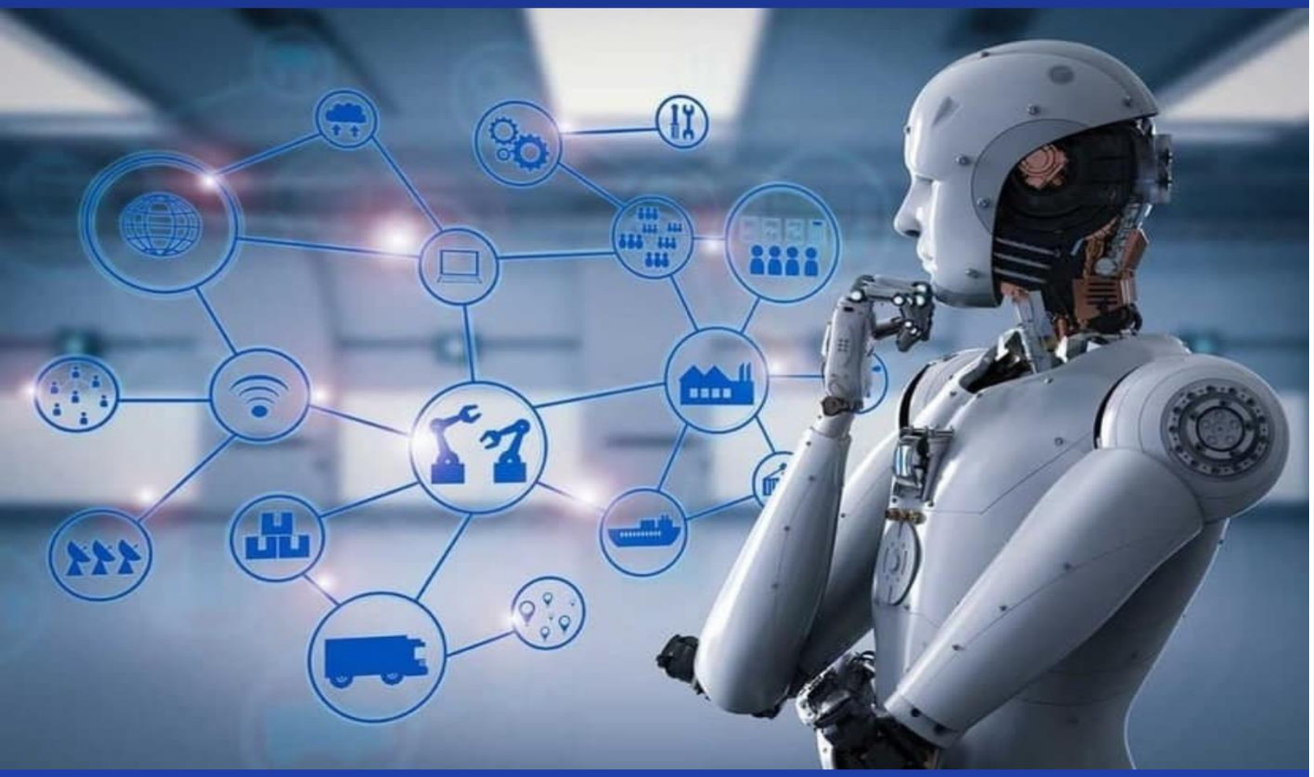
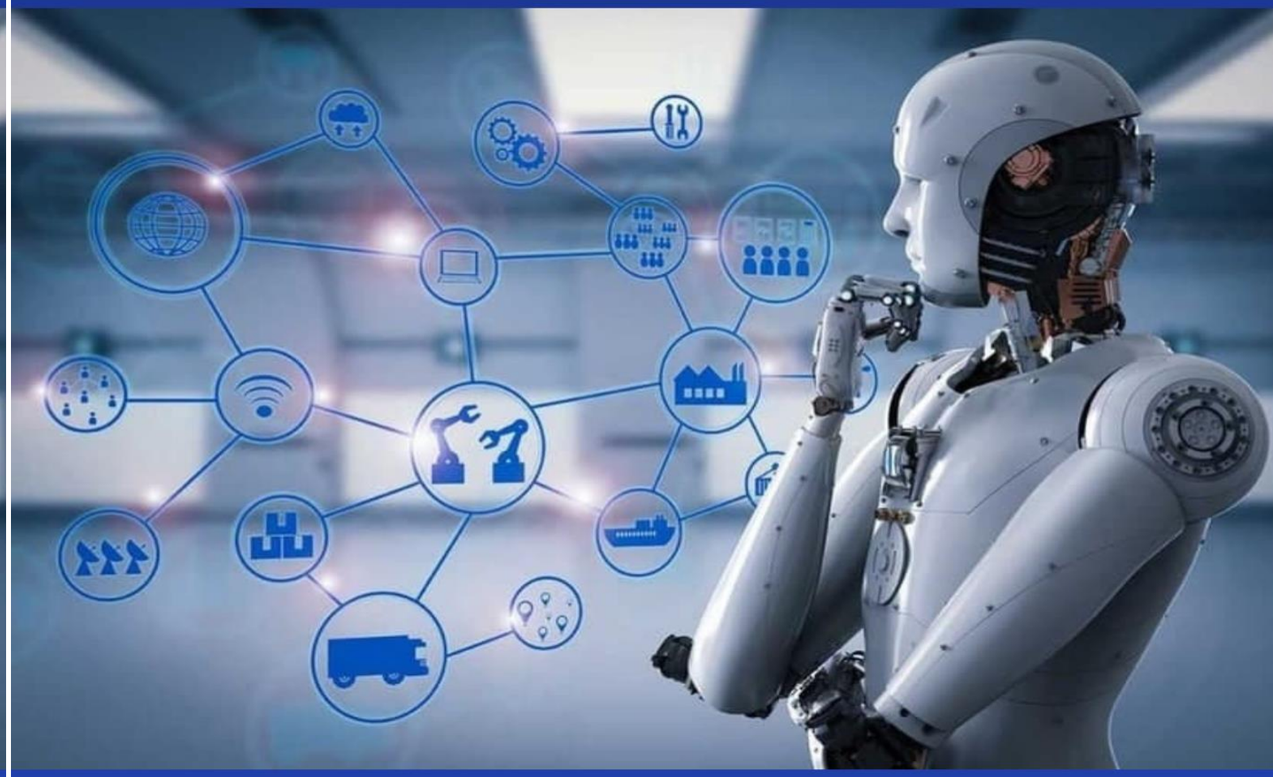


MUHAMEDIYEVA D.T., VARLAMOVA L.P.,
BAXROMOV S.A., ELOV B.B.,
ABDURAXMANOV O.A.

# DASTURIY INJINIRING ASP.NET MVC 5

## O‘QUV QO‘LLANMA



MUHAMEDIYEVA D.T., VARLAMOVA L.P.,
BAXROMOV S.A., ELOV B.B.,
ABDURAXMANOV O.A.

**O’ZBEKISTON RESPUBLIKASI OLIY VA O’RTA MAXSUS TA’LIM VAZIRLIGI**

**MIRZO ULUG‘BEK NOMIDAGI O’ZBEKISTON MILLIY UNIVERSITETI**

**Muhamediyeva D.T., Varlamova L.P.,**

**Baxromov S.A., Elov B.B., Abduraxmanov O.A.**

# “Dasturiy injiniring ”

# ASP.NET MVC 5

**o‘quv qo‘llanma**

**Toshkent 2023**

“Dasturiy injiniring” kursi uchun o‘quv qo‘llanma 5330100-“Axborot tizimlari, matematika va dasturiy ta’minot” mutaxassisligi talabalari uchun dasturlash tillaridan foydalanish, dasturiy ta’minot ishlab chiqish va ma’lumotlar bazasini boshqarish tizimlarini o‘rgatish maqsadida tuzilgan. Ushbu qo‘llanmaning 3-qismi ASP.NET MVC 5 da dasturlashni o‘z ichiga oladi.

The textbook for the course "Software Engineering" was compiled for students of the specialty 5330100 - "Information systems, mathematics and software" with the aim of teaching the use of programming languages, software development and database management systems. This course is taught to students for three semesters, according to which the manual includes three parts. “The Part Three” of this tutorial covers programming in ASP.NET MVC 5.

**Mualliflar:**

| | |
|---|---|
| **D.T.Muhamediyeva** | - Toshkent irrigatsiya va qishloq xo’jaligini mexanizatsiyalash muhandislari instituti milliy tadqiqot universiteti professori, t.f.d. |
| **L.P.Varlamova** | - Mirzo Ulug‘bek nomidagi O‘zbekiston Milliy universiteti, Hisoblash matematikasi va axborot tizimlari kafedrasi professori, t.f.d. |
| **S.A. Baxromov** | - Mirzo Ulug‘bek nomidagi O‘zbekiston Milliy universiteti, Hisoblash matematikasi va axborot tizimlari kafedrasi dotsenti, t.f.n. |
| **B.B.Elov** | - Mirzo Ulug‘bek nomidagi O‘zbekiston Milliy universiteti, Hisoblash matematikasi va axborot tizimlari kafedrasi dotsenti, t.f.n. |
| **A.A.Abduraxmanov** | - Mirzo Ulug‘bek nomidagi O‘zbekiston Milliy universiteti, Hisoblash matematikasi va axborot tizimlari kafedrasi o’qituvchisi |

**Taqrizchilar:**

**Matyaqubov A.S** – O’zMU, Amaliy matematika va computer taxlili kafedrasi mudiri

**Yakubov M.S.** – TATU, Axborot texnologiyalari kafedrasi professori

**O’quv qo’llanma O’zbekiston Respublikasi Oliy va o’rta maxsus ta'lim vazirligining 2022 yil “09” sentyabrdagi “302” –sonli buyrug’iga asosan nashrga tavsiya etilgan. Ro’yxatga olish raqami 302-0100.**

# Мундарижа

Ушбу ўқув қўлланма **5330200 –** информатика ва ахборот технологиялари, **5330300 –** ахборот хавфсизлиги, **5330100 –** ахборот тизимларининг математик ва дастурий таъминоти, **5130200 –** амалий математика ва информатика таълим йўналишлари талабаларга дастурлаш тиллари ва технологиялари (дастурлаш асослари) ва маълумотлар базаси, веб-дастурлаш, веб-технологиялар, ахборот тизимларини лойиҳалаштириш ва яратиш фанларидан фойдаланиш мақсадида ёзилган бўлиб, талабаларга **ASP.NET MVC 5** технологияси орқали сайтлар ва порталларни яратиш ўргатилади.

Ушбу ўқув услубий қўлланмани ўзлаштириш учун талабаларда объектга йўналтирилган дастурлаш, **C#** тили синтаксиси, **.Net Framework** библиотекаси, **T-SQL** тили, **ADO.NET** асоси, **Visual Studio 2013** муҳитида ишлаш кўникмасига эга бўлишлари лозим.

Ўқув қўлланмада **C#** тилида **ASP.NET MVC** лойиҳаларни яратиш, **MS SQL SERVER 2012** маълумотлар базаси бошқарув тизимида дастурлаш, **ASP.NET MVC, EntityFramework, EF Code First, JSON, Ajax, Razor** технологиялари, **HTML 5.0** тили ҳақида маълумотлар ва амалий дастурлар келтирилган.

Ўқув қўлланмада келтирилган ўқув материалларини ўзлаштириш натижасида талабалар:

- **ASP.NET MVC 5.0 C#** технологиясидан фойдаланиб сайтлар ва порталларни яратишлари;
- Статик саҳифаларни яратишлари;
- Динамик саҳифаларни **Razor** технологияси орқали яратишни;
- Сайтнинг моделини яратишни;
- **Object Relational Mapping: EntityFramework Code First, Databse First** ва **Model First** технологияларидан фойдаланишни;
- Контроллерлар ва амал методларини;
- Сайт учун турли **Routing**ларни шакллантиришни;
- Сайтда фойдаланувчиларни рўйхатдан ўтказишни ва роллар асосида муайян саҳифаларга авторизацияни амалга оширишни;
- **Nuget** пакетларидан фойдаланишни;
- **Ajax** технологиясини;
- **DataMigrations** амалларини бажаришни;
- ахборот тизимларини лойиҳалаштиришни;
- сайт ва проектлар яратишни ўрганадилар.

## Фойдаланилган қисқартмалар

**LINQ —** Language Integrated Query (тузилмаланган сўровлар тили)

**EF –** Entity Framework (маълумотларга доступни таъминловчи объектга-мўлжалланган технология)

**ORM —** object-relational mapping ()

**EDM —** Entity Data Model ()

**DB —** Database (маълумотлар базаси)

**VS 2013 —** Visual Studio 2013

**OOP —** Object-Oriented Programming (объектга йўналтирилган дастурлаш)

**UML —** Unified Modeling Language (унифицирланган моделлаштириш тили)

**TPH —** Table Per Hierarchy (класслар иерархиясига мос жадвал)

**TPT —** Table Per Type (типга мос жадвал)

**TPC —** Table Per Concrete Type (ҳар бир алоҳида типга мос жадвал)

## Фойдаланилган терминлар

**Класс** (класс, class) — мантиқий тузилма ҳисобланиб, майдонлар (атрибутлар), методлар ва ҳодисалар мажмуасидан иборат.

**Объектга йўналтирилган дастурлаш** (объектно ориентированное программирование, Object-Oriented Programming) — асосий концециялари объект ва класс тушунчалари ҳисобланган дастурлаш парадигмаси.

**Объект** (объект, object) — ўзнинг ҳолати ва ҳаракатига эга бўган виртуал фазодаги бирор элемент бўлиб, хусусиятлари(атрибутлар)нинг қийматлари аниқланган бўлиб, улар устида бир қанча амаллар (методов)ларни бажариш мумкин. «**Класс нусхаси**» ва «**объект**» терминлари синоним ҳисобланади.

**Класс майдони** ёки **атрибут** (***переменная-член, data member, class field, instance variable***) — класс ёки объектга мансуб ўзгарувчи. Объектнинг барча маълумотлари ёнинг майдонларида сақланади. Ушбу майдонларга мурожаат унинг номи орқали амалга оширилади. Ҳар бир майдон типи класс аниқланганда кўрсатилади.

**Маълумотлар типи** (*тип, type*) — қийматлар ва улар устида аниқланган амаллар тўплами

**Метод** (метод, method) **—** муайян класс ёки объектга тегишли функция ёки процедура. Метод бирор амални бажариш учун зарур бўлган бир нечта операторлардан иборат бўлиб, кирувчи аргументларга эга бўлиши мумкин.

**Элемент** (сущность, entity) **—** реал ҳаётдаги бирор объект.

**Контроллер** (controller) – фойдаланувчи ва тизим, кўриниш ва маълумот сақлагич ўртасидаги алоқани таъминловчи класс ҳисобланади. Ушбу объект орқали фойдаланувчи томонидан киритилган маълумот қабул қилинади ва уларни қайта ишлайди. Қайта ишлаш натижасига кўра фойдаланувчига натижани (масалан, кўриниш шаклида) қайтаради.

**Кўриниш (view) –** дастурнинг фойдаланувчи интерфейси ёки визуал қисми.

**Модель (model) –** ишлатилаётган маълумотларнинг мантиғини ифодаловчи класс.

**Амал методлари** (action methods) – муайян URL бўйича сўровга мос контроллернинг методларини ифодалайди.

**MySQL** – эркин фойдаланиладиган маълумотлар базаси бошқарув тизими.

**ADO.NET** – Microsoft .NETга асосланган маълумотларга доступни таъминловчи технология.

**Entity SQL –** SQL тилига ўхшаш тил ҳисобланиб, Entity Framework да концептуал моделларда сўровларни амалга оширишга мўлжалланган.

**LINQ to Entities –** LINQ воситалари орқали маълумотлар базасига мурожаат қилиш интерфейси.

**Model First –** Визуал муҳаррир асосида edmx модел ва маълумотлар базасини шакллантириш технологияси.

**Code First –** C# тилида ёзилган модел коди асосида маълумотлар базасини шакллантириш технологияси.

**Database First –** Маълумотлар базаси асосида edmx моделни шакллантириш технологияси.

**JSON**

**Ajax**

**Razor**

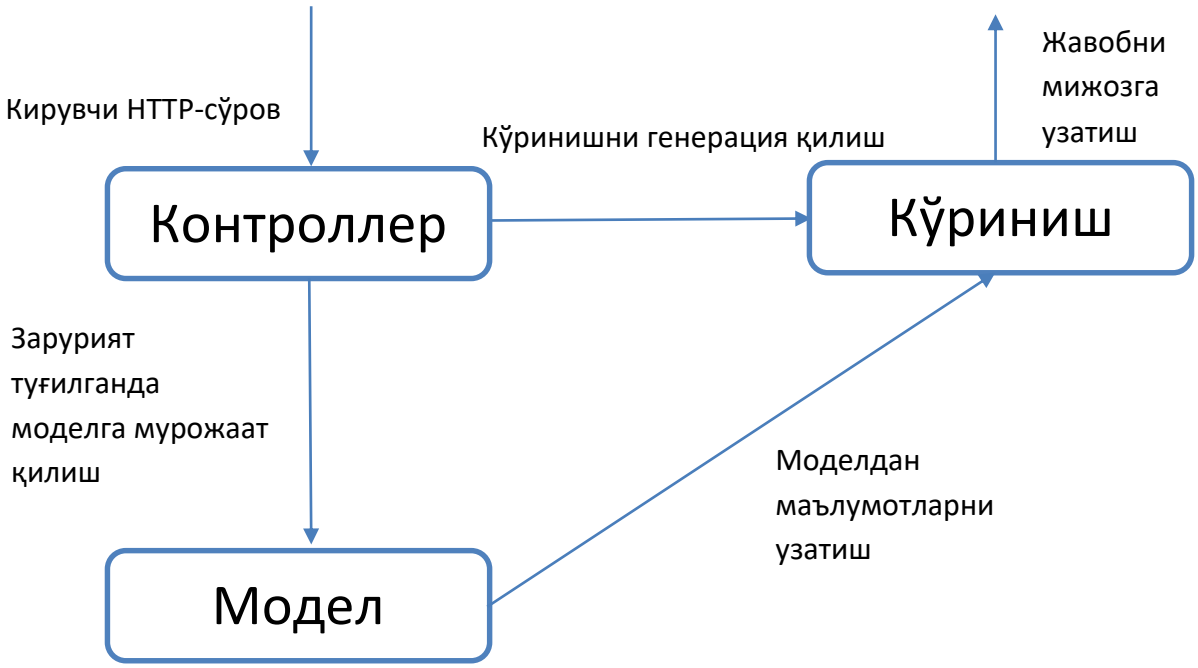
# I. ASP.NET MVC га кириш

**ASP.NET MVC** платформаси сайтларни ва веб-дастурларни яратиш фреймворки бўлиб, **MVC** паттерни асосида амалий дастурларни тадбиқ қилишга йўналтирилган.

**MVC** (**model - view - controller**) концепцияси асосида дастур учта қисмга ажратилади:

- **Контроллер** (**controller**) – фойдаланувчи ва тизим, кўриниш ва маълумот сақлагич ўртасидаги алоқани таъминловчи класс ҳисобланади. Ушбу объект орқали фойдаланувчи томонидан киритилган маълумот қабул қилинади ва уларни қайта ишлайди. Қайта ишлаш натижасига кўра фойдаланувчига натижани (масалан, кўриниш шаклида) қайтаради.
- **Кўриниш (view) –** дастурнинг фойдаланувчи интерфейси ёки визуал қисми.
- **Модель (model) –** ишлатилаётган маълумотларнинг мантиғини ифодаловчи класс.

Ушбу компонентларнинг ўзаро боғланишини қуйидаги расмдан кўриш мумкин:



Ушбу схемада модел мустақил компонент бўлиб, контроллерда ёки кўринишда амалга оширилган ихтиёрий ўзгаришлар моделга таъсир ўтказмайди. Контроллер ва кўриниш нисбатан мустақил компонентлар ҳисобланиб, уларни бир-бирига боғлиқсиз ҳолда ўзгартириш мумкин.

Натижада **жавобгарликни тақсимлаш** концепцияси тадбиқ қилиниб, алоҳида компонентлар устида осон амал бажарилади. Шунингдек, яратилган дастурий таъминотни яхши тестлаштириш мумкин. Дастурнинг визуал қисми ёки фронтэндига кўра кўринишларни контроллердан мустақил тарзда тестдан ўтказишимиз мумкин. Ёки бэкендга асосланган тарзда контроллерни тестдан ўтказишимиз мумкин.

Жорий паттернни тадбиқ қилиш ва ифодалашда фарқ мавжуд бўлиши мумкин. Аммо ўзининг мослашувчанлиги ва соддалиги сабабли ушбу технология веб-лойиҳаларни яратишда кенг тарқалди.

## ASP.NET MVC нинг асоси. MVC 5 даги янгиликлар

**2013** йилда **ASP.NET MVC 5** версияси ва **VS 2013** тадбиқ қилинди. Кўпгина аспектларда **MVC 5** версия **MVC 4** дан унча фарқ қилмасада, ўз навбатида баъзи ўзгаришлар амалга оширилган:

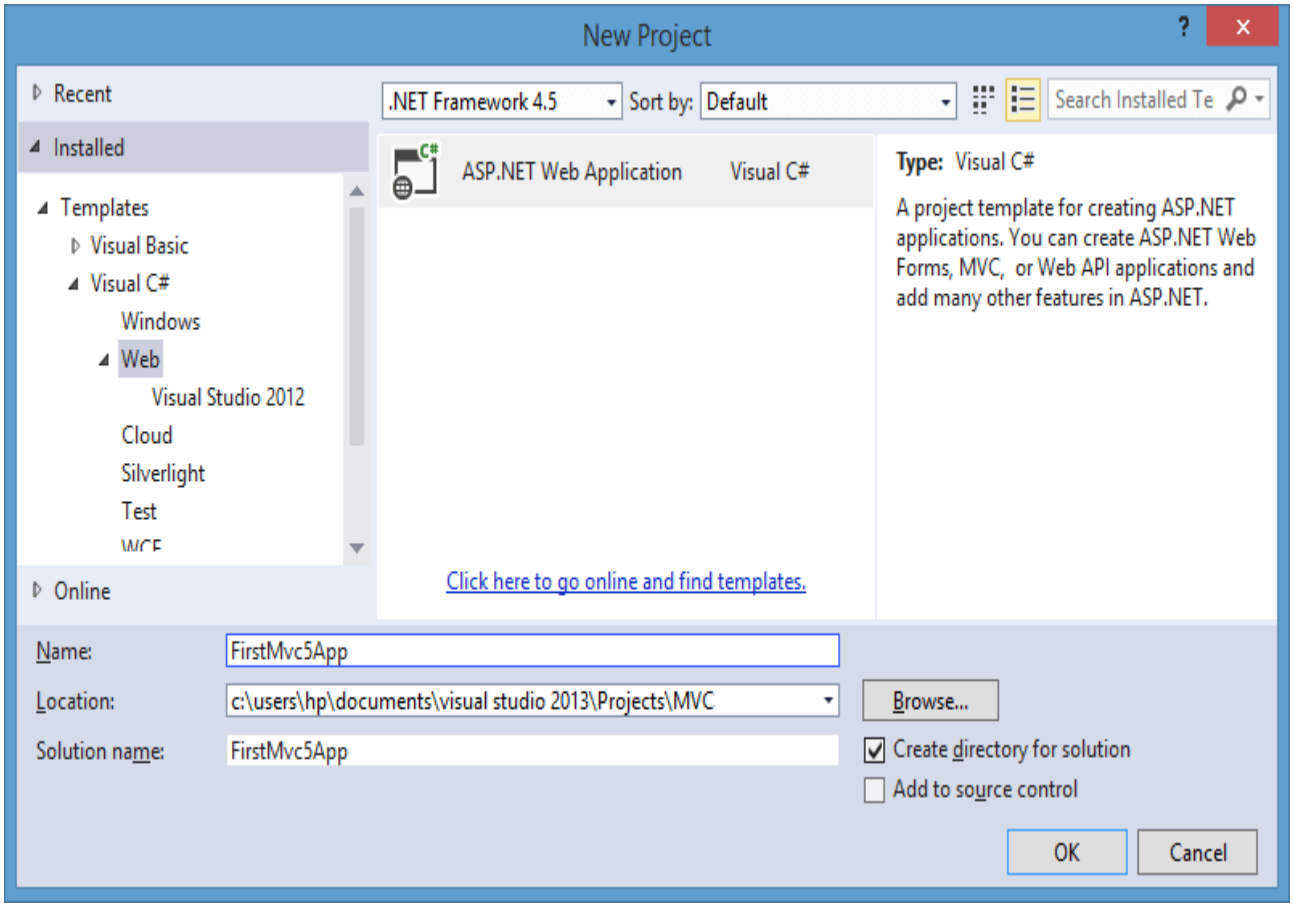
- **MVC 5** да аутентификация ва авторизация концепцияси ўзгартирилган. **SimpleMembershipProvider** ўрнида **ASP.NET Identity** тизими қўлланган. Ушбу тизимда **OWIN** ва **Katana** компонентлари ишлатилади;
- Мослашувчан ва кенгайтириш имконияти мавжуд интерфейсни яратиш мақсадида **MVC 5** да **Bootstrap css**-фраймворки ишлатилади;
- Аутентификация фильтри ва фильтрларни қайта аниқлаш функционали ишлаб чиқилган;
- **MVC 5** да маршрутлаштириш атрибутлари қўшилган.

Юқорида келтирилганлар **MVC 5** да тақдим қилинган муҳим амаллар ҳисобланади. **MVC 4** технологиясидаги кўникмаларни **MVC 5** да тўлиқ қўллаш мумкин.

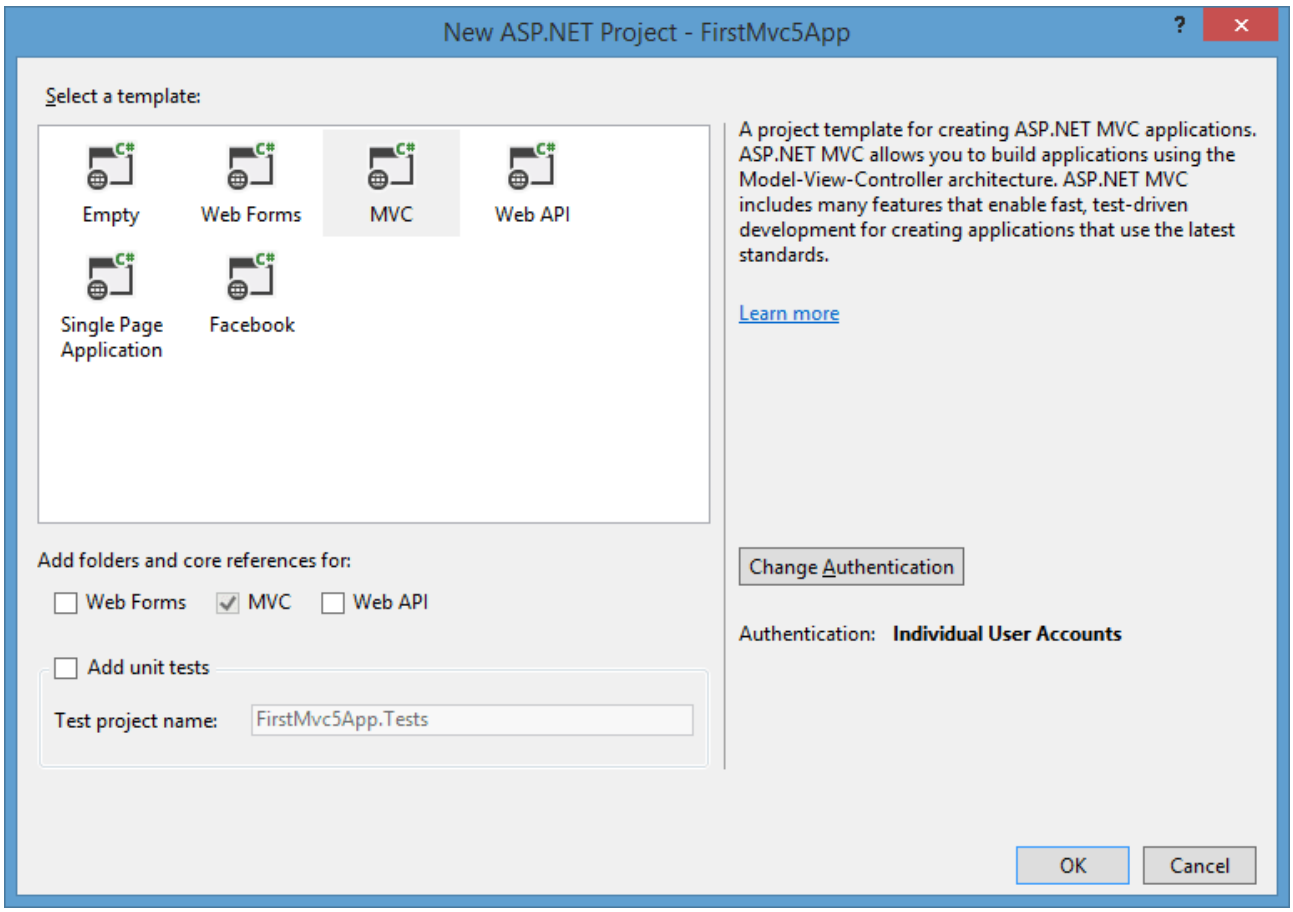
## ASP.NET MVC 5 билан ишлаш

**ASP.NET MVC 5** платформасида веб-дастурларни яратиш учун **Visual Studio Express 2013 for Web** ёки **Visual Studio 2013**нинг бошқа версияси зарур. Ушбу дастурий таъминотни **http://www.microsoft.com/ru-ru/download/details.aspx?id=40747** манзилидан кўчириб олиш мумкин. Шунингдек, **Visual Studio 2012** муҳитида ҳам **MVC 5** ни ишлатиш мумкин. Аммо бу ҳолда қўшимча тарзда **http://www.microsoft.com/ru-ru/download/details.aspx?id=41532** инструментарийни ўрнатиш лозим. Бизнинг курсимизда дастурчилар **Visual Studio 2013** дастурий таъминотидан фойдаланамиз.

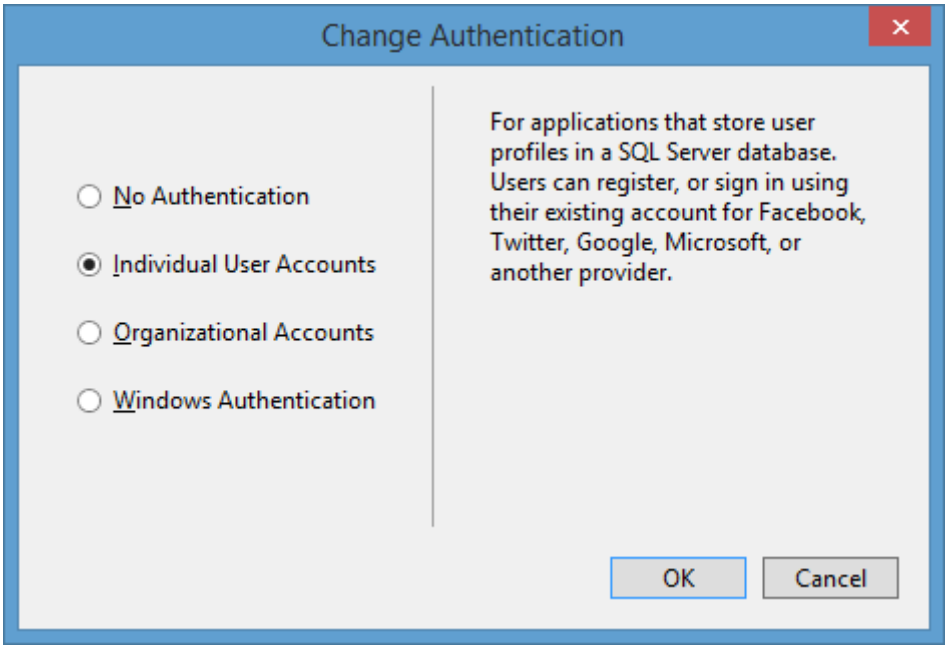
**Microsoft Visual Studio Express 2013 for Web** дастурий таъминоти ўрнатилгач, менюнинг **File (Файл)** қисмидан **New Project...(Создать проект)** пунктини танлаймиз. Натижада янги лойиҳани яратиш мулоқот ойнаси тақдим этилади. **Microsoft** компаниясида **One ASP.NET** каби курсга асосланганлиги сабабли, **Visual Studio** нинг бошқа версиялари каби турли лойиҳалар тақдим этилмайди. Уларнинг ўрнига фақатгина битта тип тақдим этилади:



Янги ҳосил қилинаётган лойиҳага бирор номни бериб, **ОК** тугмасини босамиз. Натижада янги лойиҳа учун шаблонни танлаш мулоқот ойнаси тақдим этилади:



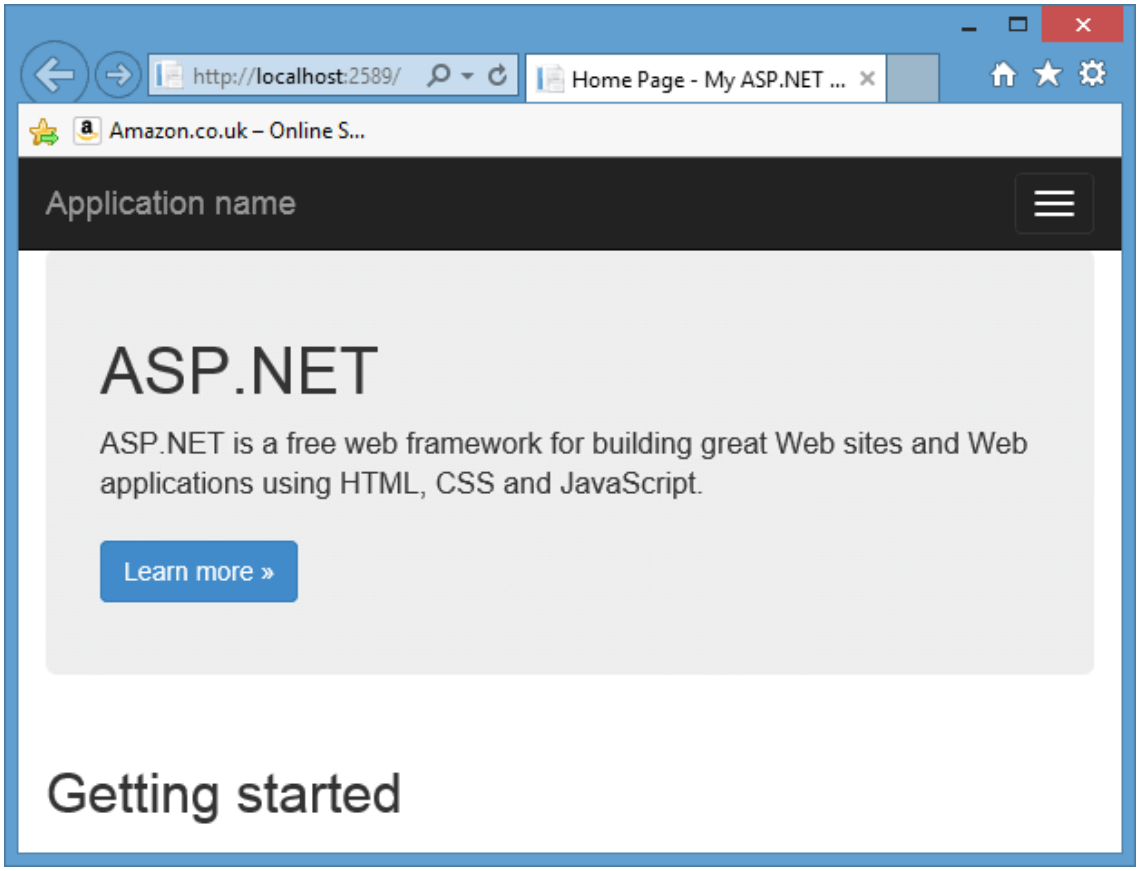

Мулоқот ойнасида **MVC** шаблони автоматик тақдим қилинади. Шунингдек, сиз жорий мулоқот ойнасида лойиҳани тестдан ўтказиш опциясини танлашингиз мумкин. Шунингдек, ойнанинг ўнг қисмида дастурнинг аутентификация механизми (**Change Authentication** тугмаси)ни ҳам танлаш мумкин. Хусусий ҳолда **Individual User Accounts** типи ўрнатилган. Ушбу опцияни ўзгартирмаймиз. Аммо **Change Authentication** тугмаси босилса, бизга қуйидаги мулоқот ойнаси тақдим этилади:




Ушбу мулоқот ойнасида дастурда аутентфикация жараёнининг қуйидаги қисмлари мавжуд:

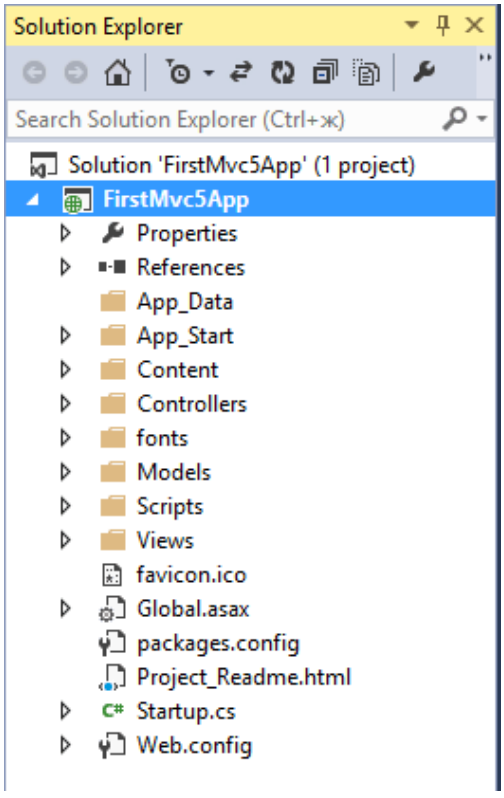
- **No Authentication**: дастур фойдаланувчи аутентификациясини талаб қилмайди;
- **Individual User Accounts**: индивидуал аутентификация талаб қилинади ва фойдаланувчиларнинг маълумотлари маълумотлар базасида сақланади. Шунингдек, социал тармоқлар орқали аутентификация қилиш ҳам мумкин;
- **Organizational Accounts**: асосан коорпоратив дастурлар учун мўлжалланган бўлиб, **Active Directory** ёки **Office 365**дан фойдаланилади;
- **Windows Authentication**: **windows** аутентификациясидан фойдаланадиган **intranet**-дастурлардаги фойдаланувчилар ҳақидаги маълумотларни бошқаришда фойдаланилади.

Бироздан сўнг биз дастурлардаги аутентификация механизмини кенгроқ кўриб чиқамиз. Шу сабабли, **OK** тугмасини босамиз ва янги лойиҳани яратамиз. Ушбу лойиҳа тармоқларган тузилма ва баъзи қийматлар тўлдирилган элементлардан иборат. Янги лойиҳани ишга туширамиз ва қуйидаги кўринишдаги дастур ишга тушади:




**MVC 5 лойиҳаси тузилмаси**

Янги яратилган **MVC 5** лойиҳасида қуйидаги функционаллар мавжуд:



Кетма-кет ушбу папка ва файлларни кўриб чиқамиз:

- **App_Data**: дастурда ишлатиладиган файллар, ресурслар ва маълумотлар базасини ўзида сақлайди;
- **App_Start**: дастурнинг ишга туширилишидаги мантиқдан иборат бир қатор статик файлларни ўзида сақлайди;
- **Content**: ёрдамчи файлларни ўзида сақлаб, уларда **C#** ёки **javascript** да ёзилмаган маълумотлар мавжуд бўлади. Ушбу турдаги маълумотларга **css** стили файлларини мисол сифатида келтириш мумкин;
- **Controllers**: контроллерлар классларидан иборат файлларни ўзида сақлайди. Лойиҳа яратилган вақда ушбу папкада иккита контроллер автоматик яратилади: **HomeController** ва **AccountController;**
- **fonts**: дастурда ишлатиладиган қўшимча шрифтларни ўзида сақлайди;
- **Models**: модел файлларини ўзида сақлайди. **Visual Studio** да бошланғич ҳолатда бир қатор моделлар автоматик яратилади. Ушбу моделлар асосида фойдаланувчиларни аутентификация қилиш учун зарур бўлган маълумотлар сақланади;
- **Scripts**: **javascript** тилида ёзилган скриптлар ва библиотекаларни ўзида сақлайди;
- **Views**: ушбу папкада кўринишлар сақланади. Барча кўринишлар папкаларда гуруҳлаштирилиб, ҳар бири битта контроллерга мос келади. Сўров қайта ишланганидан сўнг контроллер ушбу кўринишлардан бирини мижозга узатади. Шунингдек, ушбу папкада **Shared** каталоги мавжуд бўлиб, барча учун умумий кўринишни ўзида сақлайди;
- **Global.asax**: ушбу файлда дастур ишга туширилган вақтда бошланғич инициализация амалга оширилади. Шунингдек ушбу қисмда **App_Start** папкасида мавжуд класс методлари қайта ишланади;
- **Startup.cs**: **MVC 5** дастурларида **OWIN** спецификациясини қўлловчи библиотекалар ишлатилганлиги сабабли, жорий файл орқали **OWIN** ва дастур ўртасида алоқа ўрнатилади. **OWIN** спецификацияси орқали дастур компонентлари ўртасидаги алоқа тавсифланади;
- **Web.config**: дастур конфигурацияси файли.

Ҳар бир дастурнинг тузилмаси турлича ҳисобланиб, **MVC** технологиясининг мослашувчанлиги сабабли дастурни мос равишда созлаш мумкин. Юқорида келтирилган мулоҳазалар кўпгина лойиҳалар учун умумий ҳисобланади. **MVC** лойиҳаси тузилмаси билан танишиб чиққач, биринчи дастуримизни яратишимиз мумкин.

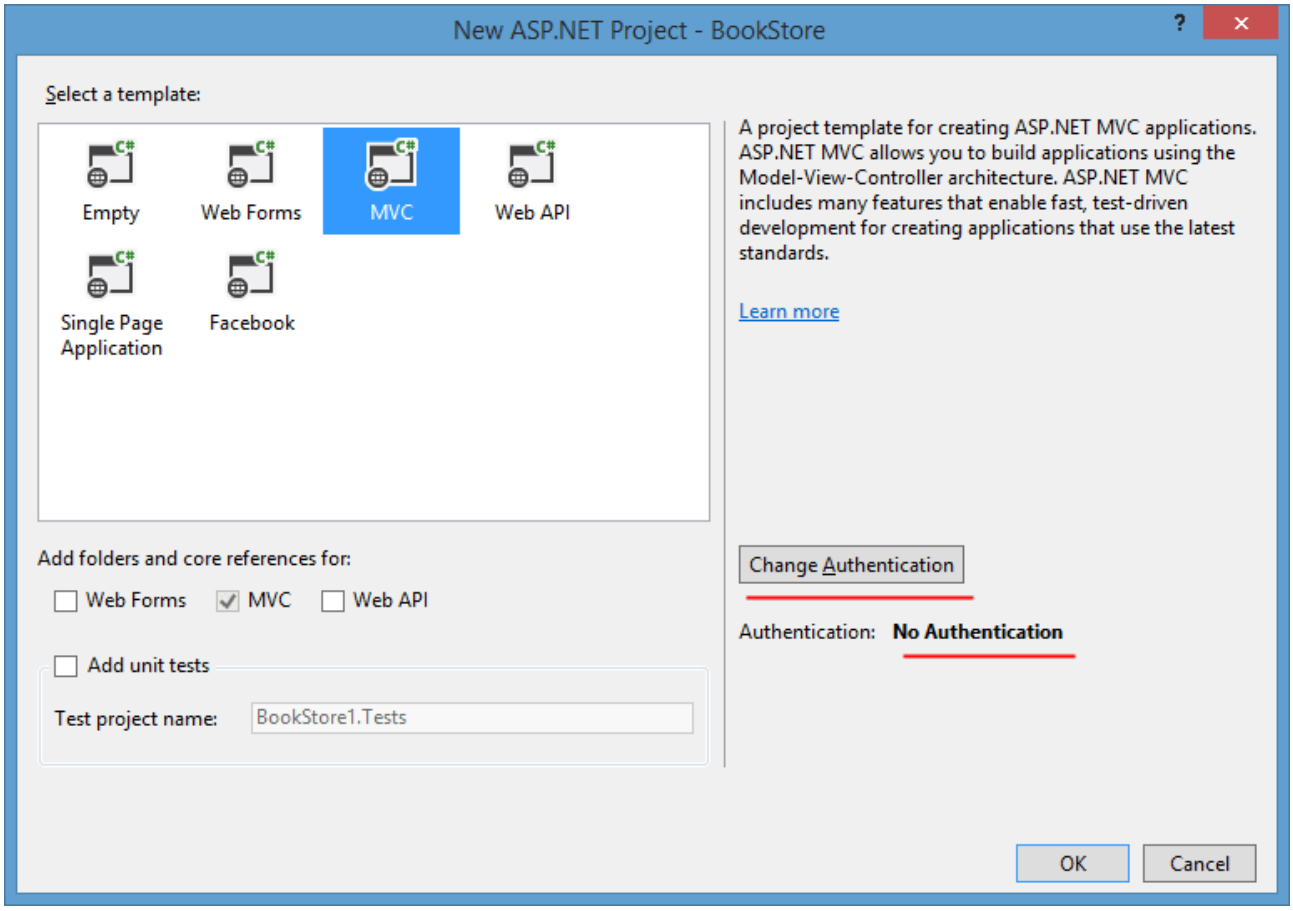
# II. ASP.NET MVC 5 даги биринчи дастур

## Лойиҳани яратиш

Биз аввалги бўлимда **MVC** паттернининг асосий тушунчаларини кўриб чиқдик. Энди **ASP.NET MVC 5** технологияси асосида биринчи дастурни яратишимиз мумкин. Бунинг учун **ASP.NET MVC 5** технологияси асосида оддий дастурни яратамиз.

Ушбу дастур китоб магазини фаолиятини эмуляция қилади. Бизда китоблар тўплами мавжуд бўлиб, сайтимизга кирган фойдаланувчи китобларни харид қилиш имкониятига эга бўлиши лозим.

Аввало **Visual Studio 2013** муҳитини ишга тушириб, менюнинг **File -> New Project..** қисмини танлаймиз. Сўнгра янги **ASP.NET MVC** лойиҳасини яратмиз ва унга **BookStore** номини берамиз. Сўнгра, таклиф қилинган технологиялар орасидан **MVC**ни танлаймиз. Мулоқот ойнасининг ўнг томонидан дастур айтентификациясида **No Authentication** ни танлаймиз.



Натижада деярли ҳеч қандай функционалликка эга бўлмаган, аммо бошланғич тузилмага эга бўлган бўш **MVC** лойиҳаси яратилади.

Биринчи навбатда кутубхона лойиҳамизнинг маълумотлар моделини ҳосил қилишимиз лозим. Лойиҳамизда китоб ва уни сотиб олиш модели мавжуд бўлиши шарт.

Яратган лойиҳамизда **Models** папкаси мавжуд бўлиб, ушбу папкада бизнинг моделларимиз жойлашади. Ушбу папка устига сичқончанинг ўнг тугмасини босиб, тақдим қилинган менюдан **Add->Class..** қисмни танлаймиз. Яратилаётган биринчи классимизни **Book** деб номлаймиз ва унга қуйидаги дастурий кодни шакллантирамиз.

```
using System;
namespace BookStore.Models
{
    public class Book
    {
        // китоб ID си
        public int Id { get; set; }
        // китоб номи
        public string Name { get; set; }
        // китоб муаллифи
        public string Author { get; set; }
        // нархи
        public int Price { get; set; }
    }
}
```

Албатта ҳақиқий дастурда ушбу моделда китобнинг расми, саҳифалари сони ва ишлаб чиқарилган йили ҳақидаги маълумотлар ҳам қатнашиши мумкин. Аммо бизнинг мисолда **Book** классида эълон қилинган атрибутлар етарли.

Худди юқоридаги усулда иккинчи **Purchase** классини ҳосил қиламиз. Ушбу класс хариф қилинаётган китобга жавоб беради.

```
using System;
namespace BookStore.Models
{
    public class Purchase
    {
        // харид ID си
        public int PurchaseId { get; set; }
```

```
        // харидор исми ва фамилияси
        public string Person { get; set; }
        // харидор манзили
        public string Address { get; set; }
        // китоб ID си
        public int BookId { get; set; }
        // хариф санаси
        public DateTime Date { get; set; }
    }
}
```

**Моделларни яратишда шартлар**

Юқоридаги ҳосил қилинган **Book** ва **Purchase** класслари **C#** тилидаги оддий классларни ифодалайди. Ушбу моделлар объектнинг реал хусусиятларидан ташкил топган. Аммо модел яратилаётганда баъзи шартларни ҳисобга олиш зарур. Биз маълумотлар базаси сифатида **SQL Server** дан фойдаланаётганимиз учун базада ушбу атрибутлардан бирламчи калитни аниқлаб олишимиз лозим. Ушбу атрибут орқали объектни идентификация қилиш мумкин. Шунинг учун ҳар бир моделда **Id** хусусиятини ҳосил қилдик. Ушбу майдонда бирламчи калит маълумотлари сақланиши назарда тутилган.

Ушбу жараёнда қуйидаги шартлар амалга оширилади: идентификатор хусусияти **Модел_номиId** ёки **Id** каби номланиши зарур. Бизнинг мисолда **Book** классида **Id** хусусияти аниқланган бўлиб, бирламчи калит ҳақидаги маълумотни сақлашга хизмат қилади. **Purchase** классида эса ушбу вазифани **PurchaseId** атрибути бажаради.

Иккинчи усул орқали калитни **Key** атрибути ёрдамида шакллантириш мумкин.

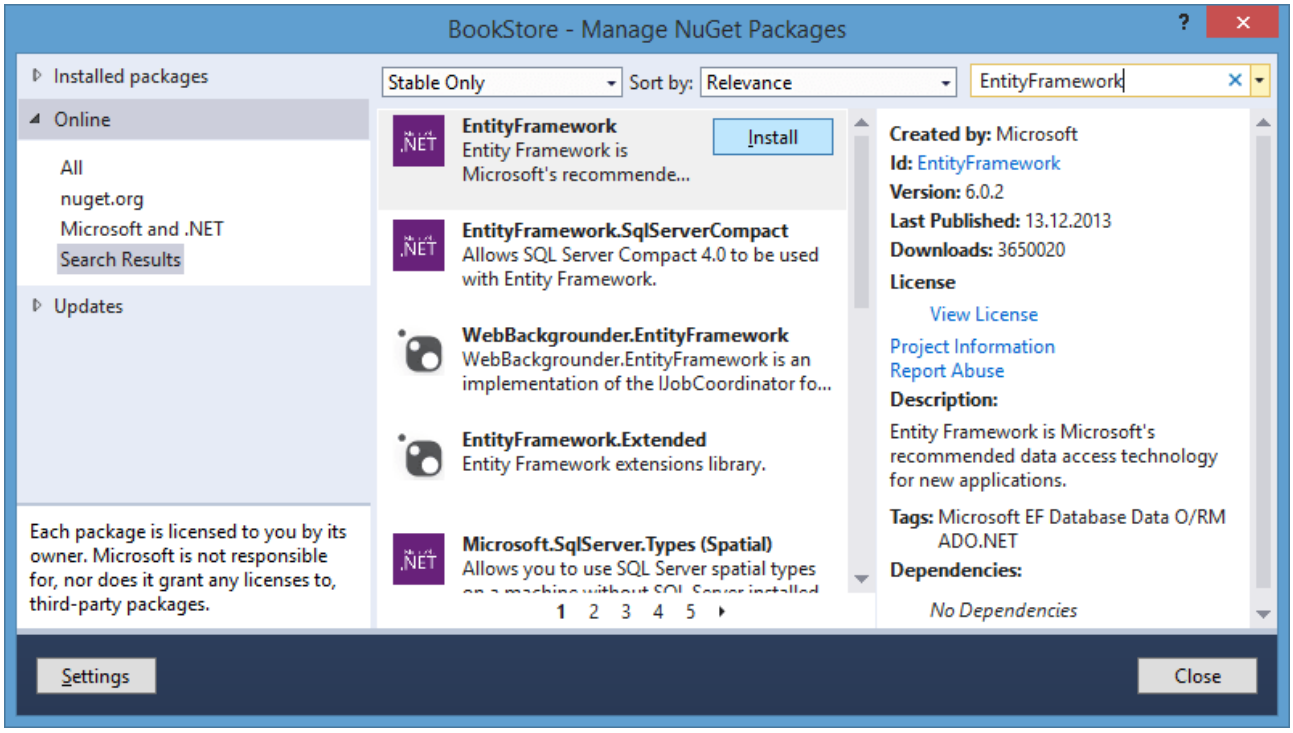
**Entity Framework**

**ASP.NET MVC** да маълумотлар билан ишлаш учун **Entity Framework** фреймворкидан фойдаланиш тавсия қилинади. Аммо ушбу фреймворкни ишлатиш мажбурий ҳисобланмайди. Дастурий таъминотни яратувчи дастурчилар ўз малака ва кўникмасидан келиб чиққан ҳолда **Entity Framework** фреймворкидан фойдаланиши ёки фойдаланмаслиги мумкин.

