель нысандарын құрған болатынбыз. Біздің жағдайда Bookstore деректер қорында Books және Buys кестелері құрылды. Көп жағдайда құрылған модель құрылымын өзгерту қажеттілігі туындайды. Алайда бізде белгілі-бір деректер сақталған деректер қоры бар және онымен байланысты контроллер мен көрсетілім нысандары құрылған. Мұндай жағдайда деректер қорын шығынсыз жаңарту үшін ASP.NET MVC миграция механизмін ұсынады. Мысалы, жоғарыда құрылған Books кестесіне Author және Price жаңа өрістерін қосу қажет.

```
public string Author { get; set;}
public int Price { get; set; }
```

Сонымен бірге Buys кестесіне BookId өрісін қосамыз. Алдағы уақытта бұл өріс әр кітапқа сұраныстар жасау үшін қажет болады.

```
public int BookId { get; set; }
```

Егер модель нысанына және көрсетілім үшін бұл өзгерістерді енгізіп, бағдарламаны орындауға жіберсек, қате туралы хабарламаны көреміз.

Сол себепті, миграция механизмін іске қосамыз. Ол үшін *Вид – Другие окна – Консоль диспетчера пакетов* командасы арқылы консоль тереезесін ашамыз. Бағдарламаның төменгі бөлігінде ашылған терезеде enable-migrations командасын енгізіп Enter батырмасына шертеміз.

Сурет 8.33 – Миграция механизмін іске қосу

Бұдан кейін жоба құрылымында пайда болған Migrations бумасында Configuration.cs файлын көруге болады. Бұл файл конфигурациялық баптауларды тағайындайтын Configuration класын хабарлайды.

```
namespace Book_Market.Migrations
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Migrations;
```

```
using System.Linq;
internal sealed class Configuration :
   DbMigrationsConfiguration<Book_Market.Models.BookContext>
{
   public Configuration()
   {
      AutomaticMigrationsEnabled = false;
      ContextKey = "Book_Market.Models.BookContext";
   }
   protected override void Seed(Book_Market.Models.BookContext context)
   {     }    } }
```

Seed әдісінде бастапқы деректер сақталған деректер қорын инициализациялауға болады.

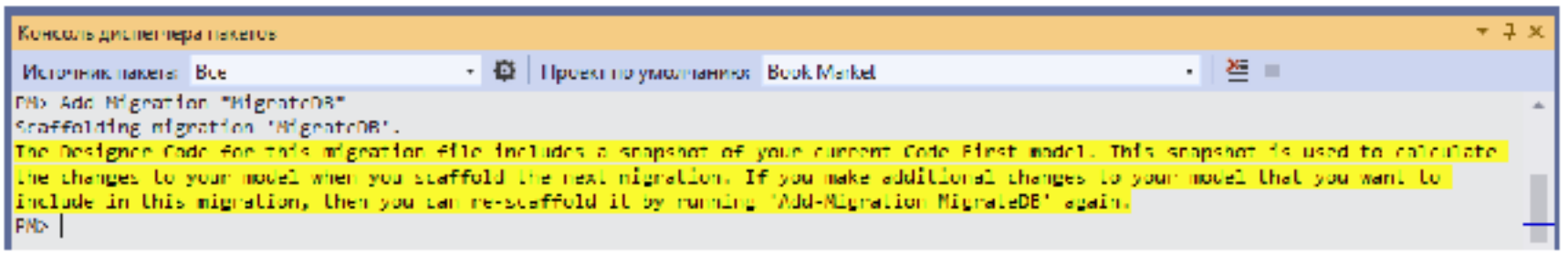
Енді бізге миграцияның өзін құру қажет. Ол үшін Консоль диспетчера пакетов терезесінде келесі команданы енгіземіз:

```
PM> Add-Migration "MigrateDB"
```

Бұдан кейін Visual Studio автоматты түрде миграция класын генерациялайды:

```
namespace Book_Market.Migrations
{
   using System;
   using System.Data.Entity.Migrations;
    public partial class MigrateDB : DbMigration
    {
      public override void Up()
      {
         AddColumn("dbo.Books", "Author", c => c.String());
         AddColumn("dbo.Books", "Price", c => c.Int(nullable: false));
         AddColumn("dbo.Buys", "BookId", c => c.Int(nullable: false));
      }

      public override void Down()
      {
         DropColumn("dbo.Buys", "BookId");
         DropColumn("dbo.Books", "Price");
         DropColumn("dbo.Books", "Author");        } } }
```

Консоль терезесінде миграция арқылы модель құрылымына өзгерістер енгізілгені туралы хабарлама шығады.



Сурет 8.34 – Миграция механизмі арқылы модель құрылымына өзгерістер енгізілгені туралы хабарлама

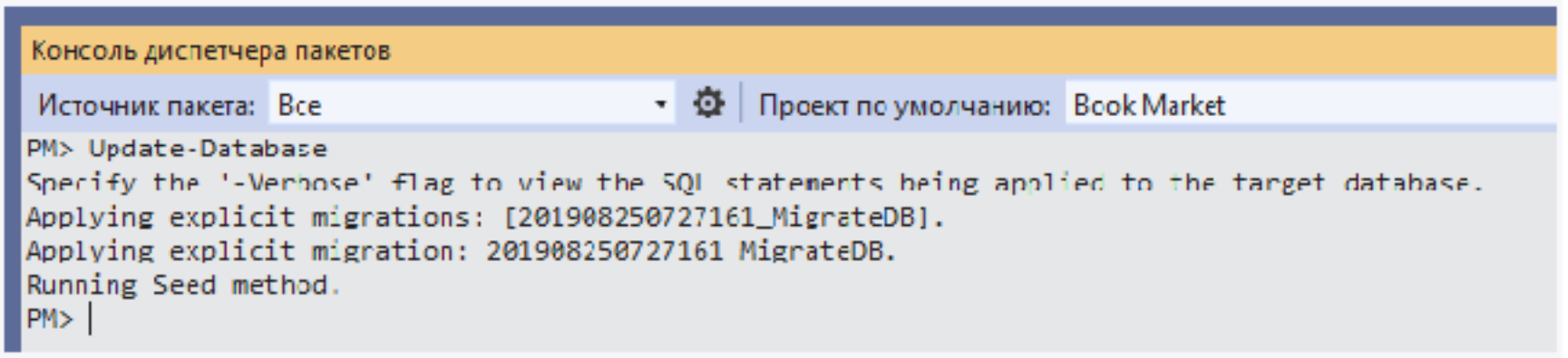
Up әдісінде AddColumn әдісін шақыру арқылы Books Author, Price және Buys кестесіне BookId өрістері енгізіледі.

