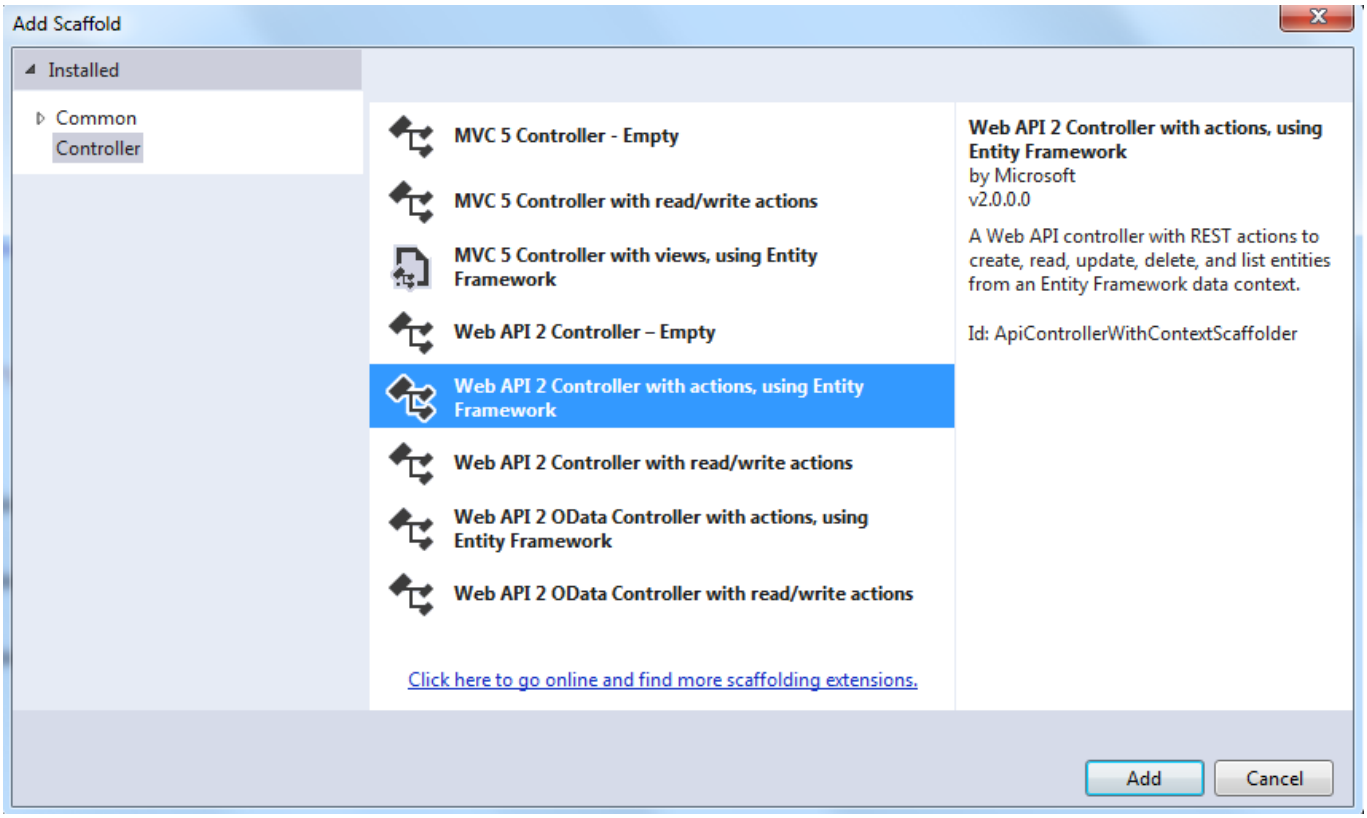
Шаблоны формирования

Так как большинство приложений так или иначе основываются на стандартных CRUD-операция (Create - Read - Update - Delete), то зачастую разработчики вынуждены многократно создавать контроллеры и представления для одних и тех же действий: добавления, изменения, удаления и просмотра записей из БД. И чтобы облегчить разработчикам жизнь, команда MVC добавила такую полезную функциональность, как **шаблоны формирования** (scaffolding templates). Эти шаблоны позволяют по заданной модели и контексту данных сформировать весь необходимый базовый код контроллеров, а также всю разметку для представлений, с помощью которых можно управлять записями в БД.