Ушбу фреймворкнинг устунлиги муайян маълумотлар базаси структурасидан келиб чиққан ҳолда абстракцияни амалга ошириш имконини беради. Шунингдек, маълумотлар устидаги барча амаллар модел орқали амалга оширилади.

Бизнинг лойиҳада **Entity Framework** библиотекалари мавжуд эмас. Уларни лойиҳага бириктириб қўйиш учун **NuGet** пакетли менеджеридан фойдаланишимиз лозим. **Solution Explorer (Обозреватель решений)** ойнасида лойиҳа структурасидаги **References** қисмига сичқончанинг ўнг тугмасини босиб, тақдим этилган менюдан **Manage NuGet Packages...** қисмни танлаймиз.

**NuGet** пакетларни бошқариш ойнасининг юқори ўнг қисмида **Entity Framework** ни танлаб, **Enter** ни босинг. Шундан сўнг ўрта устунда барча аниқланган пакетлар рўйхати тақдим қилинади. Рўйхатдан **Entity Framework** ни танлаб, уни ўрнатамиз:




**Install** тугмасини босиб, **Entity Framework** пакетини ўрнатамиз.

### Маълумотлар контекстини яратиш

**Entity Framework** пакети ўрнатилгач, маълумотлар контекстини ҳосил қилишимиз лозим. Маълумотлар контекси **Entity Framework** орқали маълумотлар базасига муайян модел асосида мурожаат қилади. Демак, **Models** папкасига янги **BookContext** классини қўшиб қўямиз:

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Data.Entity;
```

```
namespace BookStore.Models
{
    public class BookContext : DbContext
    {
        public DbSet<Book> Books { get; set; }
        public DbSet<Purchase> Purchases { get; set; }
    }
}
```

Контекстни ҳосил қилиш учун янги класс **DbContext** классидан ворисланиши лозим.

```
public DbSet<Book> Books { get; set; }
```

каби аниқланган хусусият маълумотлар базасидан муайян типдаги маълумотлар тўпламини олишда хизмат қилади.

**Code First**

Ушбу мисолда маълумотлар базасидан фойдаланилсада, ундаги объектларни ҳосил қилиш билан шуғулланмаймиз. Ушбу амалларни барчасини биз учун **Entity Framework** бажаради. Ушбу усул **Code First** деб юритилиб, биздаги мавжуд моделлар асосида фреймворк маълумотлар базасида жадвалларни ҳосил қилади.

Лойиҳанинг модел қисмидаги охирги амалимиз маълумотлар базаси билан уланиш сатрини ҳосил қилиш ҳисобланади. Бунинг учун **web.config** файлини очиб, **configSections** секциясини топамиз ва ундан сўнг **connectionStrings** секциясини қўшиб қўямиз:

```
<configSections>
    <connectionStrings>
      <add name="BookContext" connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename='|DataDirectory|\Bookstore.mdf';I
ntegrated Security=True"
   providerName="System.Data.SqlClient" />
    </connectionStrings>
</configSections>
```

Ушбу секцияда маълумотлар базасига йўл аниқланади. **|DataDirectory|** ифодаси маълумотлар базаси **App_Data** папкасида ҳосил қилинишини англатади. Кейинги бўлимларда биз маълумотлар базасига уланишлар ҳақида мулоҳаза юритамиз.

## Контроллер ва кўринишларни ҳосил қилиш

Моделлар ва контекстларни созлашни юқоридаги бўлимда кўриб чиқдик. Энди дастурнинг бошқа компоненти ҳисобланган – контроллерлар билан шуғулланамиз. Контроллерлар учун лойиҳада **Controllers** папкаси мўлжалланган. Янги лойиҳа ҳосил қилинаётган вақтда ушбу папкада **HomeController** контроллери жойлаштирилган. Аммо ушбу контроллерда ҳеч қандай функционал мавжуд эмас. Ушбу контроллердаги код қуйидагича ҳосил қилинган:

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

namespace BookStore.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult About()
        {
            ViewBag.Message = "Your application description page.";

            return View();
        }

        public ActionResult Contact()
        {
            ViewBag.Message = "Your contact page.";

            return View();
        }
    }
}
```

Ушбу контроллерда учта метод автоматик шакллантирилган: **Index, About** ва **Contact**. Бизга бундай контроллерлар керак эмас. Уларга қуйидагича ўзгартиришларни амалга оширамиз:

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

namespace BookStore.Controllers
{
    public class HomeController : Controller
    {
        // маълумотлар контекстини ҳосил қиламиз
        BookContext db = new BookContext();

        public ActionResult Index()
        {
            // маълумотлар базасидаги барча Book объектларини аниқлаймиз
            IEnumerable<Book> books = db.Books;
            // барча объектларни Books динамик хусусиятини ViewBag га
узатамиз
            ViewBag.Books = books;
            // кўринишни қайтарамиз
            return View();
        }
    }
}
```

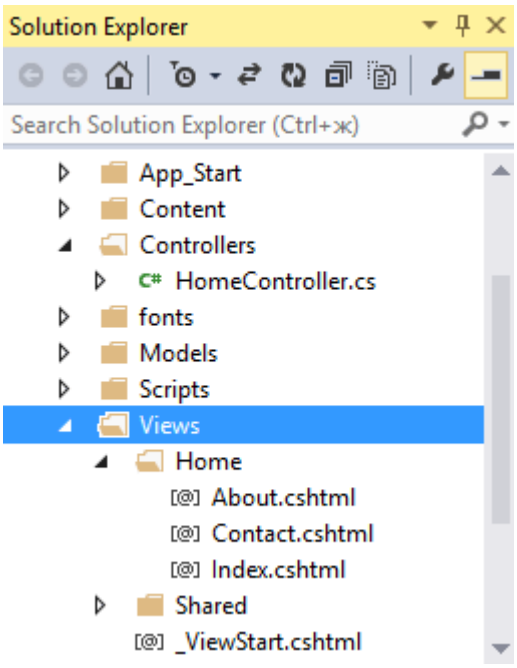
Аввало контроллеримизда моделлар номлар фазоларини юклаб оламиз. Ушбу моделлар турли номлар фазоларида жойлашган бўлиши мумкин. Сўнгра маълумотлар контекси объектини ҳосил қиламиз. Ушбу объект орқали маълумотлар базаси билан алоқа ўрнатилади:

```
BookContext db = new BookContext();
```

Сўнгра **db.Books** методи орқали маълумотлар базасидаги **Book** объектлари рўйхатини шакллантирамиз. Энди ушбу рўйхатни кўринишга узатишимиз лозим.

**Book** объектлари рўйхатини кўринишга узатиш учун **ViewBag** объектидан фойдаланамиз. **ViewBag** объекти орқали ихтиёрий ўзгарувчини аниқлаб, унга қиймат бериб, сўнгра ушбу қийматни кўринишда намойиш қилишни таъминлайди. Бизнинг мисолда **ViewBag.Books** ўзгарувчиси аниқланиб, унда китоблар рўйхати сақланади.

Энди биз янги кўринишни ҳосил қилиб, унда китоблар рўйхатини шакллантиришни амалга оширамиз. Кўринишларни сақлаш учун лойиҳамизда **Views** папкаси мавжуд. Ушбу папкада **Home** контроллерига мос кўрииниш учун қисм каталог яратиб қўйилган. Ушбу қисм каталогда учта кўриниш мавжуд: **About.cshtml, Contact.cshtml** ва **Index.chtml**.




Биринчи иккита кўринишнинг бизга кераги йуқ, шунинг учун уларни ўчириб ташлаш мумкин. **Index.cshtml** кўринишга эса қуйидаги ўзгартиришларни амалга оширамиз:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Китоб магазини</title>
</head>
<body>
    <div>
        <h3>Арзонлаштирилган китоблар</h3>
        <table>
            <tr>
                <td><p>Китоб номи</p></td>
```

```
                <td><p>Муаллиф</p></td>
                <td><p>Нарх</p></td>
                <td></td>
            </tr>
            @foreach (var b in ViewBag.Books)
            {
                <tr>
                    <td><p>@b.Name</p></td>
                    <td><p>@b.Author</p></td>
                    <td><p>@b.Price</p></td>
                    <td><p><a href="/Home/Buy/@b.Id">Харид
килиш</a></p></td>
                </tr>
            }
        </table>
    </div>
</body>
</html>
```

Ушбу файлдаги биринчи ифода **Layout = null** ҳисобланиб, мастер саҳифа ушбу кўринишга қўлланилмаслигини англатади. Кейинги дастурларда мастер саҳифалардан фойдаланишни кўриб чиқамиз. Кейинги сатрлардаги код **html** тилида ёзилган бўлиб, сотилаётган китоблар ҳақидаги маълумотларни чиқаришни амалга оширади. Ушбу **html** кодда

```
@foreach (var b in ViewBag.Books)
```

конструкцияси ҳам қўлланган. Ушбу конструкция **Razor** синтаксисини қўллайди. **Razor** технологияси ҳақида бироздан кейин фикр юритилади. Ҳозирча @ символи орқали **C#/VB.NET** тилида ёзилган синтаксисни қўллашимиз мумкинлигини билишимиз етарли.

Ушбу қисмда цикл ҳосил қилинган бўлиб, циклда **ViewBag.Books** объектидаги барча элементлар олинади. Ушбу объект контроллер методида ҳосил қилинган эди. Ушбу цикл ичида **Razor: @b.Name** синтаксиси орқали ҳар бир элементга мурожаат қилишимиз мумкин.

Жадвалнинг охирги устунида ҳар бир элемент учун `<a href="/Home/Buy/@b.Id">Харид килиш</a>` узатмаси қўшиб қўйилган. Ушбу узатма босилган вақда **Home** контроллеридаги **Buy** методи чақирилади ва **@b.Id** ўрнига китобнинг **id** си узатилади. Ҳозирча бизнинг мисолда **Buy** методи мавжуд эмас, аммо бироздан сўнг ушбу метод шакллантирилади.

**Маршрутлаштириш асослари**

**HomeController** контроллерига мурожаат қилиш учун ва унга сўровни узатиш учун сўров сатрида унинг номини (**Home**) кўрсатишимиз зарур. Шунингдек, контроллер номидан сўнг, слеш белгиси орқали контроллернинг методи ёки амалини кўрсатишимиз лозим. Агар биз бошқа маршрут кўрсатмасак, лойиҳа ишга туширилган вақтда **HomeController** контроллерининг **Index** амалини чақиради.

**/Home/Buy** йўли **HomeController** контроллерининг **Buy** методига мурожаат қилишимизни англатади. Сўровга **/Home/Buy/@b.Id** параметрини узатиш эса ушбу метод параметр қабул қилишини англатади. Ушбу методни ҳосил қилишда аввал дастурни маълумотлар билан тўлдиришимиз зарур.

**Модел учун бошланғич маълумотлар**

**Code First** ёндашуви орқали биз маълумотларни базадан олиш ва унда шакллантириш амалларини бажараётганимиз сабабли, махсус класс орқали маълумотлар базасида зарур амалларни бажарамиз. Бунинг учун **Models** папкасида **BookDbInitializer** классини шакллантирамиз:

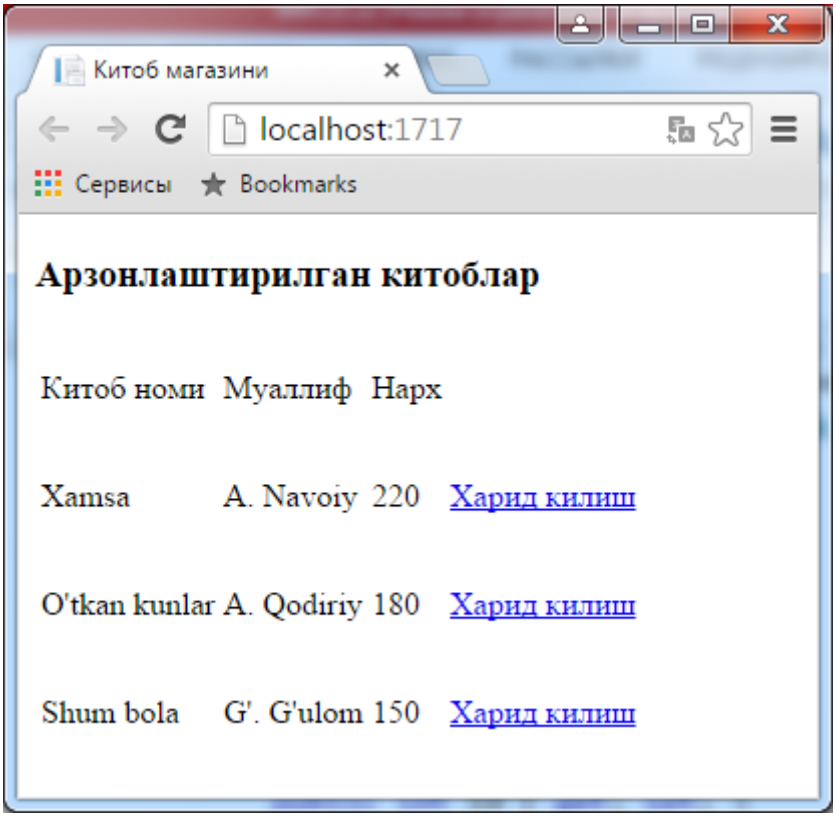
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;

namespace BookStore.Models
{
    public class BookDbInitializer :
DropCreateDatabaseAlways<BookContext>
    {
        protected override void Seed(BookContext db)
        {
            db.Books.Add(new Book { Name = "Xamsa", Author = "A. Navoiy",
Price = 220 });
            db.Books.Add(new Book { Name = "O'tkan kunlar", Author = "A.
Qodiriy", Price = 180 });
            db.Books.Add(new Book { Name = "Shum bola", Author = "G'.
G'ulom", Price = 150 });

            base.Seed(db);
        }
    }
}
```

**DropCreateDatabaseAlways** класси ҳар сафар ишга туширилганда маълумотлар базасини бошланғич маълумотлар билан тўлдиради. Бизнинг мисолда учта **Book** объекти ҳосил қилинади.

**db.Books.Add** методи орқали ҳар бир **Book** объектини маълумотлар базасига қўшиб қўямиз. Аммо ушбу класс тўғри ишлаши учун ва маълумотлар базаси тўлдирилиши учун уни дастур биринчи маротаба ишга туширилган қисмда кўрсатишимиз лозим. Дастурнинг барча созланмалари ва конфигурациялари **Global.asax** файлида жойлашган. Ушуб файлни очиб, **Application_Start** методига қуйидаги сатрни қўшиб қўямиз:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;
using BookStore.Models;
using System.Data.Entity;

namespace BookStore
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            Database.SetInitializer(new BookDbInitializer());

            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }
}
```

Энди лойиҳамизни ишга тушириб, браузерда веб-саҳифани кўришимиз мумкин:




Лойиҳа папкасини очиб (қаттиқ дискда жойлашган), ундаги **App_Data** каталогига кирамиз. Ушбу каталогда **Bookstore.mdf** файли мавжудлигини кўришимиз мумкин.

Энди юқорида келтирилган **Buy** методини шакллантиришимиз мумкин. Ушбу метод орқали танланган китобни сотиб олиш амалга оширилади. **HomeController** контроллерига қуйидаги иккита методни қўшиб қўямиз:

```
[HttpGet]
        public ActionResult Buy(int id)
        {
            ViewBag.BookId = id;
            return View();
        }

[HttpPost]
        public string Buy(Purchase purchase)
        {
            purchase.Date = DateTime.Now;
            // хариф килинган китобларни МБга қўшиб қўямиз
            db.Purchases.Add(purchase);
            // МБдаги ўгаришларни сақлаймиз
            db.SaveChanges();
            return purchase.Person + ", харидингиз учун рахмат!";
        }
```

Ушбу дастурий кодда иккита **Buy** методи мавжуд бўлсада, улар яхлит амални бажаришга хизмат қилади. Биринчи метод сўровни қабул қилишда **GET**, иккинчиси сўровни **POST** орқали қабул қилишда қўлланилади. **[HttpGet]** ва **[HttpPost]** атрибутлари орқали зарурий қайта ишлаш методи кўрсатилади.

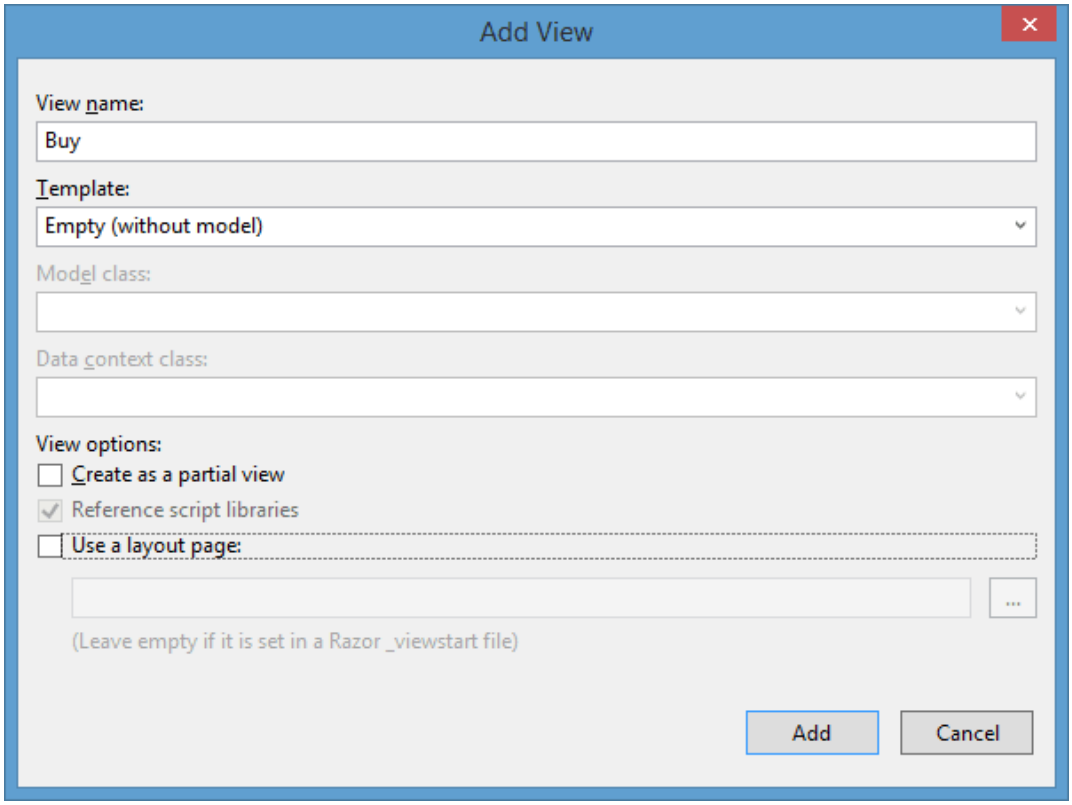
**Buy** методига китобнинг **id** си узатилишини назарда тутиб, биз қуйидаги методни аниқлашимиз лозим: **public ActionResult Buy(int id)**. Сўнгра ушбу параметр **ViewBag** орқали кўринишга узатилади.

**public string Buy(Purchase purchase)** методи бироз мураккаб кўринишга эга. У **POST** сўрови орқали узатилган маълумотни **purchase** моделига қабул қилади ва маълумотлар базасига ёзиб қўяди. Натижа сифатида фойдаланувчи кўриши лозим бўлган сатр ҳосил қилинади.

Маълумотлар базасидаги янги объектни қўшиш **Entity Framework** орқали қуйидагича амалга оширилади:

```
db.Purchases.Add(purchase);
db.SaveChanges();
```

Кейинги қадамда **Buy.cshtml** кўринишини шакллантиришимиз лозим. Бунинг учун методига ўнг тугмани босиб, рўйхатдан **Add View...(Добавить представление)**ни танлаймиз. Натижада қуйидаги мулоқот ойнаси тақдим этилади:



Барча элементларни ўзгартиришсиз қолдириб, **Use a layout page** қисмидан байроқчани олиб ташлаймиз. Чунки ҳали мастер-саҳифалар билан ишлашни кўриб чиққанимиз йўқ. Сўнгра **Add (Добавить)** тугмасини босамиз ва ҳосил бўлган кўринишда қуйидаги ўзгартиришларни амалга оширамиз:

```
@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Харид килиш</title>
</head>
<body>
    <div>
        <h3>Китоб харид килиш</h3>
        <form method="post" action="">
            <input type="hidden" value="@ViewBag.BookId" name="BookId" />
            <table>
                <tr>
                    <td><p>ФИО : </p></td>
                    <td><input type="text" name="Person" /> </td>
                </tr>
                <tr>
                    <td><p>Манзил :</p></td>
                    <td>
                        <input type="text" name="Address" />
                    </td>
                </tr>
                <tr><td><input type="submit" value="Юбориш" />
</td><td></td></tr>
            </table>
        </form>
    </div>
</body>
</html>
```

Китоблар рўйхатидан танланган китобни **«Харид қилиш»** тугмаси босилгач, контроллер **"/Home/Buy/2"** каби форматдаги узатмани қабул қилади ва **Buy** методи **id** параметри **2** қиймати билан чақирилади.

Ушбу турдаги сўров **GET** туридаги сўровни ифодалайди ва фойдаланувчига натижа форма шаклида қайтарилади.

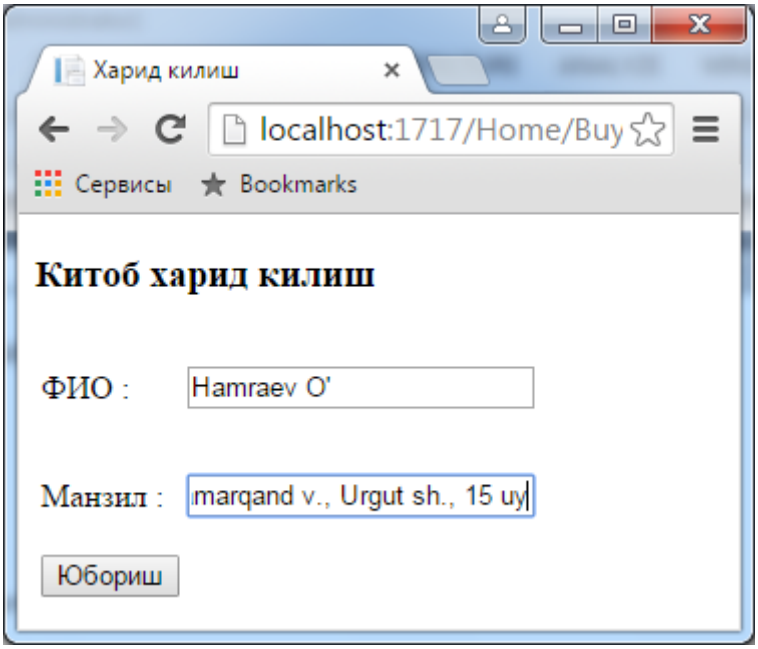
Кўринишлар маълумотларни киритиш формаси сифатида ифодаланади. Биз **BookId** қийматини ўзгартиришимиз лозим эмас, аммо ушбу қиймат **Purchase** моделини шакллантиришда ишлатилади ва ушбу қиймат форма бошида қўшиб қўямиз.

Форма зарур маълумотлар билан тўлдириб, тугма босилганда форма **POST** сўрови орқали маълумотларни узатади. Чунки биз **<form method="post" action="">** сатри орқали аниқлаган эдик. Контроллер яна бир маротаба **Buy** методига сўровни амалга оширади. Аммо эндиликда сўров натижаси **public string Buy(Purchase purchase)** методи орқали қайта ишланади.

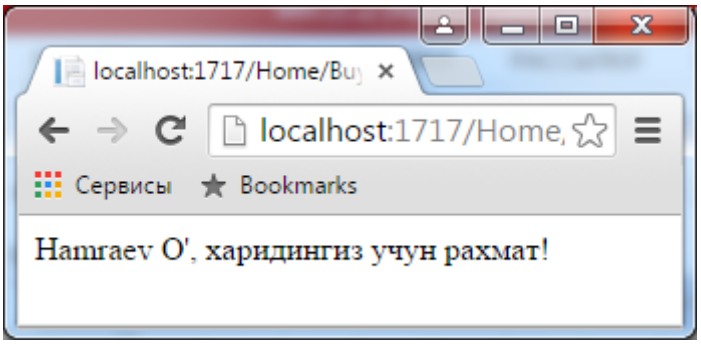
**MVC** тизими **Purchase** моделига **post** сўровини узатганлигимизни қаердан билади? Матнли майдонга қиймат киритишда **<input type="text" name="Person" />** ва **<input type="text" name="Address" />** ларга эътибор беринг.

Уларнинг **name** атрибутлари **Purchase** модели хусусиятлари номларига мос келади. Тугма босилгач, сўров натижасида дастур ушбу майдон қийматларига эга бўлади. **MVC** тизими ўзаро келишув асосида ушбу қийматларни моделдаги мос хусусиятларга беради ва алоҳидаги қийматларни модел хусусияти билан боғлайди.

Энди дастуримизни ишга туширамиз. Бош саҳифадан бирор китобни танлаб, "**Харид қилиш**" тугмасини босамиз. Натижада ҳосил қилинган формадаги барча матнли майдонларни қиймат билан тўлдириб, "**Юбориш**" тугмасини босамиз.



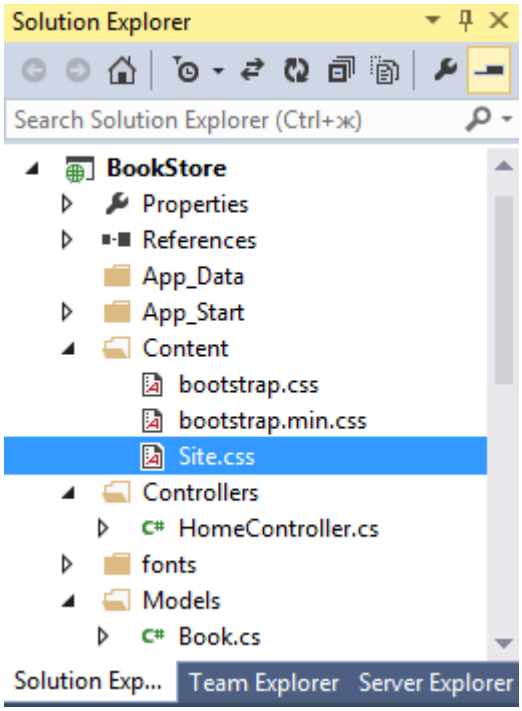
Тугма босилгач, харид ҳақидаги маълумотлар базага сақланади ва браузер бу ҳақда маълумот чиқаради:




Шунинг билан бизнинг кичкина **MVC** лойиҳамиз якунланди. Эндиликда биз дастурларимизга стилларни қўллаш натижасида чиройли шаклга олиб келамиз. Шунингдек, мастер-саҳифалар билан ишлашни кўриб чиқамиз.

## Дастурни стиллаштириш ва мастер-саҳифалар

Бизнинг лойиҳага бироз примитив тарздаги стилизацияни қўллаймиз. Аввало стиллар файлини аниқлаб оламиз. **Visual Studio** муҳити **Content** папкасига **Site.css** стиллар файлини жойлаштириб қўяди.




**Site.css** файлидан ташқари **Content** папкасида **Bootstrap** css-фреймворк файли жойлашган. Аммо ушбу файл ҳозирча бизга керак эмас. **Site.css** файлини очиб унда қуйидаги ўзгартиришларни амалга оширамиз:

```
body {
    font-size: 13px;
    font-family: Verdana, Arial, Helvetica, Sans-Serif;
    background-color: #f7f7fa;
    padding-left:40px;
}

nav{
    display: block;
}

.menu {
    padding-left:10px;
}
.menu ul {
    list-style:none;
}
.menu li {
    display:inline;
}
.menu a:hover {
    color:red;
}
table {
   vertical-align:middle;
   text-align:left;
 }
.header {
    font-weight:bold;
}
 td {
    padding-right:10px;
 }
input {
    width: 150px;
}
```

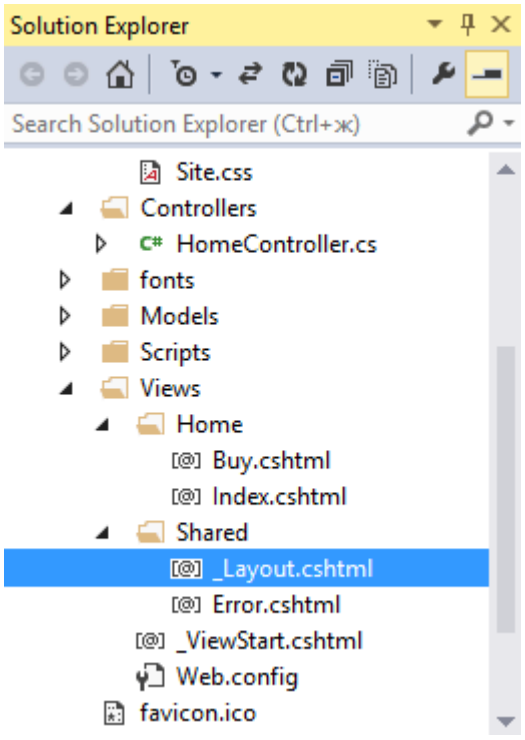
Ушбу файлдаги **.menu** класси сайтдаги менюнинг навигациясини амалга оширишга хизмат қилади. Бизнинг дастур жуда катта бўлмасада, ушбу менюда фақат бош саҳифага узатма жойлашади. Аммо зарурият туғилганда сиз ушбу менюга қўшимча қисмларни жойлаштиришингиз мумкин.

Стиллардан **html**-саҳифаларда фойдаланиш учун қуйидаги дастурий коддан фойдаланиш мумкин:

```
<head>
    <meta name="viewport" content="width=device-width" />
    <link type="text/css" rel="stylesheet" href="../../Content/Site.css"
/>
</head>
```

Бизнинг ҳолда жорий кодни ўзимизнинг иккита кўринишимизга қўшиб қўйишимиз етарли ҳисобланади. Аммо бу яхши усул ҳисобланмай, стиллар иккала кўринишда ҳам умумий бўлганлиги сабабли, баъзи дастурларда кўринишлар бир қанча бўлиши мумкин. Шунинг учун битта стил файлда қилинган ўзгаришни барча кўринишларда ўзгартириб чиқишга тўғри келади.

Ушбу муаммони ҳал қилиш учун **ASP.NET MVC** фреймворки бизга мастер-саҳифалар деб номланувчи функционалликни тақдим қилади. Мастер-саҳифа бир нечта кўринишлар учун умумий шаблонни ифодалайди. Хусусий ҳолда лойиҳада ушбу типдаги мастер-саҳифа мавжуд - **_Layout.cshtml**. Ушбу файлни **Views -> Shared** папкасидан топиш мумкин.




**_Layout.cshtml** файлида қуйидаги ўзгартиришларни амалга оширамиз:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
```

```
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet"
type="text/css" />
</head>

<body>
    <nav>
        <ul class="menu">
            <li>@Html.ActionLink("Главная", "Index", "Home")</li>
        </ul>
    </nav>
    @RenderBody()
</body>
</html>
```

Стилларга уланиш учун **Url.Content** методи ҳам мавжуд. Кейинчалик **бандл** деб номланувчи усул билан ҳам танишамиз.

Мастер-саҳифада **head** секциясидан сўнг, менюни шакллантириш лозим. Бизда иккита кўриниш мавжудлиги сабабли, ягона пункт сифатида менюда бош саҳифага узатма жойлашган. Узатмани шакллантириш учун **Html.ActionLink** методидан фойдаланилади. Ушбу метод орқали узатма генерация қилинади.

Кейинги қадамда **RenderBody()** методини чақириш амалга оширилган. Ушбу метод орқали ушбу жойда муайян кўринишларга узатиш амалга оширилади.

Энди кўринишимизда мастер-саҳифадан фойдаланишни кўриб чиқамиз. **Index.cshtml** кўринишни қуйидаги шаклда ўзгартириб оламиз:

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

<div>
        <h3>Арзонлаштирилган китоблар</h3>
        <table>
            <tr>
                <td><p>Китоб номи</p></td>
                <td><p>Муаллиф</p></td>
                <td><p>Нарх</p></td>
                <td></td>
            </tr>
            @foreach (var b in ViewBag.Books)
            {
                <tr>
```

```
                    <td><p>@b.Name</p></td>
                    <td><p>@b.Author</p></td>
                    <td><p>@b.Price</p></td>
                    <td><p><a href="/Home/Buy/@b.Id">Харид
килиш</a></p></td>
                </tr>
            }
        </table>
</div>
```

Худди шу тартибда **Buy.cshtml** файлида ўзгартиришларни амалга оширамиз:

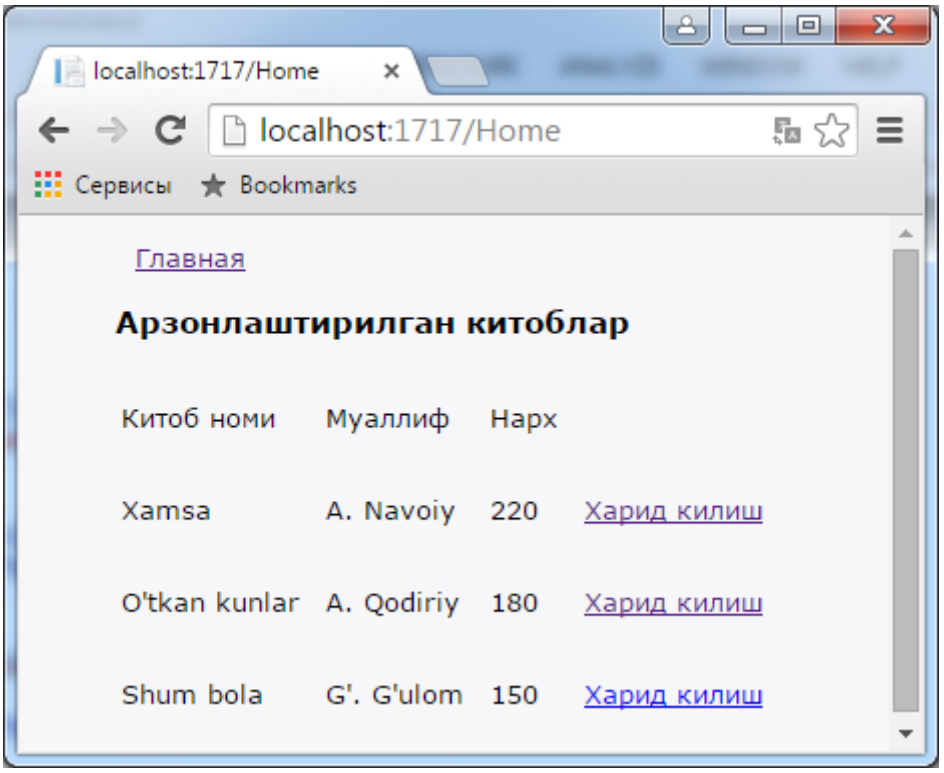
```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div>
        <h3>Китоб харид килиш</h3>
        <form method="post" action="">
            <input type="hidden" value="@ViewBag.BookId" name="BookId" />
            <table>
                <tr>
                    <td><p>ФИО : </p></td>
                    <td><input type="text" name="Person" /> </td>
                </tr>
                <tr>
                    <td><p>Манзил :</p></td>
                    <td>
                        <input type="text" name="Address" />
                    </td>
                </tr>
                <tr><td><input type="submit" value="Юбориш" />
</td><td></td></tr>
            </table>
        </form>
    </div>
```

Ишлатилаётган мастер-саҳифани кўрсатиш учун кўринишнинг бошида мастер-саҳифага йўл ёзилади:

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

Эндиликда **head** ва **body** секциялар кераги йўқ. Ушбу секцияларни ўчириб ташлаш мумкин. Ўзгартиришлар киритилган лойиҳани қайта ишга туширамиз

ва бизнинг дастурда мастер-саҳифалар ва стиллаштиришнинг қўлланганлигини кўришимиз мумкин:




Шунинг билан дастур устида амал бажаришни якунлаб, **MVC** лойиҳанинг асосий компонентларини тўлиқроқ ўрганиб чиқамиз.

# III. Контроллерлар

## Контроллерлар асослари

**MVC** архитектурасининг асосий компоненти контроллер ҳисобланади. Фойдаланувчи киритган маълумотни контроллер қабул қилиб, қайта ишлайда ва кўриниш (ёки бошқа) шаклда қайта узатади.

Контроллерларни ишлатишда баъзи шартлар мавжуд. Аввало контроллерлар номланишида "**Controller**" суффикси билан якунланиши, қолган қисми контроллер номи ҳисобланади.

```
public class HomeController : Controller
      {
            ...
      }
```

Контроллерга веб-браузердан мурожаат қилиш учун адреслар сатрида **сайт_манзили/ контроллер_номи/** каби ёзувни киритиш лозим. Шунингдек, **сайт_манзили/Home/** сўрови орқали тизим **HomeController** контроллеридаги **Index** методини чақиради. Муайн контроллернинг методига сўров ташкил қилмоқчи бўлсак, қуйидаги ифодадан фойдаланиш мумкин:

**сайт_манзили/ контроллер_номи/контроллер_методи**

Мисол, **сайт_манзили/Home/Buy** орқали **HomeController** контроллерининг **Buy** методига мурожаат қилинади.

Контроллер оддий класс ҳисобланиб, **System.Web.Mvc.Controller** бошланғич классидан ворисланади. Шунингдек, **Controller** класси **ControllerBase** абстракт бош классини ва у орқали **IController** интерфейсини тадбиқ қилади. Шу сабабли, контроллер учун ўзимизнинг классимизни ҳосил қилишимиз учун, **IController** интерфейсини тадбиқ қилади ва унинг номида **Controller** суффикси мавжуд.

**IController** интерфейси ягона **Execute** методига эга бўлиб, ушбу метод орқали сўров контексти қайта ишланади.

```
public interface IController
        {
            void Execute(RequestContext requestContext);
        }
```

Энди бирор оддий контроллер ҳосил қилиб, жорий интерфейсни тадбиқ қиламиз. Лойиҳа сифатида аввалга бўлимда келтирилган лойиҳани танлаймиз. Демак, **Controllers** папкасига янги класс ҳосил қиламиз.

```
using System.Web.Mvc;
using System.Web.Routing;

namespace BookStore.Controllers
{
    public class MyController : Controller
    {
        //
        // GET: /My/
        public void Execute(RequestContext requestContext)
        {
            string ip =
requestContext.HttpContext.Request.UserHostAddress;
            var response = requestContext.HttpContext.Response;
            response.Write("<h2>Ваш IP-адрес: " + ip + "</h2>");
        }
     }
}
```

Ихтиёрий контроллерга мурожаат қилинганда, тизим унга сўров контекстини узатади. Ушбу сўров контекстида қуйидаги маълумотлар ўз аксини топган: куки, узатилган форма маълумотлари, сўров сатри, фойдаланувчининг идентификацион маълумотлари ва б.қ. **IController** интерфейси тадбиғи асосида ушбу сўров контекстини **Execute** методидаги **RequestContext** параметри орқали аниқлаш мумкин. Бизнинг ҳолда фойдаланувчи **IP-манзили**ни **requestContext.HttpContext.Request.UserHostAddress** хусусити орқали оламиз.

Шунингдек, фойдаланувчига **Response** объектини ва унинг **Write** методини узатишимиз мумкин.

Фойдаланувчи манзили **сайт_манзили/My/** узатма орқали ўз **IP-манзили**га эга бўлади.

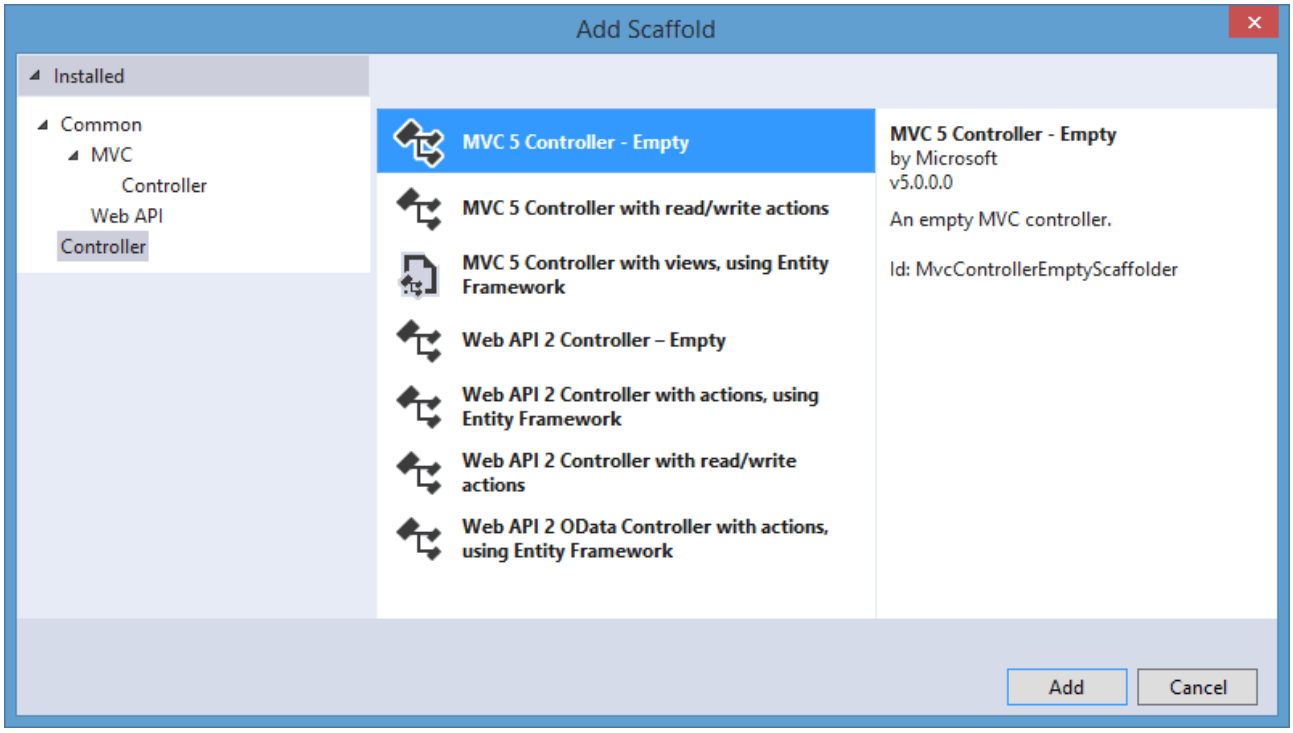
**IController** интерфейси орқали осонгина контроллерларни ҳосил қилиш мумкин. Аммо кўп ҳолларда **Controller** каби юқори даражадаги класслардан фойдаланилади. Чунки ушбу класс сўровларни қайта ишловчи қудратли воситалардан иборат. **IController** интерфейсини тадбиқ қилишда фақатгина **Execute** методи мавжуд бўлса ва барча сўровлар ушбу контроллерга узатилса, ҳамда сўровлар фақатгина битта метод орқали қайта ишланса, **Controller** класси ворисланиши орқали бир қатор методларни шакллантиришимиз мумкин. Ушбу

методлар кирувчи сўровларни қайта ишлаш ва турли шаклдаги натижаларни қайтаришга мўлжалланган.

Стандарт контроллер ҳосил қилиш учун **Controllers** папкасига оддий класс ва **Controller** классидан ворисланган классни қўшиб қўямиз:

```
using System.Web.Mvc;

namespace BookStore.Controllers
{
    public class BookShopController : Controller
    {
        //
        // GET: /BookShop/
        public ActionResult Index()
        {
            return View();
        }
     }
}
```

Аммо **Visual Studio** бизга контроллерларни ҳосил қилишнинг қулай воситаларини ва уларнинг мослашувчан созланмаларини таклиф қилади. Ушбу воситалардан фойдаланиш учун **Controllers** папкасига сичқончанинг ўнг тугмасини босиб, таклиф қилинган менюдан **Add->Controller....** қисмни танлаймиз. Шундан сўнг бизга қуйидаги мулоқот ойнаси тақдим этилади:



**MVC 5** контроллерларига ушбу ойнадаги биринчи учта пункт тегишли бўлиб, қолганлари **Web API 2** га тегишли. Ушбу рўйхатнинг биринчи ўрнида жойлашган **MVC 5 Controller - Empty** қисмни танлаймиз. Ушбу қисм орқали бўш контроллер шакллантирилади. Қолган иккита қисм **CRUD**-функционалликка эга классларни генерация қилади.

Кейинги қадамда янги ҳосил қилинаётган контроллер номи киритилади. Натижада лойиҳага ягона **Index** методидан иборат янги контроллер яратилади. Худди шу усулда яратилган контроллер учун **Views** папкасида мос каталог ҳосил қилинади. Ушбу каталогда ушбу контроллер билан боғлиқ амалларни анклантирувчи кўринишлар сақланади.

## Амал методлари ва уларнинг параметрлари

**Амал методлари (action methods)** – муайян **URL** бўйича сўровга мос контроллернинг методларини ифодалайди. Мисол сифатида, юқоридаги лойиҳани оламиз. Унда қуйидаги контроллер шакллантирилган:

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

namespace BookStore.Controllers
{
    public class HomeController : Controller
    {
        // маълумотлар контекстини ҳосил қиламиз
        BookContext db = new BookContext();

        public ActionResult Index()
        {
            // маълумотлар базасидаги барча Book объектларини аниқлаймиз
            IEnumerable<Book> books = db.Books;
            // барча объектларни Books динамик хусусиятини ViewBag га
узатамиз
            ViewBag.Books = books;
            // кўринишни қайтарамиз
            return View();
        }

        [HttpGet]
```

```
        public ActionResult Buy(int id)
        {
            ViewBag.BookId = id;
            return View();
        }

        [HttpPost]
        public string Buy(Purchase purchase)
        {
            purchase.Date = DateTime.Now;
            // хариф килинган китобларни МБга қўшиб қўямиз
            db.Purchases.Add(purchase);
            // МБдаги ўгаришларни сақлаймиз
            db.SaveChanges();
            return purchase.Person + ", харидингиз учун рахмат!";
        }
    }
}
```

Бу ерда **Index** ва **Buy** лар амал методлари ёки оддийгина қилиб контроллер амаллари ҳисобланади. **/Home/Index** шаклидаги сўров қабул қилинганда контроллер сўровни қайта ишлашни **Index** амалига узатади.

Сўровлар турли шаклда (**GET** ва **POST**) амалга оширилиш сабабли, **ASP.NET MVC** фреймворки қайта ишланаётган сўров типини аниқлаб, унга мос атрибутни қўллайди: **[HttpGet], [HttpPost], [HttpDelete]** ёки **[HttpPut]**. **Buy** амали иккита методга ажратилганлиги сабабли ҳар бир турдаги сўровга мос амал шакллантирилган.

Аммо контроллернинг барча методлари амал методлари бўлмаслиги мумкин. Амал методлари доимо **public** модификаторига эга бўлади. Ёпиқ (**private**) ва ҳимояланган (**protected**) амал методлари мавжуд эмас. Шунингдек контроллерда оддий методлар ҳам мавжуд бўлиши мумкин бўлиб, улар ёрдамчи мақсадда фойдаланилади.

Мисол,

```
[HttpPost]
public string Buy(Purchase purchase)
        {
            purchase.Date = getToday();
            // хариф килинган китобларни МБга қўшиб қўямиз
            db.Purchases.Add(purchase);
            // МБдаги ўгаришларни сақлаймиз
            db.SaveChanges();
            return purchase.Person + ", харидингиз учун рахмат!";
        }
```

```
private DateTime getToday()
        {
            return DateTime.Now;
        }
```

Биз браузерга **Home/getToday/** сўровини амалга ошира олмаймиз. Чунки **getToday()** методи амал методи ҳисобланмайди.

**Контроллерга маълумот ва параметрларни узатиш**

Юқоридаги бўлимда келтирилган дастурда **Buy** методида **purchase** параметри ишлатилган эди. Жорий метод **POST**-сўровларни қайта ишлаганлиги сабабли, биз унга қуйидаги формани узатишимиз мумкин:

```
<form method="post" action="">
        <input type="hidden" value="@ViewBag.BookId" name="BookId" />
        <p>ФИО: </p>
        <input type="text" name="Person" />
        <p>Манзилингизни киритинг: </p>
        <input type="text" name="Address" />
        <input type="submit" value="Отправить" />
</form>
```

Ушбу формадаги барча майдонларнинг **name** атрибути моделдаги хусусият номига мос келади. Шунинг учун тизими майдондаги қийматларни мос хусусиятлар билан боғлаб қўяди. **Buy** методида эса ушбу хусусиятлар рўйхати **Purchase** моделига алмаштирилади.

Бундан ташқари **POST**-сўровларда **GET**-сўровлар ҳам мавжуд бўлиб, бунда барча параметрлар сўровлар сатри орқали узатилади. Масалан, **Buy** методининг иккинчи шаклида параметр сифатида **int** типига мансуб қиймат узатилади:

```
public ActionResult Buy(int id)
```

Стандарт **GET**-сўров қуйидаги форматга эга:
**ресурс_номи?параметр1=киймат1&параметр2=киймат2**

Жорий методга мурожаат **Home/Buy?id=2** форматда мурожаат қилиш мумкин. Метод параметрлари номи сўров сатридаги параметр номи билан мос

бўлиши шарт. Ушбу мослик ўрнатилгач, тизим уларни ўзаро боғлаши мумкин. Аммо биз ушбу параметрни ва қийматни ўз хохишимизга кўра ишлатишимиз мумкин.

Бундан ташқари маршрутлаштириш тизими орқали маршрутлар ҳосил қилиш мумкин. Масалан, **MVC** лойиҳада қуйидагича маршрут аниқланган: **Контроллер/Метод/id.**

Охирги параметр опционал ҳисобланади. Шунга кўра биз **id** параметрни қуйидаги форматда узатишимиз мумкин: **Home/Buy/2**

Мисол сифатида учбурчак юзасини ҳисоблайдиган амални ҳосил қиламиз:

```
public string Square(int a, int h)
        {
            double s = a * h / 2;
            return "<h2>" + a + " асосга ва " + h + " баландикка эга
бўлган учбурчак юзаси " + s + " га тенг.</h2>";
        }
```

Ушбу ҳолда биз амал методига қуйидаги тарзда мурожаат қилишимиз мумкин:

**/Home/Square?a=10&h=3**

Дастур бизга керакли натижани қайтаради.

Шунингдек, параметрларни бошланғич ҳолда ўрнатишимиз ҳам мумкин:

```
public string Square(int a = 10, int h = 3)
        {
            double s = a * h / 2;
            return "<h2>Площадь треугольника с основанием " + a +
                    " и высотой " + h + " равна " + s + "</h2>";
        }
```

Ушбу ҳолда сўров сатрида битта параметрни кўрсатишимиз ёки умуман кўрсатмаслигимиз ҳам мумкин:

**Home/Square?h=5**

### Сўров контекстидан маълумотларини қабул қилиш

Шунингдек, параметрлар, сўров билан боғлиқ бошқа маълумотларни сўров контекси объектларидан олишимиз мумкин. Бизга контекстнинг қуйидаги объектлари маълум: **Request**, **Response**, **RoutedData**, **HttpContext** ва **Server**.

**Request** объекти **Params** коллекциясини ўзида сақлайди. Ушбу коллекцияда сўров орқали узатилган барча параметрлар мавжуд. Уларни қуйидаги тарзда олишимиз мумкин:

```
public string Square()
        {
            int a = Int32.Parse(Request.Params["a"]);
            int h = Int32.Parse(Request.Params["h"]);
            double s = a * h / 2;
            return "<h2>Площадь треугольника с основанием " + a + " и
высотой " + h + " равна " + s + "</h2>";
        }
```

## Амал натижаси

Фойдаланувчи муайян ресурсга мурожаат қилганда у муайянан жавобни кутади. Масалан, муайян маълумотларга эга веб- саҳифа шаклида натижа қайтиши мумкин. Сервер томонида контроллер методи параметрларни қабул қилиб, уларни қайта ишлайди ва муайян жавобни натижа сифатида қайтаради.

Аввалги мавзуда учбурчак юзаси ҳисобланиб, **html**-кодда сатр сифатида натижа қайтарилган эди. Қайтариладиган натижа **ActionResult** классига мос бўлиши лозим.

**ActionResult** абстракт класс ҳисобланиб, ягона **ExecuteResult** методи мавжуд.

```
public abstract class ActionResult
    {
        public abstract void ExecuteResult(ControllerContext context);
    }
```

Ўзимизнинг амал натижамизни яратамиз. Улар жуда оддий бўлиб, бирор бир лойиҳани оламиз ва унга янги **Util** папкасини қўшиб қўямиз. Ушбу папкада бир қатор класслар сақланади. Сўнгра папкада биринчи классни шакллантирамиз. Унга **HtmlResult** деб ном берамиз. У қуйидаги коддан иборат бўлади:

```
using System.Web.Mvc;

namespace BookStore.Util
{
    public class HtmlResult : ActionResult
```

```
    {
        private string htmlCode;
        public HtmlResult(string html)
        {
            htmlCode = html;
        }
        public override void ExecuteResult(ControllerContext context)
        {
            string fullHtmlCode = "<!DOCTYPE html><html><head>";
            fullHtmlCode += "<title>Главная страница</title>";
            fullHtmlCode += "<meta charset=utf-8 />";
            fullHtmlCode += "</head> <body>";
            fullHtmlCode += htmlCode;
            fullHtmlCode += "</body></html>";
            context.HttpContext.Response.Write(fullHtmlCode);
        }
    }
}
```

**HtmlResult** класси конструкторида узатилган **html**-кодни қабул қиламиз ва **Execute** методида уни **html**-саҳифага шакллантирамиз. Сўнгра чиқарилувчи потокка уни ёзиб қўямиз:

**context.HttpContext.Response.Write(fullHtmlCode)**

Ушбу классни ишлатиш учун контроллерда номлар фазосини юклаб оламиз: **using BookStore.Util;** ва янги метод қўшиб қўямиз.