Down әдісіндегі DropColumn әдісі егер мұндай өрістер болған жағдайда жояды.

Бұдан соң, миграцияны орындау үшін бұл класты қолданып консольда келесі команданы орындаймыз:

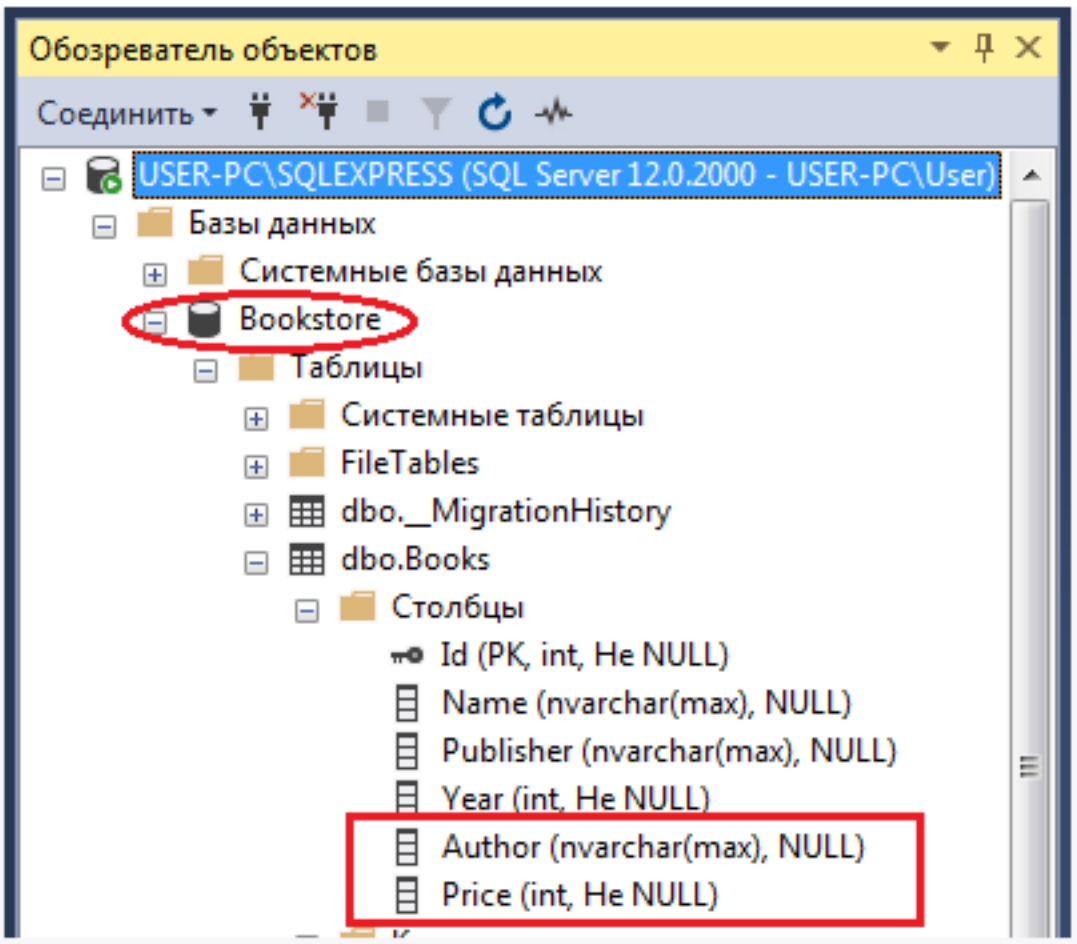
```
PM> Update-Database
```

Команда орындалғаннан кейін консоль терезесінде миграцияның қолданылғаны туралы хабарлама шығады.



Сурет 8.35 – Миграцияның қолданылғаны туралы хабарлама

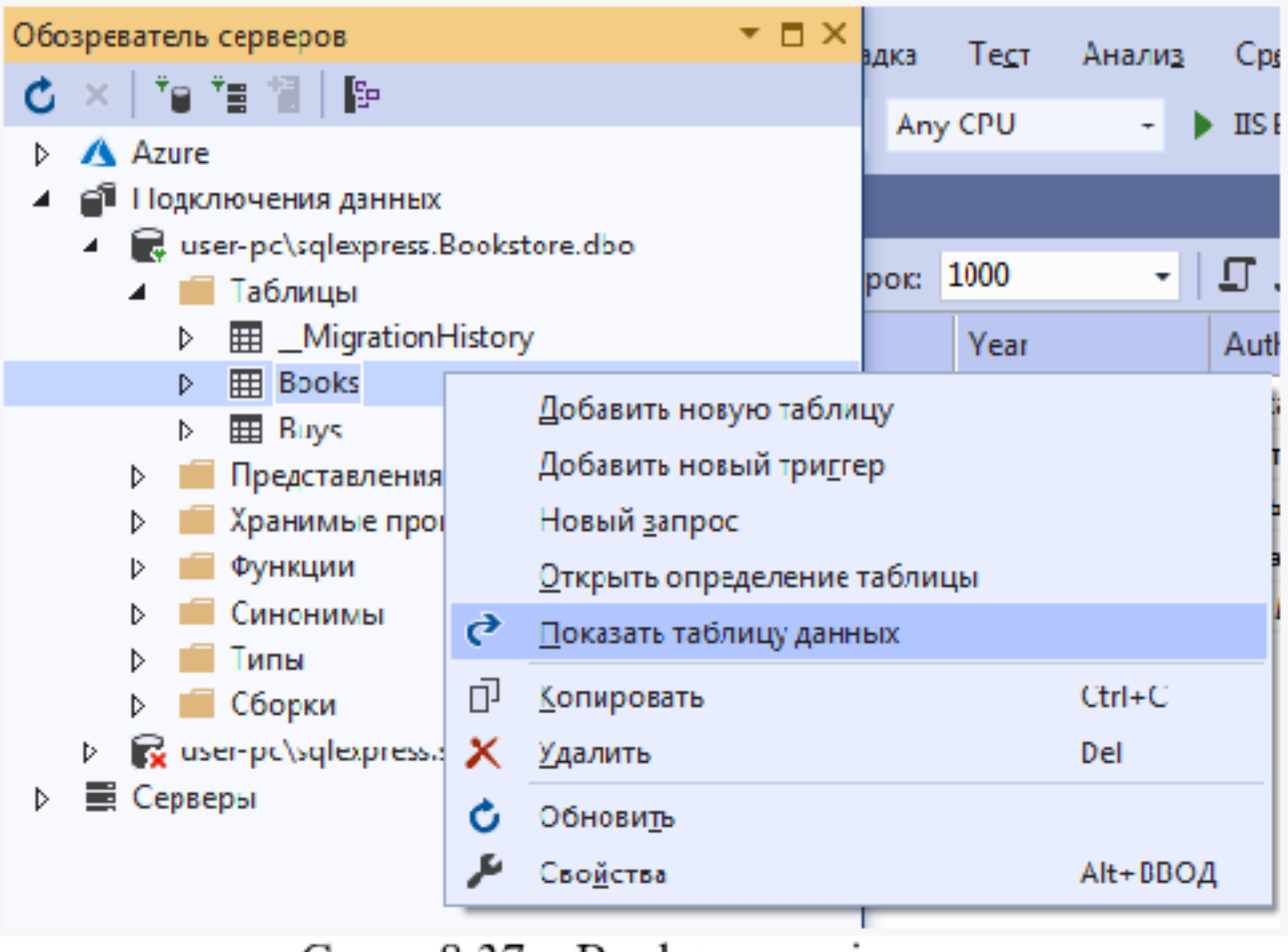
Өзгеріс нәтижесін тексеру үшін SSMS арқылы Bookstore деректер қорын ашамыз. Айтылған өрістердің қосылғанына көз жеткіземіз.