Чтобы воспользоваться данной функциональностью, добавим новый контроллер. Нажмем правой кнопкой мыши на папку Controllers и выберем **Add -> Controller...**. Далее в окне добавления нового контроллера нам будет предложено выбрать шаблон контроллера:



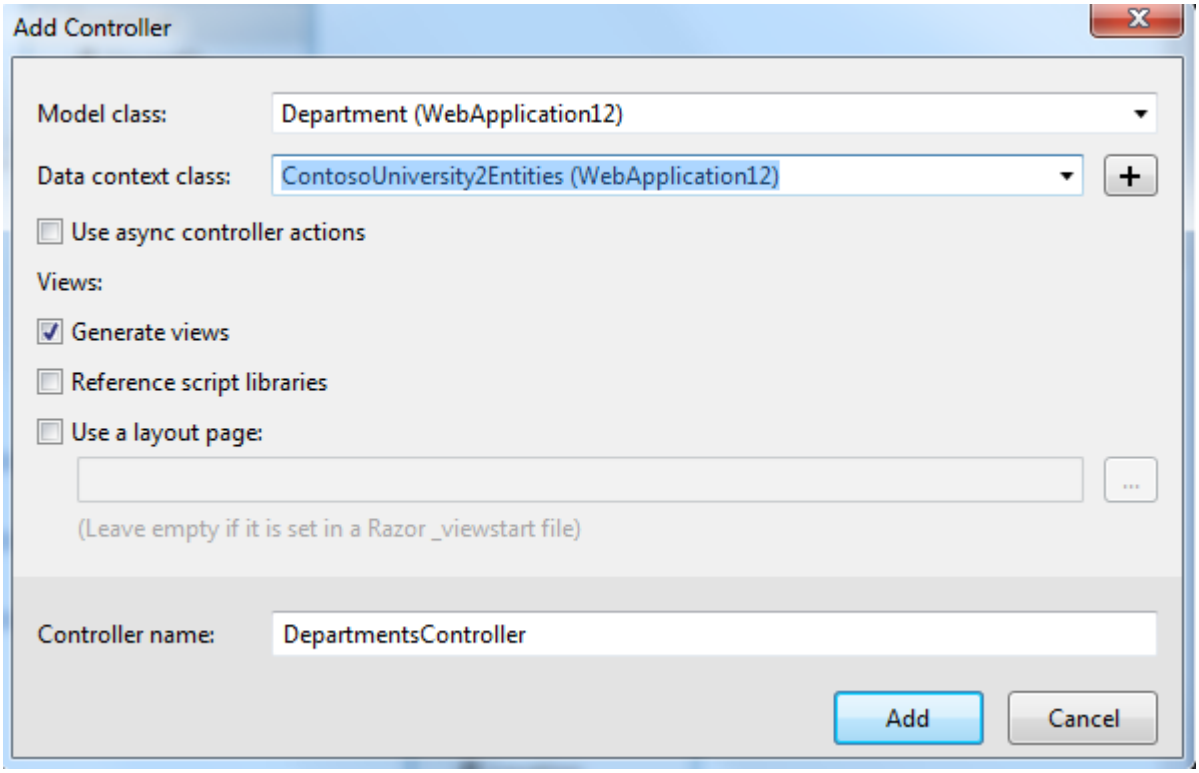
Собственно к MVC относятся только первые три шаблона:

- **MVC 5 Controller - Empty**. Этот шаблон добавляет в папку Controllers пустой контроллер, который имеет один единственный метод Index. Данный шаблон не создает представлений
- **MVC 5 Controller with read/write actions**. Данный шаблон добавляет в проект контроллер, который содержит методы Index, Details, Create, Edit и Delete. Однако эти методы не содержат никакой логики работы с базой данных. И нам предлагается самим создать для них код и представления.
- **MVC 5 Controller with views, using Entity Framework**. Это уже более интересный шаблон, который создает контроллер с методами Index, Details, Create, Edit и Delete, а также все необходимые представления для этих действий и добавляет код для извлечения информации из базы данных.