```
public ActionResult GetHtml()
        {
            return new HtmlResult("<h2>Привет мир!</h2>");
        }
```

Энди браузер орқали ушбу методга **Home/GetHtml** каби мурожаат қилиб, **html** саҳифага эга бўламиз. Ушбу мисол жуда содда бўлсада, у орқали контроллерлардаги амалларнинг бажарилиши намойиш қилинган.

Яна битта **ImageResult** натижа классини ҳосил қиламиз. Уни ҳам **Util** папакасига ёзиб қўямиз.

```
using System.Web.Mvc;

namespace BookStore.Util
{
    public class ImageResult : ActionResult
    {
```

```
        private string path;
        public ImageResult(string path)
        {
            this.path = path;
        }
        public override void ExecuteResult(ControllerContext context)
        {
            context.HttpContext.Response.Write("<div
style='width:100%;text-align:center;'>" +
                "<img style='max-width:600px;' src='" + path + "'
/></div>");
        }
    }
}
```

Ушбу класс аввалги классга ўхшаш бўлиб, расмни **html**-кодда ифодалашни амалга оширади. Ушбу амал натижасини ишлатувчи метод қуйидаги кўринишга эга:

```
public ActionResult GetImage()
        {
            string path = "../Images/visualstudio.png";
            return new ImageResult(path);
        }
```

Ушбу кодда лойиҳада **Images** папкаси мавжуд бўлиб, у **visualstudio.png** файлини ўзида сақлайди. Браузерга **Home/GetImage** каби мурожаат қилсак, расмни кўришимиз мумкин.

**ActionResult дан ҳосил қилинган ички класслар**

Баъзи ҳолларда натижа амали учун ўз классларимизни ҳосил қилишга эҳтиёж туғилмайди. **ASP.NET MVC** фрейморки бир қанча натижа амалларидан иборат класслар палитрасини тақдим этади.

- **ContentResult**

Ушбу класс орқали кўрсатилган контентни тўғридан тўғри сатр кўринишида жавоб сифатида қайтаради. Қуйидаги мисолда бу ўз аксини топган:

```
public string Square(int a, int h)
```

```
        {
            int s = a * h / 2;
            return "<h2>Площадь треугольника с основанием " + a +
                    " и высотой " + h + " равна " + s + "</h2>";
        }
```

Ушбу мисолда **ContentResult** ни қуйидагича қўллаш мумкин:

```
public ContentResult Square(int a, int h)
        {
            int s = a * h / 2;
            return Content("<h2>Площадь треугольника с основанием " + a +
                    " и высотой " + h + " равна " + s + "</h2>");
        }
```

Агар биз натижа типи **ContentResult** ўрнига **string** ни ёзсак, фреймворк қайтариладиган тип **ActionResult** объекти эмаслигини аниқлайди ва автоматик тарзда **ContentResult** объекти автоматик ҳосил қилинади.

- **EmptyResult –** бўш қиймат қайтаради.
- **FileResult –** чиқарилувчи потокка бинар жавоб қайтарадиган барча объектлар учун бош класс ҳисобланиб, жўнатиш учун мўлжалланган.
- **FileContentResult – FileResult**дан ҳосил қилинган ворисланган класс бўлиб, жавоб сифатида байтлар массивини қайтаради.
- **FilePathResult – FileResult**дан ҳосил қилинган ворисланган класс бўлиб, кўрсатилган манзилда жойлашган файлга жавобни ёзиб қўяди.
- **FileStreamResult – FileResult**дан ҳосил қилинган ворисланган класс бўлиб, жавоб сифатида бинар потокни қайтаради.
- **HttpStatusCodeResult –** мижозга муайян статусли **HTTP** кодни қайтарадиган натижа амали.
- **HttpUnauthorizedResult** – **HttpStatusCodeResult** дан ворисланган класс бўлиб, мижозга **HTTP 401** кодли жавоб қайтаради. Яъни фойдаланувчи авторизациядан ўтмаганлиги ва сўралган ресурсга рухсати йўқлигини англатади.
- **HttpNotFoundResult – HttpStatusCodeResult** дан ворисланган класс бўлиб, мижозга **HTTP 404** кодли жавоб қайтаради. Яъни сўралган ресурснинг мавжуд эмаслиги ҳақидаги маълумот қайтаради.
- **JavaScriptResult – JavaScript** коддан иборат жавобни қайтаради.

- **JsonResult –** жавоб сифатида **JSON** форматдаги объект ёки объектлар тўпламини қайтаради.
- **PartialViewResult –** чиқарилувчи потокка хусусий кўринишнинг рендерингини қайтаради.
- **RedirectResult –** фойдаланувчини бошқа **URL** га узатади ва вақтинчалик переадресация учун **302** статусли кодини ёки доимий пераадресацияли **301** кодини қайтаради.
- **RedirectToRouteResult – RedirectResult** класси каби фаолият кўрсатади. Аммо фойдаланувчини муайян **URL** манзилига кўрсатилган маршрут бўйича узатади.
- **ViewResult –** кўринишни рендирингини амалга оширади ва рендиринг натижасини **html-саҳифа** шаклида мижозга узатади.

Ушбу классларнинг баъзиларини кўриб чиқамиз.

### ViewResult ва кўринишларни генерация қилиш

**ViewResult** класси контроллерда энг кўп қайтариладиган натижа амали ҳисобланади. У веб-саҳифа кўринишида рендерингни амалга ошириб, уни жавоб сифатида мижозга узатади.

**ViewResult** объектни қайтариш учун **View** методидан фойдаланалади.

```
public class HomeController : Controller
    {
       public ActionResult Index()
        {
           return View();
        }
     }
```

**View** методини чақириш **ViewResult** объектини қайтаради. Сўнгра **ViewResult** муайян кўринишни рендиринг қилиб жавоб сифатида узатади. Контроллер лойиҳадаги кўринишлар рўйхатидан **/Views/ контроллер_номи/ кўриниш_номи.cshtml** манзил бўйича мос кўринишни излайди.

Бошланғич созланмаларга кўра агар кўриниш ошкор тарзда кўрсатилмаган бўлса, кўриниш сифатида контроллер номи билан мос

келадиган кўриниш ишлатилади. Масалан, **Index** контроллери учун **/Views/Home/** папкасидаги **Index.cshtml** кўриниши изланади.

Шунингдек, кўринишни ошкор тарзда кўрсатиш ҳам мумкин:

```
public class HomeController : Controller
    {
       public ActionResult Index()
        {
           return View("Index");
        }
     }
```

Натижада кўриниш сифатида **/Views/Home/Index.cshtml** кўриниши танланади. Шунингдек, тизим излаши лозим бўлган йўлни тўлиқ кўрсатишимиз мумкин:

```
public class HomeController : Controller
    {
       public ActionResult Index()
        {
           return View("~/Views/Some/Index.cshtml");
        }
     }
```

Агар бундай манзилда кўриниш мавжуд бўлмаса, хатолик қайтарилади.

**Контроллердан кўринишга маълумотларни узатиш**

Контроллердан кўринишга маълумот узатишнинг бир қанча усуллари мавжуд. Бу усуллардан бири **ViewData** объектидан фойдаланиш ҳисобланади. **ViewData** калит-қиймат жуфтликларидан иборат словарни ташкил қилади.

```
public ViewResult SomeMethod()
        {
            ViewData["Head"] = "Привет мир!";
            return View("SomeView");
        }
```

Ушбу ҳолда **SomeView.cshtml** кўринишини қуйидагича шакллантириш мумкин:

```
@{
    Layout = null;
```

```
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>SomeView</title>
</head>
<body>
    <div>
        <h2>@ViewData["Head"]</h2>
    </div>
</body>
</html>
```

Маълумотларни кўринишга узатишнинг яна бир усули **ViewBag** объекти ҳисобланади. Ушуб объект турли хусусиятларин аниқлашни ва уларга ихтиёрий қийматни беришни таъминлайди. Юқорида келтирилган мисолга ўзгартиришларни амалга оширамиз:

```
public ViewResult SomeMethod()
        {
            ViewBag.Head = "Привет мир!";
            return View("SomeView");
        }
```

Ушбу методга мос кўриниш қуйидагича:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>SomeView</title>
</head>
<body>
    <div>
        <h2>@ViewBag.Head</h2>
    </div>
</body>
```

```
</html>
```

**ViewBag** объектининг ҳеч қандай **Head** хусусиятига эга эмаслиги муҳим ҳисобланмайди. Ушбу хусусият динамик тарзда аниқланади. Бунда, **ViewBag** хусусияти нафақат **string** ёки **int** каби оддий объектларни ўзида сақлаши, балки мураккаб объектлардан ҳам иборат бўлиши мумкин. Аввалги бўлимда **ViewBag** объектига модел объектлари рўйхатини қўшиб қўйган эдик.

```
public ActionResult Index()
        {
           IEnumerable<Book> books = db.Books;
           ViewBag.Books = books;
           return View();
        }
```

**ViewData** ва **ViewBag** ўхшаш бўлсада, шу ўринда улар эквивалент ҳисобланмайди. Масалан, **ViewBag** дан динамик қийматни кўринишдаги кенгайтириш методларига узатиб бўлмайди.

Масалан, биз **@Html.TextBox("name", ViewBag.Name)**каби ёза олмаймиз. Чунки **C#** компилятори ҳар бир параметрнинг типини компиляция жараёнида билиши лозим. Ушбу ҳолда биз

**ViewData:@Html.TextBox("name", ViewData["Name"])**

ёки

**@Html.TextBox("name", (string)ViewBag.Name)** ни қўллашимиз лозим.

Контроллердан кўринишга маълумот узатишда **TempData** объектидан фойдаланиш мумкин. **TempData** объекти **ViewData** каби калит-қиймат жуфтликларидан иборат словарни ифодалайди. **TempData** объекти узатилган қийматни ва сўровни сақлаши мумкин. **TempData** объекти **ViewData** каби ишлатилади.

## Хатолик ва статус кодини узатиш, бошқа манзилга узатиш

Икки турдаги бошқа манзилга узатиш усули мавжуд: **вақтинчалик** ва **доимий**. Бошқа манзилга узатиш турига қараб сервер браузерга икки хил **HTTP** кодидан бирини узатади.

- **301** статусли код орқали доимий бошқа манзилга узатиш амалга оширилади. Бундай турдаги узатишда сўралаётган ҳужжат бошқа манзилга жойлаштирилган бўлиши керак. Ушбу турдаги статусли кодни қабул қилган браузер сўровларни янги ресурсга мослаштиради. Шунинг учун бундай турдаги бошқа манзилга узатиш тавсия этилмайди.
- **301** статусли код орқали вақтинчалик бошқа манзилга узатиш амалга оширилади. Вақтинчалик бошқа манзилга узатишда сўралаётган ҳужжат вақтинчалик бошқа саҳифага жойлаштирилади.

Иккала ҳолда ҳам бошқа манзилга узатишда **RedirectResult** объекти ишлатилади. Аммо ушбу объектни қайтарадиган метод фарқ қилади.

Вақтинчалик бошқа манзилга узатишда **Redirect** методи қўлланилади.

```
public RedirectResult SomeMethod()
        {
            return Redirect("/Home/Index");
        }
```

Доимий бошқа манзилга узатишда **RedirectPermanent** методи қўлланилади.

```
public RedirectResult SomeMethod()
        {
            return RedirectPermanent("/Home/Index");
        }
```

Бунда методдан **RedirectResult** объектини қайтаришимиз шарт эмас. Баъзида бошланғич талабларга кўра муайн манзилга ёки бошқа ресурсга узатишга тўғри келади. Масалан, фойдаланучи авторизация қилинган бўлса, сўралган веб-саҳифа, акс ҳолда логин учун бошқа саҳифага узатилади.
Мисол:

```
public ActionResult Buy(int id)
        {
            if (id > 2)
            {
                return Redirect("/Home/Index");
            }
            ViewBag.BookId = id;
            return View();
        }
```

Параметр сифатида **2** дан катта қиймат берилса, **Home/Index** манзилига редирект амалга оширилади. Бошқа ҳолларда фойдаланувчига кўриниш узатилади.

Бошқа манзилга узатишнинг яна бир класси **RedirectToRouteResult** ҳисобланади. Ушбу класс орқали бошқа манзилга узатиш кенгроқ амалга оширилади. У иккита метод орқали амалга оширилади: **RedirectToAction** ва **RedirectToRoute**.

**RedirectToRoute** методи домен ичида муайян маршрут бўйича бошқа манзилга узатишни амалга оширади:

```
public RedirectToRouteResult SomeMethod()
        {
            return RedirectToRoute(new { controller = "Home", action = 
"Index" });
        }
```

**RedirectToAction** методи муайян контроллернинг муайян методига мурожаат қилади.

```
public RedirectToRouteResult SomeMethod()
        {
            return RedirectToAction("Square", "Home", new { a = 10, h = 
12 });
        }
```

**Redirect/RedirectToAction** методлари вақтинчалик бошқа манзилга узатишни амалга оширади. Уларга ўхшаш доимий бошқа манзилга узатишни амалга оширувчи методлари ҳам мавжуд: **RedirectPermanent/RedirectToActionPermanent**. Улар бажарадиган амаллар **Redirect/RedirectToAction** методларига ўхшаш бўлиб, фарқи уларнинг браузерга **301** статусли коди узатиши ҳисобланади. **RedirectPermanent** ва **RedirectToActionPermanent** методларини ишлатиш тавсия этилмайди. Нотўғри созланган доимий бошқа манзилга узатиш изловчи тизимларда яхши позиция бўмаслигига олиб келади.

**Хатолик ва статусли кодларни қайтариш**

Баъзи ҳолларда мумкин бўлмаган бирор ресурсга мурожаат қилинганда хатоликни узатишга тўғри келади. Агар ресурсга доступ мавжуд бўлмаса, **mvc**-фреймворк ушбу амалга автоматик тарзда статусли код орқали жавоб қайтаради.

Аммо баъзи ҳолларда қабул қилинган сўровга аниқроқ амални бажаришга тўғри келади. Масалан, бизнинг лойиҳада мавжуд контентларга фойдаланувчиларнинг ёшига чеклагич қўйилган бўлсин. Биз фойдаланувчи ёшини сўраймиз ва агар у чеклагични қаноатлантирмаса, статусли кодли хатоликни қайтарамиз.

```
public ActionResult Check(int age)
        {
            if (age < 21)
            {
                return new HttpStatusCodeResult(404);
            }
            return View();
        }
```

Ушбу амал методи қуйидагича чақирилади: **http://localhost:2044/Home/Check?age=7.**

Худди юқоридаги усулда биз браузерга бошқа статусли кодни узатишимиз мумкин.

Алтернатив тарзда **HttpNotFound** методи орқали **HttpNotFoundResult** объектини қайтаришимиз мумкин.

```
public ActionResult CheckAge(int age)
        {
            if (age < 21)
            {
                return HttpNotFound();
            }
            return View();
        }
```

Ушбу амал методи қуйидагича чақирилади: **http://localhost:2044/Home/CheckAge?age=17.**

Статусли кодларни узатишнинг яна бир усули бу **HttpUnauthorizedResult** класси ҳисобланади. У фойдаланувчига муайян ресурсга доступи йўқлигини хабар қилади ва браузерга **401** статусли кодни қайтаради.

```
public ActionResult AgeCheck(int age)
        {
            if (age < 21)
            {
                return new HttpUnauthorizedResult();
            }
            return View();
        }
```

Ушбу амал методи қуйидагича чақирилади: **http://localhost:2044/Home/AgeCheck?age=17.**

## ASP.NET MVC 5 орқали файлларни узатиш

Файлларни мижозга узатиш учун **FileResult** классидан фойдаланилади. Ушбу класс абстракт бўлгани сабабли, биз ундан ворисланган класслар билан иш кўрамиз:

- **FileContentResult**: файлдан ўқиб олинган байтлар массивини мижозга узатади;
- **FilePathResult**: сервер томонидан файлни узатишни амалга оширади;
- **FileStreamResult**: ушбу класс **System.IO.Stream** поток объектини ҳосил қилади ва файлни мижозга узатади.

Юқоридаги учта ҳол учун ҳам **File** методидан фойдаланилади. Ушбу метод **FileResult** объектини қайтаради. Танланган усулга қараб ушбу методнинг перегрузка қилинган варианти чақирилади.

Файллар тизимидан файлни узатиш (**FilePathResult** объектни ишлатиш) учун биз **File** методида учта параметрни кўрсатишимиз лозим: **сервер томонидан файлга йўл**, **файл мазмуни** ва **файл номи** (қабул қилувчи томон учун). Параметрлар рўйхатидаги файл номини келтириш шарт эмас. Биринчи иккита параметр эса мажбурий параметрлар ҳисобланади.

```
public FileResult GetFile()
        {
            // Файлга йўл
            string file_path = Server.MapPath("~/Files/PDFIcon.pdf");
            // Файл тури - content-type
            string file_type = "application/pdf";
            // Файл номи
            string file_name = "PDFIcon.pdf";
            return File(file_path, file_type, file_name);
        }
```

Бизнинг лойиҳада **Files** папкаси ва **PDFIcon.pdf** унда файли мавжуд деб фараз қилинади. **Server.MapPath** методи лойиҳа каталогига тўлиқ йўлни ҳосил қилади. Шунингдек, ушбу йўлнинг ўрнига абсолют йўлдан ҳам фойдаланиш мумкин. Бунинг учун файллар системасидаги ихтийрий файлга

**string file_path = @"C:\Book\PDFIcon.pdf";**

каби мурожаат қилиш мумкин.

**Home/GetFile** каби контроллердаги амалга мурожаат қилинганда, жорий файлни бизнинг компьютерга сақлаш таклиф қилинади.

Худди шу усулда **FileContentResult** класси иш бажаради. Фақат **File** методида файл номи ўрнига байтлар массиви кўрсатилади:

```
// Отправка массива байтов
public FileResult GetBytes()
        {
            string path = Server.MapPath("~/Files/PDFIcon.pdf");
            byte[] mas = System.IO.File.ReadAllBytes(path);
            string file_type = "application/pdf";
            string file_name = "PDFIcon.pdf";
            return File(mas, file_type, file_name);
        }
```

Агар биз **FileStreamResult** объектини қайтармоқчи бўлсак, **File** методидаги биринчи аргумент сифатида **Stream** объектини узатишимиз лозим.

```
// Отправка потока
public FileResult GetStream()
        {
            string path = Server.MapPath("~/Files/PDFIcon.pdf");
            // Объект Stream
            FileStream fs = new FileStream(path, FileMode.Open);
```

```
            string file_type = "application/pdf";
            string file_name = "PDFIcon.pdf";
            return File(fs, file_type, file_name);
        }
```

## HttpContext сўрови контексти. Куки. Сессиялар.

Баъзи ҳолларда сўров контексти ҳақидаги ахборотни қайта ишлашга тўғри келади. Ушбу турдаги ахборотлар: **фойдаланувчи браузери**, **IP-манзили**, **қайси саҳифа**дан фойдаланувчи бизнинг саҳифага келганлиги ва ҳ.з.

**ASP.NET MVC** орқали ушбу турдаги ахборотларни **HttpContext** объектини ишлатиш орқали олиш мумкин.

Контроллер орқали **HttpContext** хусусиятига эга бўлган **ControllerContext** объектига мурожаат қилишимиз мумкин бўлса-да, улар ўртасида бироз фарқ ҳам мавжуд. **HttpContext** объекти муайян **http-сўров** маълумотларини тавсифлайди. **ControllerContext** эса жорий контроллерга мос бўлган **http-сўров** ни ифодалайди.

Сўров контекстидаги барча маълумотларга биз **HttpContext** объекти хусусияти орқали мурожаат қилишимиз мумкин. Чунки сўровдаги барча маълумотлар **Request.HttpContext.Request** хусусиятида мавжуд. У эса ўз навбатида **HttpRequestBase** классидан ворисланган ва ушбу класснинг барча хусусиятларини ўзида сақлайди. Уларнинг баъзиларини кўриб чиқамиз:

- Фойдаланувчи браузери ҳақидаги маълумотлар: **HttpContext.Request.Browser**
- Баъзи ҳолларда браузерлар ҳақидаги маълумот етарли бўлмайди. Ушбу ҳолда фойдаланувчи агентига мурожаат қилиш мумкин: **HttpContext.Request.UserAgent**
- **url** сўровга эга бўлиш: **HttpContext.Request.RawUrl**
- Фойдаланувчи **IP-адреса** адреси: **HttpContext.Request.UserHostAddress**
- Рефererга эга бўлиш: **HttpContext.Request.UrlReferrer == null ? "" : HttpContext.Request.UrlReferrer.AbsoluteUri.** Реферер мавжуд бўлмаслиги сабабли унинг **null** га тенглигини текширамиз.

**Мисол:** (**http://localhost:2044/Home/MyIndex**)

```
public string MyIndex()
        {
            string browser = HttpContext.Request.Browser.Browser;
            string user_agent = HttpContext.Request.UserAgent;
            string url = HttpContext.Request.RawUrl;
            string ip = HttpContext.Request.UserHostAddress;
            string referrer = HttpContext.Request.UrlReferrer == null ?
"" : HttpContext.Request.UrlReferrer.AbsoluteUri;
            return "<p>Browser: " + browser + "</p><p>User-Agent: " +
user_agent + "</p><p>Url запроса: " + url +
                "</p><p>Реферер: " + referrer + "</p><p>IP-адрес: " + ip
+ "</p>";
        }
```

**Жавобни узатиш**

Агар **HttpContext.Request** сўров ҳақидаги маълумотни ўзида сақласа, **HttpContext.Response** хусусият жавобни бошқаради. У **HttpResponse** объектини ифодалаб, мижоз томонга бир қанча қийматларни узатади: **куки, хизматчи сарлавҳалар** ва **html** шаклидаги жавоб.

Масалан, жавоб кодировкасини қуйидагича ўрнатамиз: **HttpContext.Response.Charset = "iso-8859-2";**

**HttpResponse** объекти методи жавобни бошқаришга масъул. Масалан, **AddHeader** методи жавобга қўшимча сарлавҳа қўшади.

Шунингдек, биз мижозга жавоб узатишимизда албатта **View()** методини чақиришимиз шарт эмас. Биз **HttpContext.Response.Write()** методидан бемалол фойдаланишимиз мумкин. **(http://localhost:2044/Home/ContextData)**

```
public string ContextData()
        {
            HttpContext.Response.Write("<h1>Hello World</h1>");

            string user_agent = HttpContext.Request.UserAgent;
            string url = HttpContext.Request.RawUrl;
            string ip = HttpContext.Request.UserHostAddress;
            string referrer = HttpContext.Request.UrlReferrer == null ?
"" : HttpContext.Request.UrlReferrer.AbsoluteUri;
            return "<p>User-Agent: " + user_agent + "</p><p>Url запроса:
" + url +
```

```
                "</p><p>Реферер: " + referrer + "</p><p>IP-адрес: " + ip
+ "</p>";
        }
```

Ёки:

```
public void ContextData()
        {
            HttpContext.Response.Write("<h1>Hello World</h1>");
        }
```

Ушбу кодда **HttpContext.Response.Write** методи орқали потокка муайян сатрни узатади. Аммо реал дастур тузаётганда, **ActionResult** объектларни генерация қилиш мақсадга мувофиқ.

**Фойдаланувчини аниқлаш**

Шунинг **HttpContext** объектида фойдаланувчи ҳақидаги маълумотлар **HttpContext.User** хусусиятида сақланади.

```
bool IsAdmin = HttpContext.User.IsInRole("admin");
// фойдаланувчининг администраторлар оиласига тегишли эканлиги аникланади
bool IsAuth = HttpContext.User.Identity.IsAuthenticated;
// фойдаланувчининг аутентификация қилинганми
string login = HttpContext.User.Identity.Name;
// авторизация қилинган фойдаланувчи логини
```

Фойдаланувчининг идентификацияси ҳақидаги маълумотлар кейинги бобларда кўриб чиқилади.

**Куки билан ишлаш**

**HttpContext.Request.Cookies** хусусияти орқали кукига эга бўлиш мумкин.

```
public void GetCookie()
        {
            string id = HttpContext.Request.Cookies["id"].Value;
        }
```

Ушбу ҳолда мижоз томонидан **"id"** кукисида қиймат мавжуд бўлса, унинг қийматини олишимиз мумкин. Аммо куки қийматини олишдан аввал унинг

қийматини ўрнатиш лозим. Кукига қиймат бериш учун **HttpContext.Response** хусусиятидан фойдаланишимиз мумкин. Масалан, "**id**"га қиймат бериш учун
**HttpContext.Response.Cookies["id"].Value = "ca-4353w";**
дан фойдаланиш мумкин.

**Сессиялар**

Сессиялар худди кукига ўхшаб, бирор маълумотни сақлаш учун ишлатилади. Ушбу маълумотга дастурнинг ихтиёрий жойидан мурожаат қилиш мумкин. Сессиялар билан ишлаш учун **Session** объектидан фойдаланилади. Биз контроллернинг бирор методига сессия қийматини ўрнатамиз:

```
public ActionResult MyIndex()
        {
            Session["name"] = "Tom";
            return View();
        }
```

Бошқа усул орқали ушбу қийматга эга бўлиш мумкин:

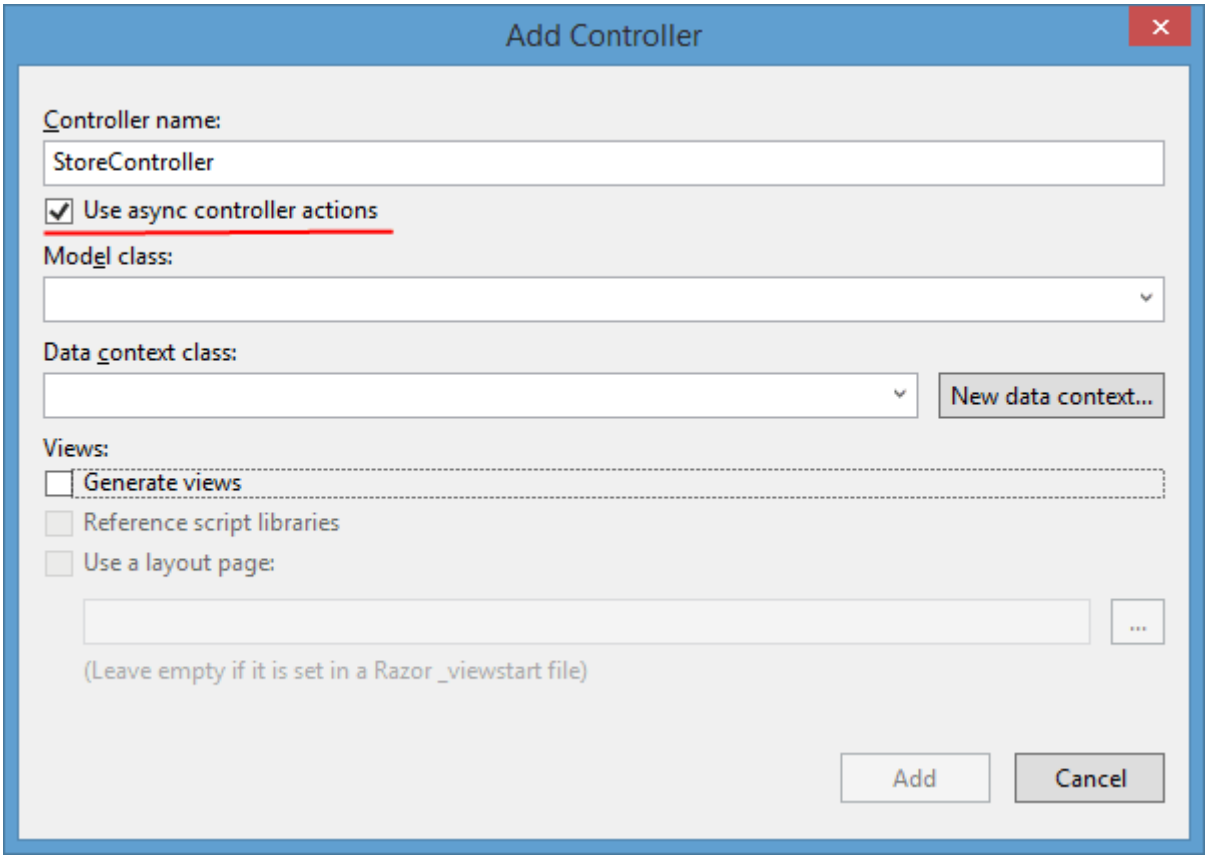
```
public string GetName()
        {
            var val = Session["name"];
            return val.ToString();
        }
```

Агар биз ушбу сессиядаги **name** калити қийматини ўчирмоқчи бўлсак,
**Session["name"]=null;**
ифодадан фойдаланишимиз лозим.

## Асинхрон методлар

**.Net** фреймворкнинг охирги версиясида асинхрон методларни қўллаб қувватлаш амалга оширилган. **Task Parallel Library** библиотекаси яратилиши натижасида асинхронликни ташкил этиш анча қулай ҳолга келтирилди ва асинхрон методлар билан ишлаш формати ўзгартирилди. **async** ва **await** калит сўзлар орқали асинхрон методларни яратиш имконияти ишлаб чиқилди.

Янги контроллерни ишлаб чиқишда созлаш жараёнида биз унинг синхрон ёки асинхрон тарзда ташкил қилиниши кўрсатилади. **Visual Studio** муҳити стандарт контроллерларга **ActionResult** типдаги объектларни қайтаришни шакллантиради. Янги контроллер ҳосил қилаётганда **Controller with views, using Entity Framework** қисмни танлаймиз ва унинг асинхрон методи мавжудлигини кўрсатамиз:




Котроллерларда асинхрон методлар нима учун керак?

Асинхрон методлар дастурнинг самарадорлигини оширади ва узоқ вақт қайта ишланиши зарур бўлган сўровларни қайта ишлашда ишлатилади. Масалан, маълумотлар базаси ёки тармоқ ресурсига мурожаат қилганда, катта ҳажмдаги маълумотлар қабул қилинади. Асинхрон методларни қўллаш дастурда параллел равишда бошқа сўровларни амалга ошириш имконини беради.

Асинхрон ва синхрон методларнинг фарқини англаш учун **IIS** нинг кирувчи сўровларни қайта ишлаш механизмини кўриб чиқамиз. Веб-серверда сўровларни қайта ишловчи потоклар пули қўллаб қувватланади. **IIS** даги веб-ресурсга фойдаланувчи мурожаат қилганда, жорий сўровга мос пулдаги поток

ажратилади. Жорий поток жорий сўровни қайта ишламагунча, бошқа сўровлар қайта ишланмайди.

Контроллер методи сўровни қайта ишлаш жараёнида бошқа ресурсга ёки маълумотлар базасига сўровни амалга ошириши лозим бўлсин. Тармоқ ресурсига ёки МБсига сўров бироз вақтни талаб қиилиш мумкин. Потокни синхрон қайта ишлашда, қайта ишланаётган сўров тармоқ ресурси ёки МБ зарур натижани қайтармагунча, вақтинчалик блокировка қилинади.

Агар сўровни қайта ишлаш узоқ муддат давом этса, **IIS** бошқа кирувчи сўровларни қайта ишлашни бошлайди. Аммо потоклар сонига чеклагич мавжуд бўлиб, потоклар сони максимал сонга яқинлашганда, янги ҳосил қилинган сўровлар навбатга жойлаштирилади. Аммо навбатдаги сўровлар сонига ҳам чеклагич мавжуд. Ушбу қиймат ҳам максимал қийматга яқинлашгач, **IIS** бошқа барча сўровларни **503 (Service Unavailable)**статусли код орқали қабул қилмайди.

Асинхрон тарзда потокни қайта ишлашда МБ маълумотни қайтаришни кутмасдан, бошқа фойдаланувчи сўровини қайта ишлайди. Тармоқ ресурси ёки МБдан зарур маълумотлар қабул қилингач, поток аввал қайта ишланаётган сўровни оддий режимга қайтаради.

Дастур кодига қайтамиз. Асинхрон методларни яратишда **async** ва **await** модификаторларидан фойдаланилади. Ушбу модификаторлар орқали асосий потокни блокировка қилмасдан узоқ муддада бажариладиган амалларни бажаради. Синхрон ва асинхрон методларни чақиришни кўриб чиқамиз: (**http://localhost:2044/Home/Index** ва **http://localhost:2044/Home/BookList**)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using BookStore.Models;
using System.Threading.Tasks;
using System.Data.Entity;


namespace BookStore.Controllers
{

    public class HomeController : Controller
    {
```

```
        // маълумотлар контекстини ҳосил қиламиз
        BookContext db = new BookContext();

        public ActionResult Index()
        {
            IEnumerable<Book> books = db.Books;
            ViewBag.Books = books;
            return View();
        }

        // асинхронный метод
        public async Task<ActionResult> BookList()
        {
            IEnumerable<Book> books = await Task.Run(() => db.Books);
            ViewBag.Books = books;
            return View("Index");
        }


    }
}
```

Иккала метод ҳам бир хил амални бажаради. Яъни маълумотлар базасидан маълумотларни олади ва бир хил натижага эришади. **Index** синхрон методи оддий ёзувни ифодаласа, **BookList** асинхрон методи эса бошқача кўринишга эга.

**BookList** асинхрон методи **ActionResult** объектини ўрнига **Task<ActionResult>** объектини қайтаради. **Task** узоқ давом этадиган асинхрон амални ифодалайди.

Шунингдек, ушбу методни асинхрон тарзда ташкил қилиш учун қайтариладиган типдан аввал **async** калит сўзи ёзилади.

Учинчи калит сўзи сифатида **await** сўзидан фойдаланилади. Ушбу калит сўзи асинхрон методларга қўлланилади ва зарурий натижа қабул қилингунча кутилади. Бизнинг мисолда **МБ**даги маълумотларга ушбу усул орқали эга бўиш амалга оширилган.

Шунингдек, **Task** объекти қайтарадиган методларга **await** калит сўзи ишлатилади. Шунинг учун МБдан маълумотларни олишда

```
await Task.Run(() => db.Books)
```

ифодасидан фойдаланилади.

**Task.Run** методи параметр сифатида бирор бир лямба ифодани қабул қилади. Ушбу амал орқали МБдан маълумотлар қабул қилинади: `db.Books.`

**Entity Framework 6** нинг янги версиясида алоҳидаги методларга ҳам асинхронлик шакли берилади. Юқоридаги мисолдаги асинхрон **BookList** методига қуйидагича ўзгартиришларни амалга оширамиз:

```
public async Task<ActionResult> BookList()
        {
            ViewBag.Books = await db.Books.ToListAsync();
            return View("Index");
        }
```

**db.Books.ToListAsync()** методи объектлар рўйхатини қайтади. **Entity Framework 6** бизга МБ билан ишлаш учун бир қатор асинхрон методларни таклиф қилади.

Асинхрон методларни қачон ишлатиш зарур?

Биринчи навбатда улардан маълумотлар базаси ва тармоқ ресурслари билан ишлашда фойдаланиш зарур. Аммо қўйилган масалага мос равишда дастурчи методни синхрон ёки асинхрон тарзда ўзи шакллантиради.

# IV. Кўринишлар

## Кўринишлар асоси

**MVC** дастури контроллерлар орқали бошқарилсада, фойдаланувчига дастур кўриниш шаклида намойиш қилинади. Кўринишлар орқали дастурнинг ташқи шакли ҳосил қилинади. **ASP.NET MVC 5**да кўринишлар **cshtml** кенгайтмасига эга бўлган файллар бўлиб, асосан **html** тилида ёзилган дастурчи кодидан иборат бўлади. Стандарт кўриниш қуйидагича:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>SomeView</title>
</head>
<body>
    <div>
        <h2>@ViewBag.Message</h2>
    </div>
</body>
</html>
```

Кўринишларда асосан **html** код мавжуд бўлсада, у **html**-саҳифа ҳисобланмайди. Дастур компиляция жараёнида кўриниш аввало C# тилида ёзилган классга ўтказилади ва компиляция қилинади. Юқорида келтирилган кўриниш қуйидаги шаклидаги классга генерация қилинади:

```
#pragma checksum "c:\users\hp\documents\visual studio
2013\Projects\MVC\BookStore\BookStore\Views\Home\SomeView.cshtml" "{ff1816ec-aa5e-4d10-87f7-
6f4963833460}" "1F05F4D370C9D00F8CBDFB8BD1F51D74189D0617"

namespace ASP {
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Linq;
    using System.Net;
    using System.Web;
    using System.Web.Helpers;
    using System.Web.Security;
```

```
    using System.Web.UI;
    using System.Web.WebPages;
    using System.Web.Mvc;
    using System.Web.Mvc.Ajax;
    using System.Web.Mvc.Html;
    using System.Web.Optimization;
    using System.Web.Routing;
    using BookStore;

    public class _Page_Views_Home_SomeView_cshtml : System.Web.Mvc.WebViewPage<dynamic>
    {

    #line hidden

    public _Page_Views_Home_SomeView_cshtml() {
    }

    protected ASP.global_asax ApplicationInstance {
    get {
    return ((ASP.global_asax)(Context.ApplicationInstance));
    }
    }

    public override void Execute() {
    BeginContext("~/Views/Home/SomeView.cshtml", 0, 2, true);

    WriteLiteral("\r\n");

    EndContext("~/Views/Home/SomeView.cshtml", 0, 2, true);


    #line 2 "c:\users\hp\documents\visual studio
2013\Projects\MVC\BookStore\BookStore\Views\Home\SomeView.cshtml"

    Layout = null;


    #line default
    #line hidden
    BeginContext("~/Views/Home/SomeView.cshtml", 27, 48, true);

    WriteLiteral("\r\n\r\n<!DOCTYPE HTML>
    \r\n\r\n
    <html>
    \r\n
    <head>
        \r\n
        <meta"); endcontext("~ />Views/Home/SomeView.cshtml", 27, 48, true);

        BeginContext("~/Views/Home/SomeView.cshtml", 75, 16, true);

        WriteLiteral(" name=\"viewport\"");

        EndContext("~/Views/Home/SomeView.cshtml", 75, 16, true);

        BeginContext("~/Views/Home/SomeView.cshtml", 91, 29, true);

        WriteLiteral(" content=\"width=device-width\"");

        EndContext("~/Views/Home/SomeView.cshtml", 91, 29, true);

        BeginContext("~/Views/Home/SomeView.cshtml", 120, 74, true);
```

```
        WriteLiteral(" />\r\n
        <title>SomeView</title>\r\n
    </head>\r\n
    <body>
        \r\n    <div>
            \r\n       <h2>
                ");

                EndContext("~/Views/Home/SomeView.cshtml", 120, 74, true);

                BeginContext("~/Views/Home/SomeView.cshtml", 195, 15, false);


                #line 15 "c:\users\hp\documents\visual studio
2013\Projects\MVC\BookStore\BookStore\Views\Home\SomeView.cshtml"
                Write(ViewBag.Message);


                #line default
                #line hidden
                EndContext("~/Views/Home/SomeView.cshtml", 195, 15, false);

                BeginContext("~/Views/Home/SomeView.cshtml", 210, 38, true);

                WriteLiteral("
            </h2> \r\n
        </div>\r\n
    </body>\r\n
</html>\r\n");

EndContext("~/Views/Home/SomeView.cshtml", 210, 38, true);

        }
    }
}
```

Ушбу код ўқишга қулай бўлмасада, компиляция жараёнида **System.Web.Mvc.WebViewPage<T>** классидан ворисланган класс ҳосил қилинади. Бу ерда **T** – ишлатиладиган класс модели ҳисобланади.

Кўриниш қатъий типлашмаган бўлсада, модел класси номи ўрнига калит **dynamic** сўзи ишлатилган. Ушбу класснинг барча амаллари **Execute** методида жойлашган. Ушбу методда **WriteLiteral** методи ёрдамида барча мавжуд кўринишлардаги **html** разметкалар қайта ишланади.

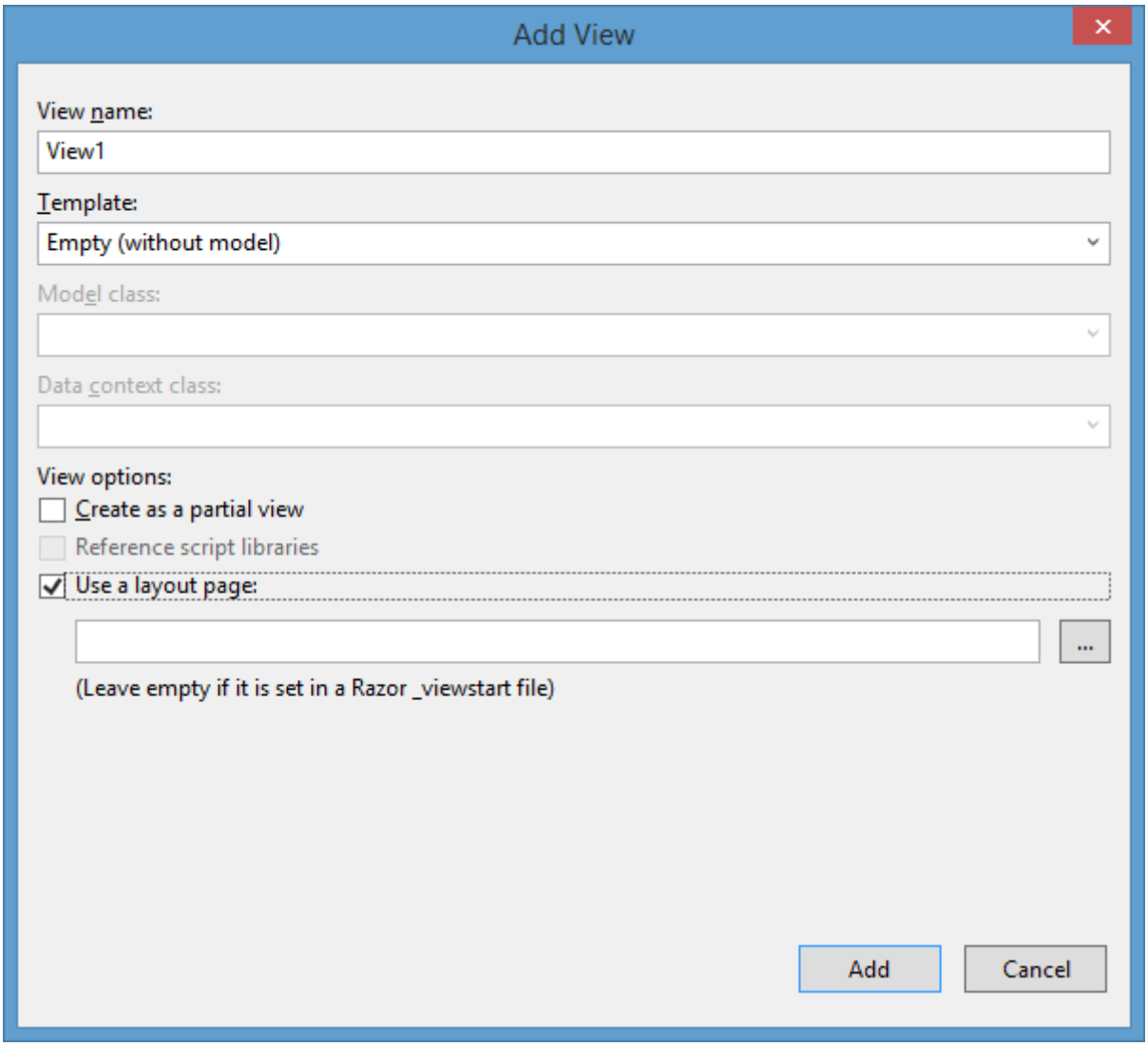
Генерация қилинган кўринишларни **C:\Users\Логин_Номи\AppData\Local\Temp\Temporary ASP.NET Files\root.** йўлдан топиш мумкин.

Барча папкалар шифрланган номга эга бўлиб, дастурга мос зарурий папкани аниқлаш учун бир қанча вақт сарфлашга тўғри келади. Шунингдек, генерация қилинган файллар ҳам шифрланган форматга эга. Юқорида

келтирилган класс **App_Web_nri53fza.1.cs** файли каби **root\307f1c1d\a36bbd4f** папкасида жойлашган бўлиши мумкин.

**Янги кўриниш ҳосил қилиш**

Жорий лойиҳада янги кўринишни ҳосил қилиш учун лойиҳадаги **Views** папкасини ва унда жойлашган **Home** папкасини топиб сичқончанинг ўнг тугмасини босамиз **Add→View (Представление)** қисмни танлаймиз. Натижада қуйидаги мулоқот ойнаси ҳосил қилинади:




Ушбу мулоқот ойнасидаги элементларни кўриб чиқамиз:

- **View Name**: **янги кўриниш номи**. Ушбу ном орқали ҳосил қилинган кўринишга автоматик тарзда **cshtml** кенгайтмаси берилади.
- **Template**: **янги ҳосил қилинаётган кўриниш шаблони**. Биз шаблонлар орасидан бирини танлашимиз лозим:

- **Empty (without model)**: бошланғич ҳолатдаги бўш кўриниш ҳосил қилинади.
- **Empty**: бўш кўриниш ҳосил қилинади. Ушбу ҳолда @model директиваси орқали моделни танлашимиз мумкин.
- **Create**: моделдаги янги объектларга мос кўриниш генерация қлиинади. Ушбу формада ҳар бир хусусиятга мос майдон ҳосил қилинади.
- **Delete**: генерируется представление с формой для удаления объектов модели. В этой форме для каждого свойства модели создается отдельное поле.
- **Details**: моделдаги барча хусусмиятларни ўзида сақлайдиган кўриниш генерация қилинади.
- **Edit**: моделдаги объектлардан иборат бўлган ва созлаш имконияти мавжуд кўриниш генерация қилинади. Ушбу формада ҳар бир хусусият учун алоҳида майдон ҳосил қилинади.
- **List**: моделда мавжуд барча объектларни ўзида акслантирадиган жадвалдан иборат кўриниш ҳосил қилинади. Ушбу кўринишдаги объектларни генерация қилиш учун контроллер методидан **IEnumerable<Тип_модели>** типдаги қийматни узатиш лозим. Шунингдек, ушбу кўринишда методларга ҳосил қилиш/ўзгартириш/ўчириш амалларига мос узатмалар шакллантирилади.

- **Model class**: Юқоридаги опциялардан бири танланганда (**Empty (without model)** дан ташқари) бирор модел классига мос қатъий типлаштирилган кўриниш генерация қилинади.
- **Data context class**: Юқоридаги опциялардан бири танланганда (**Empty (without model)** дан ташқари) маълумотлар контекстини танлаш имконияти тақдим этилади.
- **Create as a partial view**: қисмий кўринишларни яратишга имкон беради.
- **Reference Script Libraries**: ушбу опция орқали кўринишда **jQuery** библиотекаси ва бошқа **JavaScript** файлларини автоматик қўшиш мумкин.
- **Use a layout page**: Ушбу опция орқали кўринишда мастер-саҳифа ёки оддий саҳифа ишлатиши кўрсатилади. Ушбу опция ўрнатилгач, мастер-саҳифа танланиши мумкин. **Razor** технологияси учун мастер-саҳифани кўрсатиш шарт ҳисобланмайди. Агар сиз мастер-саҳифани ишлатмоқчи бўлсангиз, **_ViewStart.cshtml** дан фойдаланилади. Аммо мастер-саҳифани қайта аниқламоқчи бўлсангиз, ушбу опциядан фойдаланишингиз мумкин.

**Кўриниш файлларига йўл**

Ҳосил қилинаётган кўринишлар контроллерлар бўйича гуруҳларга ажратилади ва **Views** каталогидаги мос папкага сақланади. **Home** контроллерига мос кўринишлар лойиҳадаги **Views/Home** папкасида жойлашган. Аммо эҳтиёж туғилганда биз **Views** каталогида янги папка ҳосил қилишимиз ва унга мос қўшимча кўринишни сақлашимиз мумкин.

Чиқарилувчи потокдаги кўринишга рендерингни амалга ошириш учун **View()** методидан фойдаланилади. Агар ушбу метод орқали кўриниш номи кўрсатилмаса, амал методи номига мос кўриниш чақирилади. Қуйидаги мисолда, амал методи орқали **Index.cshtml** кўринишга мурожаат қилинади:

```
public ActionResult Index()
        {
            IEnumerable<Book> books = db.Books;
            ViewBag.Books = books;
            return View();
        }
```

Кўриниш сақланадиган мос йўлни ошкор тарзда кўрсатиш мумкин:

```
public ActionResult Index()
        {
            IEnumerable<Book> books = db.Books;
            ViewBag.Books = books;
            return View("~/Views/Some/SomeView.cshtml");
        }
```

**Razor синтаксиси**

Стандарт кўринишлар оддий веб-саҳифаларга ўхшаш бўлиб, бир нечта **html** тэглардан иборат коддан иборат. Аммо ушбу кодда **C#** тилида ёзилган коддан фойдаланиш мумкин. Ушбу кодлар @ белгиси орқали шакллантирилади. Ушбу белги **Razor** технологияси орқали **C#** тилида ёзилган кодга ўтказилади. **Razor** технологиясидаги систаксис

**Razor**

Контроллерда **View** методи чақирилгач, контроллер кўринишни рендеринг ёки **html** файлни генерация қилмайди. Контроллер зарурий маълумотлар ва кўриниш учун мос **ViewResult** объекти қайтарилади. Сўнгра **ViewResult** объекти **Razor** га мурожаат қилиб кўриниш рендерингини амалга оширади ва натижани қайтаради.

Аввалги версияларда **ViewResult** ва **Visual Studio** икки турдаги кўриниш движокларини (**Web Forms** ва **Razor**) қўллаб қувватлаган бўлса, ҳозирги версияда **Razor** асосий движок сифатида ишлатилади. **Razor** ни қўллаш **C#,** кодини чақиришда синтаксисни камайтиришга олиб келади.