Сурет 8.36 – Миграция нәтижесін деректер қорынан тексеру

Сонымен, миграция орындалды, әріқарай жаңартылған модель мен деректер контекстін өз жобамызға қолдана аламыз.

Миграция механизмі сәтті орындалғанын тексергеннен кейін Bookstore деректер қорын жазбалармен толтырамыз. Ол үшін *Обозреватель серверов* терезесіне жоғарыда сипатталғандай деректер қорын қосып, *Показать таблицу данных* командасы арқылы кесте құрылымын ашамыз.



Сурет 8.37 – Books кестесін ашу

Кестені қажетті деректермен толтырамыз.

7.4.3 Controller құру

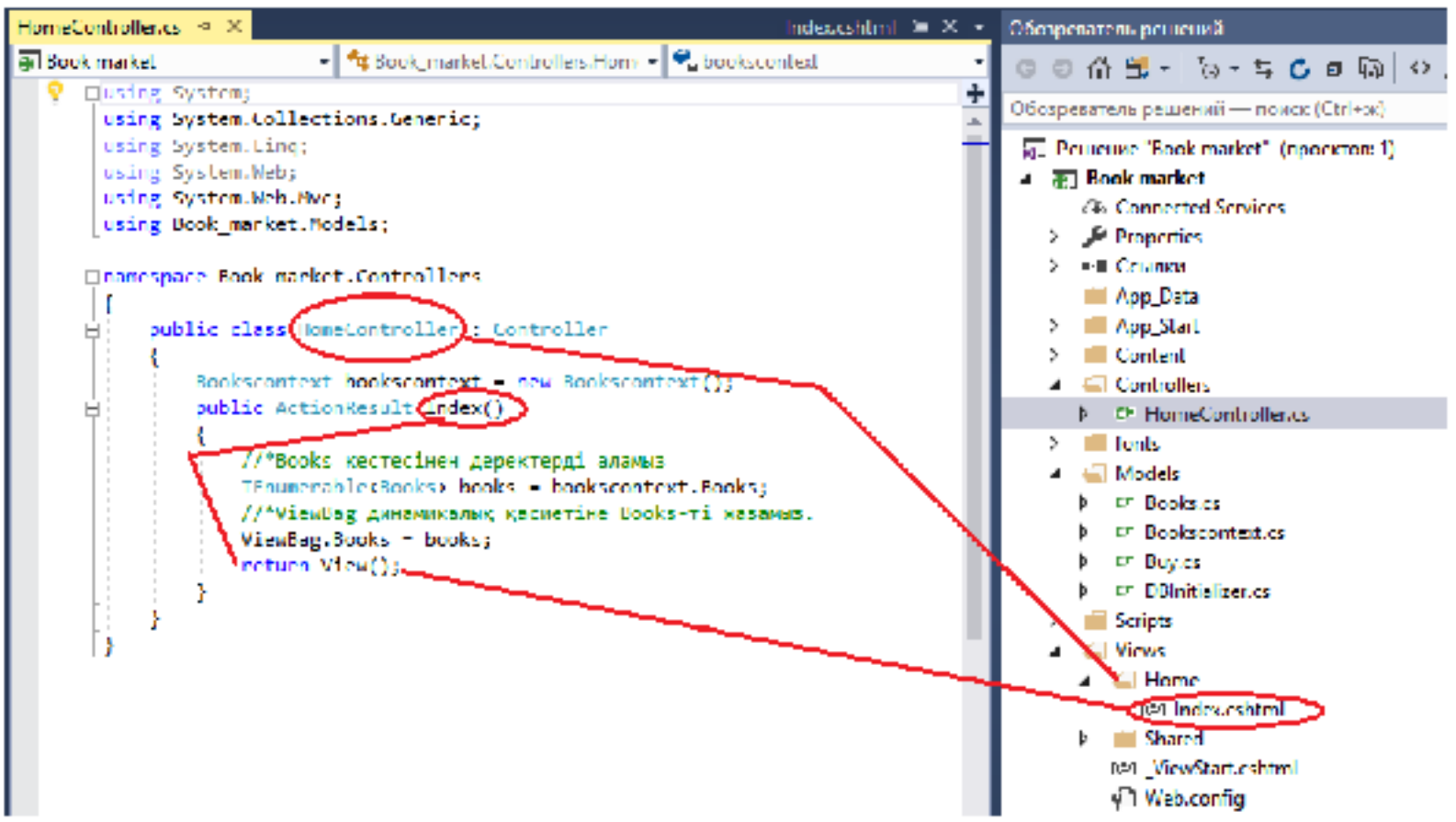
Шешімдер терезесінен Controllers бумасын ашып, ондағы HomeController.cs файлын ашамыз (бағдарлама құрылымына талдау қосымша түсініктеме ретінде келтірілген).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;//* Бұл тізім бағдарламаға қосылған сборкалар
тізімін көрсетеді.
namespace Book_market.Controllers //* атаулар кеңістігі
{
   public class HomeController : Controller //* Контроллердің базалық класы
   {
public ActionResult Index() //* Index(),About(),Contact() әдістер
      {
   return View();
}
public ActionResult About()
{
   ViewBag.Message = "Your application description page.";
   return View();
}
public ActionResult Contact()
{
   ViewBag.Message = "Your contact page.";
   return View();
}   } }
```

Controller класында көрсетілген үш әдістің ішінен, осы жолы тек Index() әдісімен жұмыс жасаймыз. Сол үшін, бағдарлама кодынан About() және Contact() әдістерін жоямыз.

Views бумасын ашатын болсақ, HomeController атауына сәйкес Home бумасын көреміз. Home бумасы контроллердегі әдістерге сәйкес үш файлдан тұрады. About() және Contact() әдістерін бағдарламадан алып тастауымызға

байланысты, бұл бумадағы About.cshtml және Contact.cshtml файлдарын да жоямыз. Контроллердің жұмысы үшін қажетті бағдарлама коды келесі түрде болады.



Сурет 8.38 – Контроллер файлын құру