Выберем последний пункт, то есть **MVC 5 Controller with views, using Entity Framework.**

После этого откроется окно добавления нового контроллера, в котором нам будет предложено установить некоторые настройки:

- **Controller name**: имя контроллера
- **Use async controller actions**: будут ли автоматические сгенерированные методы контроллера асинхронными. Установим данную опцию.
- **Model class**: класс модели. Выберем созданную ранее модель Book (либо какую-то другую имеющуюся модель)
- **Data context class**: класс контекста данных. Выберем контекст данных для выбранной модели.
- **Generate views**: надо ли генерировать представления к создаваемым действиям контроллера. При установке этой опции становятся доступными две следующие опции. Установим все эти опции.
- **Reference script libraries**: будут ли подключать представления библиотеки jquery и другие необходимые файлы javascript
- **Use a layout page**: будут ли генерируемые представления использовать мастер-страницу.



Установив все опции, нажмем кнопку Add, и в проект будет добавлен новый контроллер. Он будет выглядеть примерно следующим образом:

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

```
using System.Web.Mvc;
using WebApplication12;

namespace WebApplication12.Controllers.EnrollmentController
{
    public class EnrollmentsController : Controller
    {
        private ContosoUniversity2Entities db = new ContosoUniversity2Entities();

        // GET: Enrollments
        public ActionResult Index()
        {
            var enrollments = db.Enrollments.Include(e => e.Course).Include(e => e.Person);
            return View(enrollments.ToList());
        }


        // GET: Enrollments/Details/5
        public ActionResult Details(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Enrollment enrollment = db.Enrollments.Find(id);
            if (enrollment == null)
            {
                return HttpNotFound();
            }
            return View(enrollment);
        }



        // GET: Enrollments/Create
        public ActionResult Create()
        {
            ViewBag.CourseID = new SelectList(db.Courses, "CourseID", "Title");
            ViewBag.StudentID = new SelectList(db.People, "ID", "LastName");
            return View();
        }



        // POST: Enrollments/Create
        // Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства,
для которых следует установить привязку. Дополнительные
        // сведения см. в статье http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create([Bind(Include = "EnrollmentID,CourseID,StudentID,Grade")]
Enrollment enrollment)
        {
            if (ModelState.IsValid)
            {
                db.Enrollments.Add(enrollment);
                db.SaveChanges();
                return RedirectToAction("Index");
            }

            ViewBag.CourseID = new SelectList(db.Courses, "CourseID", "Title",
enrollment.CourseID);
            ViewBag.StudentID = new SelectList(db.People, "ID", "LastName",
enrollment.StudentID);
            return View(enrollment);
        }
```

```
        // GET: Enrollments/Edit/5
        public ActionResult Edit(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Enrollment enrollment = db.Enrollments.Find(id);
            if (enrollment == null)
            {
                return HttpNotFound();
            }
            ViewBag.CourseID = new SelectList(db.Courses, "CourseID", "Title",
enrollment.CourseID);
            ViewBag.StudentID = new SelectList(db.People, "ID", "LastName",
enrollment.StudentID);
            return View(enrollment);
        }



 // POST: Enrollments/Edit/5
        // Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства,
для которых следует установить привязку. Дополнительные
        // сведения см. в статье http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit([Bind(Include = "EnrollmentID,CourseID,StudentID,Grade")]
Enrollment enrollment)
        {
            if (ModelState.IsValid)
            {
                db.Entry(enrollment).State = EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            ViewBag.CourseID = new SelectList(db.Courses, "CourseID", "Title",
enrollment.CourseID);
            ViewBag.StudentID = new SelectList(db.People, "ID", "LastName",
enrollment.StudentID);
            return View(enrollment);
        }


        // GET: Enrollments/Delete/5
        public ActionResult Delete(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Enrollment enrollment = db.Enrollments.Find(id);
            if (enrollment == null)
            {
                return HttpNotFound();
            }
            return View(enrollment);
        }



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

```
        [ValidateAntiForgeryToken]
        public ActionResult DeleteConfirmed(int id)
        {
            Enrollment enrollment = db.Enrollments.Find(id);
            db.Enrollments.Remove(enrollment);
            db.SaveChanges();
            return RedirectToAction("Index");
        }



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

А в папке *Views/Enrollments* мы найдем все необходимые представления со всем необходимым кодом, который теперь нам не надо набирать вручную. И теперь мы можем запустить проект и перейти в адресной строке браузера к нашему контроллеру:

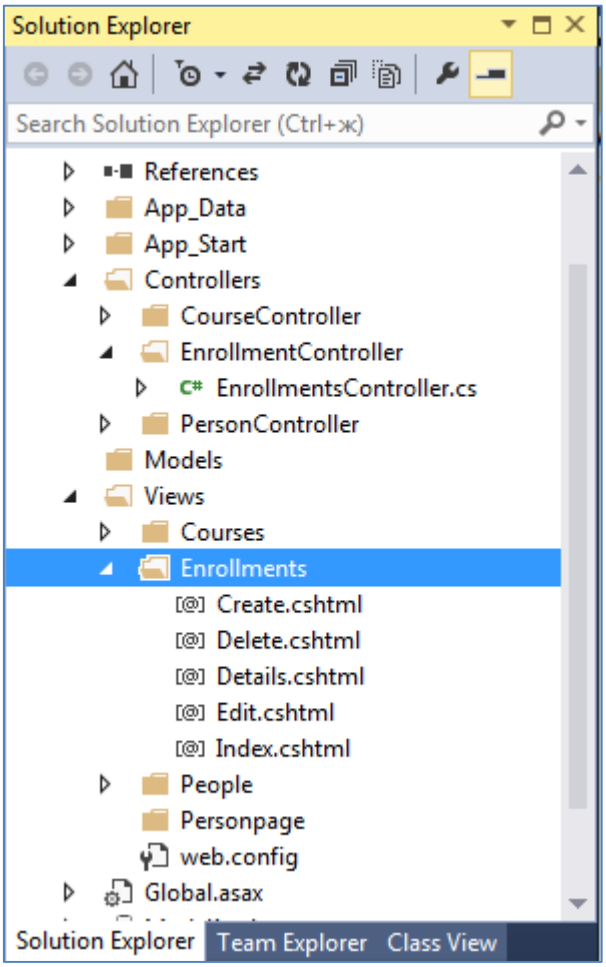
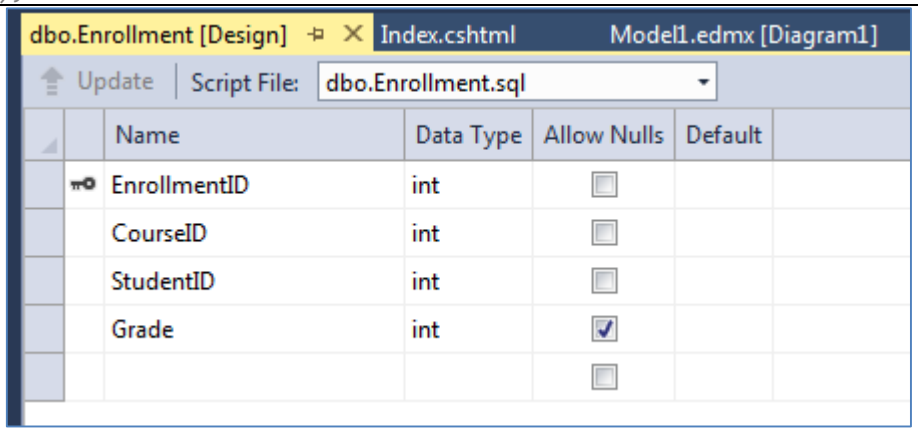


Таблица *Enrollment* состоит из следующих полей: EnrollmentID, CourseID, StudentID, Grade (рисунок).

```
CREATE TABLE [dbo].[Enrollment] (
    [EnrollmentID] INT IDENTITY (1, 1) NOT NULL,
    [CourseID]     INT NOT NULL,
    [StudentID]    INT NOT NULL,
    [Grade]        INT NULL,
    CONSTRAINT [PK_dbo.Enrollment] PRIMARY KEY CLUSTERED ([EnrollmentID] ASC),
    CONSTRAINT [FK_dbo.Enrollment_dbo.Course_CourseID] FOREIGN KEY ([CourseID]) REFERENCES
[dbo].[Course] ([CourseID]) ON DELETE CASCADE,
    CONSTRAINT [FK_dbo.Enrollment_dbo.Person_StudentID] FOREIGN KEY ([StudentID]) REFERENCES
[dbo].[Person] ([ID]) ON DELETE CASCADE
);
```



В методе `Index`()контроллера EnrollmentsController с помощью метода Include фреймворк подгружается для каждой записи таблицы Enrollment соответственно поля Title и LastName таблиц Course и Person:

```
public ActionResult Index()
        { var enrollments = db.Enrollments.Include(e => e.Course).Include(e => e.Person);
          return View(enrollments.ToList());
        }
```