**Razor** янги тил ҳисобланмасдан, у кўринишни рендеринг қилишда **html** разметкани **C#** кодга генерация қилади.

**Razor синтаксиси асоси**

**Razor** синтаксисини қўллашда код олдидан **@** белгиси қўйилади. Шу белгидан сўнг **C#** тилида ёзилган код келтирилади. Икки турдаги код келтирилиши мумкин: **ифода** ва **блок** коди. Ифода кодига мисол:

```
<p>@b.Name</p>
```

**Razor** технологияси **Name** хусусият **b** объектга мансублигини аниқлайди. Шунингдек, стандарт класс ва методлардан фойдаланиш мумкин:

```
<h3>Сана: @DateTime.Now.ToShortTimeString()</h3>
```

Кўринишда блокни ишлатиш учун **@** белгисидан фойдаланилади:

```
@foreach (var b in ViewBag.Books)
            {
                <tr>
                    <td><p>@b.Name</p></td>
                    <td><p>@b.Author</p></td>
                    <td><p>@b.Price</p></td>
                </tr>
            }
```

Кўринишда кодлар блокини шакллантириб, ушбу блокда ўзгарувчиларни C# тилида эълон қилиш каби шакллантириш мумкин.

```
@{
    string head = "Салом дунё!!!";
    head = head + " Сайтга хуш келибсиз!";
}
<h3>@head</h3>
```

## Катъий типлаштирилган кўринишлар

Аввалги мисолларда контроллердан кўринишга ахборот узатишда **ViewBag** объектидан фойдаланилди.

```
@foreach (var b in ViewBag.Books)
            {
                <tr>
                    <td><p>@b.Name</p></td>
                    <td><p>@b.Author</p></td>
                    <td><p>@b.Price</p></td>
                    <td><p><a href="/Home/Buy/@b.Id">Харид
килиш</a></p></td>
                </tr>
            }
```

Ушбу дастурий код орқали **ViewBag.Books** коллекция элементларига мурожаат қилинган. **var** калит сўзи орқали ўзгарувчи ошкормас тарзда эълон қилинган. Худди шу кодни модел типини ошкор тарзда эълон қилишимиз мумкин:

```
@foreach (BookStore.Models.Book b in ViewBag.Books)
            {
                <tr>
                    <td><p>@b.Name</p></td>
                    <td><p>@b.Author</p></td>
                    <td><p>@b.Price</p></td>
                    <td><p><a href="/Home/Buy/@b.Id">Харид
килиш</a></p></td>
                </tr>
            }
```

**ViewBag** объекти билан ишлаш қулай ҳисобланиб, қатъий типлашган кўринишларни ташкил қилиш орқали **View** методи параметри орқали маълумотларни узатиш мумкин. Контроллердаги метод коди қуйидагича кўринишга эга бўлиши мумкин:

```
// маълумотлар контекстини ҳосил қиламиз
BookContext db = new BookContext();

public ActionResult Index()
        {
            return View(db.Books);

        }
```

Энди кўринишни параметр орқали узатилган қиймат билан боғлаш учун **@model** директивасидан фойдаланилади. **books** ўзгарувчиси **IEnumerable<Book>** типга мансуб бўлганлиги сабабли, кўриниш қуйидаги шаклда ҳосил қилинади:

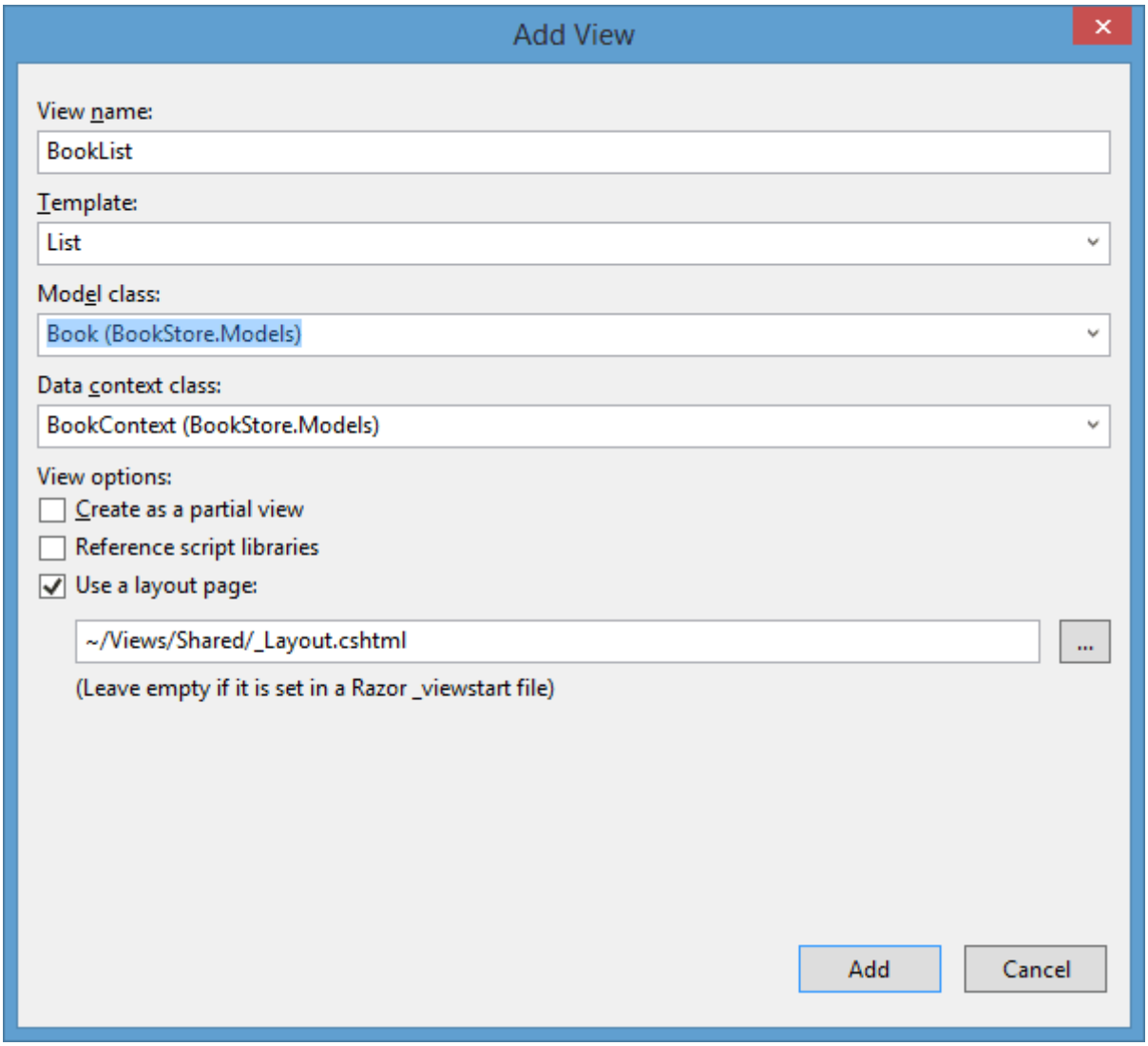
```
@model IEnumerable<BookStore.Models.Book>
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<div>
    <h3>Распродажа книг</h3>
    <table>
        <tr class="header">
            <td><p>Название книги</p></td>
            <td><p>Автор</p></td>
            <td><p>Цена</p></td>
            <td></td>
        </tr>
        @foreach (BookStore.Models.Book b in Model)
        {
            <tr>
                <td><p>@b.Name</p></td>
                <td><p>@b.Author</p></td>
                <td><p>@b.Price</p></td>
                <td><p><a href="/Home/Buy/@b.Id">Купить</a></p></td>
            </tr>
        }
    </table>
</div>


</body>
</html>
```

**Model** объекти **@model** директивасида кўрсатилган модел типини ифодалайди ва контроллердан узатилган маълумотларни сақлайди.

Модел типини тўлиқ ёзмаслик учун кўринишда номлар фазосини импорт қилишимиз мумкин:

```
@using BookStore.Models
@model IEnumerable<Book>
```

Бундан ташқари, қатъий типлаштирилган кўринишни автоматик яратиш учун кўриниш ҳосил қилинаётан вақтда қуйидаги параметрларни танлаш лозим:




Бунинг учун **Template** майдонида ихтиёрий бошқа шаблонни танлаш мумкин. **Empty (without model)** дан ташқари зарур модел класси ва маълумотлар контекстини кўрсатиш мумкин. Агар биз **List** шаблонни танласак,

автоматик генерация қилинган кўриниш ўзининг функционалига кўра юқори келтирилган мисолга ўхшаш бўлади.

## Мастер саҳифалар

Мастер саҳифалар ягона, унифицирланган шаклдаги сайтларни яратишда ишлатилади. Мастер-саҳифалар бошқа кўринишларни ўзида сақламайдиган кўринишлар ҳисобланади. Мисол сифатида мастер саҳифаларни бошқа кўринишлар учун умумий ҳисобланиб, умумий стил ва скриптларни ўзида сақлайди.

Натижада ҳар бир кўринишда стиллар файлига йўлни кўрсатишимиз ва зарурият туғилганда ўзгартиришимиз шарт эмас. Мастер саҳифаларни тўлдирувчиси ёки плейсхолдерлар орқали бошқа кўринишларда ишлатиш мумкин.

**ASP.NET MVC 5** янги лойиҳасида **_Layout.cshtml** деб номланган мастер-саҳифа ҳосил қилинади. Ушбу мастер-саҳифани **Views/Shared** каталогидан топиш мумкин. Ушбу бўлимдаги лойиҳада биз уни қуйидагича ўзгартирган эдик:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet"
type="text/css" />
</head>

<body>
    <nav>
        <ul class="menu">
            <li>@Html.ActionLink("Бош сахифа", "Index", "Home")</li>
        </ul>
    </nav>
    @RenderBody()
</body>
</html>
```

Биринчи кўринишда бу оддий кўринишга ўхшайди. Аммо ушбу кўринишда **@RenderBody()** методи ишлатилган бўлиб, ушбу қисмда жорий мастер саҳифадан фойдаланадиган бошқа кўринишлар жойлаштирилади.

Натижада веб-дастурлар учун ягона стил ишлаб чиқишимиз мумкин. Кўринишда мастер-саҳифани қўллаш учун **Layout** секциясида зарур мастер-саҳифа кўрсатилиши лозим. **Index.cshtml** кўринишида мастер-саҳифа қуйидагича қўлланилиши мумкин:

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

Агар жорий кўринишда мастер-саҳифалар ишлатилмаса, **Layout = null;** каби коддан фойдаланилади.

Шунингдек, барча кўринишлар учун ягона мастер-саҳифадан фойдаланиш шарт эмас. Бир нечта мастер-саҳифаларни шакллантириб, турли кўринишларга қўллаш мумкин. Масалан, форум учун алоҳида, блог учун алоҳида мастер саҳифадан фойдаланиш мумкин. Мастер-саҳифалар ҳам оддий саҳифалар каби лойиҳага қўшилади.

Мастер-саҳифада бир қанча секциялардан иборат бўлиб, ушбу қисмларда мастер-саҳифа мазмуни жойлаштирилади. Мастер саҳифадаги **footer** секцияси қуйидагича шакллантирилиши мумкин:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet"
type="text/css" />
</head>

<body>
    <nav>
        <ul class="menu">
            <li>@Html.ActionLink("Бош сахифа", "Index", "Home")</li>
        </ul>
    </nav>
    @RenderBody()
    <footer>@RenderSection("Footer")</footer>
</body>
```

```
</html>
```

Эндиликда аввалги келтирилган мисолдаги **Index** кўриниши чақирилганда, хатолик қайтарилади. Чунки **Footer** секцияси аниқланмаган. Мастер-саҳифасининг ҳар бир секцияси мазмуни шакллантирилиши лозим. Шунинг учун **Index** кўринишга **footer** секциясини қўшиб қўямиз. Бунинг учун **@section** ифодасидан фойдаланамиз:

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

...

@section Footer {
    Барча ҳуқуқлар ҳимояланган. Syte Corp. 2015.
}
```

Ушбу ёндашув орқали агар бизда бир қанча кўринишлар мавжуд бўлса ва биз мастер-саҳифада янги секция ҳосил қилмоқчи бўлсак, биз барча мавжуд кўринишларни ўзгартиришимиз лозим. Бу эса бир қанча ноқулайликларни келтириб чиқариши мумкин. Бизнинг ҳолда секцияларни созлаш вариантларидан бирини танлашимиз лозим.

Биринчи ҳолда **RenderSection** методининг перегрузка қилинган вариантидан фойдаланишимиз мумкин. Бу усул орқали секцияни кўринишда аниқлаш шарт эмас. **Footer** секциясини қуйидагича шакллантиришимиз лозим:

```
<footer>@RenderSection("Footer", false)</footer>
```

Иккинчи усул орқали секцияни қуйидагича шакллантириши мумкин:

```
<footer>
    @if (IsSectionDefined("Footer"))
    {
        @RenderSection("Footer")
    }
    else
    {
        <span> footer элементини мазмуни</span>
    }
</footer>
```

**ViewStart**

Агар бизнинг лойиҳада икки учта кўриниш мавжуд бўлса, ҳар бирида мастер-саҳифани **Layout** секцияда тавсифлаш мумкин. Агар турли мастер-саҳифаларни ва бир қанча кўрринишларни ишлатиш лозим бўлса, бу ноқулайликларни келтириб чиқарииши мумкин.

Кўринишларда мослашувчан созлашни амалга ошириш учун **_ViewStart.cshtml** саҳифасидан фойдаланамиз. Ушбу саҳифанинг коди у жойлашган каталогдаги барча кўринишлар учун бажарилади. Ушбу файл битта каталогда жойлашган ҳар бир кўриниш учун кетма-кет қўлланилади.

**ASP.NET MVC 5** лойиҳасини ҳосил қилаётган вақда **Views** каталогида автоматик тарзда **ViewStart.cshtml** файл файллантирилган. Ушбу файл барча кўринишлар учун умумий мастер-саҳифани ифодалайди.

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

Ушбу код барча кўринишларда бажарилади. Шунинг учун **Layout** секциясини бошқа барча кўринишлардан олиб ташлашимиз мумкин. Агар кўринишда бошқа мастер-саҳифани ишлатмоқчи бўлсак, **Layout** хусусиятини қайта аниқлашимиз лозим.

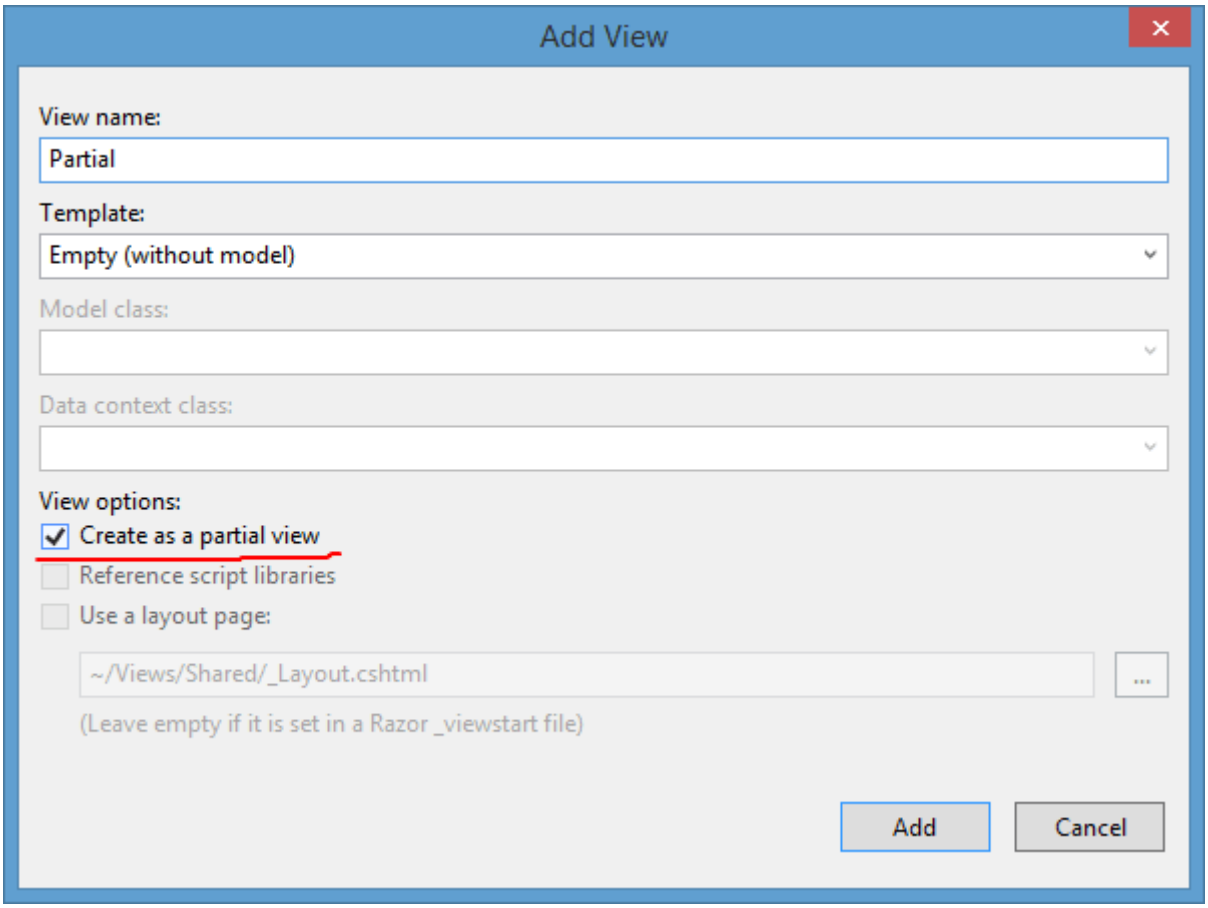
## Қисмий кўринишлар

Амал методи оддий кўринишлардан ташқари қисмий кўринишларни ҳам қайтариши мумкин. Қисмий кўринишларни бошқа кўринишларда қўшиб қўйиш мумкин. Уларни бошқа кўришларда зарурий жойда қўллаб, **AJAX**-сўров натижасини рендиринг қилиш мумкин.

Қисмий кўринишларни рендеринг қилиш учун **PartialView** методи орқали қайтарилувчи **PartialViewResult** объекти жавоб беради:

```
public ActionResult Partial()
        {
            ViewBag.Message = "Бу қисмий кўриниш.";
            return PartialView();
        }
```

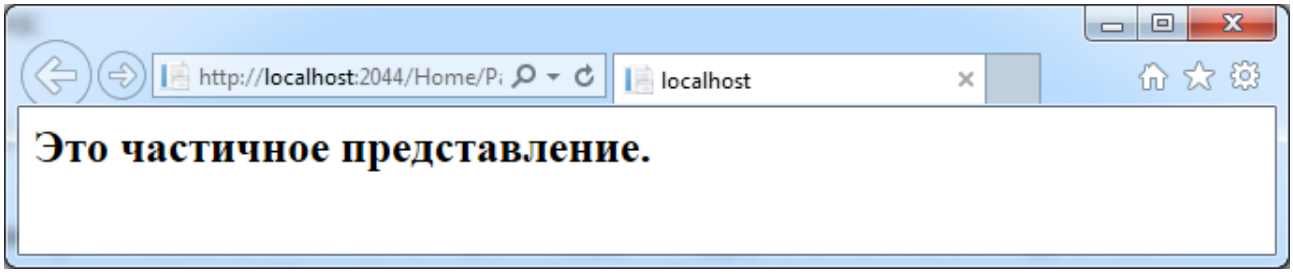
Энди янги **Partial.cshtml** кўринишни шакллантирамиз. Бунинг учун янги кўринишни ҳосил қилиш вақтида қуйидаги параметрларни созлаймиз:




Бундан сўнг лойиҳада бўш янги қисмий кўриниш файли ҳосил қилинади. Ушбу файл оддий кўриниш файли мазмуни каби бўлиб, аммо ушбу кўринишга мастер-саҳифани мос қўйиб бўлмайди. Қисмий кўринишга қуйидаги сатрни қўшиб қўямиз:

```
<h2>@ViewBag.Message</h2>
```

Шундан сўнг **Partial** амалига контроллердаги оддий амал каби мурожаат қилишимиз мумкин. (**http://localhost:2044/Home/Partial**)



Қисмий кўринишларни бошқа бирор кўринишга жойлаштиришимиз мумкин. Бунинг учун **Html.Partial** каби хелперни бошқа кўринишнинг ихтиёрий жойига жойлаштириш мумкин.

**@Html.Partial("Partial")**

Ушбу ҳолда параметр сифатида қисмий кўриниш номини кўрсатишимиз лозим. Ушбу ҳолда **Partial** методидан **ViewBag.Message** ни узатишимиз мумкин эмас. Аммо биз қисмий кўринишга ушбу қийматни узатмоқчи бўлсак, уни бош кўриниш билан боғлиқ бўлган контроллер методидан узатиш мумкин. Масалан, биз **Index.cshtml** кўринишда қисмий кўринишни ишлатаётган бўлсак, **Index** методида қуйидагича ёзиш мумкин:

```
public ActionResult Index()
        {
            ViewBag.Message = "Это вызов частичного представления из
обычного";
            return View();
        }
```

**ViewBag.Message** орқали хабар бош ва қисмий кўринишга узатилади. **Html.Partial** хелпердан ташқари қисмий кўринишни бошқа **Html.RenderPartial** хелпер орқали ташкил қилиш мумкин. Ушбу хелпер шунингдек, кўриниш номини қабул қилади. Фақат у **Razor** сатрий ифодаси шаклида ишлатилмайди.

**@{Html.RenderPartial("Partial");}**

Ушбу иккита усулнинг ўзаро фарқи **Html.RenderPartial** чинг чиқарилувчи потокка тўғридан-тўғри ёзишида ҳисобланади. Шунинг учун у **Html.Partial** дан бироз тезроқ ишлайди.

Оддий кўринишлар каби қатъий типлашган қисмий кўринишларни **@model** директивасини кўрсатиб, ҳосил қилишимиз мумкин.

```
@model IEnumerable<string>
<h2>Мамлакатлар рўйхати</h2>
<ul>
    @foreach (string t in Model)
    {
```

```
        <li>@t</li>
    }
</ul>
```

Ушбу ҳолда биз ушбу кўринишни қуйидагича чақиришимиз мумкин:

```
@Html.Partial("Partial", new string[] { "Russia", "USA", "Canada", "France" })
```

## HTML-хелперлар

Аввалги келтирилган мисоллардан шу маълумки, мазмунни шакллантиришда кўринишлар **html** разметкалардан фойдаланади. Аммо **ASP.NET MVC** фреймворкда **HTML-хелпер** деб номланувчи **html-код** ни генерация қилувчи қудратли инструмент мавжуд.

### Сатрли хелперлар

Сатрли хелперлар **C#** тилида ёзилган оддий методларга ўхшайди. Аммо улар **@helper** тэги билан бошланади.

```
@helper BookList(IEnumerable<BookStore.Models.Book> books)
{
    <ul>
        @foreach (BookStore.Models.Book b in books)
        {
            <li>@b.Name</li>
        }
    </ul>
}
```

Ушбу хелперни кўринишнинг ихтиёрий жойида ишлатишимиз мумкин. Шунингдек, кўринишнинг ихтиёрий жойида **IEnumerable<BookStore.Models.Book>** объекти орқали узитиш мумкин.

```
<h3>Список книг</h3>
@BookList(ViewBag.Books)
<!—ёки катъий типлаштирилган кўриниш ишлатилган бўлса -->
@BookList(Model)
```

Сатрли **html**-хелперлардан кўринишда бир неча мартоба методларни ишлатиш лозим бўлганда фойдаланилади.

```
@helper CreateList(string[] all)
{
    <ul>
        @foreach (string s in all)
        {
            <li>@s</li>
        }
    </ul>
}

@{
    string[] cities = new string[] { "Лондон", "Париж", "Москва" };
}

@{
    string[] countries = new string[] { "Великобритания", "Франция",
"Россия" };
}

<h3>Города</h3>
@CreateList(cities)
<br />
<h3>Страны</h3>
@CreateList(countries)
```

Зарур бўлган хелпер мавжуд бўлмаса, биз рўйхатни шакллантириш учун **html-код**ни кўпайтиришимиз лозим. Аммо ушбу хелпер жуда оддий бўлиб, биз эҳтиёж туғилганда мураккаб бўлган хелперларни шакллантиришимизга тўғри келади.

Аммо ушбу усулнинг битта камчилиги мавжуд. Агар хелпер жуда катта бўлса, кўринишдаги разметкаларни жуда мураккаблаштиради. Аммо ушбу ҳолда у кодни алоҳидаги файлга жойлаштириш лозим. Юқоридаги мисолни кўриб чиқамиз. Бунинг учун янги класс ҳосил қилишмиз ва мавжуд класслар функционалигини ошириш лозим. Ушбу класслар методнинг биринчи параметр сифатида узатилади. Лойиҳамизда **Helpers** папкасини ҳосил қиламиз ва унга янги **ListHelper** классини шакллантирамиз.

```
using System;
using System.Web;
```

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

namespace BookStore.Helpers
{
    public static class ListHelper
    {
        public static MvcHtmlString CreateList(this HtmlHelper html,
string[] items)
        {
            TagBuilder ul = new TagBuilder("ul");
            foreach (string item in items)
            {
                TagBuilder li = new TagBuilder("li");
                li.SetInnerText(item);
                ul.InnerHtml += li.ToString();
            }
            return new MvcHtmlString(ul.ToString());
        }
    }
}
```

Янги классдаги хелперда битта **CreateList** статик методи аниқланган бўлиб, у параметр сифатида метод ҳосил қилинаётган объектни қабул қилади. Ушбу метод **html**-хелперлар **HtmlHelper** классини ифодаловчи функционаллигини кенгайтирсада, ушбу типдаги объект методнинг биринчи параметри сифатида узатилади. **CreateList** методининг иккинчи параметри сифатида сатр-қиймат массиви ҳисобланиб, рўйхатни ҳосил қилишда ишлатилади.

**TagBuilder** объекти орқали **ul** элементлардан иборат стандарт **html** ҳужжат ҳосил қилинади. Массив элементлари кетма-кет кўздан кечирилганда, барча сатрий қийматлар **li** тэгга айлантирилади ва рўйхатга қўшиб қўйилади. Якуний натижа сифатида **ul** элементлардан иборат қиймат қайтарилади.

**TagBuilder** класси бир қатор аъзоларга эга бўлиб, уларни қуйидаги ҳолларда ишлатиш мумкин:

- **InnerHtml** хусусияти орқали мазмун тэги сатр шаклида ўрнатилади ва қабул қилинади;
- **MergeAttribute(string, string, bool)** методи орқали элементга битта атрибут қўшиб қўйилади. Барча атрибутларни олиш учун **Attributes** коллекциясидан фойдаланилади;

- **SetInnerText(string)** методи орқали элементни ичида матнли мазмун ўрнатилади;
- **AddCssClass(sting)** метод **css** классни элементга қўшиб қўяди.

Янги хелпер ҳосил қилингач, биз уни кўринишда қўллашимиз мумкин. Юқоридаги мисолга қуйидаги ўзгартиришларни киритамиз:

```
@{
    string[] cities = new string[] { "Лондон", "Париж", "Москва" };
}
@{
    string[] countries = new string[] { "Великобритания", "Франция",
"Россия" };
}

@using BookStore.Helpers

<h3>Города</h3>
@Html.CreateList(cities)
<br />

<h3>Страны</h3>
<!-- или можно вызвать так -->
@ListHelper.CreateList(Html, countries)
```

## Формалар билан ишлаш

Биз ихтиёрий хелперни ўзимиз ёзишимиз мумкин бўлсада, **MVC** фреймворк бир қатор стандарт **html-хелпер**лар мавжуд. Ушбу **html-хелпер**лар орқали бирор турдаги **html**-разметкаларни генерация қилади. Шунинг учун кўп ҳолларда янги хелперларни яратишга эҳтиёж қолмайди.

### Html.BeginForm хелпери

Формани ҳосил қилиш учун стандарт **html**-элементлардан фойдаланишмиз мумкин:

```
<form method="post" action="/Home/Buy">
    <input type="hidden" value="@ViewBag.BookId" name="BookId" />
    <table>
        <tr>
```

```
            <td><p>Исмингизни киритинг </p></td>
            <td><input type="text" name="Person" /> </td>
        </tr>
        <tr>
            <td><p>Манзилингизни киритинг :</p></td>
            <td><input type="text" name="Address" /> </td>
        </tr>
        <tr>
            <td><input type="submit" value="Юбориш" /> </td>
            <td></td>
        </tr>
    </table>
</form>
```

Бу оддий **html-форма** ҳисобланиб, «**Юбориш**» тугмаси босилганда, барча киритилган маълумотларни **POST** сўрови орқали манзилига **/Home/Buy** узатади. Ички **BeginForm/EndForm** хелпери орқали юқоридаги амални бажариш мумкин:

```
@using (Html.BeginForm("Buy", "Home", FormMethod.Post))
{
    <input type="hidden" value="@ViewBag.BookId" name="BookId" />
    <table>
        <tr>
            <td><p>Исмингизни киритинг </p></td>
            <td><input type="text" name="Person" /> </td>
        </tr>
        <tr>
            <td><p>Манзилингизни киритинг :</p></td>
            <td><input type="text" name="Address" /> </td>
        </tr>
        <tr>
            <td><input type="submit" value="Юбориш" /> </td>
            <td></td>
        </tr>
    </table>
}
```

**BeginForm** методи параметр сифатида амал методи, контроллерни ва сўровни қабул қилади. Ушбу хелпер очилувчи **<form>** тэгини ва ёпилувчи **</form>** тэгини ҳосил қилади. Шунинг учун кўриниш рендеринг бўлаётганда чиқарилувчи потокка аввалги мисолда келтирилган **html-код** ҳосил қилинади. Шунинг учун иккала усул ҳам бир хил ҳисобланади.

Аммо юқоридаги мисолда битта жиҳат мавжуд. Агар бизнинг контроллерда битта методнинг иккита версияси (**POST** ва **GET**) мавжуд бўлса:

```
[HttpGet]
public ActionResult Buy(int id)
        {
            if (id > 2)
            {
                return Redirect("/Home/Index");
            }
            ViewBag.BookId = id;
            return View();

        }

[HttpPost]
public string Buy(Purchase purchase)
        {
            db.Purchases.Add(purchase);
            db.SaveChanges();
            return "Сегодня: " + purchase.Date + "Спасибо," +
purchase.Person + ", за покупку!";
        }
```

Яъни саҳифани формадан чақиришда ягона **Buy** методидан фойдаланилса, **Html.BeginForm** хелперида параметрларни кўрсатмаса ҳам бўлади.

```
@using(Html.BeginForm())
{
    ............
}
```

**Маълумотни киритиш**

Юқоридаги мисолда **Html.BeginForm** хелпери билан биргаликда стандарт **html** элементларидан фойдаланилди. Аммо стандарт **html-хелперлар** ёрдамида фойдаланувчининг маълумот киритишини таъминлаш мумкин. **MVC**да бир қатор хелперлар мавжуд бўлиб, деярли барча киритилувчи **html**-элементлар учун ишлаб чиқилган. Хелпер ёки стандарт **html** элементлардан фойдаланиш дастурчига ҳавола.

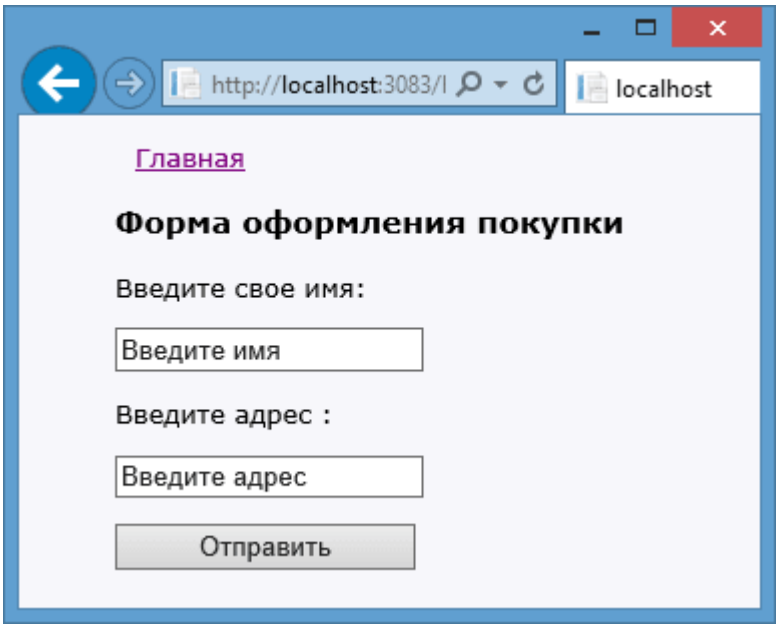
Барча стандарт **html-хелпер**ларда камида иккита параметр ишлатилади: биринчи параметр: **id** ва **name** атрибутлари қийматларини ўрнатиш учун, иккинчи параметр - **value** атрибути қийматини ўрнатиш учун қўлланилади.

**Html.TextBox**

**Html.TextBox** хелпери **input** тэгини **type** атрибутини **text** қийматига тенглаштириб ҳосил қилади. **TextBox** хелпери фойдаланувчининг маълумот киритиши учун хизмат қилади. Аввалги мисолда келтирилган формани **Html.TextBox** хелпер билан алмаштириамиз:

```
@using (Html.BeginForm("Buy", "Home", FormMethod.Post))
{
    <input type="hidden" value="@ViewBag.BookId" name="BookId" />
    <p>Введите свое имя: </p>
    @Html.TextBox("Person", "Введите имя")
    <p>Введите адрес :</p>
    @Html.TextBox("Address", "Введите адрес")
    <p><input type="submit" value="Отправить" /></p>
}
```

Натижада биз қуйидаги натижага эришамиз:




**Html.TextArea**

**TextArea** хелпери **<textarea>** элементини ҳосил қилиш учун ишлатилади. Ушбу хелпер орқали кўп сатрли матнли майдон генерация қилинади.

**@Html.TextArea("text", "привет <br/> мир")** ифоданинг натижаси қуйидаги **html-разметка** бўлади:

```
<textarea cols="20" id="text" name="text" rows="2">привет <br/> мир
</textarea>
```

Юқоридаги мисолда хелпернинг киритилаётган қийматнинг ва **html-тег**ни декодлаштиради. **TextArea** хелпернинг бошқа версиясида матнли майдоннинг ўлчови, сатрлар ва устунлар сонини кўрсатиш мумкин:

```
@Html.TextArea("text", "привет <br /> мир", 5, 50, null)
```

Ушбу хелпер қуйидаги **html-разметка**ни генерация қилади:

```
<textarea cols="50" id="text" name="text" rows="5">привет <br /> мир
</textarea>
```

**Html.Hidden**

Форма мисолида биз яширинган **input type="hidden"** майдонидан фойдаланган эдик. Унинг ўрнига **Html.Hidden** хелперидан фойдаланишимиз мумкин:

```
@Html.Hidden("BookId", "2")
```

Ушбу хелпер қуйидаги **html-разметка**ни генерация қилади:

```
<input id="BookId" name="BookId" type="hidden" value="2" />
```

Ўзгарувчи қийматини **ViewBag** дан олиш учун уни **string** типига олиб келиш лозим:

```
@Html.Hidden("BookId", @ViewBag.BookId as string)
```

**Html.Password**

**Html.Password** орқали паролни киритиш учун майдон генерация қилинади. Ушбу хелпер **TextBox** хелперига ўхшаш бўлиб, киритилаётган қийматларга маскани шакллантиради:

```
@Html.Password("UserPassword", "val")
```

Ушбу хелпер қуйидаги **html-разметка**ни генерация қилади:

```
<input id="UserPassword" name="UserPassword" type="password" value="val"
/>
```

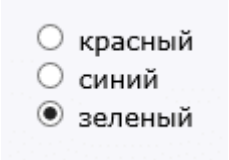
**Html.RadioButton**

Переключателларни ҳосил қилишда **Html.RadioButton** хелпердан фойдаланилади. У **input** элементини **type="radio"** қиймат билан генерация қилади. Переключателлар гуруҳини яратиш учун уларга эгона ном (**name** хусусияти) бериш лозим.

```
@Html.RadioButton("color", "red")
<span>красный</span> <br />
@Html.RadioButton("color", "blue")
<span>синий</span> <br />
@Html.RadioButton("color", "green", true)
<span>зеленый</span>
```

Ушбу хелпер қуйидаги **html-разметка**ни генерация қилади:

```
<input id="color" name="color" type="radio" value="red" />
<span>красный</span> <br />
<input id="color" name="color" type="radio" value="blue" />
<span>синий</span> <br />
<input checked="checked" id="color" name="color" type="radio"
value="green" />
<span>зеленый</span>
```




**Html.CheckBox**

**Html.CheckBox** хелпери орқали иккита элемент ҳосил қилинади. Қуйидаги кодга эътибор беринг:

```
@Html.CheckBox("Enable", false)
```

Ушбу ифода қуйидаги **html** кодни генерация қилади:

```
<input id="Enable" name="Enable" type="checkbox" value="true" />
<input name="Enable" type="hidden" value="false" />
```

Яъни жорий байроқчадан ташқари яширинган байроқча ҳам генерация қилинади. Ушбу байроқча нима учун керак? Браузер байроқча қийматини у

танланган ёки бекор қилинган вақтда узатади. Яширинган майдон эса **Enable** элементи қийматининг у танланмаган вақтда ҳам ўрнатади.

**Html.Label**

**Html.Label** хелпери **<label/>** элементини ҳосил қилади. Хелперга узатилаётган параметрда **for** атрибути қиймати ва элементдаги матн узатилади. Хелпернинг перегрузка қилинган вариантида **for** атрибути ва нишон бир-бирига боғлиқ бўлмаган ҳолда ҳосил қилинади. **Html.Label("Name")** каби ҳосил қилинган хелпер қуйидаги html-кодни генерация қилади:

```
<label for="Name">Name</label>
```

**label** элементи оддий нишонни ифодалайди ва уни маълумотлани киритиш элементига бириктириб қўйиш мумкин. **label** элементининг **for** атрибути киритилувчи элементнинг **ID**сига мослаштирилиши лозим. Агар фойдаланувчи нишонга мурожаат қилса, браузер автоматик тарзда фокусни нишон билан боғланган элементга узатади.

**Html.DropDownList**

**Html.DropDownList** хелпери (**<select/>**) рўйхатни шакллантиради. Рўйхат элементларини генерация қилиш учун **SelectListItem** объектлари коллекцияси зарур. Ушбу коллекция рўйхат элементларини ифодалайди.

**SelectListItem** объекти **Text**, **Value** ва **Selected** хусусиятларига эга. **SelectListItem** объектлар рўйхатини яратиш мумкин ёки **SelectList** хелперидан фойдаланиш мумкин. Ушбу хелпер **IEnumerable** объектларини кўриб чиқади ва уларни **SelectListItem** объектлари кетма-кетлигига ўзгартиради.

```
@Html.DropDownList("countires", new SelectList(new string[]
{"Russia","USA", "Canada","France"}),"Countries")
```

коди қуйидаги **html**-кодни генерация қилади:

```
<select id="countires" name="countires">
<option value="">Countries</option>
<option>Russia</option>
<option>USA</option>
<option>Canada</option>
<option>France</option>
</select>
```

Энди бироз мураккаб мисолни кўриб чиқамиз. Китоблар рўйхатидан иборат коллекцияни ҳосил қиламиз ва контроллердаги амал методи орқали уни **ViewBag** га узатамиз:

```
BookContext db = new BookContext();
public ActionResult Index()
        {
            SelectList books = new SelectList(db.Books, "Author",
"Name");
            ViewBag.Books = books;
            return View();
        }
```

Бу ерда биз **SelectList** объектини ҳосил қилдик ва унга **db.Books** қийматлари рўйхатидан **Book** модели хусусияти номини ва қиймат сифатида **Author** ни узатдик. Ушбу ҳолда иккита турли хусусиятларни ўрнатиш шарт эмас. Энди кўринишда биз ушбу **SelectList** объектидан фойдаланишимиз мумкин.

```
@Html.DropDownList("Author", ViewBag.Books as SelectList)
```

Кўриниш рендеринг қилинганда барча **SelectList** элементлари рўйхат шаклида ҳосил қилинади.

**Html.ListBox**

**Html.ListBox** хелпери худди **DropDownList** га ўхшаш бўлиб, **<select/>** элементини ҳосил қилади. Аммо ушбу ҳолда рўйхатдан бир нечта элементларни танлаш (**multiple** атрибути) имконини беради. Рўйхатни ҳосил қилиш учун **SelectList** ўрнига **MultiSelectList** классидан фойдаланиш лозим.

```
@Html.ListBox("countires", new MultiSelectList(new string[] { "Россия",
"США", "Китай", "Индия" }))
```

Ушбу хелпер қуйидаги **html-разметка**ни генерация қилади:

```
<select length="9" id="countries" multiple="multiple" name="countires">
  <option>Россия</option>
  <option>США</option>
  <option>Китай</option>
  <option>Индия</option>
</select>
```

Битта қийматга эга бўлган маълумотни серверга узатиш тушунарли. Аммо кўп қийматли маълумотлар қандай узатилади? Бизда кўйидаги форма мавжуд бўлсин:

```
@using (Html.BeginForm())
        {
        @Html.ListBox("countries",
            new MultiSelectList(new string[] { "Россия", "США", "Китай",
"Индия" }))
        <p><input type="submit" value="Отправить" /></p>
        }
```

У ҳолда контроллердаги метод ушбу қийматларни қуйидагича олиши мумкин:

```
[HttpPost]
public string Index(string[] countries)
        {
            string result = "";
            foreach (string c in countries)
            {
                result += c;
                result += ";";
            }
            return "Вы выбрали: " + result;
        }
```

**Бир нечта тугмалардан иборат форма**

Кўринишдаги формалардаги тугма маълумотларни узатиш учун ишлатилади. Баъзи ҳолларда кўринишда бир нечта тугмалардан фойдаланишга тўғри келади. Масалан, бизнинг кўринишда иккита маълумот киритиш майдони мавжуд бўлсин. Иккита тугма эса ушбу қийматларни қўшиш ёки ўчириши лозим бўлсин:

```
@using (Html.BeginForm("MyAction", "Home", FormMethod.Post))
        {
            <input type="text" name="product" /><br />
            <button name="action" value="add">Добавить</button>
            <button name="action" value="delete">Удалить</button>
        }
```

Бунинг оддий усули ҳар бир тугмага мос **name** атрибути номи ҳосил қилиниб, **value** атрибутига эса турли қийматларни бериш лозим. Формадаги маълумотларни қабул қилувчи метод эса қуйидагича шакллантирилиши мумкин:

```
[HttpPost]
public string MyAction(string product, string action)
        {
            string res="";
            if (action == "add")
            {
                res="add";
            }
            else if (action == "delete")
            {
                res = "deleted";
            }
            // остальной код метода
            return res;
        }
```

**action** параметрида сақланаётган **value** қийматга мос конструкция ишга туширилади.

## Қатъий типлаштирилган хелперлар

**ASP.NET MVC** да стандарт хелперлардан ташқари қатъий типлаштирилган хелперлар ҳам мавжуд. Ушбу турдаги хелперлар параметр сифатида лямбда ифодаларни қабул қилади. Ушбу лямбда ифодада жорий хелпер мос қўйилган модел хусусияти кўрсатилади. Қатъий типлаштирилган хелперлар фақат қатъий типлашган кўринишлар ишлатилади. Хелперга узатиладиган модел типи эса кўринишнинг **@model** директивасида кўрсатилган моделга мос бўлиши керак.

Мисолда кўриб чиқамиз. Иккинчи бўлимда биз китоб харид қилишда **Purchase** моделидан фойдаланган эдик:

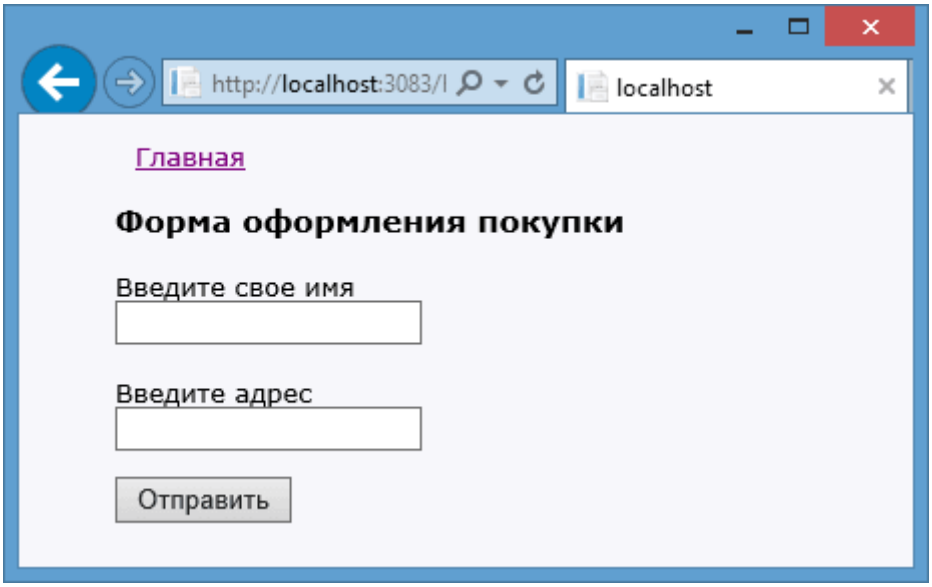
```
public class Purchase
    {
        // харид ID си
        public int PurchaseId { get; set; }
        // харидор исми ва фамилияси
        public string Person { get; set; }
        // харидор манзили
        public string Address { get; set; }
        // китоб ID си
        public int BookId { get; set; }
        // хариф санаси
        public DateTime Date { get; set; }
    }
```

Сўнгра, у учун қуйидаги формадан фойдаландик:

```
<form method="post" action="">
     <input type="hidden" value="@ViewBag.BookId" name="BookId" />
     <table>
     <tr>
       <td><p>Введите свое имя </p></td>
       <td><input type="text" name="Person" /> </td>
     </tr>
     <tr>
       <td><p>Введите адрес :</p></td>
       <td>
         <input type="text" name="Address" />
       </td>
     </tr>
     <tr>
     <td><input type="submit" value="Отправить" /> </td>
     <td></td>
     </tr>
     </table>
</form>
```

Ушбу мисолни хелпер орқали ўзгартириб оламиз:

```
@model BookStore.Models.Purchase
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div>
    <h3>Форма оформления покупки</h3>
    @using (Html.BeginForm("Buy", "Home", FormMethod.Post))
    {
        @Html.HiddenFor(m => m.BookId)
        @Html.LabelFor(m => m.Person, "Введите свое имя")
        <br />
        @Html.TextBoxFor(m => m.Person)
        <br /><br />
        @Html.LabelFor(m => m.Address, "Введите адрес")
        <br />
        @Html.TextBoxFor(m => m.Address)
        <p><input type="submit" value="Отправить" /></p>
    }
</div>
```




Қатъий типлаштирилган хелперлар оддий хелперларга ўхшайди. Аммо уларнинг охирида **For**: **LabelFor** суффикси қўшиб қўйилади. Қатъий типлаштирилган хелперлар қатъий типлаштирилган кўринишларда ишлатилиши сабабли, кўриниш бошида ишлатиладиган модел кўрсатилиши лозим:

```
@model BookStore.Models.Purchase
```

Яъни `@Html.TextBoxFor(m => m.Person)` чақирилганда **m** параметри **Purchase** модели ўзгарувчисини ифодалайди. **m=>m.Person** ифода эса жорий хелпер **Person** хусусият учун матнли майдонни генерация қилишни англатади. `@Html.TextBoxFor(m => m.Person)` хелпери

```
<input id="Person" name="Person" type="text" value="" />
```

матнли майдонни генерация қилади.

Ҳар бир стандарт хелпер учун мос равишда қатъий типлашган хелпер мавжуд:

- **Html.CheckBoxFor**
  `@Html.CheckBoxFor(m=>m.Enable, false)` ифода қуйидаги **html** кодни генерация қилади:
  ```
  <input id="Enable" name="Enable" type="checkbox" value="true" />
  <input name="Enable" type="hidden" value="false" />
  ```

- **Html.HiddenFor**
  `@Html.HiddenFor(m=> m.Name)` ифода қуйидаги **html** кодни генерация қилади:

```
<input id="Name" name="Name" type="hidden"
value="[значение_m.Name]" />
```

- **Html.LabelFor**
  `@Html.LabelFor(m => m.Name,"Имя")` ифода қуйидаги **html** кодни генерация қилади:
  `<label for="Name">Имя</label>`

- **Html.PasswordFor**
  `@Html.PasswordFor(m => m.Password)` ифода қуйидаги **html** кодни генерация қилади:
  ```
  <p><input type="submit" value="Отправить" /></p>
  <p><input id="Password" name="Password" type="password" /></p>
  ```

- **Html.RadioButtonFor**
  `@Html.RadioButtonFor(m => m.Option, "val")` ифода қуйидаги **html** кодни генерация қилади:
  `<input id="Option" name="Option" type="radio" value="val" />`

- **Html.TextBoxFor**
  `@Html.TextBoxFor(m => m.Name)` ифода қуйидаги **html** кодни генерация қилади:
  `<input id="Name" name="Name" type="text" />`

- **Html.TextAreaFor**
  `@Html.TextAreaFor(m => m.Name,10,9,null)` ифода қуйидаги **html** кодни генерация қилади:
  `<textarea cols="9" id="Name" name="Name" rows="10" ></textarea>`

# V. Моделлар

## Моделлар ва маълумотлар базаси

Дастурдаги барча берилганларни алоҳида модел сифатида шакллантириш мақсадга мувофиқ. Қўйилган масала ва унинг мураккаблигига қараб, турли сондаги ва шаклдаги моделлардан фойдаланиш мумкин. Иккинчи бўлимда келтирилган оддий мисолларда иккита модел (класс) ишлатилган эди: **китоб** (**Book**) ва **китобни харид қилиш** (**Purchase**).

Моделлар оддий классларни ифодалаб, лойиҳадаги **Models** папкасида жойлашган. Моделлар асосида дастур мантиғи ифодаланади. Масалан, китоб ва унинг харидини ифодаловчи маълумотлар модели:

```
public class Book
    {
        // китоб ID си
        public int Id { get; set; }
        // китоб номи
        public string Name { get; set; }
        // китоб муаллифи
        public string Author { get; set; }
        // нархи
        public int Price { get; set; }
    }

public class Purchase
    {
        // харид ID си
        public int PurchaseId { get; set; }
        // харидор исми ва фамилияси
        public string Person { get; set; }
        // харидор манзили
        public string Address { get; set; }
        // китоб ID си
        public int BookId { get; set; }
        // хариф санаси
        public DateTime Date { get; set; }
    }
```

Модел фақат хусусиятлардан иборат бўлиши шарт эмас. Шунингдек, моделда конструкторлар ва ёрдамчи методлар ҳам мавжуд бўлиши мумкин. Аммо модел классини мураккаблаштириш шарт эмас. У маълумотларни

ифодалаш учун ишлатилади. Маълумотларни бошқариш ва бизнес-мантиқ бу контроллернинг иши ҳисобланади.

Модел маълумотлари асосан маълумотлар базасида сақланади. МБси билан ишлаш учун **Entity Framework** фреймворкидан фойдаланиш қулай ҳисобланади. Ушбу фреймворк орқали **sql-сўровлар**ни амалга ошириш ва МБда базаларни яратиш мумкин.

**MVC 5** лойиҳасини яратишда фойдаланувчиларни аутентификация қилиш учун "**No Authentication"** қисми танланган бўлса, лойиҳа яратилганидан сўнг уни **Entity Framework** фреймворки орқали **NuGet** пакетли менеджерига улаш лозим.

**NuGet** ўрнига пакетли менеджер консолини ишлатиш мумкин. Бунинг учун **Visual Studio** менюсидан **View -> Other Windows -> Package Manager Console** қисмини танланг. Сўнгра, муҳит пастида пакетли менеджер консоли шакллантирилади. Ушбу консолга қуйидаги буйруқни амалга оширамиз:

**PM> Install-Package Entity Framework -Version 6.0.2**

Шундан сўнг **Entity Framework** пакети юклаб олинади ва ўрнатилади. Баъзи ҳолда ушбу консолдан фойдаланиш қулай ҳисобланади.

**Entity Framework** технологиясида "**Code First**" ёндашуви қўллаб-қувватланади. Ушбу ёндашув асосида **МБ**дан маълумотларни олиш ёки сақлаш мумкин. Ёки оддий класслар асосида ҳосил қилинган объектларни қаерда ва қай тарзда сақлашни **Entity Framework** амалга оширади.