Атаулар кеңістігіне using Book_market.Models – ті қосамыз. Бұдан соң, Bookscontext типті өріс қосамыз. Bookscontext класын құрған кезде, осы класпен жұмыс жасауға арналған экземплярлармен жұмыс жасауға мүмкіндік аламыз.

Public ActionResult Index() әдісі ешқандай параметрлерді қабылдамайды. ActionResult типіндегі деректерді қайтарады. Әдіс атауы Index(), қолжетімділік модификаторы public.

Деректерді алу үшін IEnumerable интерфейсі қолданылады. Яғни, деректер қорынан кітаптар туралы деректер алынады.

ViewBag - динамикалық нысан. Динамикалық нысанда кез-келген қасиет құра аламыз және оған кез-келген мән тағайындай аламыз.

Return View() – әдісті қайтарады.

Сурет 8.38-де көрсетілгендей әдіс атауы көрсетілімдегі файлмен сәйкес келеді. Контроллерде генерацияланған кодты көрсетілім HTML код түрінде қолданушыларға ұсынады.

7.4.4 View нысандарымен жұмыс

View нысандарымен жұмыс жасау үшін Views бумасынан Index.cshtml файлын ашамыз.

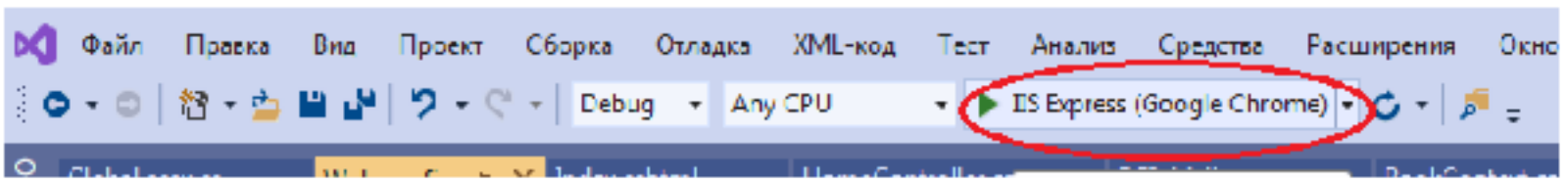
Index файлын тазалап, келесі түрдегі кодты енгіземіз:

```
@{
    Layout = null; //*беттер шебері қолданылмайды
    }
<!DOCTYPE html>
<html>
<head>
    <title>
        Index
    </title>
</head>
<body>
    <h3>Books market интернет дүкенінің тауарлары</h3>
    <br />
    <table>
        <thead>
            <tr>
            <td>Коды</td>
        <td>Кітап атауы</td>
        <td>Автор</td>
        <td>Құны</td>
        <td>Баспасы</td>
        <td>Шыққан жылы</td>
        <td> </td>
    </tr>
</thead>
<tbody>
    @{ foreach (var books in ViewBag.Books)
        {
            <tr>
                <td>@books.Id</td>
                <td>@books.Name</td>
                <td>@books.Author</td>
                <td>@books.Price</td>
                <td>@books.Publisher</td>
                <td>@books.Year</td>
                <td><a href="#">Сатып алу</a></td>
            </tr>   } }
```

```
        </tbody>
    </table>
</body>
</html>
```

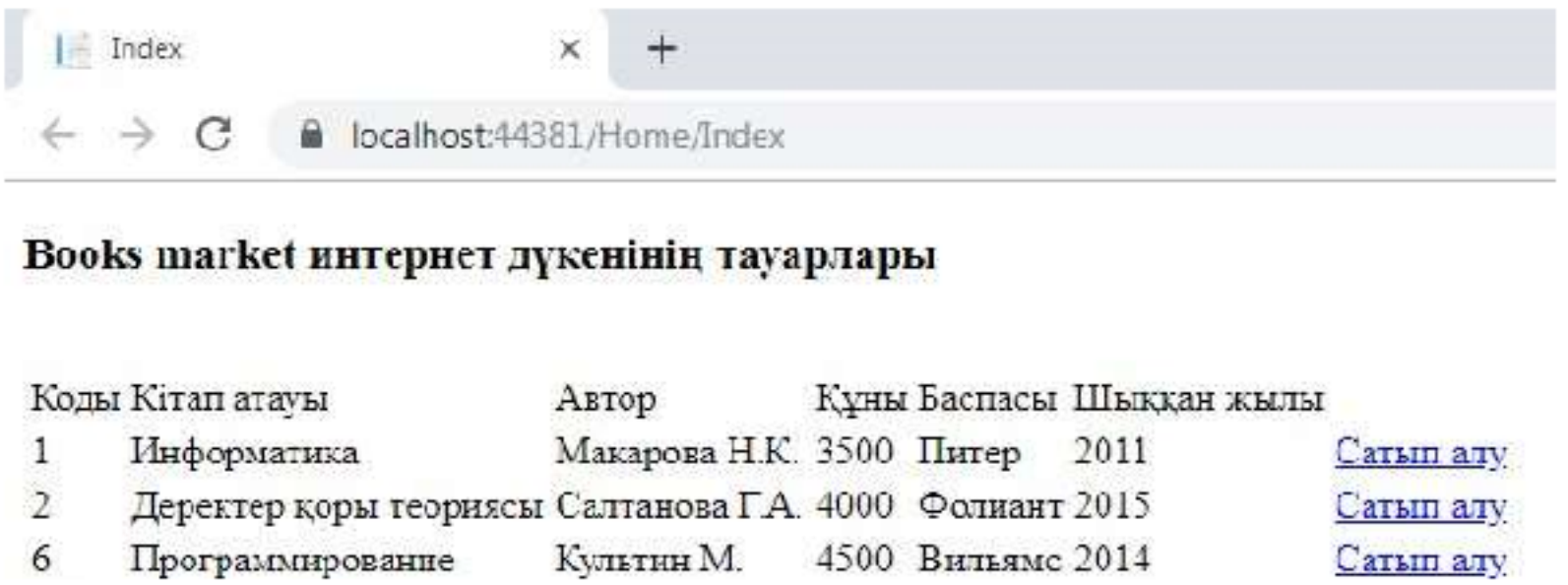
Бұл бағдарламалық кодтағы @ Razor белгісі болып табылады. Оның көмегімен HTML кодқа C# коды орнатылады.

Қосымшаны орындауға жіберу үшін құрал-саймандар панеліндегі IIS Express атауы көрсетілген орындау батырмасына шертеміз. Нәтиже Google Chrome браузері арқылы ашылады.



Сурет 8.39 – Қосымшаны орындауға жіберу батырмасы

Деректермен толтырылған web-клиенттік қосымша келесі түрде болады.



Сурет 8.40 – Интернет дүкеннің тауарлар тізімі

Әріқарай, *Сатып алу* батырмасына шерткен кезде, қажетті деректерді толтыру формасын жасауға көшейік. Ол үшін HomeController.cs файлын ашамыз.

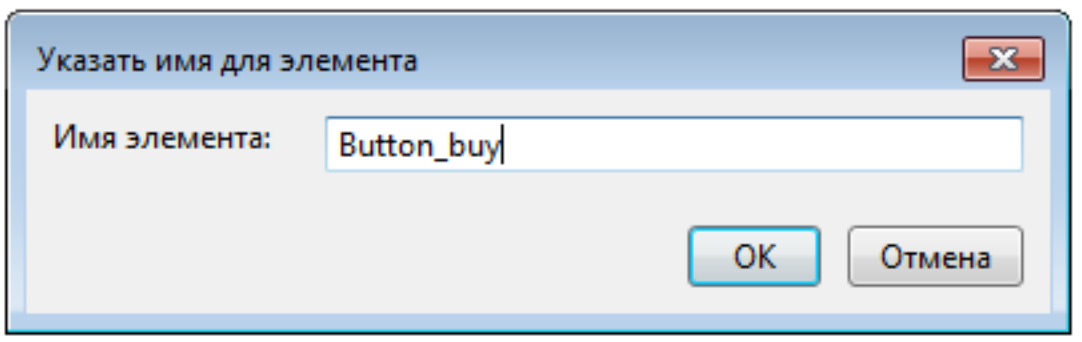
Деректерді жіберу және қабылдау үшін get, post сұраныстары жазылады. Сол арқылы http сұранысты қажетті бетке қайтару жұмысын орындайды.

```
Book Market                                   Book_Market.Controllers.HomeController
12          BookContext BookContext = new BookContext();
            0 references
13          public ActionResult Index()
14          {
15              IEnumerable<Books> books = BookContext.Books;
16              ViewBag.Books = books;
17              return View();
18          }
19          [HttpGet]
            0 references
20          public ActionResult Button_buy(int Id)
21          {
22              ViewBag.Id = Id;
23              return View();
24
25          }
26
```

Сурет 8.41 – HTTP Get сұраныстарын іске қосу

ActionResult әдісін шақырып оған Button_Buy атауын береміз. Жоғарыда айтылғандай контроллердегі бұл әдіс атауымен бірдей жаңа көрсетілім, яғни View нысаны құратынымызды естен шығармаңыз. Button_Buy әдісіне тағайындалған Id параметрі әр кітапқа жасалатын сұраныстарды айқындаушы қызметін атқарады.

Button_Buy көрсетілімін құру үшін Views–Home жолынан контексті мәзірді орындау арқылы, *Добавить – Страница представления MVC 5 (Razor)* командасын орындаймыз.



Сурет 8.42 – Buy көрсетілімін құру

Көрсетілім үшін тапсырыс берушіге форма құруға арналған бағдарлама кодын жазамыз.

Сурет 8.43 – Тапсырыс беруші формасын құру

Тапсырыс беруші формасын (сурет 8.43) құру барысында <form> әдісі post сұраныстары үшін құрылады. Input элементінде BookId нысанын құрып, оның қасиетін hidden етіп тағайындаймыз, яғни жасырып қоямыз. Алайда, бұл өріс бағдарламадағы шешуші рөлді атқарады, ол арқылы сұраныстар қай кітапқа берілгенін айқындаймыз. <Table> атрибуты арқылы тапсырыс беруші туралы деректерді енгізуге арналған өрістер құрылады. Мұндағы кесте өрістерін құрайтын бағдарлама үзіндісін талдайтын болсақ, алғашқы жол статикалық түрдегі кез-келген мәтін бола алады, ал екінші жолдағы name атауы Buy класы арқылы құрылған кестенің өріс аттарына сәйкес келу қажет. Себебі, бұл енгізілген деректер осы кестеге жазылатын болады.