**МБ**сига **Entity Framework** орқали уланиш учун бизга **маълумотлар контексти** деб номланувчи объект зарур. Маълумотлар контекси **DbContext** классидан ворисланган классни ифодалайди. Маълумотлар контекстида бир ёки бир қанча **DbSet<T>** типдаги хусусиятлар мавжуд. Бу ерда **T** маълумотлар базасида сақланувчи объект типини ифодалайди. Масалан, юқорида келтирилган китоб ва унинг харидига мос маълумотлар контекстини шакллантирамиз:

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Data.Entity;

namespace BookStore.Models
{
```

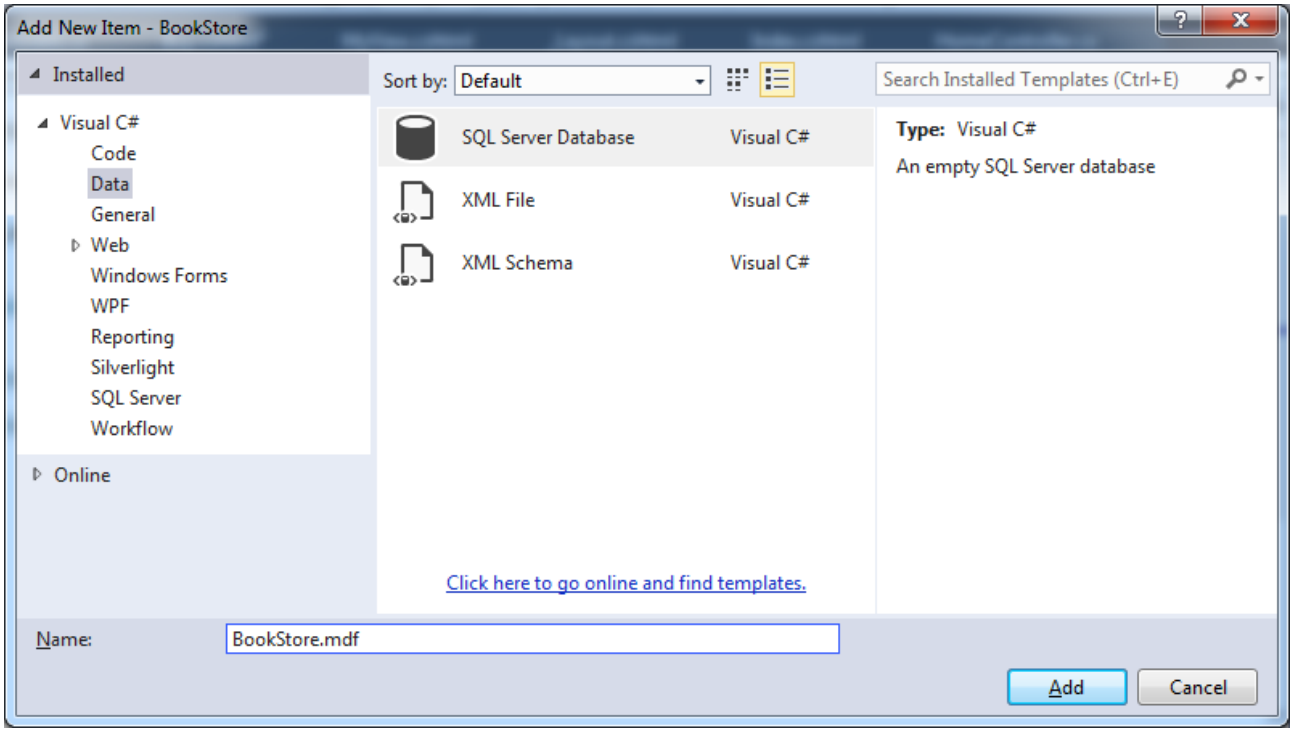
```
    public class BookContext : DbContext
    {
        public DbSet<Book> Books { get; set; }
        public DbSet<Purchase> Purchases { get; set; }
    }
}
```

**Books** ва **Purchases** хусусиятлари орқали **МБ**сида сақланувчи мавжуд маълумотлар моделларига доступга эга бўламиз.

## Маълумотлар базасига уланиш

Дастурдаги маълумотларни сақлаш учун **МБ**си зарур. Биз дастуримиз учун турли **МБ**дан фойдаланишимиз мумкин. Юқоридаги мисолларда **MS SQL Server МБ**сида жадваллар яратилиб, уларда моделларга мос қийматлар сақланди.

Биз **МБ**сини лойиҳанинг ўзида ёки **MS SQL** серверида ҳосил қилишимиз мумкин. **МБ**сини лойиҳада сақлаш учун **App_Data** папкаси мўлжалланган. **App_Data** папкасига сичқончанинг ўнг тугмасини босиб, ҳосил бўлган мулоқот ойнасидан **Add-> New Item...**ни танлаймиз. Ҳосил қилинган мулоқот ойнасидан ни **SQL Server Database** танлаб, янги база сифатида **Bookstore.mdf** ни киритамиз.



Натижада **App_Data** папкасида янги **BookStore** деб номланган **МБ**си ҳосил бўлади. Ушбу **МБ**сида жадвал ва маълумотларни киритиш ва бошқа амалларни бажариш мумкин.

Аввало ушбу **МБ**сида қандай турдаги маълумот сақланаётганлигини кўриб чиқамиз. **МБ** билан ишлаш учун аввалги бўлимларда келтирилган мисолларни оламиз (**Book** ва **Purchase** моделлари):

```
public class Book
    {
        // китоб ID си
        public int Id { get; set; }
        // китоб номи
        public string Name { get; set; }
        // китоб муаллифи
        public string Author { get; set; }
        // нархи
        public int Price { get; set; }
    }
```

ва

```
public class Purchase
    {
        // харид ID си
        public int PurchaseId { get; set; }
        // харидор исми ва фамилияси
        public string Person { get; set; }
        // харидор манзили
        public string Address { get; set; }
        // китоб ID си
        public int BookId { get; set; }
        // хариф санаси
        public DateTime Date { get; set; }
    }
```

Ушбу моделлар билан ишлаш учун қуйидаги маълумотлар контексти аниқланган бўлсин:

```
public class BookContext : DbContext
    {
        public DbSet<Book> Books { get; set; }
        public DbSet<Purchase> Purchases { get; set; }
    }
```

Бизнинг лойиҳамиз, маълумотлар контексти ва **МБ** билан ўзаро боғланишни шакллантириш учун **web.config** файлида ушбу МБсига уланишни амалга оширамиз. Сўнгра **configSections** секциясига қуйидаги кодни ёзиб қўямиз:

```
<connectionStrings>
    <add name="BookContext" connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename='|DataDirectory|\Bookstore.mdf';I
ntegrated Security=True"
   providerName="System.Data.SqlClient" />
</connectionStrings>
```

**name** атрибути қийматига (name="BookContext") эътибор қаратинг. Ушбу атрибут қиймати сифатида маълумотлар контексти келтирилган.

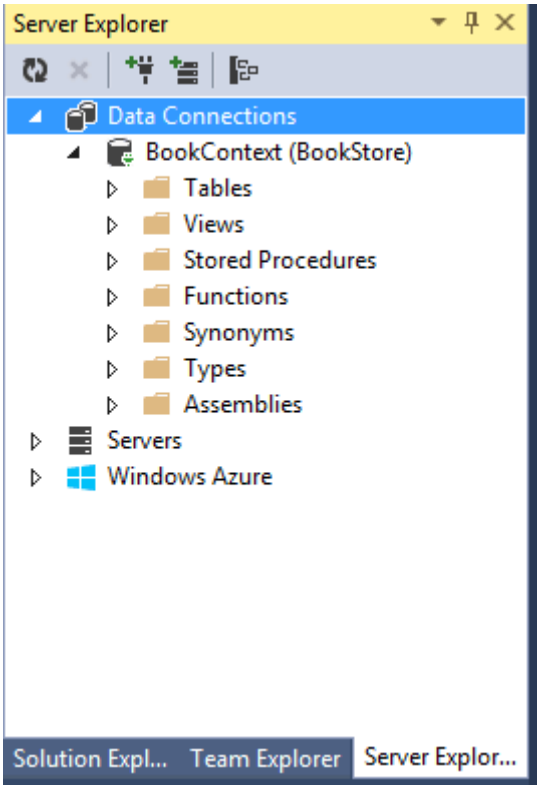
Бундан ташқари иккинчи усул ҳам мавжуд. Бизда **МБ**сига бошқа уланиш сатри мавжуд бўлсин. Масалан, **<add name="DefaultConnection"....,**

Биз ушбу сатрни ўзгартмоқчи эмасмиз. Ушбу ҳолда мавжуд маълумотлар контексти билан алоқа ўрнатиш учун бош класс конструкторига ушбу уланиш сатри номини беришимиз зарур.

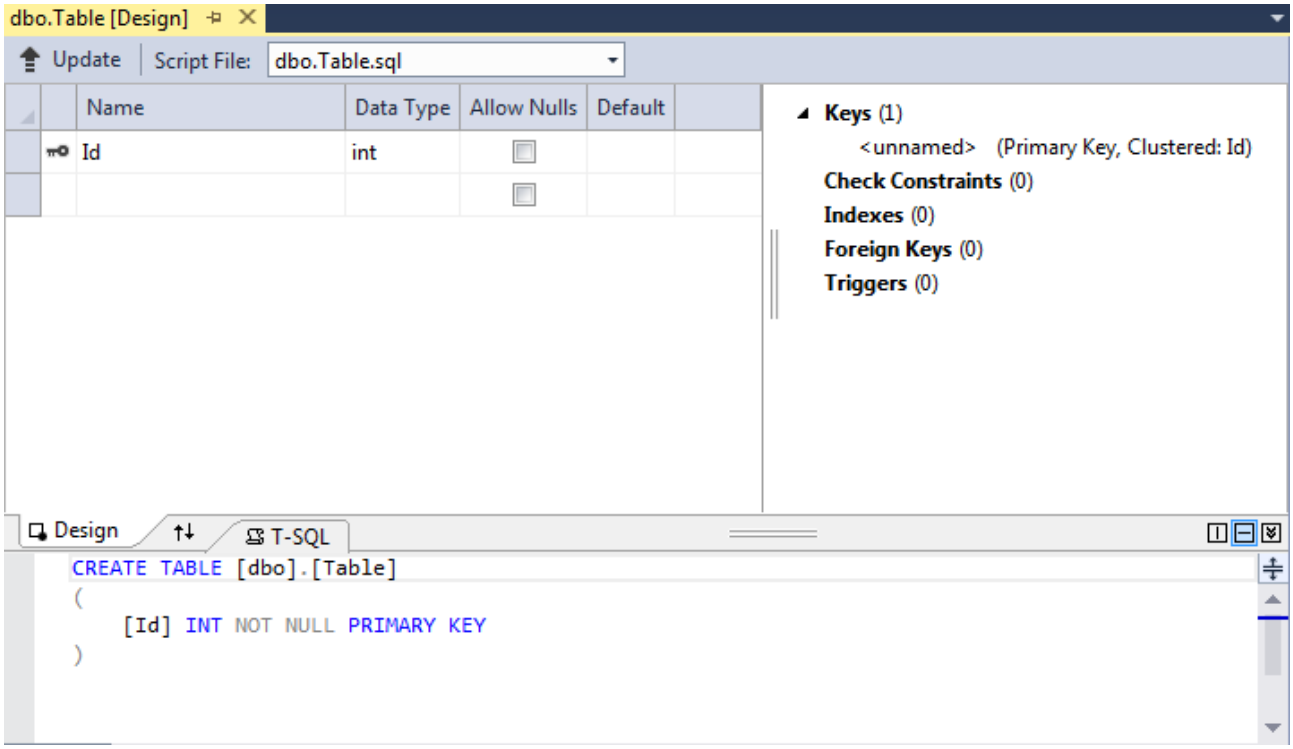
```
public class BookContext : DbContext
    {
        public BookContext()
            : base("DefaultConnection")
        { }
        public DbSet<Book> Books { get; set; }
        public DbSet<Purchase> Purchases { get; set; }
    }
```

**Visual Studio 2013** да **LocalDB**дан фойдаланишимиз мумкин. **LocalDB – SQL Sever Express МБ**сининг соддалаштирилган варианти бўлиб, дастурчилар учун махсус яратилган. Шунинг учун юқоридаги мисолларнинг барчасида **МБ** сифатида **(LocalDB)\v11.0** ишлатилган.

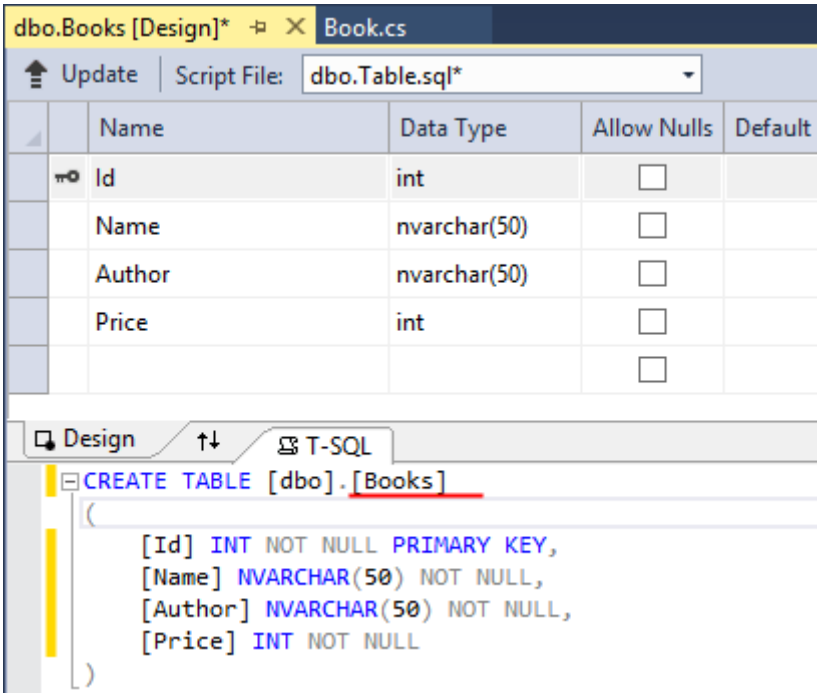
Уланиш сатридаги **|DataDirectory|** ифода **App_Data** папкасида жойлашган **МБ**сига мос тўлиқ физик йўлни ифодалайди. Энди лойиҳамиз учун зарур жадвалларни шакллантирамиз. Бунинг учун **Server Explorer** ойнасига ўтиб, **Data Connections** қисмни танлаймиз. Натижада бизга **BookContext** қисми тақдим этилади. Ушбу қисмни очсак, қуйидаги интерфейс ҳосил бўлади:



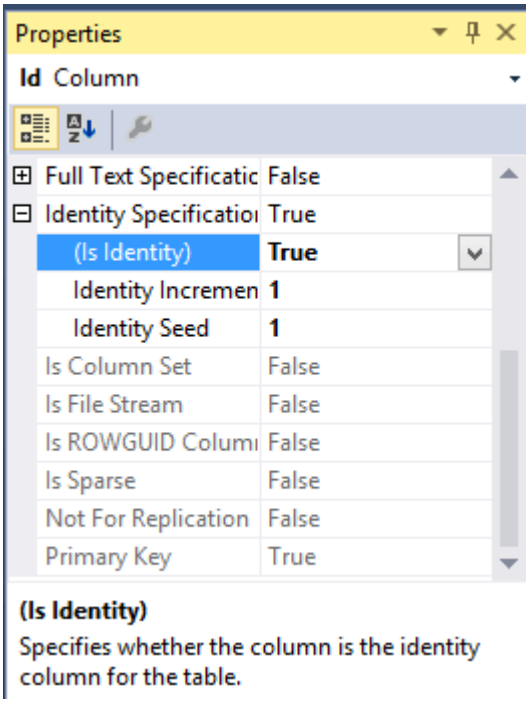

Ушбу **МБ**сига биринчи жадвалимизни яратамиз. Бунинг учун **Tables** қисмига ўтиб, сичқончанинг ўнг тугмасини босиб, ҳосил қилинган менюдан **Add New Table** қисмни танлаймиз. Натижада бизга янги жадвал ҳосил қилиш учун конструктор тақдим этилади.



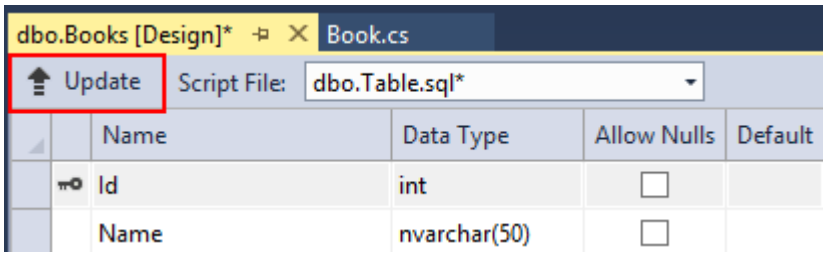
**Entity Framework** фрейворки билан ишлаш келишувларига кўра жадвал номи модел номига мос бўлиши лозим. Масалан, **Book** моделига мос жадвал номи **Books** бўлади. Демак, **Book** моделига мос жадвал тузилмасини ҳосил қиламиз:



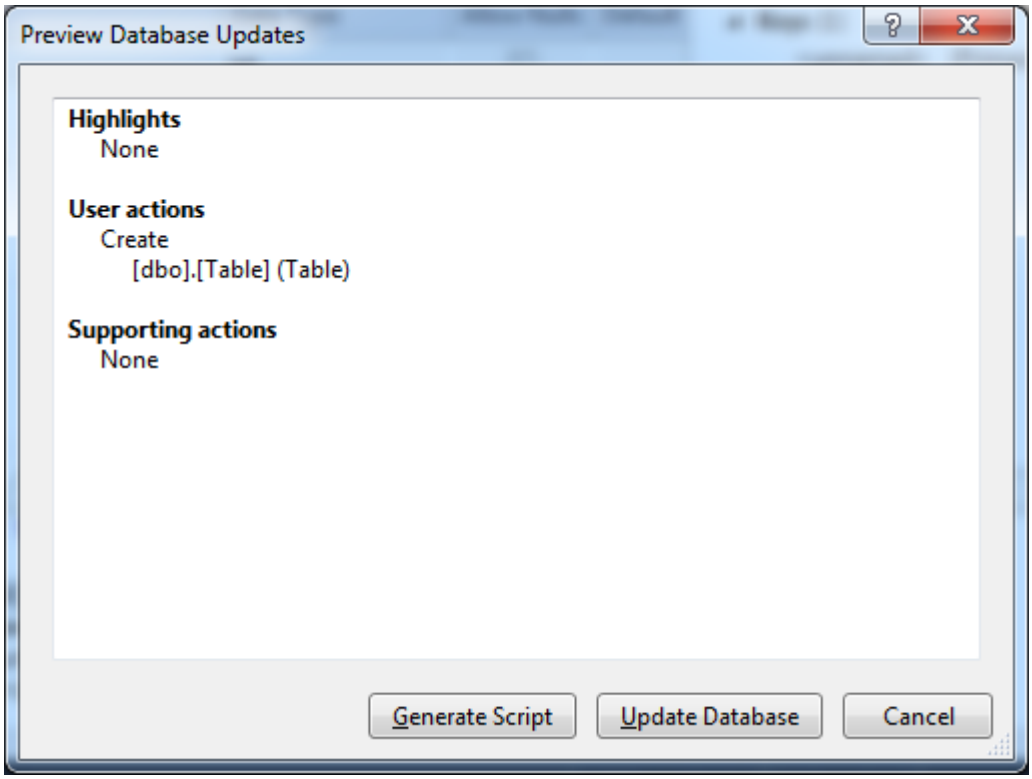

Шунингдек, **Id** майдони янги ҳосил қилинган барча жадваллар учун автоматик тарзда ҳосил қилинади. **Properties** ойнасига кириб, ушбу майдон учун автоинкрементни шакллантирамиз:



Охирги қадамда маълумотлар базасини генерация қилишимиз лозим. Бунинг учун **Update** тугмаси босилади:

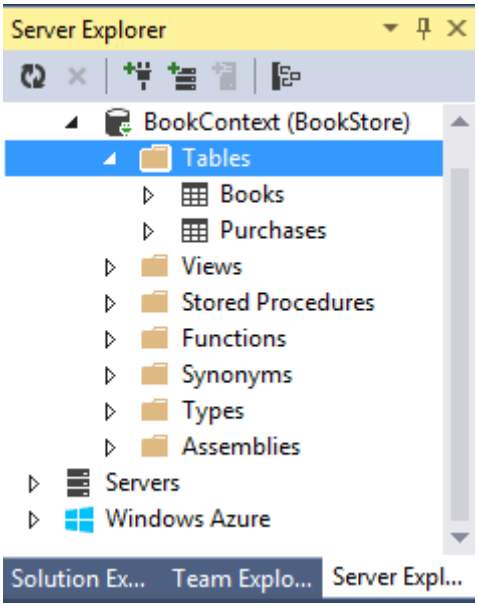



Натижада қуйидаги мулоқот ойнаси тақдим этилади:

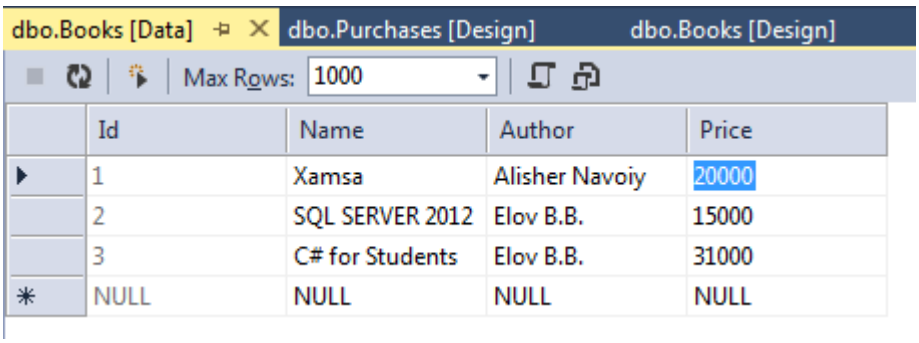



Ушбу мулоқот ойнасидаги **Update Database** тугмани босамиз. Шундан сўнг, бизнинг **МБ**да янги жадвал ҳосил бўлади. Худди шу усулда **Purchase** модели учун **Purchases** жадвалини яратамиз.

```
CREATE TABLE [dbo].[Purchases]
(
	[Id] INT NOT NULL PRIMARY KEY IDENTITY,
    [Person] NVARCHAR(50) NOT NULL,
    [Address] NVARCHAR(50) NOT NULL,
    [BookId] INT NOT NULL,
    [Date] DATETIME NOT NULL
)
```




Кейинги қадамда **МБ** дизайнерида **Books** жадвали учун бир қанча ёзувларни шакллантирамиз. Бунинг учун **Server Explorer** ойнасидаги **Books** қисмини топиб, тақдим этилган рўйхатдан **Show Table Data** қисмни танлаймиз. Мисол сифатида қуйидаги ёзувларни ҳосил қиламиз:




| | Id | Name | Author | Price |
|---|---|---|---|---|
| ▶ | 1 | Xamsa | Alisher Navoiy | 20000 |
| | 2 | SQL SERVER 2012 | Elov B.B. | 15000 |
| | 3 | C# for Students | Elov B.B. | 31000 |
| * | NULL | NULL | NULL | NULL |

Кейинги қадамда ушбу маълумотларни **Home** контроллерида оламиз ва мос кўринишга узатамиз:

```
BookContext db = new BookContext();
public ActionResult Index()
        {
            return View(db.Books);
        }
```

**Index()** амал методига мос кўринишни ҳосил қиламиз:

```
@model IEnumerable<BookStore.Models.Book>
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

```
<div>
    <h3>Распродажа книг</h3>
    <table>
        <tr class="header">
            <td><p>Название книги</p></td>
            <td><p>Автор</p></td>
            <td><p>Цена</p></td>
            <td></td>
        </tr>
        @foreach (BookStore.Models.Book b in Model)
        {
            <tr>
                <td><p>@b.Name</p></td>
                <td><p>@b.Author</p></td>
                <td><p>@b.Price</p></td>
                <td><p><a href="/Home/Buy/@b.Id">Купить</a></p></td>
            </tr>
        }
    </table>
</div>
```

**Маълумотлар базасига уланишни ёпиш**

**МБ** билан муайян амаллар бажарилгач, у билан ўрнатилган алоқани (узиш) якунлаш лозим. Бунинг учун маълумотлар констекстидаги **Dispose** методидан фойдаланиш зарур.

```
protected override void Dispose(bool disposing)
        {
            db.Dispose();
            base.Dispose(disposing);
        }
```

Ушбу **Dispose** методи контроллернинг қайта аниқланган методи ҳисобланиб, контроллер объекти ўчирилганда (фаолиятини якунлаганда) чақирилади. Унда `db.Dispose();` методи мавжуд бўлиб, у маълумотлар контекси билан боғлиқ барча алоқаларни (ресурс ва уланишларни) йўқотади.

## Шаблонли хелперлар

Муайян элементларни генерация қилувчи стандарт **html**-хелперлардан ташқари **ASP.NET MVC** да **шаблонли хелперлар** ҳам мавжуд. Аввалги бўлимларда келтирилган **html**-хелперларлан фарқли равишда улар муайн **html** элементни генерация қилмайди. Шаблонли хелперлар модел хусусиятига мурожаат қилади ва у ўз навбатида ушбу хусусиятга мос **html** элементни генерация қилади.

**ASP.NET MVC** да қуйидаги шаблонли хелперлар мавжуд:

- **Display –** ушбу элемент орқали муайян модел хусусияти намойиш қилинади: **Html.Display("Name")**
- **DisplayFor –** қатъий типлаштирилган **Display** хелперининг аналоги: **Html.DisplayFor(m => m.Name)**
- **Editor –** муайян модел хусусиятини модификация қилиш учун разметка элементини ҳосил қилади: **Html.Editor("Name")**
- **EditorFor –** қатъий типлаштирилган **Editor** хелперининг аналоги: **Html.EditorFor(m => m.Name)**
- **DisplayText –** кўрсатилган ифодага мос оддий сатрни шакллантиради: **Html.DisplayText("Name")**
- **DisplayTextFor –** қатъий типлаштирилган **DisplayText** хелперининг аналоги: **Html.DisplayTextFor(m => m.Name)**

Ушбу хелперлар якка хелперлар ҳисобланиб, моделнинг битта хусусиятидан **html-разметка**ни ҳосил қилади. Ушбу хелперлардан ташқари фреймворкда бир қанча шаблонлар мавжуд бўлиб, улар орқали моделдаги барча хусусиятлар учун майдонлар шакллантирилади:

- **DisplayForModel –** моделдаги барча хусусиятлар учун майдонларни фақат ўқиш учун шакллантиради: **Html.DisplayForModel()**
- **EditorForModel** - моделдаги барча хусусиятлар учун майдонларни ўзгартириш учун шакллантиради: **Html.EditorForModel()**

Биз контроллеримизда бирор **BookView** амалини шакллантирмоқчи бўлсак ва у мос **Id** бўйича муайян китоб ҳақидаги маълумотларни тақдим қилиши лозим бўлсин. Контроллерда **BookView** амал методи ва шунга мос кўринишни ҳосил қиламиз:

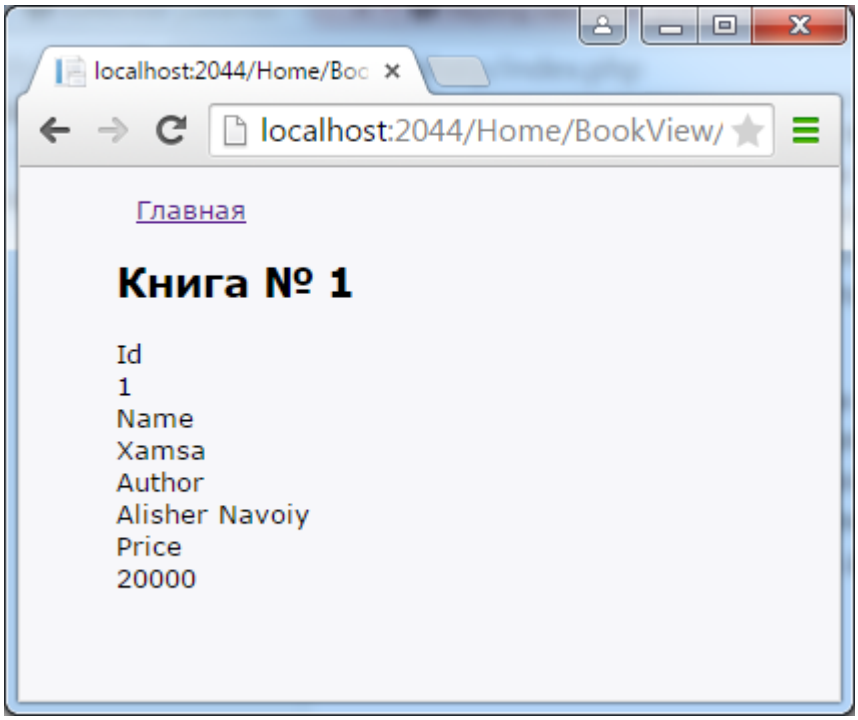
```
public ActionResult BookView(int id)
        {
            Book book = db.Books.Find(id);
```

```
            return View(book);
             }
```

ва кўринишда

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

@model BookStore.Models.Book
<h2>Книга № @Model.Id</h2>
@Html.DisplayForModel()
```

Ушбу ресурсга браузердаги манзиллар сатридан **Home/BookView/1** орқали мурожаат қиламиз.




## Моделларни таҳрирлаш

Юқоридаги мисолда биз шаблонли хелперлар орқали моделдаги маълумотларни намойиш қилишни кўриб чиқдик. Энди моделдаги маълумотларни ўзгартириш билан шуғулланамиз. Аввало контроллерда амал методини шакллантирамиз. Ушбу амал методида моделдаги **Id** га мос ўзгартиришларни ҳосил қилади:

```
[HttpGet]
public ActionResult EditBook(int? id)
        {
            if (id == null)
            {
                return HttpNotFound();
            }
            Book book = db.Books.Find(id);
            if (book != null)
            {
                return View(book);
            }
            return HttpNotFound();
        }
```

Агар фойдаланувчи **Id** ни кўрсатмаса, параметр сифатида **int** ўрнига **int?** ни ўрнатамиз. Агар ушбу параметр узатилмаган бўлса, методдан **HttpNotFound** натижани қайтарамиз.

**EditBook** амал методига мос кўринишимиз моделдаги баъзи майдонларга мос **EditorFor** хелперлар тўпламини ўзида сақлайди:

```
@{
    ViewBag.Title = "Редактировать книгу";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@model BookStore.Models.Book
<h2>Книга № @Model.Id</h2>
@using (Html.BeginForm("EditBook", "Home", FormMethod.Post))
{
    <fieldset>
        @Html.HiddenFor(m => m.Id)
        <p>
            @Html.LabelFor(m => m.Name, "Название книги")
            <br />
            @Html.EditorFor(m => m.Name)
        </p>
        <p>
            @Html.LabelFor(m => m.Author, "Автор")
            <br />
            @Html.EditorFor(m => m.Author)

        </p>
        <p>
            @Html.LabelFor(m => m.Price, "Цена")
            <br />
            @Html.EditorFor(m => m.Price)
```

```
        </p>
        <p><input type="submit" value="Отправить" /></p>
    </fieldset>
}
```

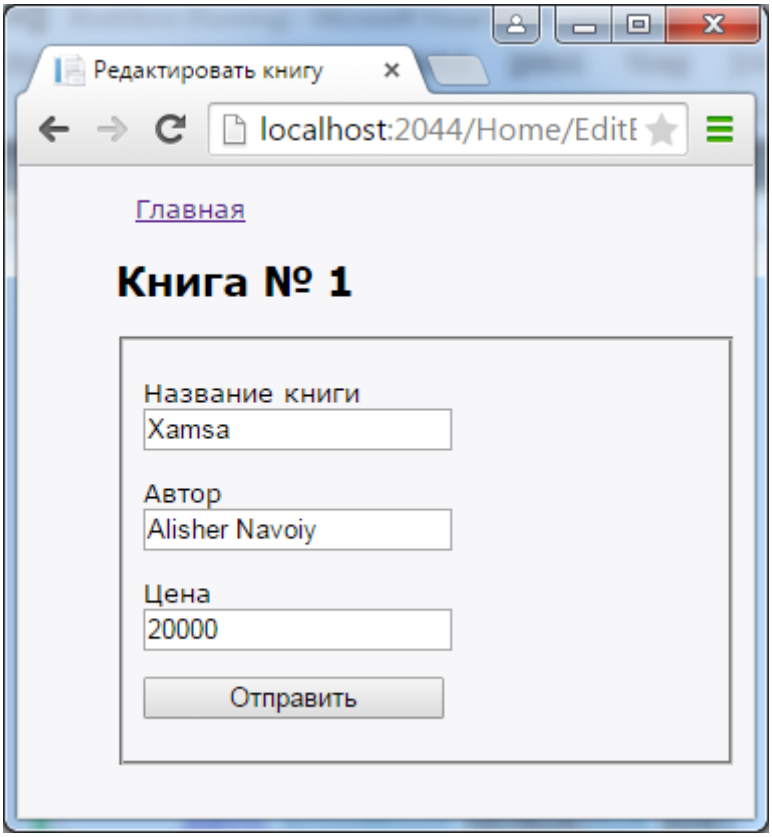
Китобнинг уникал **Id** идентификаторини ўзгартириш шарт эмас. Шунинг учун унга мос майдонни яширинган шаклда ҳосил қиламиз (**Html.HiddenFor** хелперидан фойдаланамиз).

Энди бизга амалга оширилган ўзгартиришларни **МБ**сига сақлаш лозим. Контроллерда **POST-сўров**ни амалга оширувчи **EditBook** амал методини шакллантирамиз.

```
[HttpPost]
        public ActionResult EditBook(Book book)
        {
            db.Entry(book).State = EntityState.Modified;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
```

`db.Entry(book).State = EntityState.Modified;` сатри орқали биз **book** объектининг МБсида мавжудлиги ва унда амалга оширилган ўзгартиришларни МБсида акслантириш лозимлигини кўрсатамиз. Амалга оширилган ўзгартиришлар МБсида акслантирилгач, лойиҳамизнинг бош саҳифасига ўтамиз (`return RedirectToAction("Index")`).

**Entity Framework** фреймворки орқали **sql** сўровлар бажарилиши ва МБ тузилмаси ўзгартирилишини **db.SaveChanges()** буйруғи орқали моделдаги ўзгартиришларни МБсида **UPDATE** командаси бажарилади. **EditBook** методига **Home/EditBook/1** орқали мурожаат қиламиз:




**Html.EditorFor** хелпери бизга модел учун зарур бўлган майдонларни генерация қилиб берди. Биз модел маълумотларини ўзгартириб, серверга узатсак, у МБсида ўзгартиришларни амалга оширади.

## Моделларни қўшиш ва ўчириш

### Моделларни қўшиш

Юқоридаги мисолда мавжуд моделни ўзгартиришни кўриб чиқдик. **Book** модели билан ишлашни давом эттирамиз ва **МБ**да янги ёзув қўшиш ва ўчиришни кўриб чиқамиз. Бунинг учун бир нечта амал методларини шакллантирамиз:

```
[HttpGet]
public ActionResult Create()
        {
            return View();
        }
```

```
[HttpPost]
public ActionResult Create(Book book)
        {
            db.Books.Add(book);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
```

Биринчи метод фойдаланувчи учун янги **Book** объекти маълумотларини киритиш учун зарур. Иккинчиси эса формада киритилган маълумотларни қабул қилади ва **МБ**сида янги ёзувни сақлайди. Энди янги кўринишни ҳосил қиламиз.

```
@model BookStore.Models.Book

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

<h2>Новая книга</h2>

@using (Html.BeginForm())
{
    @Html.LabelFor(model => model.Name, "Название книги")
    <br />
    @Html.EditorFor(model => model.Name)
    <br /><br />
    @Html.LabelFor(model => model.Author, "Автор")
    <br />
    @Html.EditorFor(model => model.Author)
    <br /><br />
    @Html.LabelFor(model => model.Price, "Цена")
    <br />
    @Html.EditorFor(model => model.Price)
    <br /><br />
    <input type="submit" value="Добавить" />
}
```

**book** моделини қабул қилишда **Create** методида **db.Books.Add(book)** амали орқали МБсида янги ёзув ҳосил қилинади. Сўнгра **db.SaveChanges()** методи орқали **INSERT** операцияси орқали жадвалда янги ёзув ҳосил қилиниб, сақланади. **Create** методини қуйидагича ёзиб олишимиз мумкин:

```
[HttpPost]
public ActionResult Create(Book book)
```

```
        {
            db.Entry(book).State = EntityState.Added;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
```

Амал методини **(http://localhost:2044/Home/Create)** орқали чақириш мумкин.

**Моделни ўчириш**

Моделни ўчириш учун МБсида қуйидаги амалларни бажаришимиз лозим:

```
public ActionResult Delete(int id)
        {
            Book b = db.Books.Find(id);
            if (b != null)
            {
                db.Books.Remove(b);
                db.SaveChanges();
            }
            return RedirectToAction("Index");
        }
```

Аввало биз **Id** кодига мос ёзувнинг **МБ**сида мавжудлигини аниқлаймиз. Агар бундай ёзув мавжуд бўлса, **db.Books.Remove(b)** методини чақирамиз. У модел статусини **Deleted** га ўзгартиради. **Entity Framework** фреймборки **db.SaveChanges** методини чақириш орқали **DELETE sql**-ифодасини амалга оширади. Аммо биз ошкор тарзда ушбу ишни амалга оширишимиз ҳам мумкин:

```
public ActionResult Delete(int id)
        {
            Book b = new Book { Id = id };
            db.Entry(b).State = EntityState.Deleted;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
```

Бундай ёндашувнинг ўзига хос томони мавжуд бўлиб, биринчи усулда **Book b = db.Books.Find(id);** ифодасидан фойдаланилган ҳолда МБсига қўшимча сўров амалга оширилади. Иккинчи ҳолда эса **МБ**сига битта сўров амалга

оширилади. Аммо ушбу усулнинг камчилиги ҳам мавжуд. Масалан, бизга <img src="http://Сайт_номи/Home/Delete/1" /> каби электрон хабар келсин. Натижада жадвалнинг 1-ёзуви ўчирилади. **Delete** методи **GET-сўров** каби амалга ошириш хавфсизлик жиҳатдан нотўғри амал ҳисобланади. Шунинг учун ушбу методни қуйидагича ўзгартириб оламиз:

```
[HttpGet]
public ActionResult Delete(int id)
        {
            Book b = db.Books.Find(id);
            if (b == null)
            {
                return HttpNotFound();
            }
            return View(b);
        }

[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id)
        {
            Book b = db.Books.Find(id);
            if (b == null)
            {
                return HttpNotFound();
            }
            db.Books.Remove(b);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
```

Эндиликда битта **Delete** метод ўрнига иккита методдан фойдаланилмоқда. Амал методидаги **ActionName("Delete")** атрибути **DeleteConfirmed** методининг **Delete** каби қабул қилинишини кўрсатади. Биринчи метод ўчирилиши лозим бўлган моделни кўринишга узатади. Кўринишда тугма босилиши орқали биз модел ўчирилишини тасдиқлаймиз. Ўчирилаётган **Id** га мос модел иккинчи методга **POST** сўрови каби узатилади. Натижада биз **GET**-сўровнинг камчилигини бартараф қиламиз. Кўриниш эса қуйидаги шаклда амалга оширилади:

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

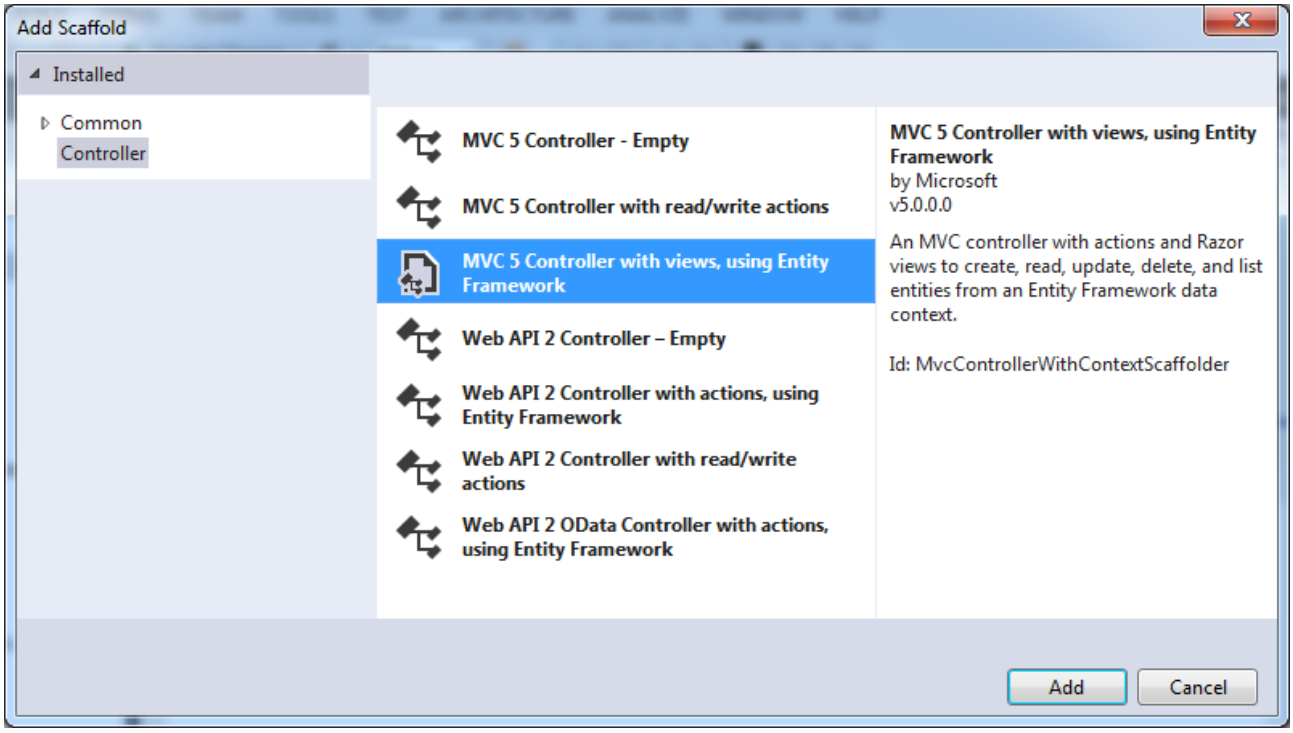
```
}
@model BookStore.Models.Book
<h2>Удаление книги</h2>
<dl>
    <dt>Название</dt>
    <dd>
        @Html.DisplayFor(model => model.Name)
    </dd>

    <dt>Автор</dt>
    <dd>
        @Html.DisplayFor(model => model.Author)
    </dd>

    <dt>Цена</dt>
    <dd>
        @Html.DisplayFor(model => model.Price)
    </dd>
</dl>

@using (Html.BeginForm())
{
    <input type="submit" value="Удалить" />
}
```

**(http://localhost:2044/Home/Delete/4**)

## Шакллантириш шаблонлари

Кўпгина дастурлар стандарт **CRUD**-амаллари (**Create - Read - Update - Delete**)га асосланган бўлади. Акс ҳолда дастурчилар бир нечта контроллерлар ва кўринишларда модел учун **МБ**сида қўшиш, ўзгартириш, ўчириш ва ёзувни кўриш каби амалларни шакллантиришлари лозим. Дастурчилар ишини осонлаштириш учун **MVC шакллантириш шаблони** (**scaffolding templates**) деб номланувчи функционалликни ишлаб чиқишган. Ушбу шаблонлар орқали берилган модел ва маълумотлар контекстига мос зарур стандарт контроллер ва кўринишларни генерация қилади.

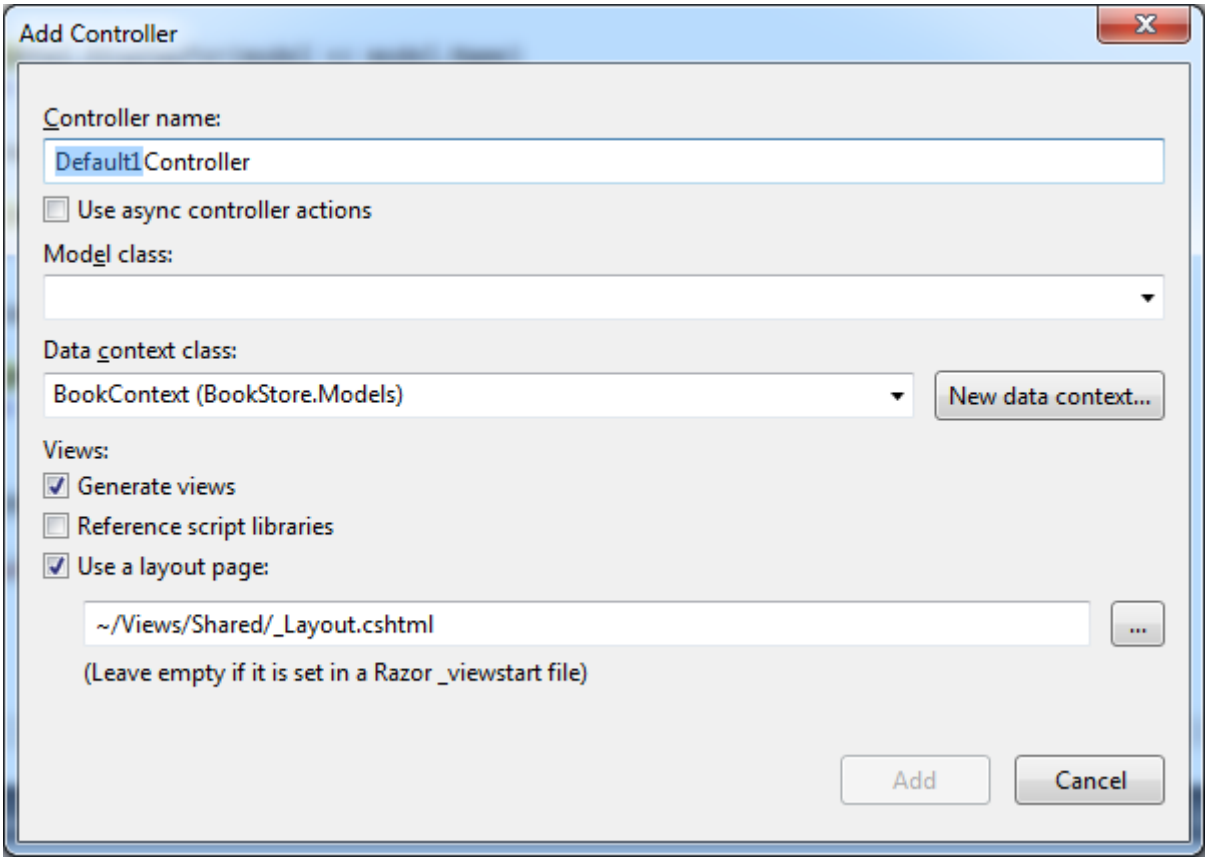
Ушбу фукнционалликдан фойдаланиш учун лойиҳамизга янги контроллер қўшамиз. **Controllers** папкасига сичқончанинг ўнг тугмасини босиб, **Add -> Controller...**ни танлаймиз. Ҳосил қилинган мулоқот ойнасидан зарур шаблонни танлаймиз:




Ушбу рўйхатдаги шаблонлардан биринчи учтаси **MVC** га таалуқли.

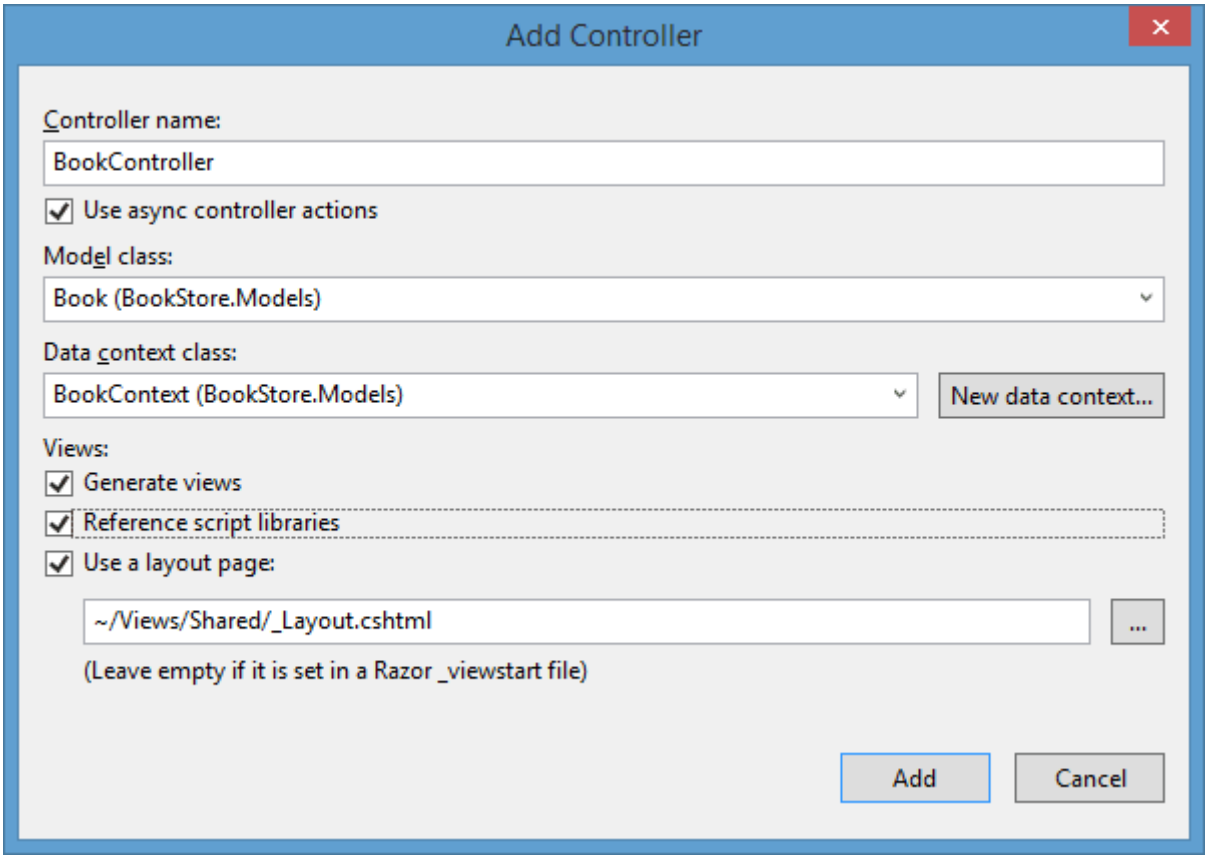
- **MVC 5 Controller - Empty**. Ушбу шаблон орқали **Controllers** папкасида бўш контроллер ҳосил қилинади. Ҳосил қилинган контроллерда ягона **Index** методи мавжуд бўлиб, ушбу шаблон орқали кўриниш ҳосил қилинмайди.
- **MVC 5 Controller with read/write actions**. Ушбу шаблон лойиҳамизга **Index, Details, Create, Edit** ва **Delete** методларидан иборат контроллерни қўшиб қўяди. Аммо ушбу методлар **МБ**си билан боғлиқ ҳеч қандай мантиқни ўзида сақламайди. Биз ўзимиз ушбу методлар учун код ва кўринишларни шакллантиришимиз лозим.
- **MVC 5 Controller with views, using Entity Framework**. Ушбу шаблон орқали **Index, Details, Create, Edit** ва **Delete** методларидан иборат контроллер ва кўринишлар ҳосил қилинади. Шунингдек, ушбу шаблон орқали **МБ**си билан боғлиқ барча амаллар ва кодлар автоматик генерация қилинади.

Биз ушбу рўйхатдаги учинчи элементни (**MVC 5 Controller with views, using Entity Framework**) танлаймиз. Натижада лойиҳамизга янги контроллер қўшиш учун мулоқот ойнаси тақдим этилади:




Ушбу мулоқот ойнасида баъзи қийматларни ўрнатиш лозим:

- **Controller name**: контроллер номи;
- **Use async controller actions**: асинхрон методлар автоматик генерация қилинадими? Ушбу опцияни ўрнатамиз;
- **Model class**: модел класси. Аввал ҳосил қилинган **Book** моделини танлаймиз (ёки мавжуд бошқа бирор модел);
- **Data context class**: маълумотлар контекси класси. Танланган моделга мос маълумотлар контекстини танлаймиз;
- **Generate views**: ҳосил қилинаётган контроллерга мос кўринишларни генерация қилиш шартми? Ушбу опция ўрнатилгач, қуйида келтирилган иккита опцияга доступ очилади. Барча опцияларни ўрнатамиз.
- **Reference script libraries**: кўринишда **jquery** библиотекаси ва бошқа зарурий **javascript** файллар мавжудлиги;
- **Use a layout page**: генерация қилинаётган кўринишларда мастер-саҳифалардан фойдаланиш.




Барча зарур опциялар ўрнатилгач, **Add** тугмасини босамиз ва лойиҳада янги контроллер қўшиб қўйилади. У қуйидагича кўринишга эга бўлади:

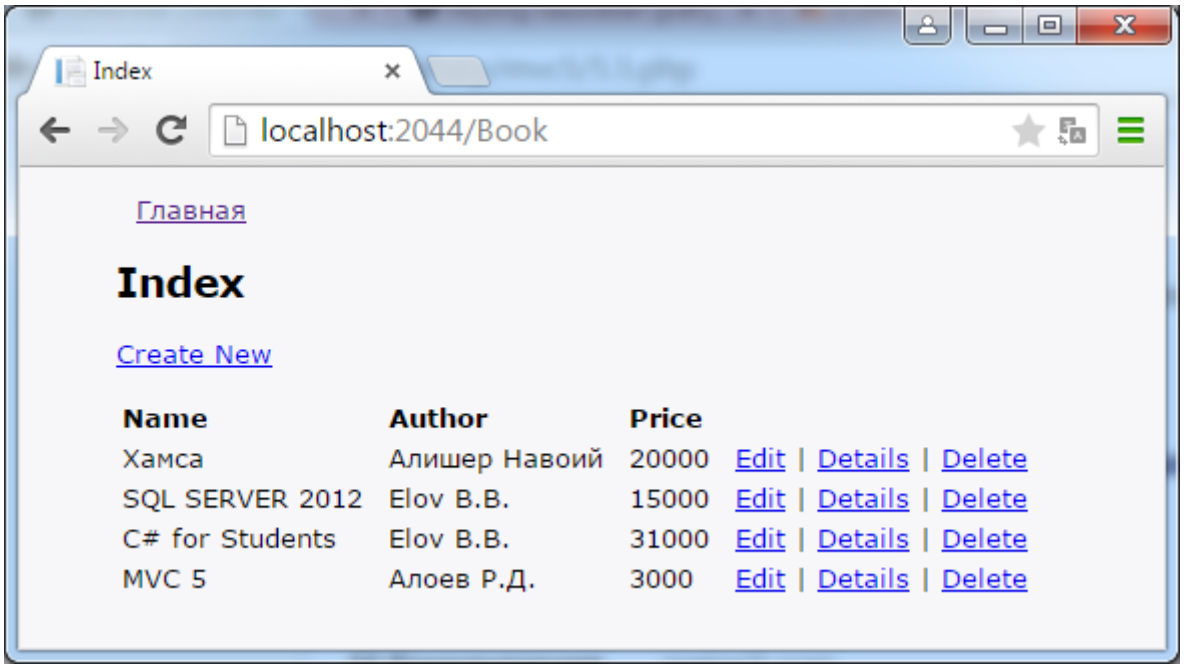
```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Threading.Tasks;
using System.Net;
using System.Web;
using System.Web.Mvc;
using BookStore.Models;

namespace BookStore.Controllers
{
    public class BookController : Controller
    {
        private BookContext db = new BookContext();

        // GET: /Book/
        public async Task<ActionResult> Index()
        {
```

```
            return View(await db.Books.ToListAsync());
        }

        // GET: /Book/Details/5
        public async Task<ActionResult> Details(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Book book = await db.Books.FindAsync(id);
            if (book == null)
            {
                return HttpNotFound();
            }
            return View(book);
        }

        // GET: /Book/Create
        public ActionResult Create()
        {
            return View();
        }

        // POST: /Book/Create
        // To protect from overposting attacks, please enable the
specific properties you want to bind to, for
        // more details see
http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
Create([Bind(Include="Id,Name,Author,Price")] Book book)
        {
            if (ModelState.IsValid)
            {
                db.Books.Add(book);
                await db.SaveChangesAsync();
                return RedirectToAction("Index");
            }

            return View(book);
        }

        // GET: /Book/Edit/5
        public async Task<ActionResult> Edit(int? id)
        {
            if (id == null)
```

```
            {
                return new
HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Book book = await db.Books.FindAsync(id);
            if (book == null)
            {
                return HttpNotFound();
            }
            return View(book);
        }

        // POST: /Book/Edit/5
        // To protect from overposting attacks, please enable the
specific properties you want to bind to, for
        // more details see
http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
Edit([Bind(Include="Id,Name,Author,Price")] Book book)
        {
            if (ModelState.IsValid)
            {
                db.Entry(book).State = EntityState.Modified;
                await db.SaveChangesAsync();
                return RedirectToAction("Index");
            }
            return View(book);
        }

        // GET: /Book/Delete/5
        public async Task<ActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Book book = await db.Books.FindAsync(id);
            if (book == null)
            {
                return HttpNotFound();
            }
            return View(book);
        }

        // POST: /Book/Delete/5
        [HttpPost, ActionName("Delete")]
```

```
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> DeleteConfirmed(int id)
        {
            Book book = await db.Books.FindAsync(id);
            db.Books.Remove(book);
            await db.SaveChangesAsync();
            return RedirectToAction("Index");
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

**Views/Book** папксида эса ушбу контроллерга мос барча зарур кўринишлар ҳосил қилинган. Энди биз лойиҳани ишга тушириб, браузернинг манзиллар сатрида қуйидаги ёзувни киритамиз: **http://localhost:2044/Book**




Шакллантириш шаблони асосида бир қанча жадваллар учун стандарт амалларни бажаришимиз осонлашади ва вақт тежалади. Аммо генерация

қилинган код ва кўринишларга бироз ўзгартиришларни амалга оширишга тўғри келади.