```
<tr>
        <td><p>Аты-жөніңізді енгізіңіз :</p></td>

                <td><input type="text" name="FIO" /> </td>
```

```
</tr>
```

Бұдан соң HomeController.cs файлын қайтадан ашып, контроллерде деректерді қабылдауға арналған HttpPost сұраныстарын өңдеу кодын жазамыз. Контроллердің толық түрдегі коды келесі түрде болады:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Book_Market.Models;

namespace Book_Market.Controllers
{
   public class HomeController : Controller
   {
      BookContext BookContext = new BookContext();
      public ActionResult Index()
      {
      IEnumerable<Books> books = BookContext.Books;
      ViewBag.Books = books;
      return View();
}
[HttpGet]
public ActionResult Button_buy(int Id)
{
   ViewBag.Id = Id;
   return View();        }

[HttpPost]
   public String Button_buy(Buy buy)
{
   buy.DateTime = DateTime.Now;
   BookContext.Buys.Add(buy);
   BookContext.SaveChanges();
   return $"Құрметті, {buy.FIO}, сізбен жақын уақытта хабарласады!";

      } } }
```

[HttpPost] сұранысы жоғарыда құрылған форманы қажетті әдіске жіберу барысында орындалады. Бұл post сұраныста әдіс атауын көрсеткен жоқпыз:

```
public String Button_buy(Buy buy)
```

Себебі, Button_buy атауы көрсетілім атауына сәйкес келеді. Сол арқылы инициализация өз ретімен орындалады.

Бұдан соң, Buys кестесіне деректерді қосуға арналған сұраныс кодын жазамыз. Бұл кестенің DateTime өрісінен басқа өрістері форманы толтыру барысында деректерді енгізу үшін қолданылған болатын, сол себепті осы өріске *деректер қосу* командасын жазамыз. Мұндағы:

```
buy.DateTime = DateTime.Now;
```

Тапсырыс жасалған уақытты есепке алуға арналған бұл өріске Now функциясы арқылы ағымдағы күн жады мен тапсырыс уақыты тіркелетін болады. Бұдан соң,

```
BookContext.Buys.Add(buy); //деректерді қосу
BookContext.SaveChanges(); //деректер қорындағы өзгерістерді сақтау
```

командалары

орындалады.

Деректердің дұрыс енгізілгенін тексеру және кері байланыс орнату үшін return функциясы арқылы хабарлама терезесін құрамыз:

```
return $"Құрметті, {buy.FIO}, сізбен жақын уақытта хабарласады!";
```

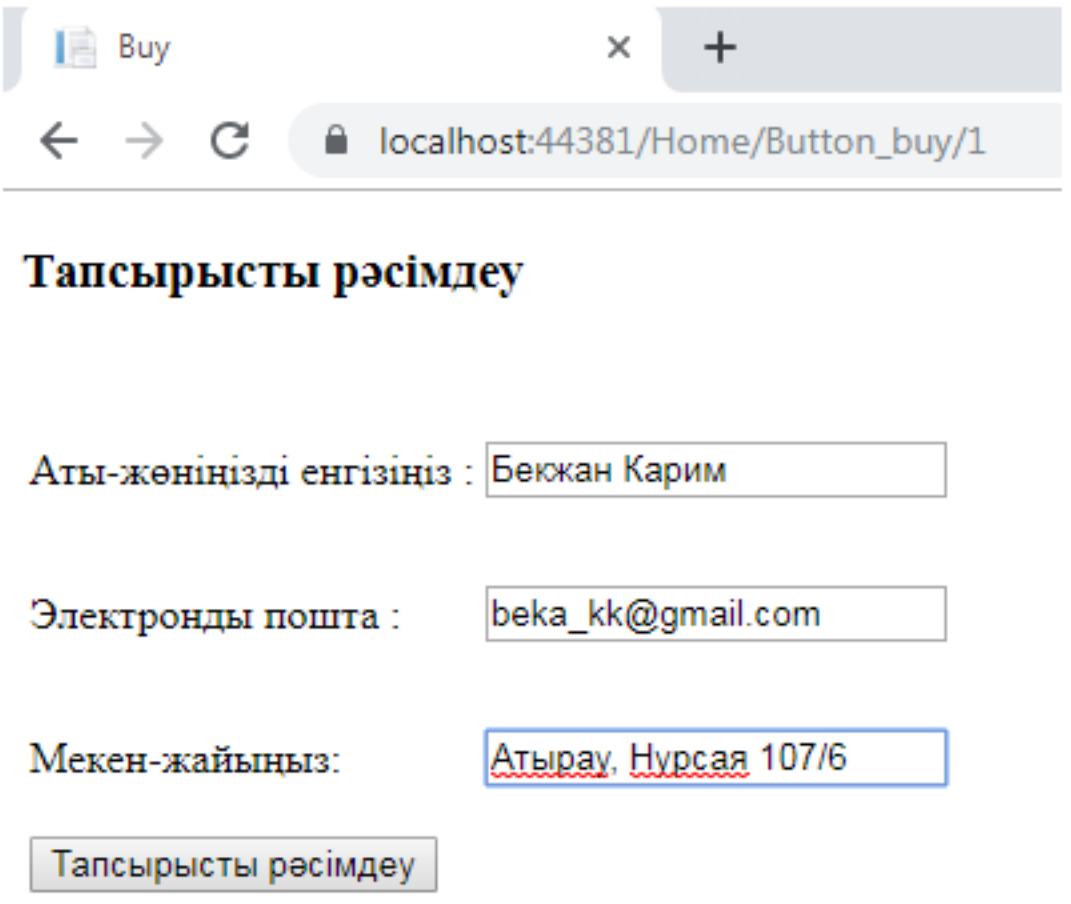
Бұдан соң, Index.cshtml файлын ашып, кітаптар тізімін шығару кодындағы *Сатып алу* батырмасына сілтеме қоямыз.

```
Web.config | Index.cshtml | HomeController.cs | BookContext.cs | Buy.cs

<table>
    <thead>
        <tr>
            <td>Коды</td>
            <td>Кітап атауы</td>
            <td>Автор</td>
            <td>Құны</td>
            <td>Басгасы</td>
            <td>Шыққан жылы</td>
            <td> </td>
        </tr>
    </thead>
    <tbody>
        @{ foreach (var books in ViewBag.Books)
            {
                <tr>
                    <td>@books.Id</td>
                    <td>@books.Name</td>
                    <td>@books.Author</td>
                    <td>@books.Price</td>
                    <td>@books.Publisher</td>
                    <td>@books.Year</td>
                    <td><a href="/Home/Button_buy/@books.Id">Сатып алу</a></td>
                </tr>
            }

        }
    </tbody>
```

Сурет 8.44 – Сатып алу формасына сілтеме қою

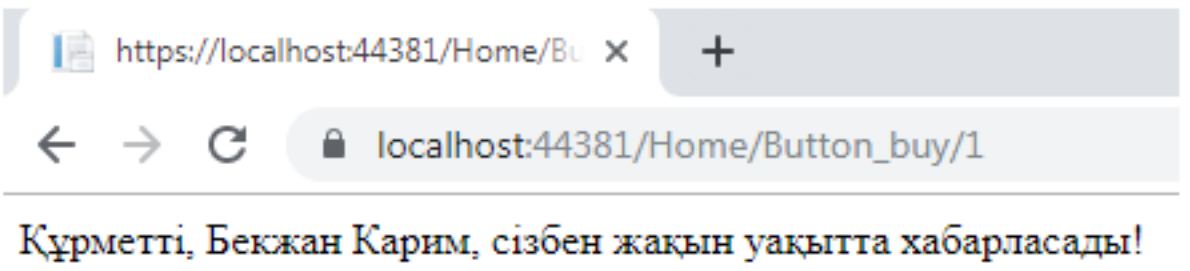
Бұл кодта орындалатын көрсетілім жолы және Razor механизмі арқылы кітаптың Id нөміріне сілтеме көрсетіледі.

Қажетті жұмыстар орындалғаннан кейін бағдарламаны орындауға жібереміз. Кітаптар тізімі көрсетілген алғашқы құрылған Index бетінен кез-келген кітап үшін Сатып алу батырмасына шертіп Тапсырысты рәсімдеу бетіне өтеміз (8.45 - сурет). Формаға қажетті деректерді енгізіп, *Тапсырысты рәсімдеу* батырмасына шертеміз.



Сурет 8.45 – Тапсырысты рәсімдеу формасын толтыру

Кері байланыс орнату үшін құрылған хабарлама коды орындалып келесі түрдегі жауап аламыз.



Сурет 8.46 – Сұраныс қабылданғаны туралы хабарлама

Бұл формада толтырылған деректер Bookstore деректер қорының Buys кестесіне қосылады. Нәтижені деректер қорын ашып, тексере аласыз.

7.4.5 Web-беттер шеберін қолдану

Жобадағы web-беттердің әрқайсысына стиль қойып, жалпы қасиеттерді қайталап жазып отырмау үшін, MVC-де құрылған жобада web-беттер шеберін қолданған тиімді. Web-беттер шебері – барлық беттерге ортақ шаблон құрады.

Web-беттер шеберін ашу үшін Views-Shared-_.ViewStart.cshtml (Visual Studio ортасының алдыңғы нұсқаларында _Layout.cshtml) файлын ашамыз.

Үнсіз келісім бойынша бұл файл төмендегі кодтан тұрады:

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

Біздің жағдайда кітаптар тізімі мен тапсырыс берушілер деректерін толтыру формасынан тұратын екі web-бет Button_buy және Index көрсетілімдері түрінде сақталған. Осы беттер үшін ортақ фон және шрифт қойып, *Басты бетке* өтуге арналған батырма құрамыз. Аталған файлдар Views-Home бумасында сақталған.

_.ViewStart.cshtml файлына келесі түрдегі кодты енгіземіз:

```
<!DOCTYPE html>
<html>
<head>
    <link type="text/css" rel="stylesheet" href="~/Content/Site.css" />
    <title></title>
</head>
<body>
    <div>
        @Html.ActionLink("Басты бетке", "Index", "Home")
        <br />

    </div>
</body>
</html>
```

Мұндағы, <link type="text/css" rel="stylesheet" href="~/Content/Site.css"/> - жолы Content/Site.css файлында сақталған стильдер файлына сілтемені көрсетеді. Ал, *@Html.ActionLink("Басты бетке", "Index", "Home")* – контроллердің белгілі-бір әдісіне сілтеме жасайтын әдіс болып табылады.

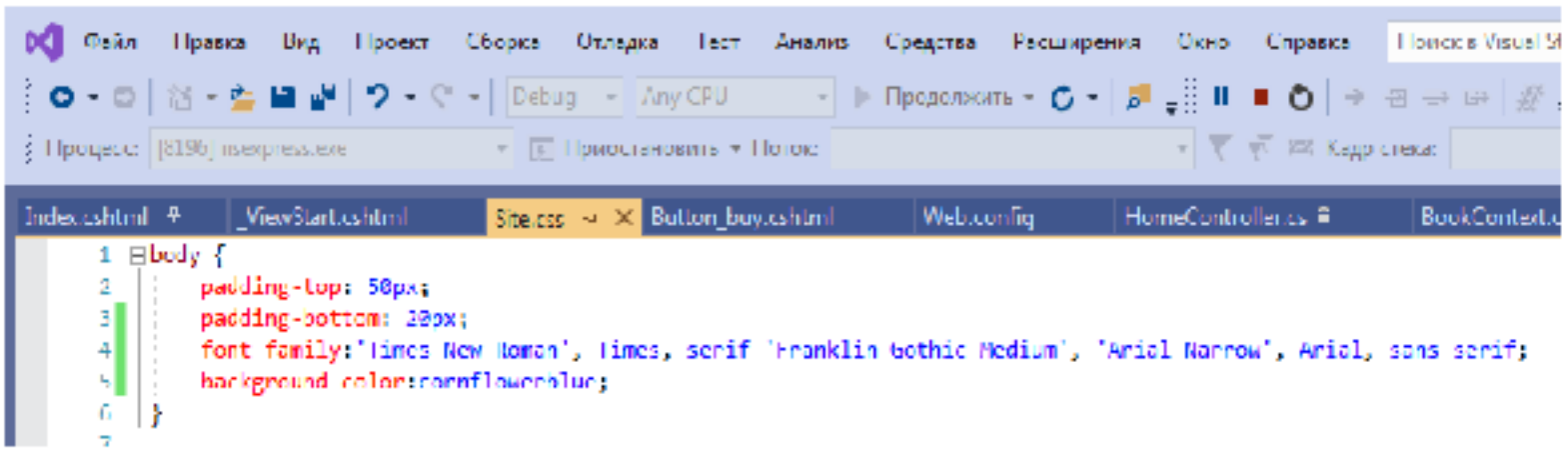
Бұдан соң, Content/Site.css адресінде орналасқан файлды ашып, стильге өзгерістер енгіземіз.