## Мураккаб тузилмали моделлар

Аввалги мисолларда биз нисюатан оддий ҳисобланган **Book** ва **Purchase** моделлари устида амаллар бажарилган эди. Аммо ҳаётий масалалардаги моделлар мураккаб тузилмага эга. Масалан, футболчи ва футбол командасини ифодаловчи моделни шакллантирамиз:

```
public class Player
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
        public string Position { get; set; }
        public int? TeamId { get; set; }
        public Team Team { get; set; }

    }

public class Team
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Coach { get; set; }
        public IEnumerable<Player> Players { get; set; }
    }
```

**Player** классида стандарт хусусиятлардан ташқари, **Team** хусусияти мавжуд бўлиб, у орқали футболчининг муайян командага мансублиги аниқланади.

**Team** хусусияти ушбу ҳолда навигацион хусусиятга эга бўлади. Навигацион хусусиятга эга бўлган объектни биз **МБ**сидан олишимиз мумкин. Бунинг учун ташқи калитни ўрнатишимиз лозим.

Ташқи калит иккита хусусиятдан иборат: **навигацион** ва **оддий**. Навигацион хусусиятни биз юқорида кўриб чиқдик. Оддий хусусият қуйидаги номлардан бирини қабул қилади:

- **Навигацион хусусият номи+боғланган жадвал калит номи** – бизнинг ҳолда навигацион хусусият номи **Team**, моделдаги **Team** калити – **ID.** Шунинг учун бизда хусусият номи - **TeamId.**
- **Боғланган жадвал класс номи+боғланган жадвал калит номи** – бизнинг ҳолда **Team** класси, **Team** классидаги калит – **Id.** Ушбу ҳолда ҳам хусусият номи - **TeamId.**

Энди моделга мос маълумотлар контекстнини ҳосил қилишимиз мумкин:

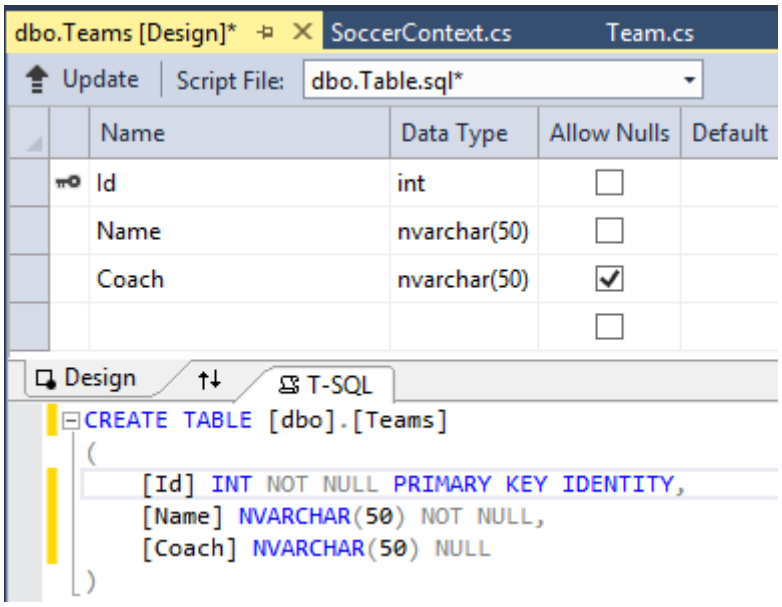
```
public class SoccerContext : DbContext
    {
        public DbSet<Player> Players { get; set; }
        public DbSet<Team> Teams { get; set; }
    }
```

Энди юқоридагиларни **МБ**сида акслантирамиз. Бунинг учун **SoccerInfo.mdf МБ**сини ҳосил қилаамиз. **Player** ва **Team** моделларни сақлаш учун мос равишда **Players** ва **Teams** жадвалларидан фойдаланилади.

**Teams** жадвалини қуйидагича шакллантирамиз:

```
CREATE TABLE [dbo].[Teams]
(
      [Id] INT NOT NULL PRIMARY KEY,
    [Name] NVARCHAR(50) NOT NULL,
    [Coach] NVARCHAR(50) NULL
)
```

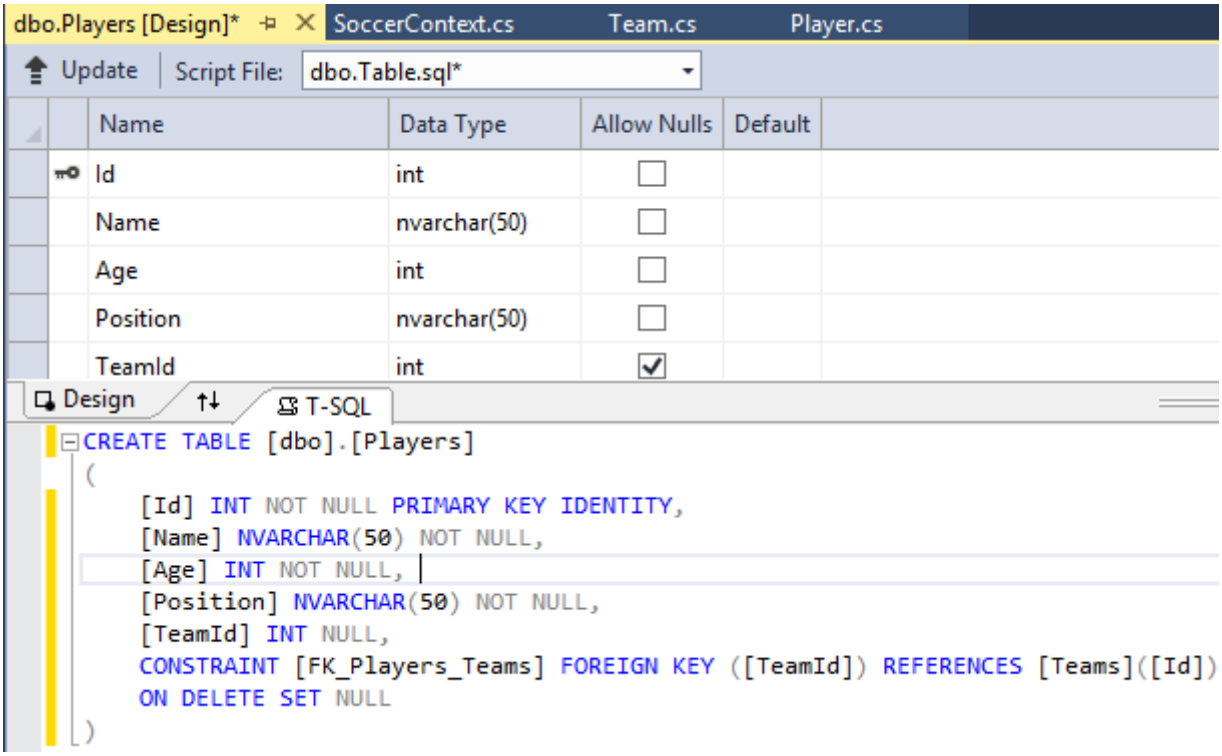
ёки



**Players** жадвалини қуйидагича аниқлаб оламиз:

```
CREATE TABLE [dbo].[Players]
(
	[Id] INT NOT NULL PRIMARY KEY IDENTITY,
    [Name] NVARCHAR(50) NOT NULL,
    [Age] INT NOT NULL,
    [Position] NVARCHAR(50) NOT NULL,
    [TeamID] INT NULL,
	CONSTRAINT [FK_Players_Teams] FOREIGN KEY ([TeamId]) references
	[Teams]([Id])
	ON DELETE SET NULL
)
```

ёки




**Teams** жадвалидан фарқли равишда **Players** жадвалида ташқи калит (**TeamId**) хусусияти орқали **Teams** жадвалидаги **Id** майдони билан боғланган. Ташқи калитни ҳосил қилиш учун **SQL** панелидаги дизайнерда қуйидаги сатрни ҳосил қилишимиз лозим:

```
CONSTRAINT [FK_Players_Teams] FOREIGN KEY ([TeamId]) references
	[Teams]([Id])
```

Бу **SQL** тилида ёзилган оддий ифода бўлиб, иккита жадвалнинг устунларини ўзаро боғлайди. Ушбу **SQL** ифоданинг охирги қисми

```
ON DELETE SET NULL
```

орқали **Teams** жадвалидаги объект ўчирилаётганда (**TeamId** хусусияти бўйича) ушбу қийматга мос боғланган бошқа жадвалдаги устунларга **null** қиймати берилади.

Ушбу амал орқали бирор команда **МБ**дан ўчирилганда, ушбу команда билан боғлиқ команда аъзоларидаги **TeamId** қийматига **null** қиймати берилади. Аммо бирор команда ўчирилаётганда ушбу командага бириктирилган ўйинчилар ҳам ўчирилиши лозим бўлса, **ON DELETE CASCADE** ёзувидан фойдаланиш лозим.

Жадваллар аниқланганидан сўнг, уларни қийматлар билан тўлдирамиз. Мисол сифатида **Teams** жадвалини қуйидагича тўлдиришимиз мумкин:

dbo.Teams [Data] | SoccerContext.cs | Team.cs

Max Rows: 1000

| Id | Name | Coach |
|---|---|---|
| 1 | Реал Мадрид | Анчелотти |
| 2 | Барселона | Мартино |
| 3 | Бавария | Гуардиола |
| 4 | Боруссия | Клопп |
| NULL | NULL | NULL |

**Players** жадвалига ҳам (**TeamId** устунида **Team** жадвалидаги **Id** устунига мос қийматлар ёзилади) маълумотларни киритамиз:

dbo.Players [Data] | dbo.Players [Design] | SoccerContext.cs* | Team.cs*

Max Rows: 1000

| Id | Name | Age | Position | TeamID |
|---|---|---|---|---|
| 2 | Месси | 26 | Хужумчи | 2 |
| 3 | Роналду | 29 | Хужумчи | 1 |
| 4 | Бейл | 24 | Ярим химоячи | 1 |
| 5 | Неймар | 22 | Хужумчи | 2 |
| 6 | Рибери | 30 | Ярим химоячи | 3 |
| NULL | NULL | NULL | NULL | NULL |

Энди дастур мантиғини шакллантирамиз. Лойиҳамизга юқоридаги моделларга мос контроллерни ҳосил қиламиз:

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

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

namespace NavigationProperty.Controllers
{
    public class SoccerController : Controller
    {
        SoccerContext db = new SoccerContext();
        // Выводим всех футболистов
        public ActionResult Index()
        {
            var players = db.Players.Include(p => p.Team);
            return View(players.ToList());
        }

     }
}
```

**Include** методи ёрдамида фреймворк командининг ҳар бир аъзосини юклаб, у билан боғлиқ маълумотларни ташқи калит орқали боғлайди. Кейинги қадамда **Index.cshtml** кўринишни ҳосил қиламиз:

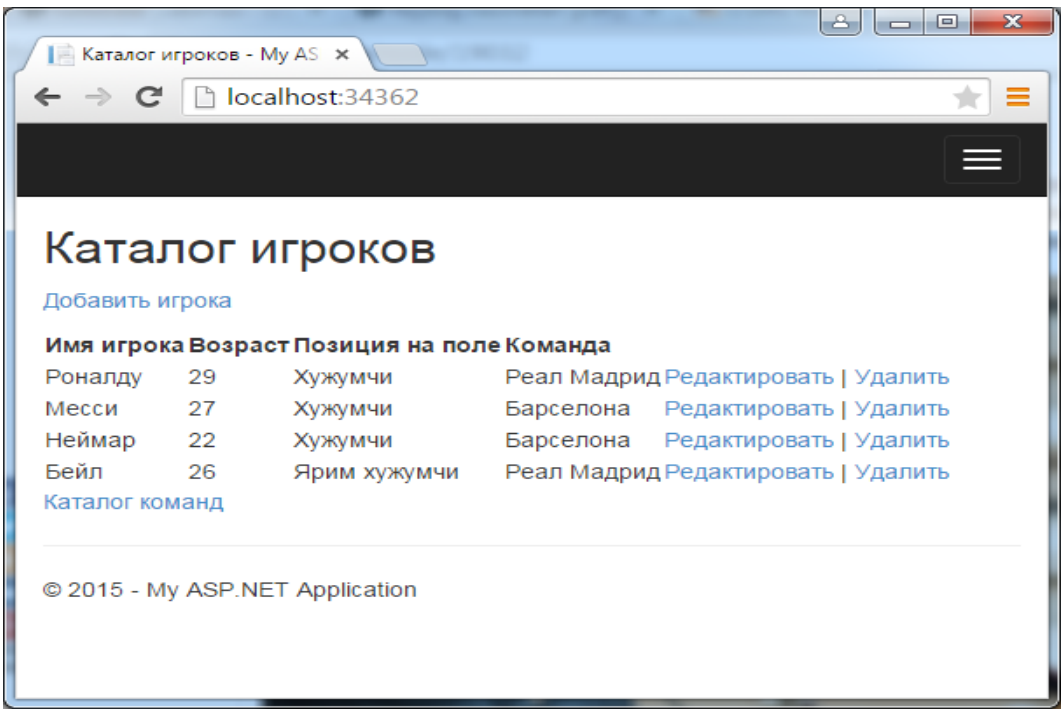
```
@model IEnumerable<NavigationProperty.Models.Player>
@{
    ViewBag.Title = "Каталог игроков";
}
<h2>Каталог игроков</h2>
<p>
    @Html.ActionLink("Добавить игрока", "Create")
</p>
<table>
    <tr>
        <th>Имя игрока</th>
        <th>Возраст</th>
        <th>Позиция на поле</th>
        <th>Команда</th>
        <th></th>
    </tr>

    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Age)
            </td>
```

```
            <td>
                @Html.DisplayFor(modelItem => item.Position)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Team.Name)
            </td>
            <td>
                @Html.ActionLink("Редактировать", "Edit", new { id =
item.Id }) |
                @Html.ActionLink("Удалить", "Delete", new { id = item.Id
})
            </td>
        </tr>
    }
</table>
<p>
    @Html.ActionLink("Каталог команд", "ListTeams")
</p>
```

ва **web.config** файлида қуйидаги сатрни қўшиб қўйиш лозим:

```
<connectionStrings>
    <add name="SoccerContext" connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename='|DataDirectory|\SoccerInfo.mdf';
Integrated Security=True"
   providerName="System.Data.SqlClient" />
  </connectionStrings>
```

(**http://localhost:2044/Soccer/Index**)



Контроллердаги амал методида **Include** методи ёрдамида барча **Player** моделига ўзининг **Team** объектига навигацион **TeamId** хусусияти орқали уланади. Кўринишда биз ушбу боғланган **Team** объектини олишимиз ва унинг хусусиятларидан фойдаланишимиз мумкин. Мисол сифатида, команда номини ифодаловчи **item.Team.Name** ни олишимиз мумкин.

Худди юқорида келтирилган усулда командалар рўйхатини ҳам шакллантириш мумкин. Аммо у ерда бажариладиган амаллар жуда оддий ва қизиқарли эмас. Чунки у жадвалда иккиламчи калитга эга бўлган майдонлар мавжуд эмас. Аммо бизнинг **Team** моделдаги **Players** хусусияти командадаги аъзолар рўйхатини ўзида сақлаши лозим. Команда ва унинг аъзоларини моделимизда шакллантирамиз. Аввало контроллерда қуйидаги методни аниқлаб оламиз:

```
public ActionResult TeamDetails(int? id)
        {
            if (id == null)
            {
                return HttpNotFound();
            }
            Team team = db.Teams.Find(id);
            if (team == null)
            {
                return HttpNotFound();
            }
            team.Players = db.Players.Where(m => m.TeamId == team.Id);
            return View(team);
        }
```

Биринчидан, узатилаётган параметр мавжуд бўлмаган ҳол учун параметр сифатида **int? id** ни ишлатамиз. Иккинчидан биз командадаги барча аъзоларни
`team.Players = db.Players.Where(m => m.TeamId == team.Id);`
ифода орқали юклаб оламиз.

Сўнгра **TeamDetails.cshtml** кўринишда маълумотларни ифодалашни амалга оширамиз:

```
@using NavigationProperty.Models
@model Team

@{
    ViewBag.Title = "Команда " + @Model.Name;
    Layout = "~/Views/Shared/_Layout.cshtml";
```

```
}

<div>
    <h4>Команда @Model.Name</h4>
    <hr />
    <dl>
        <dt>Название</dt>

        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>

        <dt>Тренер</dt>

        <dd>
            @Html.DisplayFor(model => model.Coach)
        </dd>

        <dt>Игроки</dt>

        <dd>
            <ul>
                @foreach (Player player in Model.Players)
                {
                    <li>@player.Name (@player.Position)</li>
                }
            </ul>
        </dd>
    </dl>
</div>
```

**(http://localhost:34362/Home/TeamDetails/1)**

## Мураккаб моделлар билан ишлаш

Аввалги мисолда биз иккита **Player** ва **Team** моделларини ҳосил қилдик ва **Players** жадвалидаги маълумотларни саҳифага чиқардик. Энди, модел билан боғлиқ бўлган бошқа амалларни шакллантирамиз. Ушбу амалларни оддий моделларда бажарилган амаллар каби шакллантирамиз. Улардаги ягона фарқ – мураккаб моделдаги навигацион хусусият ҳисобланади.

**Модел қўшиш**

Ташқи калитга эга бўлган моделни қўшиш учун оддий модел каби иш юритилади. Фақат қўшимча сифатида қийматлар рўйхатига бошқа жадвалдаги майдон билан боғланган устун қийматини узатиш лозим. Контроллерга **Create** амал методини шакллантирамиз:

```
[HttpGet]
public ActionResult Create()
        {
            // Формируем список команд для передачи в представление
            SelectList teams = new SelectList(db.Teams, "Id", "Name");
            ViewBag.Teams = teams;
            return View();
        }

[HttpPost]
public ActionResult Create(Player player)
        {
            //Добавляем игрока в таблицу
            db.Players.Add(player);
            db.SaveChanges();
            // перенаправляем на главную страницу
            return RedirectToAction("Index");
        }
```

Биринчи метод **GET**-сўровни қайта ишлайди ва кўринишга барча командалар рўйхати **SelectList** ни узатади.

Иккинчи метод фойдаланувчи томонидан кўринишда киритилган моделни қабул қилади ва уни **МБ**сига қўшиб қўяди. Энди эса **Create.cshtml** кўринишни ҳосил қиламиз:

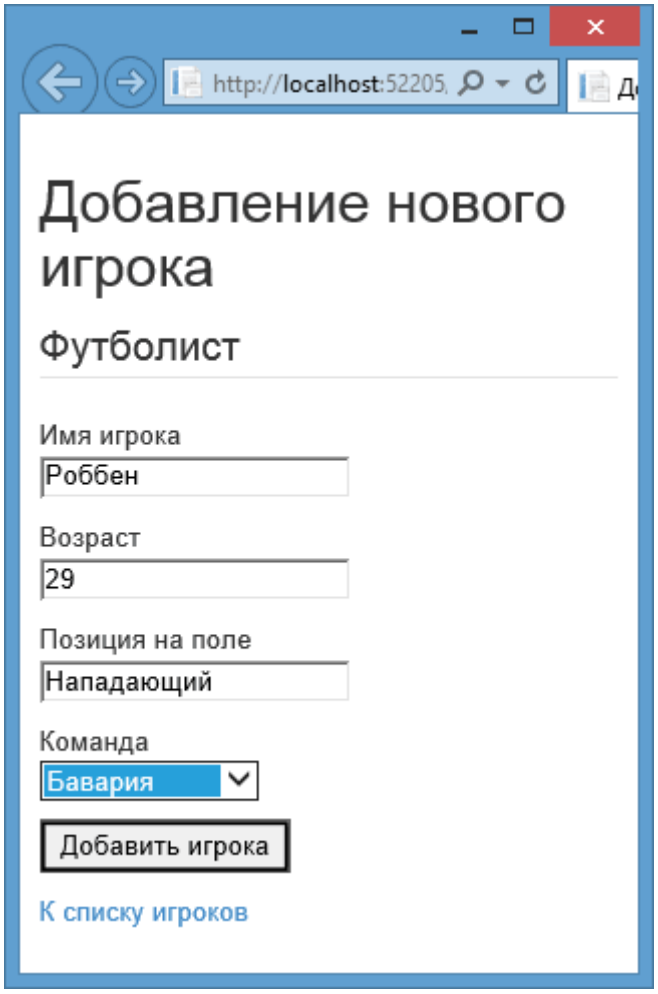
```
@model NavigationProperty.Models.Player

@{
    ViewBag.Title = "Добавление игрока";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Добавление нового игрока</h2>

@using (Html.BeginForm())
{
```

```
    <fieldset>
        <legend>Футболист</legend>

        <p>
            Имя игрока <br />
            @Html.EditorFor(model => model.Name)
        </p>

        <p>
            Возраст <br />
            @Html.EditorFor(model => model.Age)
        </p>

        <p>
            Позиция на поле  <br />
            @Html.EditorFor(model => model.Position)
        </p>
        <p>
            Команда <br />
            @Html.DropDownListFor(model => model.TeamId, ViewBag.Teams as
SelectList)
        </p>

        <p>
            <input type="submit" value="Добавить игрока" />
        </p>
    </fieldset>
}
<div>
    @Html.ActionLink("К списку игроков", "Index")
</div>
```

Оддий моделлар каби ушбу ҳолда ҳам биз майдонни муайян хусусиятга бириктириб қўямиз.

Бу ерда фақат командалар рўйхати орқали зарур команда танланади. Рўйхатдан танлаб олинган қиймат **TeamId** модели хусусиятига боғланади.

Браузерга **http://localhost:34362/Home/Create** сатри киритирлиши натижасида қуйидаги интерфейс ҳосил қилинади:




**Моделни таҳрирлаш**

Моделни таҳрирлаш юқорида келтирилган усулда амалга оширилади. Контроллерда **Edit** амал методини шакллантирамиз:

```
[HttpGet]
public ActionResult Edit(int? id)
        {
            if (id == null)
            {
                return HttpNotFound();
            }
            // Находим в бд футболиста
            Player player = db.Players.Find(id);
            if (player != null)
            {
                // Создаем список команд для передачи в представление
                SelectList teams = new SelectList(db.Teams, "Id", "Name",
player.TeamId);
                ViewBag.Teams = teams;
```

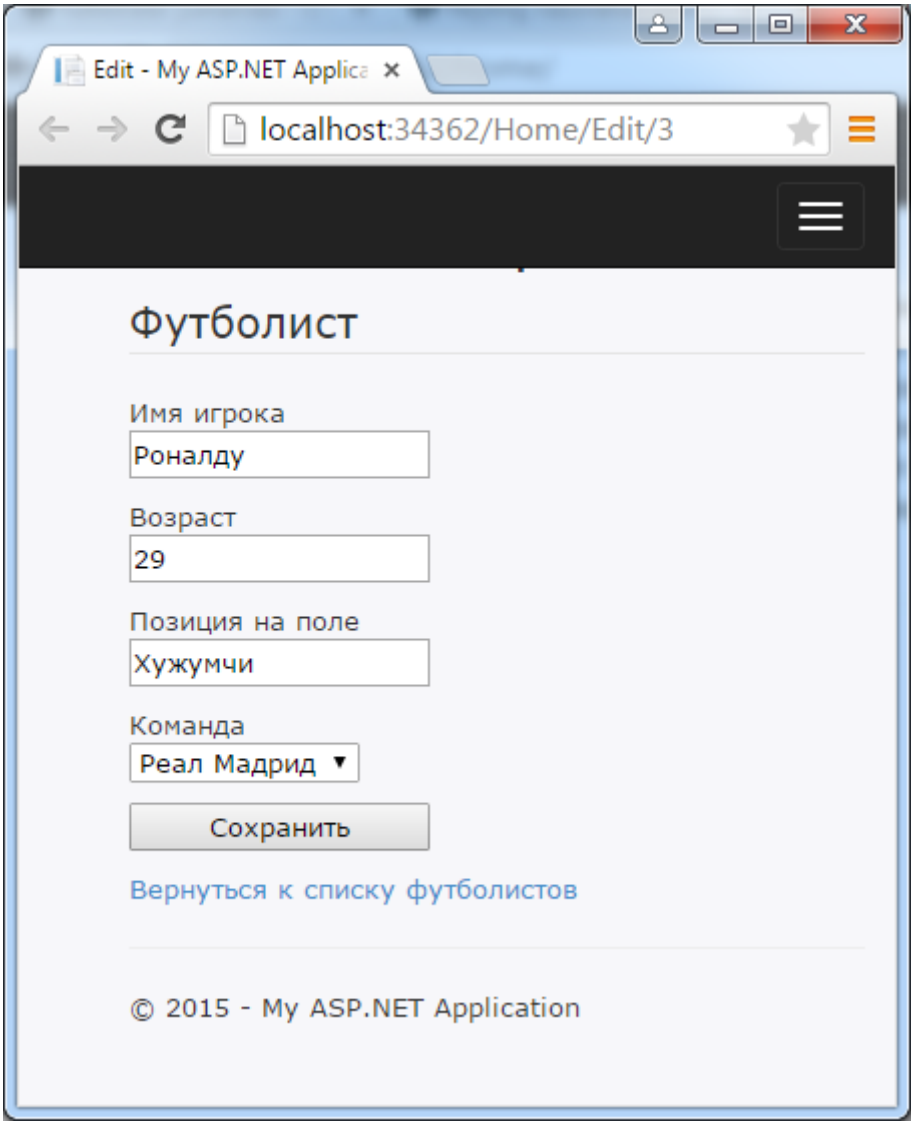
```
            return View(player);
        }
        return RedirectToAction("Index");
    }

[HttpPost]
public ActionResult Edit(Player player)
    {
        db.Entry(player).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
```

Ушбу амал методида командалар рўйхати учун **SelectList** объектидан фойдаланилган. Ушбу объектда маълумотлар **МБ**сидан ҳосил қилинади. Муайян **Player** моделини таҳрирлаш фойдаланувчи томонидан амалга оширилганда, контроллер ушбу модел ва командалар рўйхатини **Edit.cshtml** кўринишига узатади.

```
@model NavigationProperty.Models.Player
@{
    ViewBag.Title = "Edit";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Изменение игрока</h2>

@using (Html.BeginForm())
{
    <fieldset>
        <legend>Футболист</legend>

        @Html.HiddenFor(model => model.Id)

        <p>
            Имя игрока <br />
            @Html.EditorFor(model => model.Name)
        </p>

        <p>
            Возраст <br />
            @Html.EditorFor(model => model.Age)
        </p>

        <p>
            Позиция на поле  <br />
```

```
            @Html.EditorFor(model => model.Position)
        </p>
        <p>
            Команда <br />
            @Html.DropDownListFor(model => model.TeamId, ViewBag.Teams as
SelectList)
        </p>
        <p>
            <input type="submit" value="Сохранить" />
        </p>
    </fieldset>
}
<div>
    @Html.ActionLink("Вернуться к списку футболистов", "Index")
</div>
```

http://localhost:34362/Home/Edit/3



Моделни ўчириш худди оддий моделни ўчириш каби амалга оширилади. Контроллерда **Delete** амал методини қуйидагича ҳосил қиламиз:

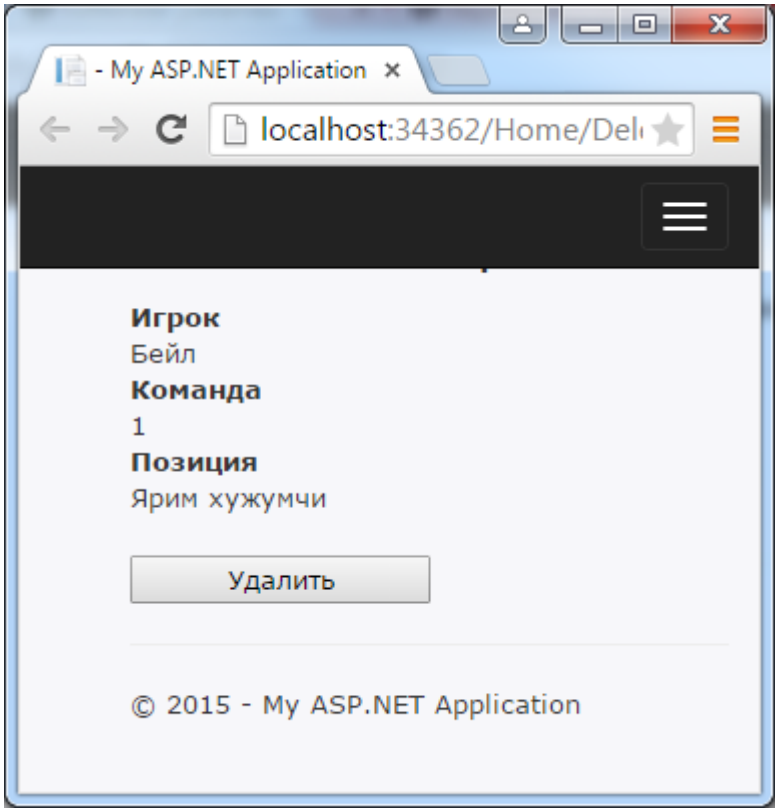
```
[HttpGet]
public ActionResult Delete(int id)
        {
            if (id == null)
            {
                return HttpNotFound();
            }

            // Находим в бд футболиста
            Player player = db.Players.Find(id);
            if (player != null)
            {
                //// Создаем список команд для передачи в представление
                //SelectList teams = new SelectList(db.Teams, "Id",
"Name", player.TeamId);
                //ViewBag.Teams = teams;
                return View(player);
            }
            return RedirectToAction("Index");
        }

[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id)
        {
            Player b = db.Players.Find(id);
            if (b == null)
            {
                return HttpNotFound();
            }
            db.Players.Remove(b);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
```

ва ушбу амал методига мос кўринишни шакллантирамиз:

```
@model NavigationProperty.Models.Player
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Удаление игрока</h2>
<dl>
```

```
    <dt>Игрок</dt>
    <dd>
        @Html.DisplayFor(model => model.Name)
    </dd>

    <dt>Команда</dt>
    <dd>
        @Html.DisplayFor(model => model.TeamId)
    </dd>

    <dt>Позиция</dt>
    <dd>
        @Html.DisplayFor(model => model.Position)
    </dd>
</dl>

@using (Html.BeginForm())
{
    <input type="submit" value="Удалить" />
}
```

**http://localhost:34362/Home/Delete/6**



## Кўпга-кўп боғланишга эга моделлар

Моделларнинг 1:1 ва 1:n муносабатидан ташқари n:m муносабатдаги моделлар ҳам мавжуд. Агар реал ҳаётга қаралса ушбу турдаги бир қанча моделларни кўриш мумкин. Олий таълим муассасасидаги ўқув жараёнида турли сондаги талабалар турли фанлардан таҳсил олишади. Муайян ўқув фани бўйича таҳсил олаётган талабалар ҳисобини юритиш ва ҳар бир талаба таҳсил олаётган бир нетча ўқув фанлари ҳисобини юритишга тўғри келади. Ушбу бизнес-жараённи **ASP.NET MVC 5** лойиҳасида моделлаштирамиз.

Аввало янги **Study** лойиҳасини яратишимиз лозим. Биринчи навбатда лойиҳамизда ишлатиладиган моделларни шакллантирамиз. Соддалик учун **Code First** ёндашувидан фойдаланамиз.

Бизда иккита модел мавжуд бўлсин: **талаба** ва **ОТМда ўқитиладиган фан**. Аввало **Student** моделини қуйидагича шакллантириб оламиз:

```
public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Surname { get; set; }
        public virtual ICollection<Course> Courses { get; set; }

    }
```

ва **Course** моделини қўшамиз:

```
public class Course
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public virtual ICollection<Student> Students { get; set; }

    }
```

**Student** ва **Course** моделлари жуда содда тарзда ташкил қилинган бўлиб, уларда қуйидаги иккита виртуал хусусиятлар мавжуд: **Students** ва **Courses.**

Ушбу хусусиятлар ёрдамида **n:m** муносабат амалга оширилади. Кейинги босқичда маълумотлар контекстини ҳосил қилиш лозим. Лойиҳага **StudentsContext** классини қўшамиз:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;

namespace Study.Models
{
    public class StudentsContext : DbContext
    {
        public DbSet<Student> Students { get; set; }
        public DbSet<Course> Courses { get; set; }

        public StudentsContext()
            : base("DefaultConnection") { }

        protected override void OnModelCreating(DbModelBuilder
modelBuilder)
        {
            modelBuilder.Entity<Course>().HasMany(c => c.Students)
                .WithMany(s => s.Courses)
                .Map(t => t.MapLeftKey("CourseId")
                .MapRightKey("StudentId")
                .ToTable("CourseStudent"));
        }
    }

}
```

Биринчидан **МБ**сига уланиш сатрини шакллантириш учун маълумотлар контекстидаги

```
public StudentsContext(): base("DefaultConnection") { }
```

Кейинги қадамда ҳосил қилинаётган **МБ**да талабалар ҳақидаги барча маълумотлар **Students** жадвалида сақланади. **ОТМ**даги ўқув фанлари ҳақидаги барча маълумотлар эса **Courses** жадвалида сақланади. Аммо ушбу жадваллар ўзаро n:m муносабатда боғланган бўлиши лозим. Ушбу алоқани **CourseStudent** жадвали амалга оширади.

Ушбу жадвални ҳосил қилиш учун **OnModelCreating** методидан фойдаланамиз. **modelBuilder** объекти ёрдамида янги жадвал ва унинг майдонларини ҳосил қиламиз.

Ундаги **CourseId** майдони **Courses** жадвали билан боғланган бўлиб, ўқув фанининг **Id** майдони қийматини ўзида сақлайди. Иккинчи майдон **StudentId**

эса талабалар жадвалига узатмани шакллантиради ва талабанинг **Id** сини сақлайди.

Натижада бизда иккита моделнинг **Id** хусусиятларидан иборат жуфтлик ҳосил бўлади ва биз **n:m** тарзда алоқани шакллантирамиз. Энди **МБ** жадвалларини бошланғич қийматлар билан тўлдирамиз. Лойиҳамизга қуйидаги классни қўшамиз:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;

namespace Study.Models
{
    public class CourseDbInitializer :
DropCreateDatabaseAlways<StudentsContext>
    {
        protected override void Seed(StudentsContext context)
        {
            Student s1 = new Student { Id = 1, Name = "Егор", Surname =
"Иванов" };
            Student s2 = new Student { Id = 2, Name = "Мария", Surname =
"Васильева" };
            Student s3 = new Student { Id = 3, Name = "Олег", Surname =
"Кузнецов" };
            Student s4 = new Student { Id = 4, Name = "Ольга", Surname =
"Петрова" };

            context.Students.Add(s1);
            context.Students.Add(s2);
            context.Students.Add(s3);
            context.Students.Add(s4);

            Course c1 = new Course
            {
                Id = 1,
                Name = "Операционные системы",
                Students = new List<Student>() { s1, s2, s3 }
            };
            Course c2 = new Course
            {
                Id = 2,
                Name = "Алгоритмы и структуры данных",
                Students = new List<Student>() { s2, s4 }
            };
```

```
            Course c3 = new Course
            {
                Id = 3,
                Name = "Основы HTML и CSS",
                Students = new List<Student>() { s3, s4, s1 }
            };

            context.Courses.Add(c1);
            context.Courses.Add(c2);
            context.Courses.Add(c3);

            base.Seed(context);
        }
    }
}
```

Биринчи қадамда МБсида талабалар ҳақидаги маълумотлар шакллантирилган. Сўнгра ўқув фанлари ва уларни таҳсил олаётган талабалар МБсида амалга оширилган. Кейинги қадамда, ўқув фанлари ва талабалар ўртасида алоқани ташкил этишга эътибор беринг: ҳосил қилинган талабалар рўйхатини **Students** коллекциясига ҳар бир курс учун қўшиб қўямиз.

```
Students = new List<Student>() { s3, s4, s1 }
```

Юқоридагиларнинг барчаси ишга тушиши учун **Global.asax.cs** файлидаги **Application_Start** методига қуйидаги ёзувни қўшиб қўямиз:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;
using System.Data;
using System.Data.Entity;
using Study.Models;

namespace Study
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            Database.SetInitializer(new CourseDbInitializer());
```

```
            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }
}
```

**Контроллер ҳосил қиламиз**. Ўқуф фанлари ва талабалар ҳақидаги маълумотларни шакллантириш мураккаб жараён бўлмаганлиги сабабли уларни таҳлил қилмаймиз. Бизга боғланган маълумотлар моделларини шакллантириш муҳим. Шу сабабли **HomeController** контроллерида **Details** амал методини ҳосил қиламиз. Ушбу метод ёрдамида муайян талаба ҳақидаги маълумотлар чиқарилади.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Study.Models;

namespace Study.Controllers
{
    public class HomeController : Controller
    {
        private StudentsContext db = new StudentsContext();
        public ActionResult Index()
        {
            return View(db.Students.ToList());
        }

        public ActionResult Details(int id = 0)
        {
            Student student = db.Students.Find(id);
            if (student == null)
            {
                return HttpNotFound();
            }
            return View(student);
        }

        protected override void Dispose(bool disposing)
```

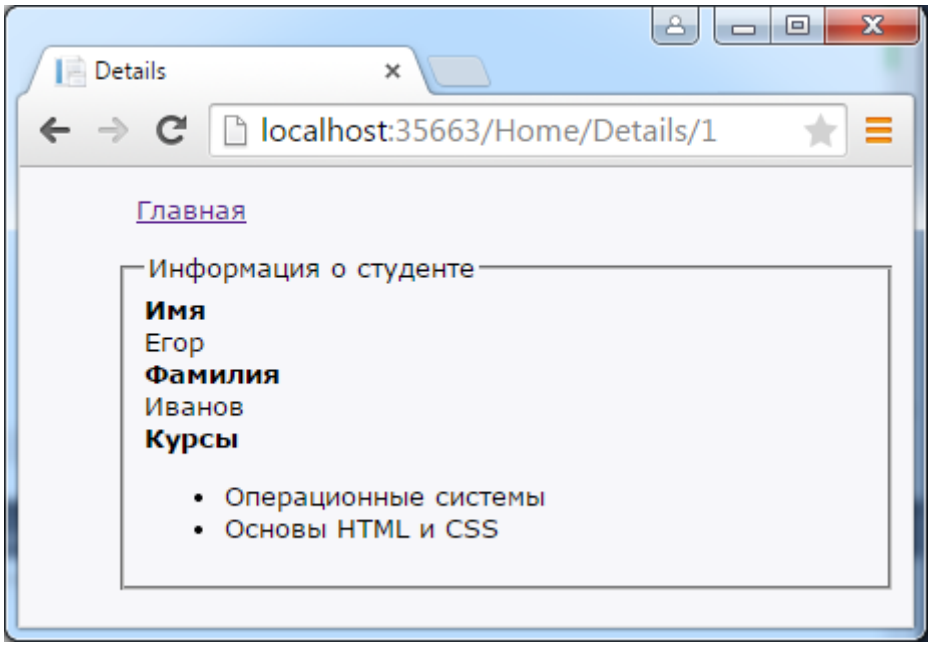
```
        {
            db.Dispose();
            base.Dispose(disposing);
        }
    }
}
```

**Details** методи орқали **Student** объекти бўйича маълумотни олиш учун ишлатилади. **Student** моделидаги **Courses** виртуал методи асосида талаба билан боғлиқ барча ўқув фанлари автоматик уланади. Энди **Details.cshtml** кўринишни ҳосил қиламиз:

```
@using Study.Models
@model Student
@{
    ViewBag.Title = "Details";
}

<fieldset>
    <legend>Информация о студенте</legend>

    <div class="display-label"><b>Имя</b></div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Name)
    </div>

    <div class="display-label"><b>Фамилия</b></div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Surname)
    </div>

    <div class="display-label"><b>Курсы</b></div>
    <ul>
        @foreach (Course c in Model.Courses)
        {
            <li>@c.Name</li>
        }
    </ul>
</fieldset>
```

Моделда боғланган маълумотлар автоматик олинганлиги сабабли, биз уларга **Model.Courses** орқали мурожаат қилишимиз мумкин. **МБ**сида бошланғич маълумотлар шакллантирилганлиги сабабли, 1 кодли талаба

ҳақидаги маълумотга эга бўлиш учун **Home/Details/1** сўрови амалга оширилади.




Шунингдек, **МБ**сига қараб, учта жадвал мавжудлигини кўришимиз мумкин. **Student** ва **Course** жадвалларини ўзаро боғловчи **CourseStudent** жадвали қуйидаги қийматларни ўзида сақлайди:

dbo.CourseStudent [Data] | Details.cshtml

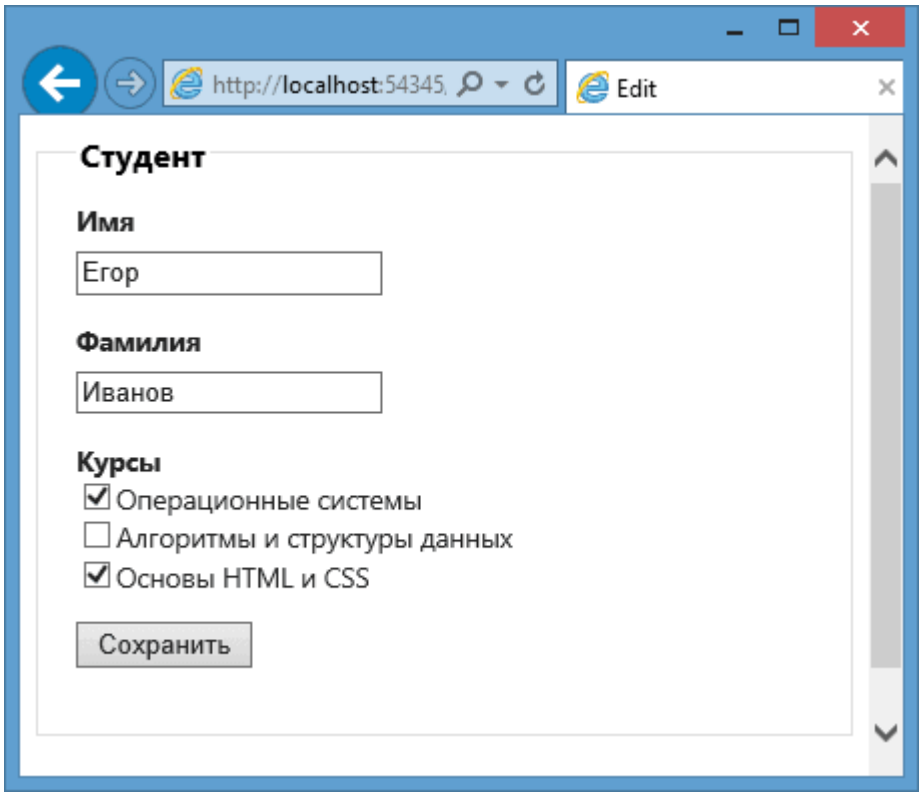
Max Rows: 1000

| CourseId | StudentId |
|---|---|
| 1 | 1 |
| 3 | 1 |
| 1 | 2 |
| 2 | 2 |
| 1 | 3 |
| 3 | 3 |
| 2 | 4 |
| 3 | 4 |
| NULL | NULL |

Яъни барча маълумотлар биз бошланғич инициализация қилган маълумотлардан иборат.

## m:n алоқадан иборат моделлар билан ишлаш

**m:n** алоқага асосланган оддий боғлиқли моделни шакллантириш катта қийинчилик туғдирмайди. Аммо ушбу турдаги моделларни яратиш ва таҳририлашни шакллантириш усули ҳозиргача келтирилмаган. Аввалги мавзуда келтирилган талаба ва ўқув фанларининг функционаллигини ошириб, талабани таҳрирлашни кўриб чиқамиз. Натижада талаба ҳақидаги маълумотларни таҳририлаш қуйидагича амалга оширилади:




Бунинг учун **HomeController** контроллерида **Edit** амал методини қўшиб қўямиз:

```
public ActionResult Edit(int id = 0)
        {
            Student student = db.Students.Find(id);
            if (student == null)
            {
                return HttpNotFound();
            }
            ViewBag.Courses = db.Courses.ToList();
            return View(student);
        }
```

```
[HttpPost]
public ActionResult Edit(Student student, int[] selectedCourses)
        {
            Student newStudent = db.Students.Find(student.Id);
            newStudent.Name = student.Name;
            newStudent.Surname = student.Surname;

            newStudent.Courses.Clear();
            if (selectedCourses != null)
            {
                //получаем выбранные курсы
                foreach (var c in db.Courses.Where(co =>
selectedCourses.Contains(co.Id)))
                {
                    newStudent.Courses.Add(c);
                }
            }

            db.Entry(newStudent).State = EntityState.Modified;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
```

**Edit** амал методи иккита **GET**-сўрови ва **POST**-сўровига мос методлар орқали шакллантирилган. **GET**-сўровини қайта ишловчи методи орқали муайян **student** моделини ва барча ўқув курсларини

```
ViewBag.Courses = db.Courses.ToList();
```

орқали таҳрирланувчи моделга узатилади.

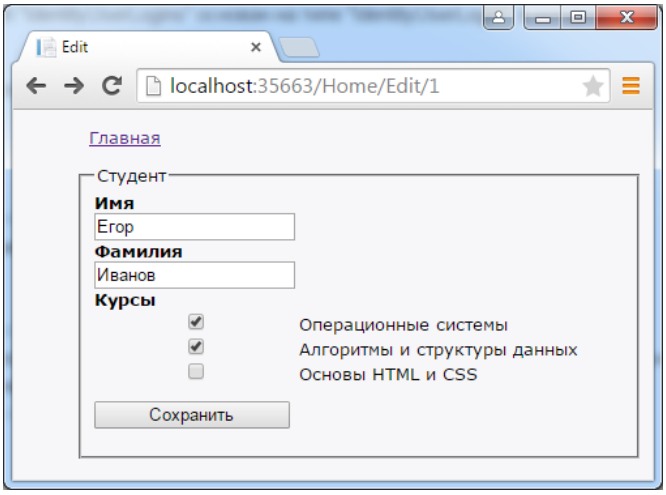
**POST** методи қабул қилинган маълумотларни (**Student** модели ва танланган ўқув фанлари **Id** лари рўйхати) олади. Ушбу массивда танланган барча ўқув курслари ҳақидаги маълумотлар сақланади. Методда эса валидауия амалга оширилгач, модел хусусиятига янги қийматларни ўрнатамиз.

Сўнгра, талабанинг **Courses** коллекциясига барча танланган курсларини ўрнатиш лозим. Бунинг учун **МБ**сидаги барча ўқув фанларини танланган ўқув фанлари билан мослигини аниқлаймиз. Агар ушбу ўқув фани рўйхатда мавжуд бўлмаса, уни қўшиб қўямиз.

Танланмаган ўқув фанларини эса рўйхатдан (агар улар мавжуд бўлса) ўчирамиз. Натижада бизнинг **МБ**да талабанинг таҳрирланган ўқув фанлари ҳавидаги маълумотлар акслантирилади. Охирги қадамда **Edit.cshtml** кўринишни қуйидагича ҳосил қиламиз:

```
@using Study.Models
@model Student
@{
    ViewBag.Title = "Edit";
}

@using (Html.BeginForm())
{
    <fieldset>
        <legend>Студент</legend>

        @Html.HiddenFor(model => model.Id)

        <div class="editor-label"><b>Имя</b></div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
        </div>

        <div class="editor-label"><b>Фамилия</b></div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Surname)
        </div>
        <div class="editor-label"><b>Курсы</b></div>
        @{
    List<Course> courses = ViewBag.Courses;
    foreach (Course c in courses)
    {
        <input type="checkbox" name="selectedCourses" value="@c.Id"
               @(Model.Courses.Contains(c) ? "checked=\"checked\"" : "")
/>@c.Name <br />
    }
        }

        <p>
            <input type="submit" value="Сохранить" />
        </p>
    </fieldset>
}
```

**http://localhost:35663/Home/Edit/1**



## Массивлар ва мураккаб тузилмали маълумотларни контроллерга узатиш

Аввалги мавзуларда биз кўринишдаги баъзи объектларни амал методига параметр қилиб узатишни кўриб чиққан эдик. Баъзи ҳолларда амал методига битта **int** типи ёки бирор модел ўрнига бир қанча объектлар узатилиши мумкин. Бир нечта мисолларни кўриб чиқамиз.

### Коллекцияларни узатиш

Кўринишда қуйидаги формани аниқлаймиз:

```
@using (Html.BeginForm())
{
    @Html.TextBox("names")
    @Html.TextBox("names")
    @Html.TextBox("names")
    @Html.TextBox("names")
    <input type="submit" />
}
```

Ушбу хелперлар орқали қуйидаги **html-код** генерация қилинади:

```
<form action="/Home/Array" method="post">
    <input id="names" name="names" type="text" value="" />
    <input id="names" name="names" type="text" value="" />
    <input id="names" name="names" type="text" value="" />
    <input id="names" name="names" type="text" value="" />
    <input type="submit" />
</form>
```

Форма узатилаётганда тўртта элементдан иборат **names** коллекцияси ҳосил қилинади. Контроллердаги амал методида ушбу элементларни қабул қилишимиз мумкин:

```
[HttpPost]
public string Array(List<string> names)
        {
            string fin = "";
            for (int i = 0; i < names.Count; i++)
            {
                fin += names[i] + ";  ";
            }
            return fin;
        }
```

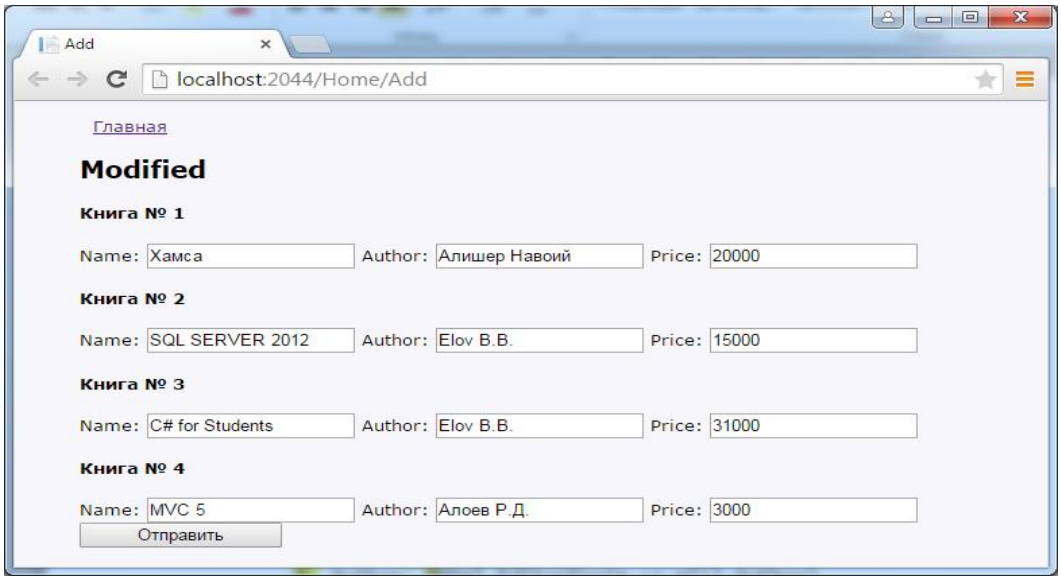
**Объектлар коллекциясини моделга узатиш**

Биз кўринишга бирор моделдан иборат объектлар коллекциясини узатишимиз мумкин.