Үнсіз келісім бойынша ашылған файлда тек отступтарға арналған келесі жолдарды ғана қалдырамыз:

```
padding-top:      50px;
padding-bottom: 20px;
```

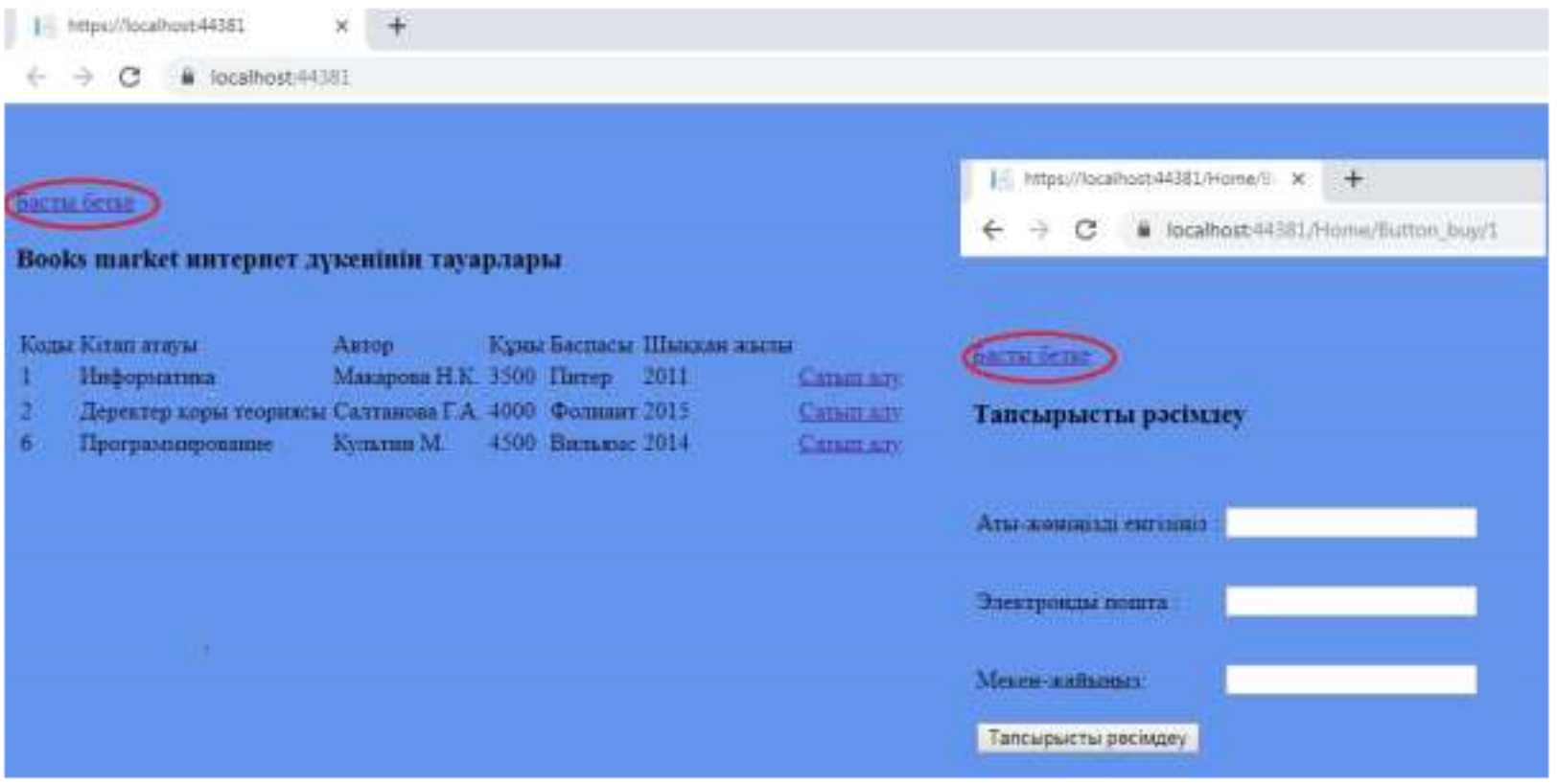
қалған екі жолға бет фонының түсі және шрифтілер түрлері енгізіледі.

font-family – қасиетіндегі шрифтілердің бірнеше түрін таңдауға болады. Егер қолданушының компьютерінде белгілі-бір шрифт орнатылмаған болса, тізімдегі келесі тұрған шрифт бағдарламаның орындалуы барысында таңдалады.



Сурет 8.47 – Site.css файлының құрылымы

Бағдарлама нәтижесінде жасақталған Index және Button_buy беттері үшін «Басты бетке» өту батырмасы құрылып, web-беттің фоны мен шрифтісіндегі өзгеріс екі бет үшін де орындалады (8.48-сурет).



Сурет 8.48 – Web-беттер шеберін қолдану нәтижесі

Тапсырма.

Сатып алушылардың тапсырыстар тізімін шығаратын web-бет құрыңыз.

Бақылау сұрақтары

1. Visual Studio интеграцияланған ортасының қызметіне сипаттама беріңіз.
2. MVC паттерні көмегімен жоба құру қалай жүзеге асырылады?
3. MVC жобасындағы AppData файлы қандай деректерді сақтайды?
4. Models арқылы класс құру қандай нәтиже береді?
5. SQLConnection параметрлері қандай деректерді қамтиды?
6. NuGET пакеті қандай жағдайда қолданылады?
7. Entity Framework орнату қандай нәтижеге әкеледі?
8. Форма арқылы деректер толтыру және оны серверге жіберу үрдісі қалай жүзеге асады?
9. Razor механизмінің қызметін сипаттаңыз.
10. Controller және View нысандарының байланысы қандай?
11. Web-беттер шеберін қолдану қандай нәтижеге әкеледі?
12. MVC жобасындағы Views бумасы қандай деректерді сақтайды?
13. ASP.NET жобасын жүзеге асырушы серверді атаңыз.
14. MVC 5 нұсқасына қандай толықтырулар енгізілген?
15. Web-клиенттік файлдар стильдеріне өзгерістер енгізу үшін қандай файлдар каталогын қолданамыз.