```
[HttpGet]
        public ActionResult Add()
        {
            return View(db.Books.ToList());
        }
```

Кўринишда барча объектларни таҳрирлаш учун қуйидаги конструкциядан фойдаланамиз: **http://localhost:2044/Home/Add**

```
@model List<BookStore.Models.Book>
@{
    ViewBag.Title = "Add";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Modified</h2>

@using (Html.BeginForm())
{
    for (int i = 0; i < Model.Count; i++)
    {
        <h4>Книга № @(i + 1)</h4>

        @: Name: @Html.EditorFor(m => m[i].Name)
        @: Author: @Html.EditorFor(m => m[i].Author)
        @: Price: @Html.EditorFor(m => m[i].Price)
    }
    <input type="submit" />
}
```



Биз ҳар бир объект учун модел хусусиятларидан иборат майдонларни генерация қилдик. Барча маълумотларга ўзгартиришлар амалга оширилгач, тугма босилади. Натижада барча маълумотлардан иборат массив серверга узатилади. Ушбу маълумотларни қуйидаги тарзда қабул қилиб олиш мумкин:

```
[HttpPost]
        public string Add(List<Book> books)
        {
            //.......................
        }
```

**Битта моделдаги турли шаклдаги объектларни узатиш**

Аввалги мисолда биз **Book** модели объектлари коллекциясини кўринишга ва амал методига узатдик. Аммо баъзи ҳолларда бир неча хил турдаги объектларни узатишга тўғри келади. Масалан, контроллердаги амал методи қуйидагича амалга ошириш мумкин:

```
[HttpPost]
        public string Add(Book book, Book myBook)
        {
            //.......................
        }
```

Ушбу мисолда иккита алоҳида **Book** объекти фойдаланилган. Биз контроллерга ушбу иккита турли объектларни қандай узатишимиз мумкин? Битта объектни кўриниш модели сифатида, иккинчисини кўринишда ҳосил қиламиз:

```
@using BookStore.Models
@model Book
@{
    ViewBag.Title = "Array";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@{
    Book myBook = new Book() { Name = "Мартин Иден", Author = "Джек
Лондон", Price = 190 };
}
<h2>Книги</h2>

@using (Html.BeginForm())
{
    @Html.EditorFor(m => myBook)
```

```
    @Html.EditorForModel()
    <input type="submit" />
}
```

Битта моделни биз контроллердан кўринишга узатамиз:

```
[HttpGet]
public ActionResult Array()
        {
            Book firstBook = db.Books.ToList<Book>().FirstOrDefault();
            return View(firstBook);
        }
```

Иккинчи модел **myBook**ни биз кўринишда ҳосил қиламиз. Моделнинг барча майдонлари **@Html.EditorFor(m=>myBook)** хелпери орқали генерация қилинади.

Модел номига эътибор беринг – **myBook**. Ушбу объект методга параметр сифатида **myBook** узатилади. Ушбу усул орқали серверга битта моделга мос иккита объект узатилади.

**http://localhost:2044/Home/Array**

### Мураккаб объектларни узатиш

Бизда қуйидаги модел мавжуд бўлсин.

```
public class Book
    {
        public int Id { get; set; }
        public string Name { get; set; }

    }

public class Author
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public ICollection<Book> Books { get; set; }
    }
```

Муаллиф моделида китобларга узатмани ўзида сақлайди. Контроллерда биз ушбу моделни оламиз:

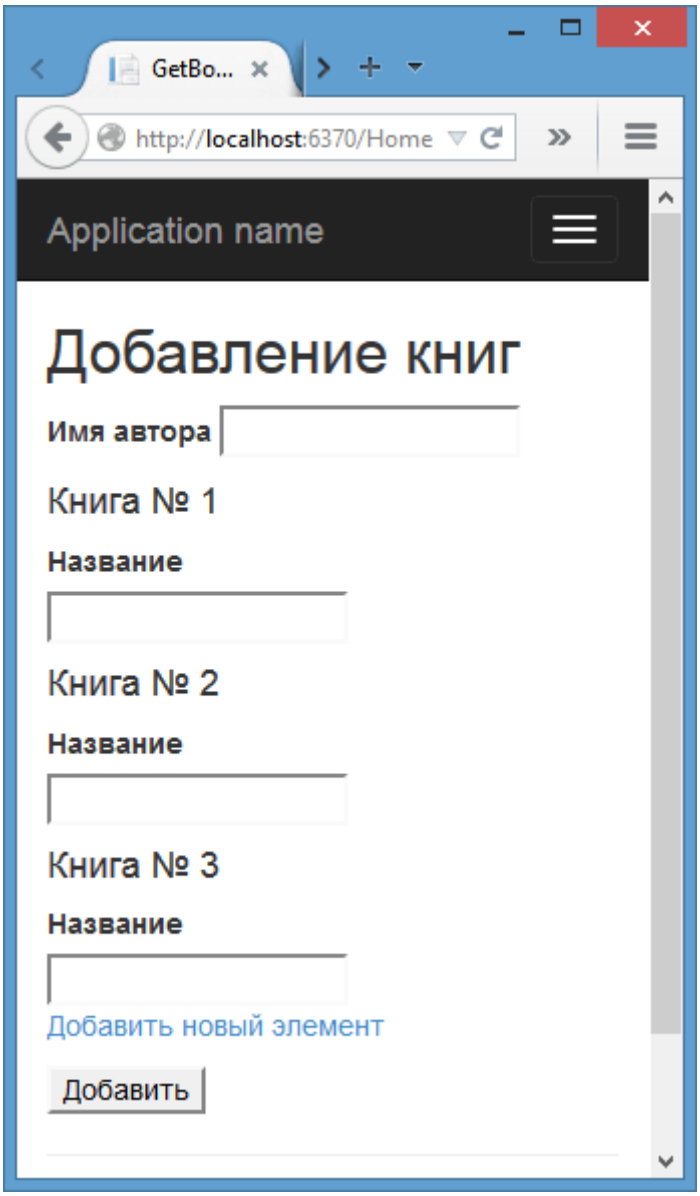
```
[HttpPost]
public ActionResult GetAuthor(Author author)
        {
            return View();
```

```
        }
```

Ушбу ҳолда кўриниш қуйидаги шаклга эга:

```
@model ArrayPostApp.Models.Author

<h2>Добавление книг</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="authorBlock">
        <label>Имя автора</label>
        <input type="text" name="name" />
    </div>
    <div id="booksBlock">
        <div class="bookItem">
            <h4>Книга № 1</h4>
            <div>
                <label>Название</label>
                <div>
                    <input type="text" name="Books[0].name" />
                </div>
            </div>
        </div>

    </div>
    <p><a class="addLink">Добавить новый элемент</a></p>
    <p><input type="submit" value="Добавить" /></p>
}

@section Scripts {
    <script>
        $(function () {
            var i = 0;
            $('.addLink').click(function () {
                i++;
                var html2Add = "<div class='bookItem'>" +
             "<h4>Книга № " + (i + 1) + "</h4>" +
             "<div><label>Название</label><div>" +
                    "<input type='text' name='Books[" + i + "].name' />"
+
                 "</div></div></div>";
                $('#booksBlock').append(html2Add);
            })
        })
    </script>
}
```

**Author** модели китоблар тўпламини ўзида **Books** хусусияти орқали сақланагани учун **name** атрибутида мос матнли элементлар **name="Books[0].name"** каби шаклга эга. **javascript** скрипти ёрдамида янги элементларни динамик тарзда қўшиб қўйишимиз мумкин.



## Маълумотлар базасини миграция қилиш

Баъзи ҳолларда модел тузилмасида ўзгаришлар амалга оширилиш мумкин. Масалан, моделга янги хусусиятлар қўшиш лозим бўлсин. Аммо бизда **МБ** мавжуд бўлиб, унда муайян маълумотлар мавжуд. Мавжуд маълумотларни йўқотмасдан **МБ**сини янгилаш учун **ASP.NET MVC** миграция деб номланувчи механизмни таклиф қилади. Бизда оддий **User** модели мавжуд бўлсин:

```
public class User
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
```

Ушбу моделга мос **МБ** билан ишловчи маълумотлар контексти:

```
public class UserContext : DbContext
    {
        public UserContext() :
            base("DefaultConnection")
        { }
        public DbSet<User> Users { get; set; }
    }
```

Бизда ушбу моделга мос инфратузилма мавжуд бўлсин: контроллерлар ва мавжуд **МБ**даги ушбу моделга мос объектлар.

1) **Web.config** файлида қуйидагиларни қўшиб қўямиз:

```
<connectionStrings>
    <add name="UserContext" connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename='|DataDirectory|\Users.mdf';Integrated
Security=True"
 providerName="System.Data.SqlClient" />

    <add name="DefaultConnection" connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename='|DataDirectory|\Users.mdf';Integrated
Security=True"
 providerName="System.Data.SqlClient" />
  </connectionStrings>
```

2) **Context\Site.css** файли:

```
body {
    font-size: 13px;
    font-family: Verdana, Arial, Helvetica, Sans-Serif;
    background-color: #f7f7fa;
    padding-left:40px;
}

nav{
```

```
        display: block;
    }

    .menu {
        padding-left:10px;
    }
    .menu ul {
        list-style:none;
    }
    .menu li {
        display:inline;
    }
    .menu a:hover {
        color:red;
    }
    table {
       vertical-align:middle;
       text-align:left;
     }
    .header {
        font-weight:bold;
    }
     td {
        padding-right:10px;
     }
    input {
        width: 150px;
    }
```

**3) Views\Shares\ _Layout.cshtml** файли:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet"
type="text/css" />
</head>

<body>
    <nav>
        <ul class="menu">
            <li>@Html.ActionLink("Главная", "Index", "Home")</li>
        </ul>
    </nav>
    @RenderBody()
</body>
</html>
```

### 4) HomeController контроллери

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

namespace MUser.Controllers
{
    public class HomeController : Controller
    {
        private UserContext db = new UserContext();
        public ActionResult Index()
        {
            return View(db.Users);
        }

        public ActionResult Details(int id = 0)
        {
            User user = db.Users.Find(id);
            if (user == null)
            {
                return HttpNotFound();
            }
            return View(user);
        }

    }
}
```

### 5) Index.cshtml файли

```
@model IEnumerable<MUser.Models.User>
@{
    ViewBag.Title = "All users";
}

<table>
    <tr>
        <th>Имя пользователя</th>
        <th></th>
    </tr>

    @foreach (MUser.Models.User item in Model)
    {
        <tr>
            <td>
                @item.Name
            </td>
            <td>
              <p><a href="/Home/Details/@item.Id">Подробно</a></p>
            </td>
        </tr>
```

```
    }
</table>
```

**6) Details.cshtml** файли:

```
@using MUser.Models
@model User
@{
    ViewBag.Title = "Details";
}

<fieldset>
    <legend>Информация о пользователе</legend>

    <div class="display-label"><b>Имя</b></div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Name)
    </div>
</fieldset>
```

Аммо дастуримиздаги моделга ўзгартириш киритишга тўғри келди. Масалан, **User** моделига янги майдон қўшиш лозим бўлсин:

```
public class User
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
```

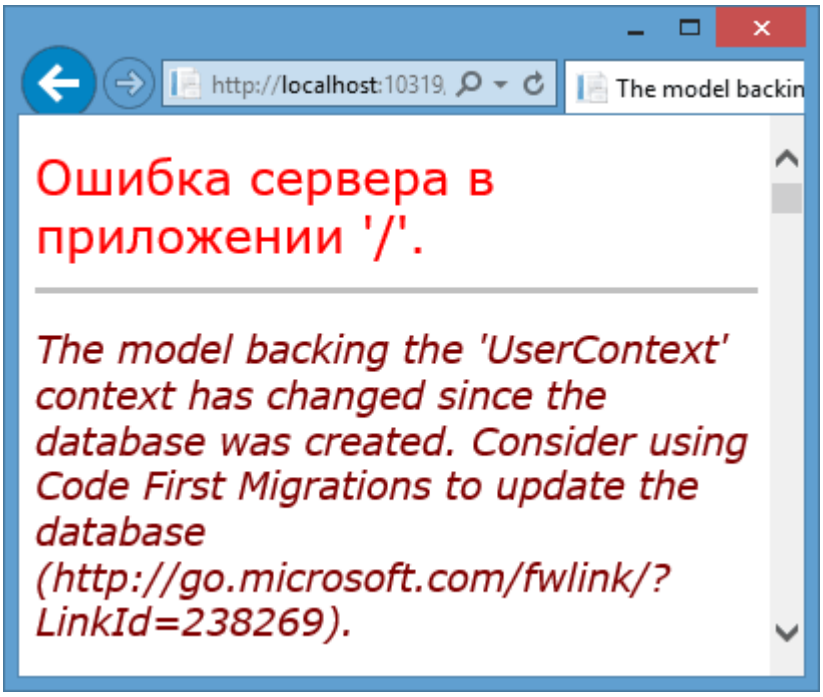
Бундан ташқари, яна битта моделни қўшиш лозим бўлсин:

```
public class Company
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
```

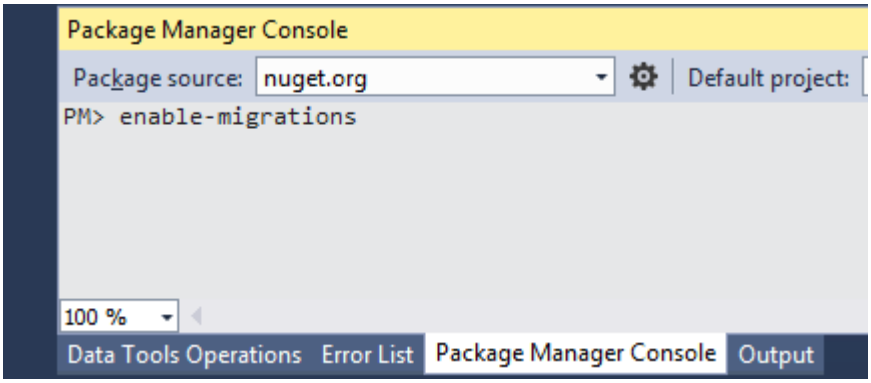
Шунингдек, маълумотлар контекстида қуйидаги ўзгартиришларни амалга ошириш лозим:

```
public class UserContext : DbContext
    {
        public UserContext() :
            base("DefaultConnection")
        { }
        public DbSet<User> Users { get; set; }
        public DbSet<Company> Companies { get; set; }
    }
```

Биз кўринишда **User** модели учун қўшимча **Age** хусусиятини қўшиб қўйиш лозим. Янги модел учун контроллер ва кўринишни ҳосил қилишимиз лозим. Янги объектни **МБ**сида сақлаш вақтида қуйидаги хатолик юзага келади:




Маълумотлар контексти ўзгарган ва биз маълумотлар миграциясини аввалги схемадан янги схемага ўтказишимиз лозим. Биринчи навбатда **Visual Studio** да **Package Manager Console** ойнасини топамиз ва унга **enable-migrations** буйруғини берамиз.




Ушбу буйруқ **Visual Studio** да амалга оширилгач, лойиҳада **Migrations** папакси ҳосил бўлади. Ушбу папкада **Configuration.cs** файлини топиш мумкин. Ушбу файл **Configuration** классидаги эълонни ўзида сақлайди:

```
namespace MUser.Migrations
{
```

```
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Migrations;
    using System.Linq;

    internal sealed class Configuration :
DbMigrationsConfiguration<MUser.Models.UserContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = false;
        }

        protected override void Seed(MUser.Models.UserContext context)
        {
            //  This method will be called after migrating to the latest
version.

            //  You can use the DbSet<T>.AddOrUpdate() helper extension
method
            //  to avoid creating duplicate seed data. E.g.
            //
            //    context.People.AddOrUpdate(
            //      p => p.FullName,
            //      new Person { FullName = "Andrew Peters" },
            //      new Person { FullName = "Brice Lambson" },
            //      new Person { FullName = "Rowan Miller" }
            //    );
            //
        }
    }
}
```

Ушбу файлдаги **Seed** методида **МБ**сини бошланғич қийматлар билан инициализация қилиш мумкин. Энди биз миграцияни амалга оширишимиз лозим. **Package Manager Console** консолида қуйидаги буйруқни амалга оширишимиз лозим:

**PM> Add-Migration "MigrateDB"**

Шундан сўнг **PM> Add-Migration "MigrateDB"** автоматик тарзда миграция классини ҳосил қилади:

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

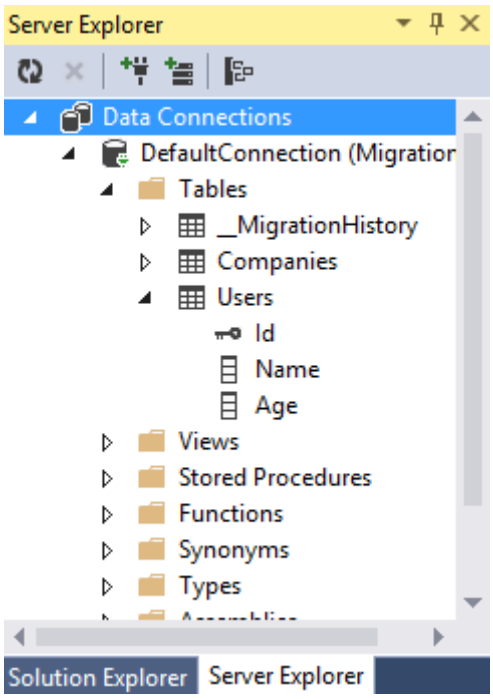
```
    public partial class MigrateDB : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.Companies",
                c => new
                    {
                        Id = c.Int(nullable: false, identity: true),
                        Name = c.String(),
                    })
                .PrimaryKey(t => t.Id);

            CreateTable(
                "dbo.Users",
                c => new
                    {
                        Id = c.Int(nullable: false, identity: true),
                        Name = c.String(),
                        Age = c.Int(nullable: false),
                    })
                .PrimaryKey(t => t.Id);

        }

        public override void Down()
        {
            DropTable("dbo.Users");
            DropTable("dbo.Companies");
        }
    }
}
```

Ушбу файлдаги **Up** методидаги **CreateTable** методи ёрдамида **dbo.Companies** жадвали ва унинг созланмаси ҳосил қилинади. Яъни, устунлар ва калитлар яратилади. Шунингдек, мавжуд **Users** жадвалида **Age** майдони қўшиб қўйилади. **Down** методи ёрдамида мавжуд устунлар ва жавдваллар ўчирилади. Ушбу методлар **SQL** тилидаги **ALTER** ифодасига мос келади. **ALTER** орқали МБ ва жадвалнинг тузилмасига ўзгартиришлар амалга оширилади.

Миграцияни амалга ошириш учун ушбу классни консолдаги қуйидаги буйруқ орқали қўллаймиз:

**PM> Update-Database**

Шундан сўнг лойиҳамиздаги маълумотлар базаси тузилмасини ва ундаги ўзгаришларни кўришимиз мумкин:




Миграция амалга оширилгач, биз янгиланган модел ва маълумотлар контексти билан иш кўришимиз мумкин.

## Пангинация

Маълумотлар билан ишлашда энг кенг тарқалган амаллардан бири пангинация муаммоси ёки маълумотларни бир нечта блок (саҳифалар)ларга ажратилган ҳолда саҳифаларда навигацияни амалга ошириш ҳисобланади.

Пангинацияни ҳосил қилиш учун мавжуд плагинлардан фойдаланишимиз мумкин. Ушбу ҳолда биз кўпсатрли киритишни шакллантирамиз. Аввало зарур моделни шакллантириб оламиз:

```
public class Phone
    {
        public int Id { get; set; }
        public string Model { get; set; }
        public string Producer { get; set; }
    }
```

Шунингдек, пагинация механизмини тавсифловчи моделни ва кўринишда маълумотларни киритиш моделини аниқлаймиз:

```
public class PageInfo
    {
        public int PageNumber { get; set; } // номер текущей страницы
        public int PageSize { get; set; } // кол-во объектов на странице
        public int TotalItems { get; set; } // всего объектов
        public int TotalPages  // всего страниц
        {
            get { return (int)Math.Ceiling((decimal)TotalItems /
PageSize); }
        }
    }

public class IndexViewModel
    {
        public IEnumerable<Phone> Phones { get; set; }
        public PageInfo PageInfo { get; set; }
    }
```

Телефон моделидаги маълумотлар билан биргаликда пагинация модели кўринишда ишлатилиши сабабли, улар **IndexViewModel** класси орқали инкапсулция қилинади.

Энди ушбу амалларга мос контроллер ва маълумотларни киритиш методини аниқлаймиз:

```
namespace ManualPaginApp.Controllers
{
    public class HomeController : Controller
    {
        List<Phone> phones;
        public HomeController()
        {
            phones = new List<Phone>();
            phones.Add(new Phone { Id = 1, Model = "Samsung Galaxy III",
Producer = "Samsung" });
            phones.Add(new Phone { Id = 2, Model = "Samsung Ace II",
Producer = "Samsung" });
            phones.Add(new Phone { Id = 3, Model = "HTC Hero", Producer =
"HTC" });
            phones.Add(new Phone { Id = 4, Model = "HTC One S", Producer
= "HTC" });
            phones.Add(new Phone { Id = 5, Model = "HTC One X", Producer
= "HTC" });
            phones.Add(new Phone { Id = 6, Model = "LG Optimus 3D",
Producer = "LG" });
            phones.Add(new Phone { Id = 7, Model = "Nokia N9", Producer =
"Nokia" });
```

```
            phones.Add(new Phone { Id = 8, Model = "Samsung Galaxy
Nexus", Producer = "Samsung" });
            phones.Add(new Phone { Id = 9, Model = "Sony Xperia X10",
Producer = "SONY" });
            phones.Add(new Phone { Id = 10, Model = "Samsung Galaxy II",
Producer = "Samsung" });
        }


        public ActionResult Index(int page = 1)
        {
            int pageSize = 3; // количество объектов на страницу
            IEnumerable<Phone> phonesPerPages = phones.Skip((page - 1) *
pageSize).Take(pageSize);
            PageInfo pageInfo = new PageInfo { PageNumber = page,
PageSize = pageSize, TotalItems = phones.Count };
            IndexViewModel ivm = new IndexViewModel { PageInfo =
pageInfo, Phones = phonesPerPages };
            return View(ivm);
        }

    }
}
```

Соддалик учун контроллер конструкторида оддий рўйхатни шакллантирамиз. Аммо ушбу ушбу маълумотларни **МБ**сидан олиш ҳам мумкин.

**Index** методи параметр сифатида саҳифа тартиб рақамини қабул қилади. Бошланғич қиймат сифатида 1 берилган. Ушбу методда **Skip** ва **Take** методлари комбинацияси орқали рўйхатдаги маълумотлардан зарурий қисми олинади. **Skip()** методи муайян сондаги маълумотларни ўтказади. **Take()** методи муайян сондаги элементларни танлаб танлаб олади.

Кўринишда пагинация механизмини ҳосил қилиш учун хелперлардан фойдаланиш мақсадга мувофиқ. Ўзимизнинг хелперимизни шакллантирамиз. Бунинг учун **Helpers** папкасига қуйидаги классни шакллантирамиз:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;
using System.Web.Mvc;
using ManualPaginApp.Models;

namespace ManualPaginApp.Helpers
{
```

```
    public static class PagingHelpers
    {
        public static MvcHtmlString PageLinks(this HtmlHelper html,
        PageInfo pageInfo, Func<int, string> pageUrl)
        {
            StringBuilder result = new StringBuilder();
            for (int i = 1; i <= pageInfo.TotalPages; i++)
            {
                TagBuilder tag = new TagBuilder("a");
                tag.MergeAttribute("href", pageUrl(i));
                tag.InnerHtml = i.ToString();
                // если текущая страница, то выделяем ее,
                // например, добавляя класс
                if (i == pageInfo.PageNumber)
                {
                    tag.AddCssClass("selected");
                    tag.AddCssClass("btn-primary");
                }
                tag.AddCssClass("btn btn-default");
                result.Append(tag.ToString());
            }
            return MvcHtmlString.Create(result.ToString());
        }
    }
}
```

Ушбу хелпер узатмалар блокини шакллантиради ва унда визуализация классларини қўшиб қўяди. Класслар турлича бўлиши мумкин. Аммо ушбу ҳолда стандарт **bootstrap** классиларидан фойдаланилган.

Энди пагинацияни ишлатувчи кўринишни яратамиз:

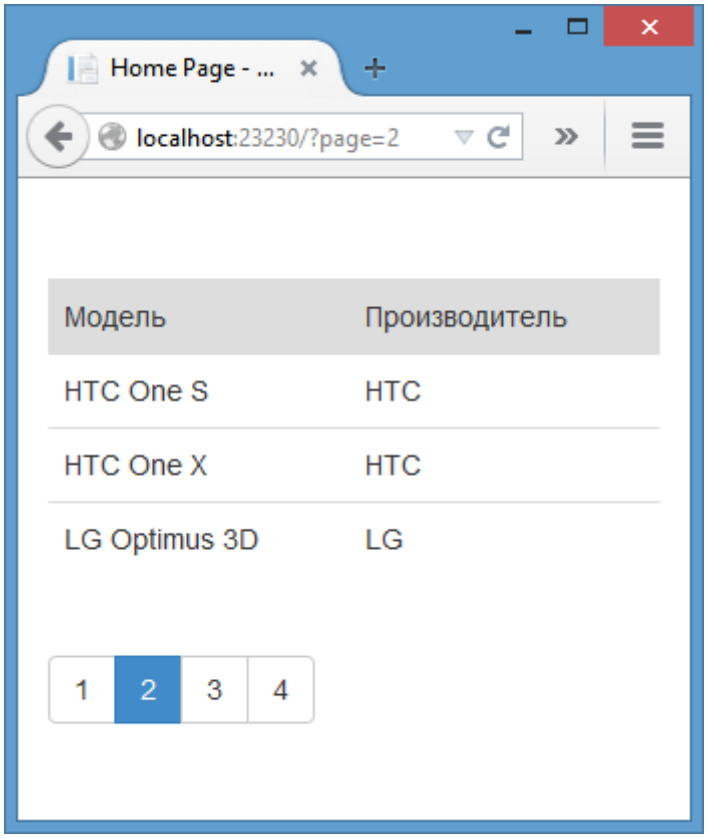
```
@model ManualPaginApp.Models.IndexViewModel
@using ManualPaginApp.Helpers
@{
    ViewBag.Title = "Home Page";
}
<table class="table">
    <tr>
        <td><b>Модель</b></td>
        <td><b>Производитель</b></td>
    </tr>
    @foreach (var item in Model.Phones)
    {
        <tr>
            <td>@item.Model</td>
            <td>@item.Producer</td>
        </tr>
    }
```

```
</table>
<br />
<div class="btn-group">
    @Html.PageLinks(Model.PageInfo, x => Url.Action("Index", new { page = x }))
</div>
```

Пагинация хелпери **Helpers** папкасида жойлашганлиги сабабли, кўриниш бошида ушбу номлар фазосини кўрсатиш лозим:

**using ManualPaginApp.Helpers**

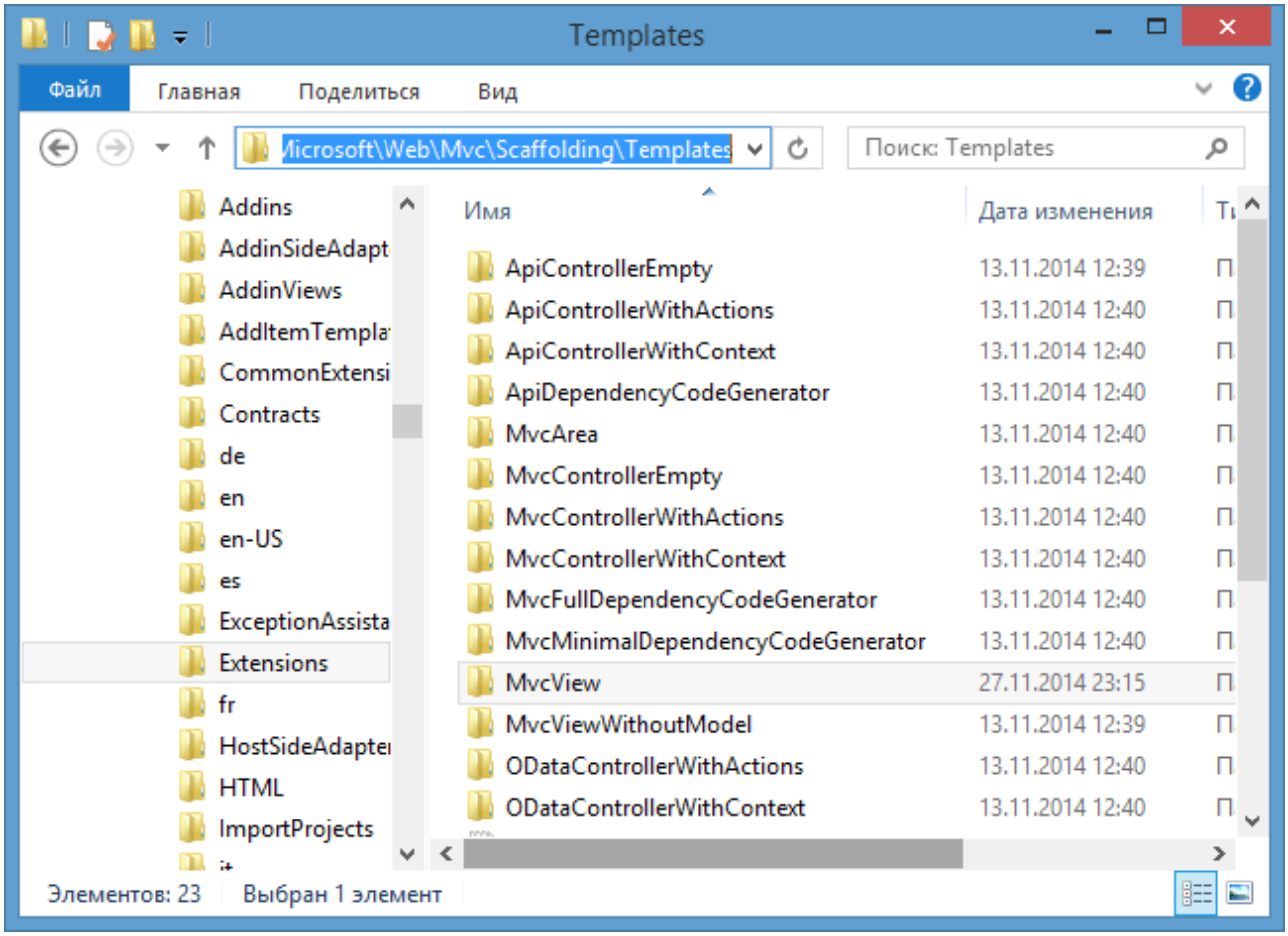
Натижада **Html.PageLinks** пагинация хелпери барча мантиқни ўзи амалга оширади. Лойиҳа ишга туширилгач, қуйидаги натижа ҳосил қилинади:



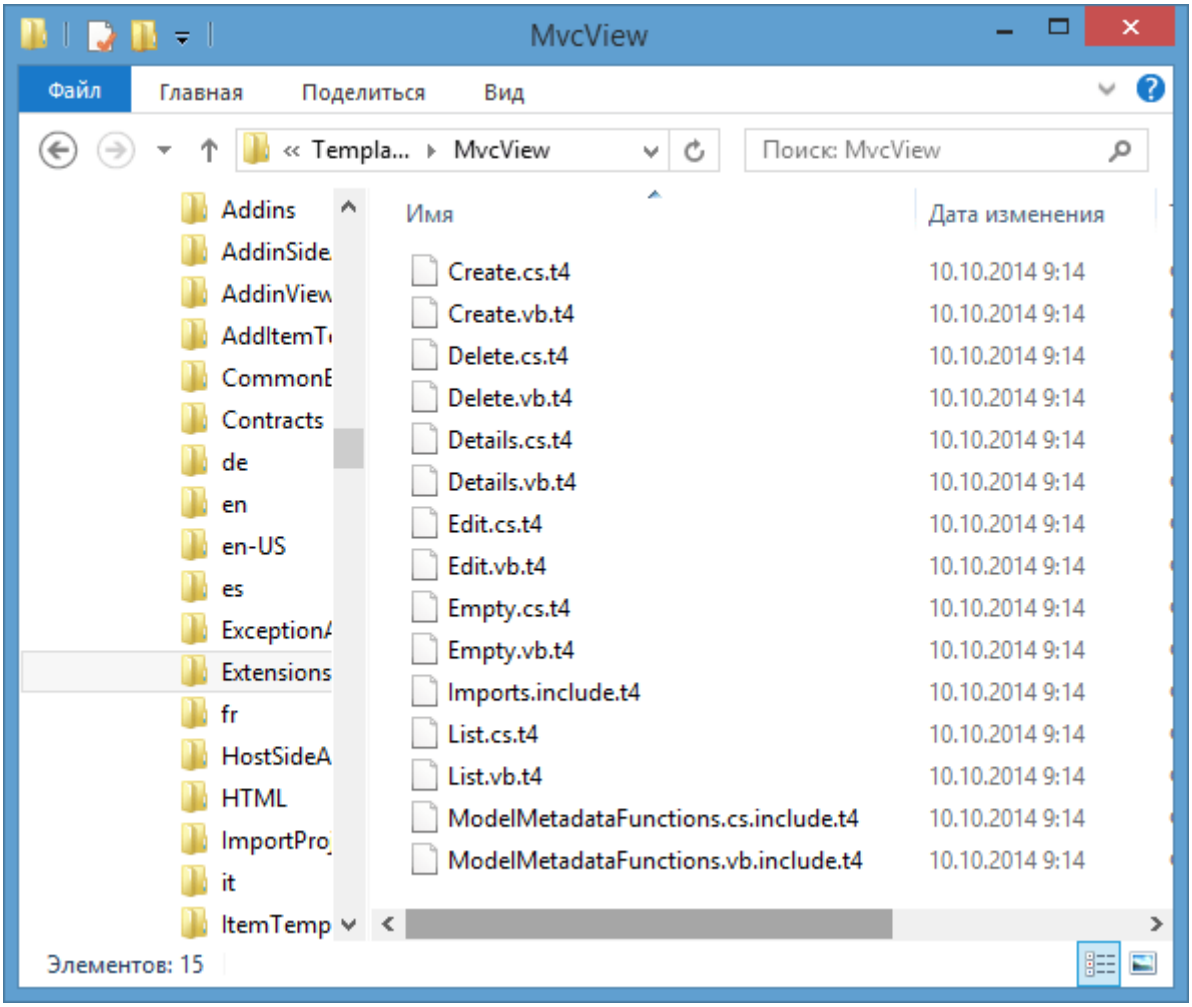
## Шакллантириш шаблонларини қайта аниқлаш

Аввалги мисолларда шакллантириш шаблонларини кўриб чиққан эдик. Ушбу шаблонлар ёрдамида контроллер ва кўринишларни ҳосил қилишни автоматлаштириш мумкин. Аммо баъзи ҳолларда уларнинг функционаллиги етарли бўлмайди ва баъзи ҳодисаларни қайта аниқлашга тўғри келади.

Аввало мавжуд шаблонларни кўриб чиқамиз. **Visual Studio 2013** да барча шаблонлар ***C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\Extensions\Microsoft\Web\Mvc\Scaffolding\Templates*** папкасида сақланади. Ушбу папкада қуйидаги қисм каталоглар мавжуд:




Барча қисм каталоглар шаблонларни ифодалайди. Улар асосида контроллерлар ва кўринишлар генерация қилинади. Масалан, **MvcView** каталогида генерация қилиниши лозим бўлган кўринишлар сақланади:




Шаблонлар **.t4** кенгайтмасига эга бўлиб, унинг ичида бир қатор директивалар мавжуд. Ушбу директивалар орқали код генерация қилинади. Масалан, **Empty.cs.t4** шаблони қуйидаги кодга эга:

```
<#@ template language="C#" hostspecific="True" #>
    <#@ output extension=".cshtml" #>
        <#@ include file="Imports.include.t4" #>
            @model <#= ViewDataTypeName #><# =ViewDataTypeName #>
                <# />/ The following chained if-statement outputs the
file header code and markup for a partial view, a view using a layout
page, or a regular view.
                if(IsPartialView) {
                #>

                <# } else if(islayoutpageselected) {
                   #>

                    @{
    ViewBag.Title = "<#= ViewName#>";
    <# =ViewName#>
```

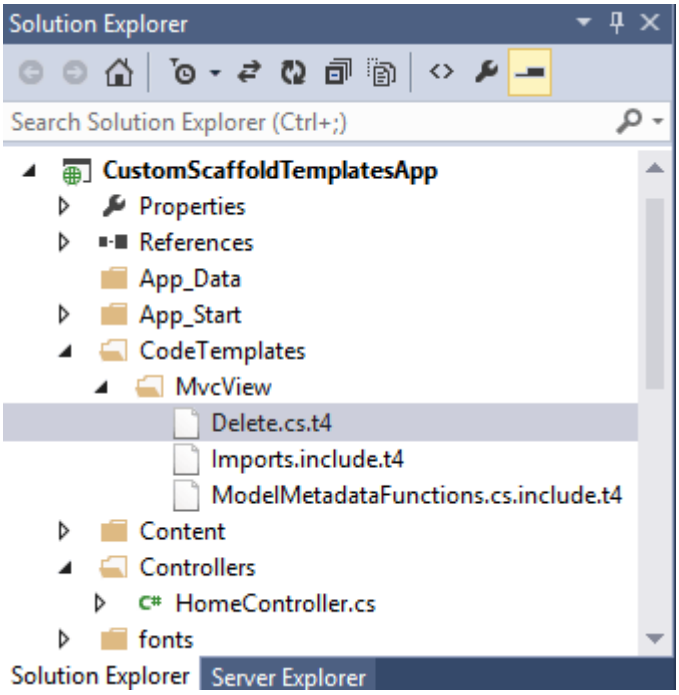
```
        <# if (!string.isnullorempty(layoutpagefile)) {
           #>
            Layout = "<# =LayoutPageFile#>
                ";
                <# }
                   #>
                    }

                    <h2><# =ViewName#></h2>

                    <# } else {
                       #>

                        @{
    Layout = null;
                        }

                        <!DOCTYPE HTML>

                        <html>
                        <head>
                            <meta name="viewport" content="width=device-
width" />
                            <title><# =ViewName #></title>
                        </head>
                        <body>
                            <# pushindent(" ");
                               }
                               #>
                                <# if(!ispartialview &&
!islayoutpageselected) {
                                   #>
                                    <div>
                                    </div>
                                    <# }
                                       #>
                                        <# />/ The following code closes
the tag used in the case of a view using a layout page and the body and
html tags in the case of a regular view page
                                        #>
                                        <# if(!ispartialview &&
!islayoutpageselected) {
                                           clearindent();
                                           #>
     </body>
</html>
<# }
#>
```

Шаблондаги барча директивалар ва бошқарувчи блоклар **<#** ва **#>**тэглари орасида жойлашади. **<#@ output extension=".cshtml" #>** директиваси ҳосил қилинувчи файлнинг **.cshtml** шаклида амалга оширади.

Оддий матн блоки <# ва **#>** тэгларидан ташқарида жойлаштирилади. Матн блоклари қайта ишланмайди ва натижавий файлга тўлиқ ўтказилади.

Биз мавжуд шаблонда ўзгартиришларни амалга оширишимиз мумкин. Аммо ушбу ўзгартиришлар шаблондан фойдаланилган кейинги барча кўриниш ёки конторелларда ўз аксини топади. **Visual Studio** лойиҳасидаги глобал шаблонлар лойиҳанинг ***C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\Extensions\Microsoft\Web\Mvc\Scaffolding\Templates*** папкасида жойлаштирилади.

Бирор шаблонни қайта шакллантириш (ўзгартириш киритиш) учун лойиҳада янги **CodeTemplates** папкасини яратиш ва унга ўзгартириш киритилиши лозим бўлган барча шаблонларни жойлаштириш лозим. Масалан, **Delete.cs.t4** шаблонини қайта аниқлаймиз:




**Delete.cs.t4** шаблон **MvcView** папкасида жойлашганлиги сабабли, лойиҳада **CodeTemplates** папкасини яратиш ва унга шаблонни кўчириш лозим. Шунингдек, қўшимча тарзда **Delete.cs.t4**га боғлиқ бўлган файлларни ҳам ҳосил қиламиз. Улар **C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\Extensions\Microsoft\Web\Mvc\Scaffolding\Templates\Mvc View** папкасида жойлашган.

Кейинги қадамда **Delete.cs.t4** шаблонда қуйидагича ўзгартиришларни амалга оширамиз:

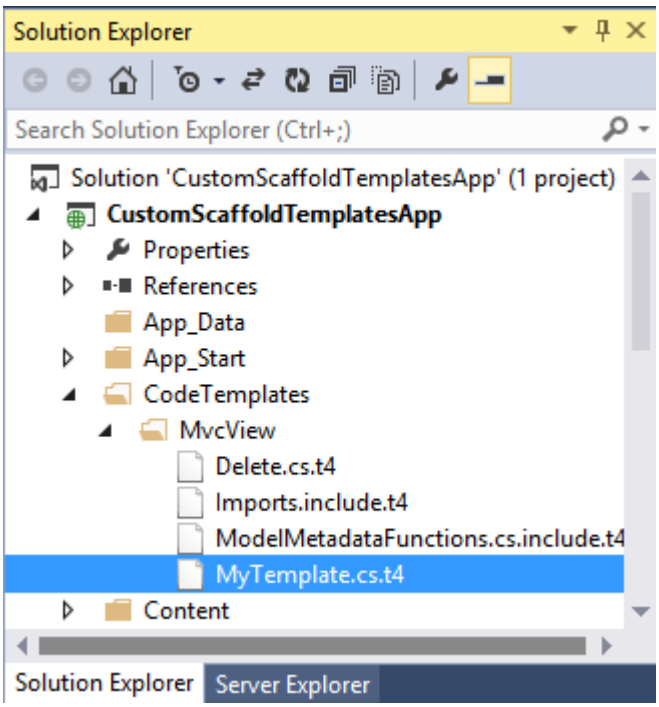
```
<#@ template language="C#" hostspecific="True" #>
<#@ output extension=".cshtml" #>
<#@ include file="Imports.include.t4" #>
<h2>Шаблон удаления</h2>
```

Ушбу шаблонда фақат сарлавҳа ҳосил қилинади. Биринчи директива **C#** тилида код ёзилиши ифодаланган. Иккинчи сатрда генерация қилинадиган файл **.cshtml** кенгайтмага эга бўлиши келтирилган. Учинчи сатрда шаблон файли келтирилган.

Энди биз **Delete** шаблони бўйича кўринишни генерация қилмоқчи бўлсак, **Visual Studio** қуйидаги файлни генерация қилади:

```
<h2>Шаблон удаления</h2>
```

Энди маънога эга бўлган шаблонни ҳосил қиламиз. Бунинг учун лойиҳага **CodeTemplates/MvcView** папкасини ҳосил қиламиз ва бирор шаблонни **MyTemplate** га ўзгартирамиз.

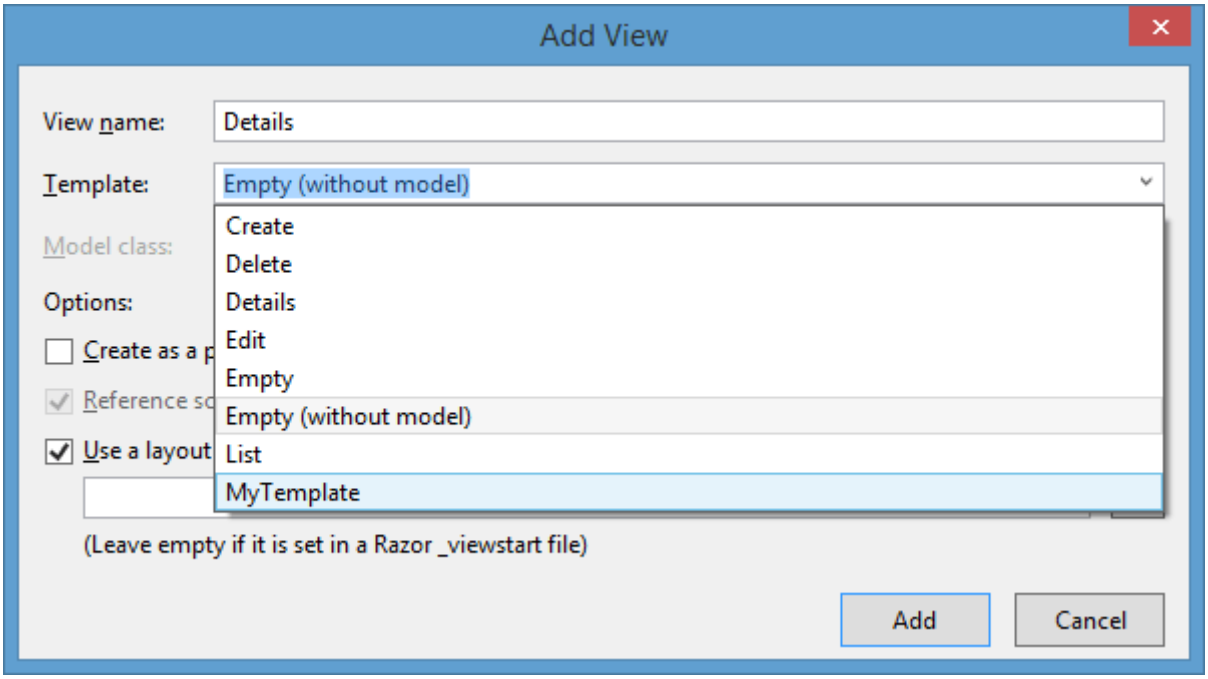



Ушбу шаблонда қуйидаги кодни шакллантирамиз:

```
<#@ template language="C#" hostspecific="True" #>
    <#@ output extension=".cshtml" #>
        <#@ include file="Imports.include.t4" #>
            @model <#= ViewDataTypeName #><# =ViewDataTypeName #>
```

```
<!DOCTYPE HTML>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>MyTemplate</title>
</head>
<body>
    <h2><# =ViewName#></h2>
    <table>
        <# foreach (propertymetadata property in
modelmetadata.properties) {
            if (property.scaffold &&
!property.isprimarykey && !property.isforeignkey) {
            #>
             <tr>
                 <td>
                     <b> @Html.DisplayNameFor(model =>
model.<#= GetValueExpression(property) #>)<#
=GetValueExpression(property) #> </b>
                 </td>

                 <td>
                     @Html.DisplayFor(model => model.<#=
GetValueExpression(property) #>)<# =GetValueExpression(property) #>
                 </td>
             </tr>
             <# }
                }
               #>
    </table>
</body>
</html>
<#@ include file="ModelMetadataFunctions.cs.include.t4" #>
```

Ушбу шаблон контроллердаги моделни қабул қилади ва ундаги барча усусиятлар қийматларини жадвал шаклида намойиш қилади. Энди биз ушбу шаблон асосида кўринишни генерация қилишимиз мумкин:




**Ифодалаш ва таҳрирлаш шаблонларини қайта аниқлаш**

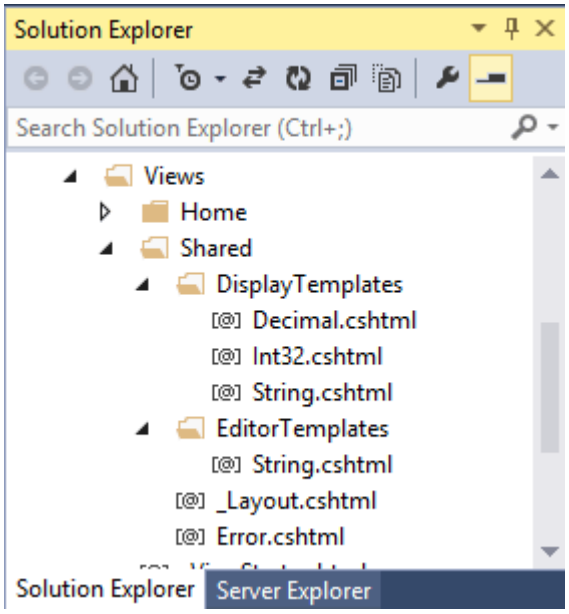
**DisplayFor()/EditorFor()** методларини **MVC** фреймворкида модел майдонларини ифодалаш ва таҳрирлашни қўллашни қай тарзда ва қандай **html** разметкани қилиш лозимлиги кўрсатилади.

Аммо биз ушбу амалларни қайта аниқлашимиз ва **MVC** нинг алоҳида типлар билан ишлаш усулини кўрсатишимиз мумкин. Бизда қуйидаги модел мавжуд бўлсин:

```
public class Book
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
    }
```

Ушбу моделда учта турли типга мансуб маълумотлар мавжуд: **int, string, decimal**. Ушбу хусусиятлар иборат разметкаларни қайта аниқлаймиз. Бунинг учун лойиҳадаги **Views/Shared** каталогда иккита янги папка ҳосил қиламиз:

- **DisplayTemplates** (ифодалаш учун шаблон)
- **EditorTemplates** (таҳрирлаш учун шаблон)




Классда турли уч хил типлардан иборат хусусиятлардан ишлатилганлиги учун, ифодалаш шаблонидаги **DisplayTemplates** папкасида ҳар бир бип учун файл ҳосил қилинган. Файл номлари типларни ифодаловчи класс номига мос ҳосил қилинган. **Int32.cshtml** номли файлда **int** қийматни визуализация қилишда қуйидаги разметкадан иборат бўлади:

```
@model Int32

<b>@Model</b>
```

`@model Int32` директиваси модел сифатида **int** ёки **Int32** типи ишлатилишини кўрсатади. Модел мазмуни эса **<b>** тэги ёрдамида шакллантирилади.

**String.cshtml** файли қуйидаги мазмунга эга:

```
@model String

<b>"@Model"</b>
```

Ушбу файлда модел қиймати қўштирноққа олинган. Ушбу қисмда турли тэглар ва уларнинг тузилмасини аниқланашимиз мумкин. Масалан, (**"@Model"**) каби ёзувни шакллантиришимиз мумкин.

**Decimal.cshtml** файлида:

```
@model decimal
@{
    IFormatProvider formatProvider =
        new System.Globalization.CultureInfo("ru-RU");
```

```
        <span class="currency">@Model.ToString("C",
formatProvider)</span>
}
```

Ушбу файлда модел қиймати муайян формат бўйича (пул форматида) шакллантирилган. Энди биз **DisplayFor()** хелперини модел хусусияти учун ишлатамиз:

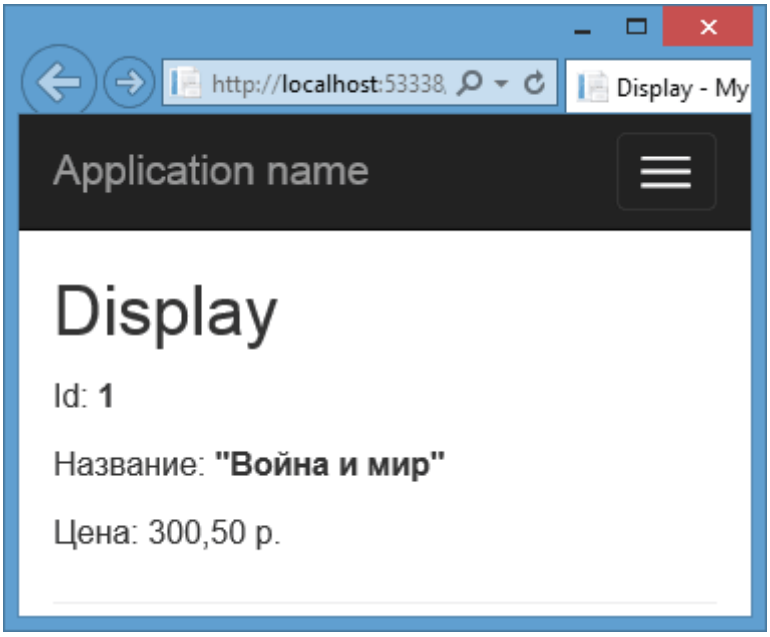
```
public ActionResult Display(int id)
        {
            if (id == null)
            {
                return HttpNotFound();
            }
            Book book = db.Books.Find(id);
            if (book != null)
            {
                return View(book);
            }
            return HttpNotFound();
        }
```

ва

```
@model CustomScaffoldTemplatesApp.Models.Book

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>MyTemplate</title>
</head>
<body>
<h2>Display</h2>
<table>
     <tr>
        <td>
           <b> @Html.DisplayNameFor(model => model.Name) </b>
        </td>

        <td>
            @Html.DisplayFor(model => model.Name)
        </td>
     </tr>
     <tr>
        <td>
           <b> @Html.DisplayNameFor(model => model.Price) </b>
```

```
        </td>

        <td>
            @Html.DisplayFor(model => model.Price)
        </td>
    </tr>
</table>
</body>
</html>
```

Ушбу кўриниш ишга тушган вақтда қуйидаги натижа ҳосил қилинади (**http://localhost:40674/Home/Display/2**):




**EditorTemplates** папкасида ягона **String.cshtml** форматлаш шаблонини қўшиб қўямиз.

```
@model string

<input type="text" name="name" style="border-color:red; height:40px;
background-color:#eee;" value="@Model" />
```

Бизда маълумотларни таҳрирлаш учун қуйидаги контроллер ва кўриниш мавжуд бўлсин:

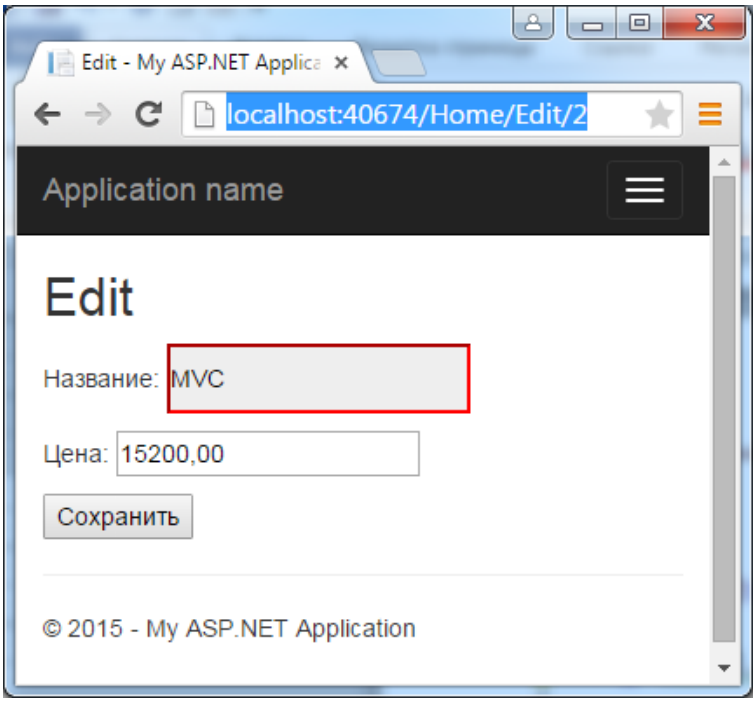
```
public ActionResult Edit(int id)
        {
            if (id == null)
            {
                return HttpNotFound();
            }
```

```
            Book book = db.Books.Find(id);
            if (book != null)
            {
                return View(book);
            }
            return HttpNotFound();
        }
```

ва

```
@model CustomScaffoldTemplatesApp.Models.Book

@{
    ViewBag.Title = "Edit";
}

<h2>Edit</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    @Html.HiddenFor(model => model.Id)
    <p>Название: @Html.EditorFor(model => model.Name)</p>
    <p>Цена: @Html.EditorFor(model => model.Price)</p>
    <p><input type="submit" value="Сохранить" /></p>
}
```

Ушбу контроллер амал методи ишга туширилганда (**http://localhost:40674/Home/Edit/2**)



Бундай шаблонларни киритиш элементлари асосида шакллантиришда уларги қўйилган чекланишларни ҳисобга олиш керак. Ушбу ҳолда элемент **name="name"**каби атрибутга эга бўлади. Агар бизда модел фақат битта сатрий хусусият мавжуд бўлса ва у **Name** деб номланган бўлса, ушбу ёндашувни қўллаш мумкин. Агар бизда бир қанча **string** типидаги хусусиятлар мавжуд бўлса, улар турлича номланади. Ушбу ҳолда киритиш ёки таҳрирлаш шаблонини тўлиқича аниқлашимиз мумкин.

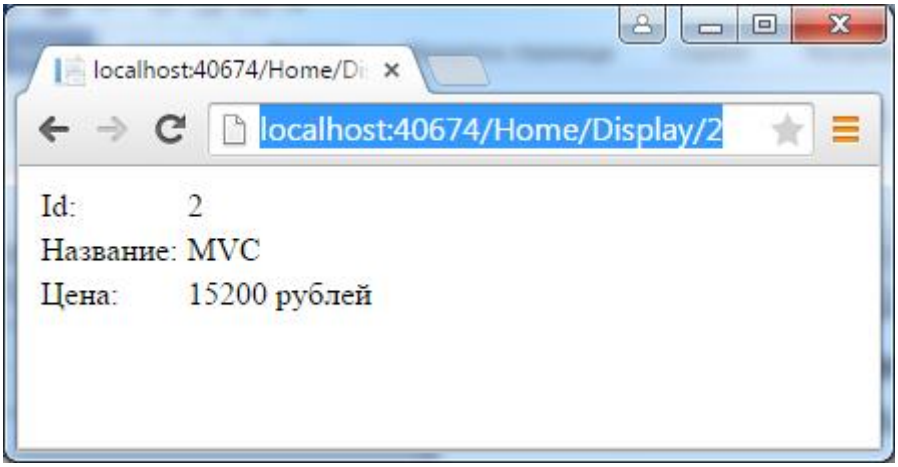
**DisplayTemplates** папкамизга **Book.cshtml** файлини қўшиб қўямиз:

```
@model CustomScaffoldTemplatesApp.Models.Book

<table>
    <tr><td>Id:</td><td>@Model.Id</td></tr>
    <tr><td>Название:</td><td>@Model.Name</td></tr>
    <tr><td>Цена:</td><td>@Model.Price рублей</td></tr>
</table>
```

CustomScaffoldTemplatesApp – лойиҳа номи бўлиб, унда **Book** класси аниқланган. Моделнинг барча хусусиятлари қиймати жадвал орқали чиқарилган. Ушбу ҳолда моделни кўринишда ифодалаш қуйидагича:

```
@model CustomScaffoldTemplatesApp.Models.Book
@{
    Layout = null;
}
@Html.DisplayForModel()
```

**http://localhost:40674/Home/Display/2**

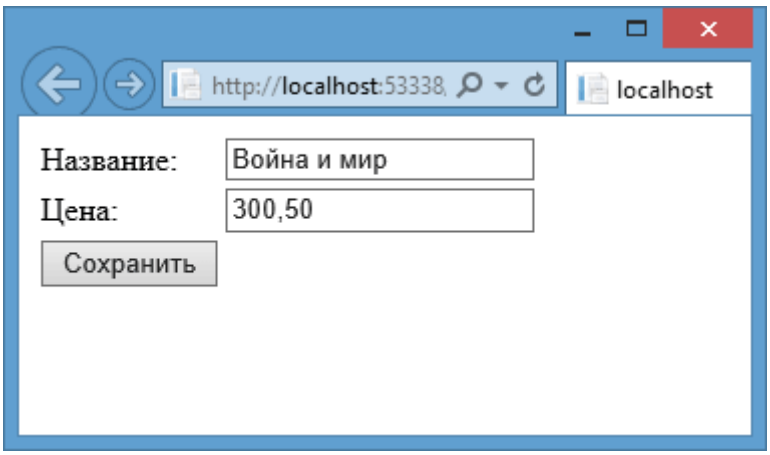



Шунингдек, **EditorTemplates** каталогида таҳрирлаш учун **Book.cshtml** шаблонини ҳосил қиламиз:

```
@model CustomScaffoldTemplatesApp.Models.Book

@using (Html.BeginForm())
{
    @Html.HiddenFor(model => model.Id)
    <table>
        <tr><td>Название: </td><td><input type="text" name="name"
value="@Model.Name" /></td></tr>
        <tr><td>Цена: </td><td><input type="text" name="price"
value="@Model.Price" /></td></tr>
        <tr><td><input type="submit" value="Сохранить"
/></td><td></td></tr>
    </table>
}
```

Ушбу шаблонни таҳрирлаш формасида ишлатамиз:

```
@model CustomScaffoldTemplatesApp.Models.Book

@{
    Layout = null;
}
@Html.EditorForModel()
```



# VI. Маршрутлаштириш

## Маршрутларни аниқлаш

Аввалги бўлимларда маршрутлаштириш масаласини кўриб чиққан эдик. Масалан, бирор контроллер амал методига мурожаат қилиш учун браузердаги буйруқлар сатрида **http://localhost:3456/Home/Index** каби ёзувдан фойдаланишимиз мумкин. Бу ерда **Home** – контроллер номи, **Index** – ушбу контроллердаги амал методи. Агар **Index** методи бирор **int** типига мансуб параметрни қабул қилиш лозим бўлса (**public ActionResult Index(int Id)**), ушбу методга буйруқлар сатрида каби мурожаат қилишимиз мумкин:
**http://localhost:3456/Home/Index/5**

Ушбу бўлимда маршрутлаштириш механизмини кўриб чиқамиз. **ASP.NET MVC 5** да барча маршрутлар **RouteConfig.cs** файлида жойлашган бўлиб, ушбу файл лойиҳадаги **App_Start** папкасида жойлашган. Агар ушбу файлни очиб кўрсак, қуйидаги мазмунга эга бўламиз:

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

namespace CustomScaffoldTemplatesApp
{
    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 = "Home", action = "Index", id
= UrlParameter.Optional }
            );
        }
    }
}
```

Маршрутларни ўрнатиш билан **RegisterRoutes** статик методи шуғулланади. Аммо ушбу амал билан маршрутлаштириш якунланмайди. Биз ушбу методни дастур ишга тушган вақтда чақиришимиз лозим. Ушбу метод **Global.asax** файлидаги **Application_Start** методидан чақирилган:

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

namespace CustomScaffoldTemplatesApp
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }
}
```

`RouteConfig.RegisterRoutes(RouteTable.Routes);` ни чақириш дастурдаги маршрутлаштиришни рўйхатга олади. Энди маршрутни аниқловчи **RouteConfig.cs** файини кўриб чиқамиз.

Ушбу файлдаги биринчи сатр

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
```

орқали муайян кенгайтмали (***.axd (WebResource.axd)**) файллардан сўровга чекланиш қўйилган.

Сўнгра

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

оқали маршрутлаш аниқланган.

**routes.MapRoute** методи сўровга мос маршрутни шакллартиради. Ушбу методнинг перегрузка қилинган вариантларида қўшимча параметрларни узатишимиз мумкин.

Ушбу метод параметрларини кўриб чиқамиз. Аввало **name** хусусияти орқали маршрут номи узаталади - **Default**. Иккинчи параметр ҳисобланган **url** орқали сўров сатридаги **Url шаблони** кўрсатилади.

**Url шаблони** ўзида бир нечта сегментлардан иборат. Ушбу сегментлар фигурали қавсларга олинган. Ушбу ҳолда сегмент сўровнинг бир қисми бўлиб, слешлар орасида жойлашган. Шаблондаги ҳар бир сегмент параметрдан иборат. Ушбу параметрлар **URL** параметрлари деб юритилади. Ушбу ҳолда параметрлар **controller, action** ва **id** дан иборат. Параметрлар турлича номларларга ва ўзида алфавитли-рақамли символлардан иборат эга бўлиши мумкин.

Сўров қабул қилинганда маршрутизация механизми **URL** сатрини бўлаклайди ва маршрут қийматини **объектRouteValueDictionary** луғатига жойлаштиради. Ушбу **объектRouteValueDictionary** объектига дастур контексти ҳисобланган **RequestContext** орқали мурожаат қилиш мумкин. Луғат калити сифатида **URL** номлари қўлланади. **URL** нинг мос сегментлари қийматлар сифатида ишлатилади. Масалан, қуйидаги **URL** сўрови:

**http://localhost/Home/Index/5**

қуйидаги **RouteValueDictionary** луғатида калитлар ва қийматлар жуфтлигига айлантирилади:

| Параметр | Значение |
|---|---|
| controller | Home |
| action | Index |
| id | 5 |

**routes.MapRoute** методининг учинчи параметри - **defaults** орқали маршрут учун **default** қиймат аниқланади. Масалан, агар сўров сатрида барча зарур параметрлар узатилмаса, сўров **http://localhost/** манзили бўйича узатилаётган бўлса, маршрутизация механизми **Home** контроллеридаги **Index** методини чақиради. Чунки ушбу қиймат **defaults** параметри орқали кўрсатилган. Шунингдек, биз контроллер методини кўрсатмасак

**http://localhost/Home/,** ушбу ҳолда ҳам **Home** контроллеридаги **Index** методини чақирилади.

Биз дастур ишга туширилганда фойдаланувчига **HomeController** контроллерининг **Index** методи ўрнига **BookController** контроллерининг **Show** методига мурожаат қилиш лозим бўлса, биз **defaults** параметри қийматини қуйидагича ўзгартиришимиз лозим:

```
defaults: new { controller = "Book", action = "Show", id = UrlParameter.Optional }
```

Охирги параметр (`id = UrlParameter.Optional`) қатнашиши шарт эмас ҳисобланади. Шунинг учун у сўровда мавжуд бўлмаса, у ҳисобга олинмайди ва **RouteValueDictionary** луғатига узатилади.

**http://localhost/Home/Create/3** сўрови орқали **HomeController** контроллеридаги **Create** методи чақирилади ва ушбу методга **3** сони параметр сифатида узатилади. Ўз навбатида **http://localhost/Home/Create/** сўров **HomeController** контроллеридаги **Create** методи чақиради. Ушбу ҳолда параметр узатилмаган.

**defaults** параметри орқали параметрларни шакллантириш орқали сўров сатрида контроллер ва унинг методини тўлиқ кўрсатиш шарт эмас. Аммо ушбу параметрлар кўрсатилмаган бўлса, биз сўров сатрида контроллер ва унинг методини келтиришимиз лозим. Лойиҳадаги маршрутларни **RouteConfig.cs** файлида қуйидагича ўзгартирамиз:

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

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}");
        }
    }
```

Ушбу маршрутлар орқали ҳосил қилинган лойиҳамизга браузердан **http://mysyte.com/** узатма орқали мурожаат қиламиз. Ушбу ҳолда бизга хатолик қайтарилади. Хатолик браузердаги буйруқлар сатрида сўровни тўлиқ амалга ошириш лозимлигини англатади. Шунинг учун ушбу ҳолда лойиҳамизга

**http://mysyte.com/Home/Index** каби мурожаат қилиш зарур.

Агар биз **http://localhost/Home/** манзилга мурожаат қилсак, яна хатолик юзага келади. Чунки буйруқлар сатридаги узатмада фақат битта сегмент кўрсатилган. Маршрутни аниқлаган вақтимизда эса иккита сегмент кўрсатилган эди: **{controller}/{action}**. Агар иккита параметр учун бошланғич қийматлар кўрсатилмаган бўлса, сўров сатрида ҳам шунча сегментлар бўлиши лозим.

Агар сўров учта сегментдан иборат бўлса, (**http://localhost/Home/Index/1**) лойиҳани ишга туширганимизда яна хатолик қайтарилади. Чунки сўровдаги сегментлар сони маршрутда кўрсатилган **URL** дан кўп.

## Маршрутлар билан ишлаш

### Янги маршрутлар яратиш

Янги маршрутларни ҳосил қилиш учун **MapRoute** методидан фойдаланилади:

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

            routes.MapRoute(
            name: "newRoute",
            url: "{controller}/{action}");
        }
    }
```

Ёки аввало **Route** объектини ҳосил қилиб, сўнгра уни **RouteCollection** маршрутлар коллекциясига қўшиб қўямиз. Иккита маршрут аниқлаймиз:

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

            routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}");
```

```
            Route newRoute = new Route("{controller}/{action}", new
MvcRouteHandler());
            routes.Add(newRoute);

        }
    }
```

Ушбу лойиҳада иккита маршрут аниқланган. Биринчи маршрут **Default** бўлиб, учта сегментдан иборат сўровга мос қўйилади. Иккинчи маршрут **newRoute** ҳисобланиб, иккита сегментдан иборат. **http://localhost:5555/Home/Index/1** сўровини амалга ошириш натижасида биринчи маршрут орқали иш кўрилади.

**http://localhost:5555/Home/Index** сўрови орқали иккинчи маршрут орқали амаллар бажарилади. **http://localhost:5555/Home** сўрови бирор маршрутга тўғри келмайди. Чунки бизда ушбу турдаги фақат битта сегментдан иборат маршрут аниқланмаган.

Ушбу иккита маршрутни ягона маршрутга олиб келишимиз мумкин:

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

Параметрлар учун бошланғич қийматни беришда параметр позициясини ўрнини ҳисобга олиш лозим. Маршрутизацияда параметрлар учун бошланғич қийматлар тўғри берилиши лозим. Масалан, қуйидаги маршрут аниқланган бўлсин:

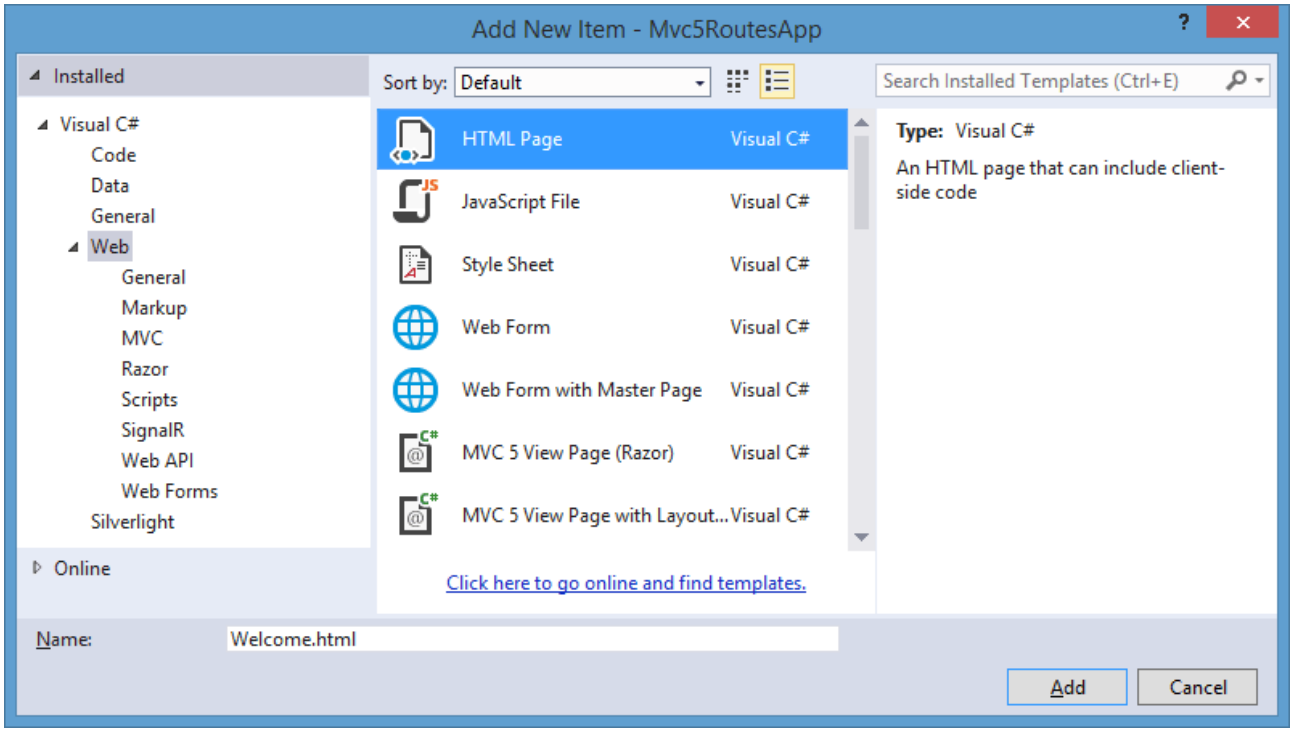
```
routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { action = "Index" });
```

у ҳолда иккита сегментдан иборат **Home/2** сўрови ушбу маршрутга мос келмайди. Шунинг учун ушбу ҳолда биз **Id** параметр учун ҳам бошланғич қийматни шакллантиришимиз лозим.

**Дискда жойлашган файлларга сўровларни амалга ошириш**

Биз контроллерлар ва уларнинг методлари орқали маршрутлар ҳосил қилишни кўриб чиқдик. Аммо сўровларни лойиҳамизнинг алоҳидаги файллари орқали (**статик html саҳифалар**) амалга оширишимиз мумкин. Ушбу усул орқали шакллантирилган маршрутлаштириш механизмида аввало сўровнинг серверда сақланаётган муайян файлга мослиги текширилади. Агар бундай мавжуд бўлмаса, сўров бошқа маршрутларга мослиги текширилади.

Лойиҳамиздаги **Context** папкасида янги html-саҳифани ҳосил қиламиз ва унга **Welcome.html** деб ном берамиз.




Лойиҳамизни тестдан ўтказиш учун ушбу саҳифага бирор контентни ҳосил қиламиз. Сўнгра унга лойиҳамиздан мурожаат қилиш учун **Content/Welcome.html** сўровидан фойдаланишимиз мумкин.
**http://localhost:30839/Content/Welcome.html**

**Сўров сатрида префикслардан фойдаланиш**

Сўров сатридаги сегментларда параметрлар учун фақат маршрутларда аниқланган қийматларни ўзида сақлаши шарт эмас. Сўров сатрида турли префикслардан фойдаланиш ва маршрутни ушбу турлаги сўровларни қайта ишлайдиган қилиш мумкин. Мисол, биз сўровда **Uz** префикси ишлатилган бўлсин**:**

**http://localhost:49326/Uz/Home/Index/1**

Ушбу ҳолда ушбу сўров учун маршрут қуйидагича шакллантирилади:

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

Шунингдек, префиксларни муайян сегментга мос ҳолда ташкил қилиш ҳам мумкин:

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

Ушбу ҳолда сўров сатри қуйидагича амалга оширилиши мумкин:

**http://localhost:49326/RuHome/Index**

**Янги маршрутларни ҳосил қилиш тартиби**

Лойиҳада янги маршрутларни шакллантиришда уларни тартибини ҳисобга олиш лозим. Кўп ҳолларда юқори статусга эга бўлган маршрутлар олдинроқ келтирилади. Бизда қуйидаги маршрутлар мавжуд бўлсин:

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

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

            routes.MapRoute(
                name: "Default2",
                url: "Ru{controller}/{action}/{id}",
                defaults: new { id = UrlParameter.Optional });
        }
```

Лойиҳага **http://localhost:49326/RuHome/Index** каби сўров амалга оширилган вақтда, агар **RuHomeController** контроллери аниқланмаган бўлса, дастур хатоликни қайтаради.

Нега? Чунки маршрутлаштириш тизими аввал биринчи маршрут билан текширилади. Агар сўров биринчи маршрутга мос келмаса, кейин иккинчи ва ундан кейинги маршрутлар билан текширилади. Аммо биринчи маршрут орқали **RuHomeController** контроллери изланади ва у ушбу контроллерни топмасдан, хатолик қайтаради. Шунинг учун ушбу ҳодиса юзага келмаслиги учун, аввал **Default2** маршрутни аниқлаш лозим.

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

            routes.MapRoute(
                name: "Default2",
                url: "Ru{controller}/{action}/{id}",
                defaults: new { id = UrlParameter.Optional });

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

Шунингдек, биз префиксларни контроллер ва унинг методи учун ишлатишимиз мумкин.

```
routes.MapRoute(
                name: "Default2",
                url: "Store/Buy",
                defaults: new { controller = "Home", action = "Index" });
```

Ушбу ҳолда **HomeController** контроллер учун **Store** псевдоними **Index,** амали учун эса **Buy** псевдоними ишлатилмоқда. Натижада ушбу маршрут **Store/Buy** каби сўрови каби мос қўйилади. Маршрутлаштириш тизими ушбу турдаги сўровга **Home** контроллерининг **Index** методига мурожаат қилади.

### Узатилган параметрларни қабул қилиш

Маршрут параметрига узатилган қийматни олиш учун биз **RouteData** объектидан фойдаланишимиз мумкин. Агар, бизда стандарт

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

```
            );
```

маршрути аниқланган бўлса, **controller** параметри қийматини қуйидагича олишимиз мумкин:

```
public string GetParam()
        {
            string controller =
RouteData.Values["controller"].ToString();
            return controller;
        }
```

**http://localhost:34362/Home/GetParam**

Қўшимча қийматлар сифатида узатилаётган маълумотларни метод параметридан қуйидагича олиш мумкин:

```
public ActionResult GetParam(int id)
        {
            ViewBag.OldId = id;
            //или так
            // ViewBag.OldId = RouteData.Values["id"];
            return View();
        }
```

**Сўровда ихтиёрий сондаги параметрларни узатиш**

Аввалги мисолларда сўров сатрида фақат учта параметр билан чекланган эдик. Аммо бизнинг мисолда метод икки ва ундан ортиқ параметр қабул қилса:

```
public ActionResult MyMethod(int id = 1, string name = "")
        {
            ViewBag.Name = name;
            return View();
        }
```

Биз маршрутга зарурий параметрлар сонини кўрсатишимиз мумкин:

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

Ушбу ҳолда жорий маршрут **Home/Index/1/name** каби сўровларни қабул қила олади. Ушбу сўров сатри **Home/Index?id=1&name=name** сўров сатрига эквивалент.

Шунингдек, биз сўров ихтиёрий сондаги сегментлардан **{*catchall}**: параметри фойдаланишмиз мумкин.

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

Энди ихтиёрий сегментли сўровларни қайта ишлаш мумкин:

| Сўров | Сўров параметри |
|---|---|
| mysyte.com | controller=Home<br>action=Index |
| mysyte.com/Book | controller=Book<br>action=Index |
| mysyte.com/Book/Show | controller=Book<br>action=Show |
| mysyte.com/Book/Show/2 | controller=Book<br>action=Show<br>id=2 |
| mysyte.com/Book/Show/2/Oldedition | controller=Book<br>action=Show<br>id=2<br>catchall=Oldedition |
| mysyte.com/Book/Show/2/Oldedition/1960 | controller=Book<br>action=Show<br>id=2<br>catchall=Oldedition/1960 |

**catchall** параметри учун қийматини олиш учун алоҳидаги сенгментларни қайта ишлаш лозим

## Маршрутлаш учун чегланишларни шаклантириш

Баъзи ҳолларда жорий маршрут учун сўров сатрининг устма уш тушиши қатъий чеклагич қўйишга тўғри келади. Масалан ҳосил қилиниаётган контроллер номи доим «**H**» ҳарфи билан бошланиши лозим бўлсин. У ҳолда ном учун муайян чекланишларни регуляр ифода орқали амалга ошириш мумкин:

```
routes.MapRoute(
                name: "Default",
```

```
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id
= UrlParameter.Optional },
                constraints: new { controller = "^H.*" }
            );
```

**constraints** параметри орқали маршрутга чеклагич қўйиш мумкин.

Агар биз лойиҳага **Book/Index** каби сўровни узатсак, ҳатто бизда **Index** методига эга **BookController** контроллери мавжуд бўлса, дастур хатоликни қайтаради. Чунки контроллер чеклагичга тушади, **Book/Index** сўровига мос маршрут эса бизда мавжуд эмас.

Биз ушбу турдаги чеклагичларни бошқа параметрларга ҳам бериш мумкин. **Id** параметри камида иккита рақамдан иборат бўлиши лозим бўлсин:

**Home/Index/1** сўрови маршрутга мос бўлмаганлиги сабабли, хатолик қайтаралида. Чунки **Id** параметр фақат битта рақамдан иборат.

Шунингдек, метод типига ҳам чеклагич қўйиш мумкин. Лойиҳада фақат **GET**-сўровлар қайта ишланиши лозим бўлсин:

```
routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}/{*catchall}",
                defaults: new { controller = "Home", action = "Index" },
                constraints: new { controller = "^H.*", id =
@"\d{2}",httpMethod = new HttpMethodConstraint("GET") }
                );
```

Эндиликда фақат **GET**-сўровлари қабул қилинади.

### Шахсий чеклагичларни яратиш

Шахсий чеклагичларни яратиш учун **IRouteConstraint** интерфейсини ягона **Match** методи орқали тадбиқ қилади:

```
public interface IRouteConstraint
        {
            bool Match(HttpContextBase httpContext, Route route, string
parameterName,
                RouteValueDictionary values, RouteDirection
routeDirection);
        }
```

**IRouteConstraint** интерфейс орқали маршрутга чеклагич қўйилади. Бунда маршрутлаштиришга **IRouteConstraint.Match** методи қўлланилади. Масалан, баъзи **url** лар бўйича сўровларни қабул қилишга чеклаш лозим бўлсин. Бунинг учун лойиҳага қуйидаги классни ҳосил қиламиз:

```
public class CustomConstraint : IRouteConstraint
        {
            private string uri;
            public CustomConstraint(string uri)
            {
                this.uri = uri;
            }
            public bool Match(HttpContextBase httpContext, Route route,
string parameterName,
                    RouteValueDictionary values, RouteDirection
routeDirection)
            {
                return !(uri == httpContext.Request.Url.AbsolutePath);
            }
        }
```

Ушбу методда агар сўров амалга оширилаётган ресурс **httpContext.Request.Url.AbsolutePath** хусусиятига мос бўлса, сўров маршрутга текширилмайди. Ушбу ҳолда маршрутни аниқлаш қуйидагича амалга оширилиши мумкин:

```
routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}/{*catchall}",
                defaults: new { controller = "Home", action = "Index" },
                constraints: new { id = @"\d{2}", myConstraint = new
CustomConstraint("/Home/Index/12") }
                );
```

Эндиликда **/Home/Index/12** сўрови барча чеглагичларни ва шартларни қаноатлантирса ҳам қайта ишланмайди.

### Сўровларни ҳисобга олмаслик

**RouteConfig** классидаги **RegisterRoutes** методида **routes.IgnoreRoute("{resource}.axd/{*pathInfo}");** сатри аниқланган. Ушбу ифода орқали биз серверга муайян ресурсларга мурожаат қилишга чеклаш қўямиз. Қуйидаги мисолда **/Home/Index/12** каби маршрутга чеклаш қўйилган:

```
public static void RegisterRoutes(RouteCollection routes)
```

```
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
            routes.IgnoreRoute("Home/Index/12");
            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id
= UrlParameter.Optional }
            );
        }
```

Эндиликда **Home/Index/12** сўрови қайта ишланмайди ва сервердаги муайян ресурсга мос қўйилади.

## Кирувчи URL манзилларни генерация қилиш

Маршрутизация механизмининг яна бир томони дастурда кирувчи манзилларни генерация қилиш ҳисобланади. Ушбу амални **a** элементи анкоридан фойдаланиш ҳисобланади:

```
<a href="Home/Index/3" />
```

Шунингдек биз махсус **Html.ActionLink** ва **Html.RouteLink** хелперларидан рендерингда фойдаланишимиз мумкин.

### Html.ActionLink

**ActionLink** хелпери орқали амал контроллерига гипермурожаат ҳосил қилинади. Агар биз амал методига узатмани жорий контроллерга ҳосил қилсак, амал методи номини кўрсатишимиз етарли:

```
@Html.ActionLink("Бу ерни босинг", "Show")
```

Ушбу хелпер қуйидаги **html** разметкани генерация қилади:

```
<a href="/Home/Show">Бу ерни босинг</a>
```

Бошқа контроллердаги амал методига кўрсатма ҳосил қилиш лозим бўлса, **ActionLink** хелперида учинчи параметр сифатида контроллер номини бериш керак. Мисол сифатида **Book** контроллеридаги **List** амал методига узатмани ҳосил қилиш лозим бўлсин:

```
@Html.ActionLink("Список книг", "List", "Book")
```

Шунингдек, бизнинг **Book** контроллеримиздаги бирор **Index** методда бир қанча параметрлар мавжуд бўлса:

```
public class BookController : Controller
        {
            public string Index(string author = "Толстой", int id = 1)
            {
                return author + "  " + id.ToString();
            }
        }
```

Ушбу ҳолда **ActionLink** хелперининг перегрузка қилинган варианти орқали **routeValues** параметри учун объектни узатиш мумкин. Бажариш муҳити объектнинг хусусиятини қабул қилади ва уларни маршрутизация қийматларини яратишда фойдаланади. Юқорида келтирилган контроллер амал методига узатмани ҳосил қиламиз:

```
@Html.ActionLink("Все книги", "Index", "Book", new { id = 10 }, null)

// ёки

@Html.ActionLink("Достоевский", "Index", "Book", new { author =
"Достоевский", id = 5 }, null)
```

Ушбу хелпердаги охирги параметр **htmlAttributes** параметри ҳисобланади. Биз ушбу параметрни **HTML** элементи атрибути қийматини ўрнатишда фойдаланишимиз мумкин. Ушбу ҳолда **null** қиймати узатилади (ҳеч қандай атрибутлар ўрнатилмайди).

Энди муайнян (**id** ва **class**) атрибутларни ўрнатишни кўриб чиқамиз:

```
@Html.ActionLink("Все книги", "Index", "Book", new { author = "Толстой",
id = 10 }, new { id = "Tolstoi", @class = "link" })
```

Натижада қуйидагича **html**-разметка генерация қилинади:

```
<a class="link"
href="/Book/Index/10?author=%D0%A2%D0%BE%D0%BB%D1%81%D1%82%D0%BE%D0%B9"
id="Tolstoi">Все книги</a>
```

**class** сўзидан аввал келтирилган **@** белгисига эътибор беринг: **class** сўзи **C#** тилида калит сўзи бўлганлиги сабабли, кўринишда тўғри рендеринг амалга оширилиши лозим.

**Html.RouteLink**

**RouteLink** хелпери **ActionLink** га ўхшаш шаблондан фойдаланилади. Ушбу хелперга маршрут номи узатилади. Аммо контроллер номи ва амал методи аргументлари талаб қилинмайди. Биринчи мисолдаги келтирилган **ActionLink** қуйидаги кодга эквивалент:

```
@Html.RouteLink("Все книги", new { controller = "Book", action = "Index",
author = "Толстой", id = 10 }, new { id = "Tolstoi", @class = "link" })
```

Маршрутни ишлатиш учун, муайян маршрут номи ва ундан сўнг қўшимча параметрларни кўрсатиш лозим. **Default** стандарт маршрутини оламиз:

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

У ҳолда биз қуйидаги узатмани ҳосил қилишимиз мумкин:

```
@Html.RouteLink("Все книги", "Default", new { action = "Show" })
```

**URL-хелперы**

**URL**-хелперлар **ActionLink** ва **RouteLink** хелперларига ўхшаш бўлиб, улар **HTML** кодни қайтармайди. Улар **URL** йўлларини ҳосил қилади ва уларни сатр шаклида қайтаради. Уч турдаги **URL**-хелперлар мавжуд:

- **Action**
- **Content**
- **RouteUrl**

**Action** хелпери **ActionLink** га ўхшаш бўлиб, аммо у якор тэгини қайтармайди. Қуйидаги код орқали узатма ўрнига **URL** манзилини қайтаради.

```
@Url.Action("Index", "Book", new { author = "Толстой", id = 10 }, null)
```

**RouteUrl** хелпери **Action** га ўхшаш шаблони ишлатади. Аммо у **RouteLink** каби маршрут номини қабул қилади ва маршрут учун аргументларни қабул қилади.

```
@Url.RouteUrl(new { controller = "Book", action = "Index", author =
"Толстой", id = 10 })
```

**Content** хелпери нисбий йўлларни абсолютга айлантиради. **Content** хелперини ишлатишни **_Layout** кўринишда кўриш мумкин.

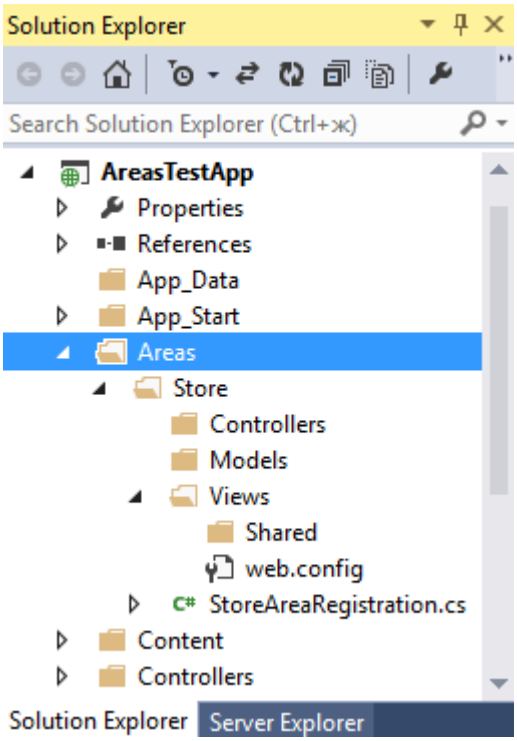
```
<script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")"
        type="text/javascript"></script>
```

Тилда **(~)** ёрдамида **Content** хелпери зарурий **URL**ни лойиҳа жойлашган манзилга мос ҳолда генерация қилади. Агар дастурни бошқа виртуал каталогга ўтказсангиз, тилдасиз **URL** коррект ҳисобланмайди.

## Соҳалар

**MVC** лойиҳада функционал қисмлар аниқ структурланган (контроллерлар, моделлар, кўринишлар) бўлсада, баъзи ҳолларда катта лойиҳаларда дастур бир қанча соҳаларга (**area**) ажратилади.

**MVC** лойиҳасига соҳани қўшиб қўямиз. Лойиҳага сичқончанинг ўнг тугмасини босиб, ҳосил қилинган менюдан **Add->Area** ни танлаймиз. Ҳосил қилинган ойнада соҳа номини **Store** деб белгилаймиз. Шундан сўнг лойиҳа тузилмасида бир қанча ўзгартиришлар амалга оширилади: лойиҳада янги **Areas** папкаси, унинг ичида **Store** папкаси ҳосил қилинади. **Store** папкаси ичида мини-лойиҳа жойлаштирилади. Ушбу папка ичида контроллер, моделлар ва кўринишлар учун папкалар ва соҳанинг класси рўйхатидан ўтказиш жойлашади.



Ҳосил қилинган **StoreAreaRegistration.cs** файли қуйидаги коддан иборат:

```
using System.Web.Mvc;

namespace AreasTestApp.Areas.Store
{
    public class StoreAreaRegistration : AreaRegistration
    {
        public override string AreaName
        {
            get
            {
                return "Store";
            }
        }

        public override void RegisterArea(AreaRegistrationContext
context)
        {
            context.MapRoute(
                "Store_default",
                "Store/{controller}/{action}/{id}",
                new { action = "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```

Ушбу файлдаги маршрутнинг автоматик генерация қилинганлиги кирувчи сўровларнинг **Store** соҳасидаги контроллер ва амалларга мос қўйилади. Аммо сўровларнинг ушбу соҳада амалга оширилиши учун барча соҳаларни **Global.asax** файлида рўйхатга олиш керак. Ушбу амални **Visual Studio** автоматик ҳосил қилади. Ушбу **Global.asax** файлни очиб амалга оширилган ўзгаришларни кўриш мумкин:

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

namespace AreasTestApp
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
```

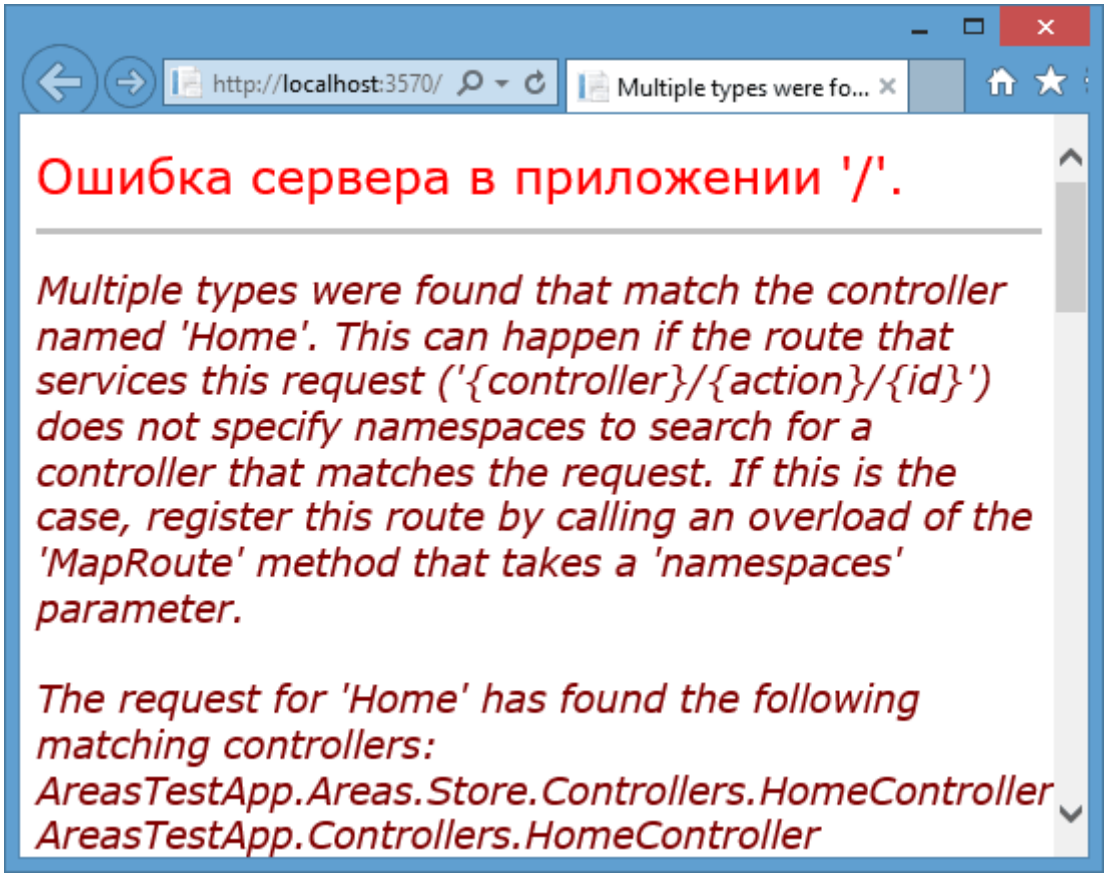
```
        {
            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }
}
```

Бизнинг соҳани синовдан ўказиш учун **Controllers** папкасида янги контроллер ва унда амал методини ҳосил қиламиз. Шунингдек, ушбу контрроллерга мос кўринишни ҳам яратамиз. Бизнинг **Store** соҳамизда **ShopController** контроллери аниқланган бўлсин:

```
public class ShopController : Controller
    {
        //
        // GET: /Store/Shop/
        public ActionResult Index()
        {
            return View("ShopController");
        }
     }
```

Лойиҳа ишга туширилганда ушбу **Index** методига **/Store/Shop/Index** манзили орқали мурожаат қилишимиз мумкин. Бунинг учун аввал соҳа номи, сўнгра контроллер ва метод номи келтирилади.

Асосий лойиҳамиздаги **Home** контроллерига **Index** методини ҳосил қилсак, стандарт маршрут орқали дастур бизни **Home** контролеридаги **Index** методига узатади. Маршрутлаштириш тизими қайси контроллерга мурожаат қилишни билмайди. Натижада қуйидаги ҳодиса (хатолик) юзага келади:




Ушбу ҳодисани олдини олиш учун лойиҳадаги **RouteConfig.cs** файлида дастур ишга туширилган вақтда чақирилувчи **Home** контроллерини кўрсатиш лозим.

```
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 = "Home", action = "Index", id
= UrlParameter.Optional },
                namespaces: new[] { "AreasTestApp.Controllers" }
            );
        }
    }
```

Ушбу ҳолда асосий дастур **AreasTestApp** номлар фазосида жойлашган. Чунки лойиҳа **AreasTestApp** деб номланган. **Home** контроллери эса **AreasTestApp.Controllers** номлар фазосида жойлашган. Бизнинг дастур ишга туширилганда **Store** соҳасидаги **Home** контроллерининг **Index** методи ишга тушиши лозим бўлса, мос номлар фазосини кўрсатиш мумкин:
**AreasTestApp.Areas.Store.Controllers**

**Соҳаларда узатмаларни генерация қилиш**

Кўринишларда узатмаларни генерация қилишда **Html.ActionLink** каби хелперлардан фойдаланиш мумкин. Аммо бу ерда ўзига хос жиҳат мавжуд. Муайян соҳа ичидаги контроллердаги бирор амал методига узатмани генерация қилиш учун қуйидаги коддан фойдаланамиз:

```
@Html.ActionLink("Все книги", "Index", "Book", new { id = 10 }, null)
```

Натижада бизга қуйидаги узатма генерация қилинади:

```
<a href="Store/Book/Index/10>Все книги</a>
```

Агар зарур амал методи ва контроллер бошқа соҳада жойлашган бўлса, мос параметрларни хелперда кўрсатишимиз лозим:

```
@Html.ActionLink("Все книги", "List", new { area = "Library", controller
= "Book" })
```

Ушбу хелпер асосида қуйидаги узатма генерация қилинади:

```
<a href="Library/Book/List />Все книги</a>
```

Агар метод ва контроллер асосий лойиҳада жойлашган бўлса, **area** параметрида бўш сатр келтирилади:

```
@Html.ActionLink("Все книги", "Index", new { area = "", controller =
"Home" })
```

## Маршрутларни қайта ишлашнинг шахсий тизими

Маршрутларни қайта ишлашни янги тизимини ишлаб чиқишдан аввал, маршрутлаштириш тизимини кўриб чиқамиз. Маршрутлаштириш жараёни қуйидаги босқичлардан иборат:

1. **UrlRoutingModule** модули жорий сўровни **RouteTable** маршрутлар жадвалидаги қийматлар билан таққослаб чиқади.
2. Агар мос қиймат аниқланса, маршрутлаштириш модулини **IRouteHandler** объектига узатади.
3. Сўнгра **IRouteHandler** объектида **GetHandler** методи чақирилади. У сўровни қайта ишлаш учун **IHttpHandler** объектини қайтаради.
4. **IHttpHandler** қайта ишлашда **ProcessRequest** методи чақирилади.


**Foydalanilgan adabiyotlar**

**Asosiy adabiyotlar**

1. Roger Pressman, Bruce Maxim, Software Engineering: A Practitioner's Approach, John Wiley & Sons, USA 2014.
2. Ian Sommerville. Software Engineering Hardcover. Pearson 2010 USA

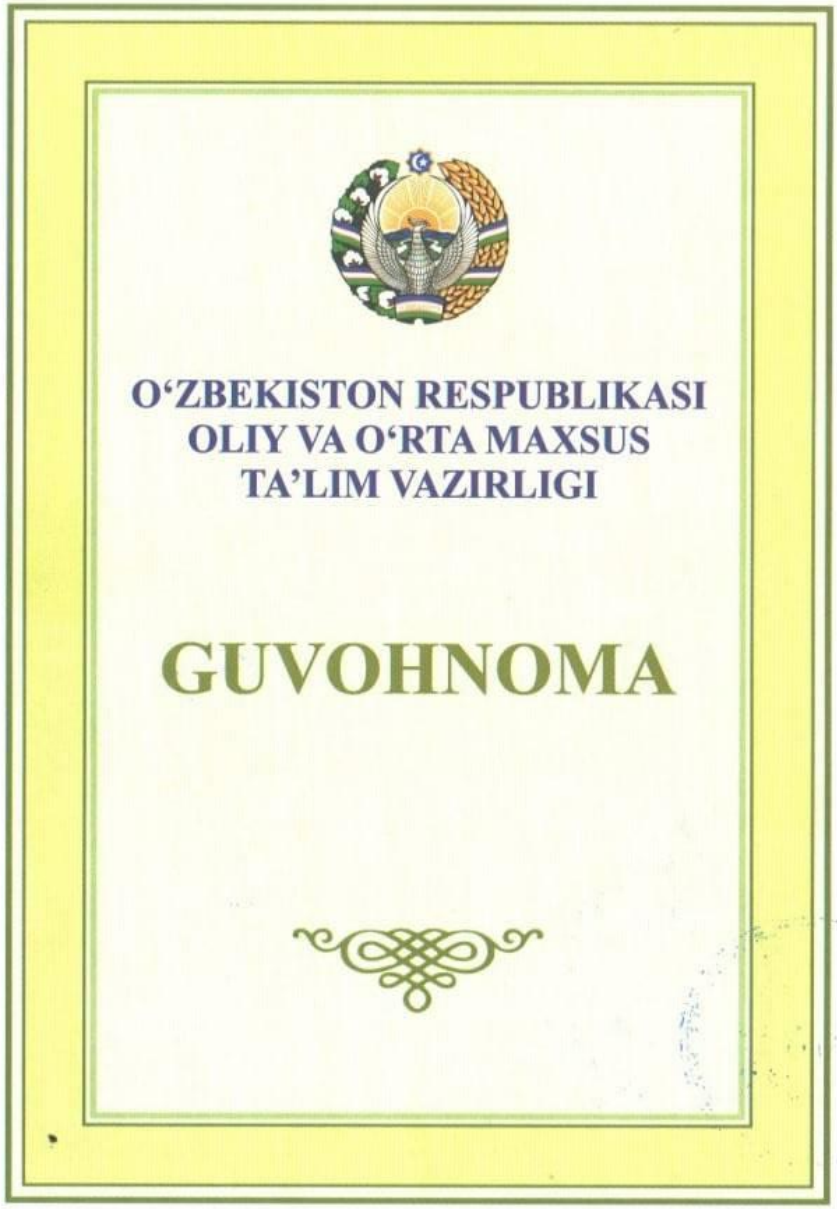
**Qo’shimcha adabiyotlar**

3. Orit Hazzan, Yael Dubinsky Agile Software Engineering Springer; 2009
4. Akao, Y., Quality Function Deployment, Productivity Press, 2004
5. Ambler, S., The Object Primer, 2d ed., Cambridge University Press, 2001
6. Larman K Primeneniye UML i shablonov proyektirovaniya, M,2002.
7. Trofimov «Rational Rose», SPb, Piter – 2002.
8. Maklarov S. V. Sozdaniye informasionnыx sistem s AllFusion Modeling Suite. M.: DIALOG – MIFI, 2002. – 224 s.
9. Maklarov S. V. BPWin i ERWin. CASE – sredstva razrabotki informasionnыx sistem. - M.: DIALOG – MIFI, 1992. – 256 s.
10. R. Lafore. Obyektno-oriyentirovannoye programmirovaniye v S++ [Tekst].

M.: BINOM 2004.

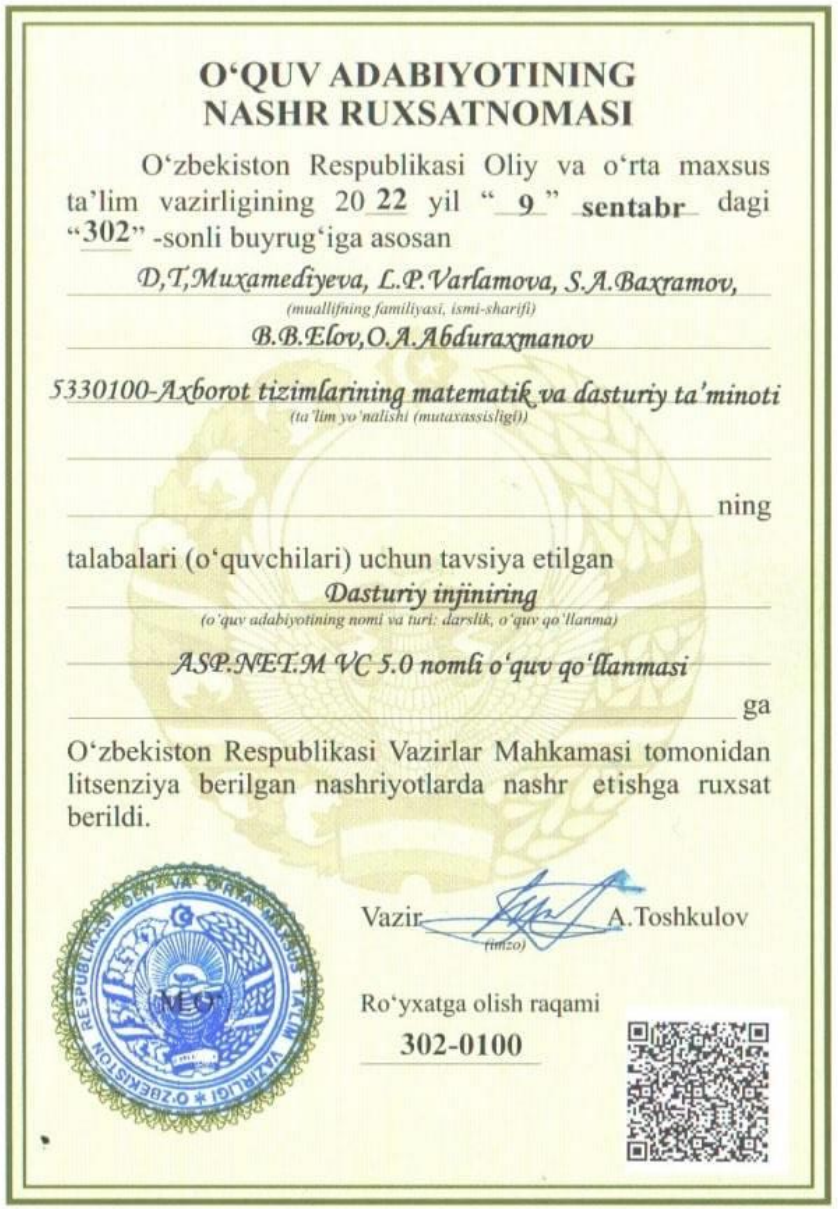
**Elektron manbalar**

1. Computerworld, PC Week, PC Magazine, PC Computing, Macworld, Inter@ctive week, Computer Shopper , PC World jurnallari.
2. Apple Computer, Dell Computer, Gateway, IBM, Hewlett-Packard, Compaq, and Sun Microsystems, Microsoft, Lotus, IBM, Oracle WWW saytlari.
3. springer.com



O‘ZBEKISTON RESPUBLIKASI
OLIY VA O‘RTA MAXSUS
TA’LIM VAZIRLIGI

# GUVOHNOMA



## O‘QUV ADABIYOTINING NASHR RUXSATNOMASI

O‘zbekiston Respublikasi Oliy va o‘rta maxsus ta’lim vazirligining 20 22 yil “ 9 ” sentabr dagi “302” -sonli buyrug‘iga asosan

*D,T,Muxamediyeva, L.P.Varlamova, S.A.Baxramov,*
(muallifning familiyasi, ismi-sharifi)
*B.B.Elov,O.A.Abduraxmanov*

*5330100-Axborot tizimlarining matematik va dasturiy ta’minoti*
(ta’lim yo’nalishi (mutaxassisligi))

ning

talabalari (o‘quvchilari) uchun tavsiya etilgan

*Dasturiy injiniring*
(o’quv adabiyotining nomi va turi: darslik, o‘quv qo’llanma)

*ASP.NET.M VC 5.0 nomli o‘quv qo‘llanmasi*

ga

O‘zbekiston Respublikasi Vazirlar Mahkamasi tomonidan litsenziya berilgan nashriyotlarda nashr etishga ruxsat berildi.

Vazir _____ A.Toshkulov
(imzo)

Ro‘yxatga olish raqami
302-0100