Представление Index.cshtml, которое будет выводить все записи:

```
@model IEnumerable<WebApplication12.Enrollment>

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <p>
        @Html.ActionLink("Create New", "Create")
    </p>
    <table class="table">
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Grade)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Course.Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Person.LastName)
            </th>
```

```
            <th></th>
        </tr>

    @foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Grade)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Course.Title)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Person.LastName)
            </td>
            <td>
                @Html.ActionLink("Edit", "Edit", new { id=item.EnrollmentID }) |
                @Html.ActionLink("Details", "Details", new { id=item.EnrollmentID }) |
                @Html.ActionLink("Delete", "Delete", new { id=item.EnrollmentID })
            </td>
        </tr>
    }

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

Вывод записей для просмотра с помощью представления Index.cshtml в браузере представлен ниже на рисунке 11.

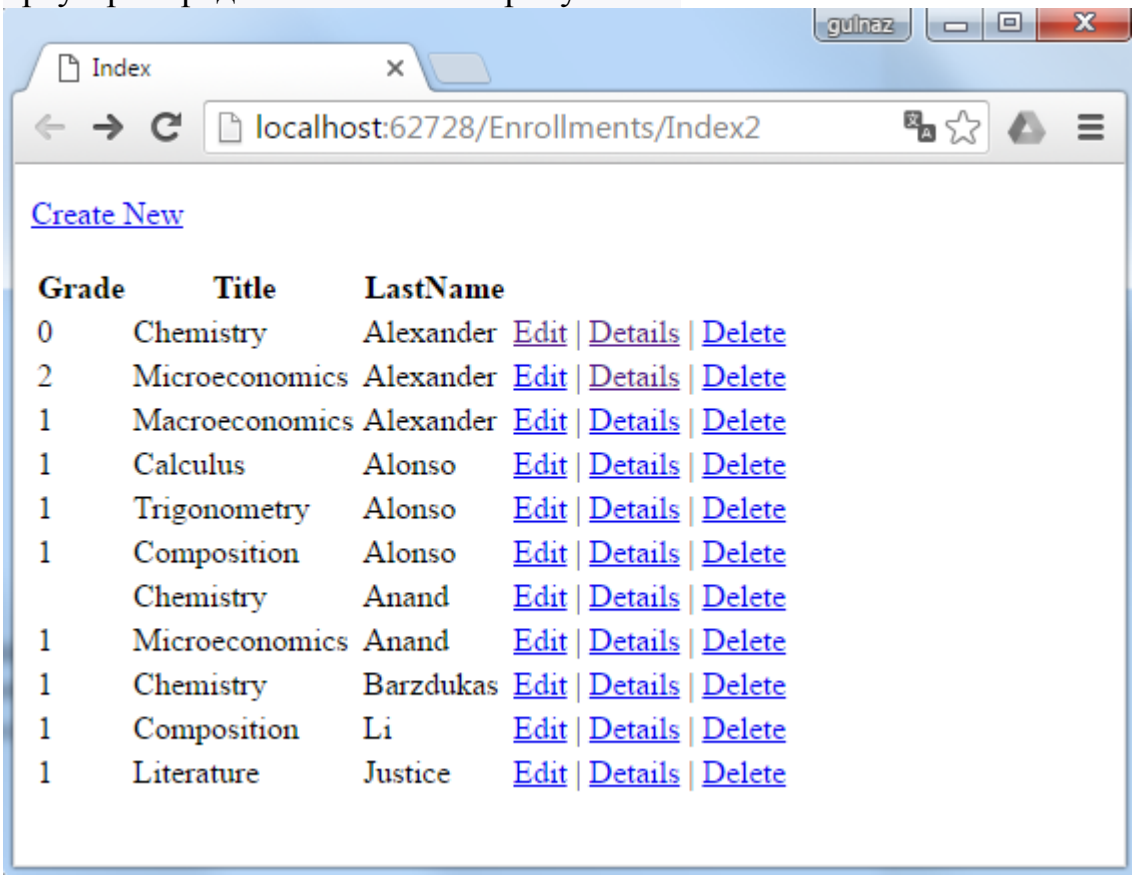


Рисунок 11-

Редактирование/ обнавление

Генерация исходящих адресов URL.

Хелпер ActionLink создает гиперссылку на действие контроллера. Если мы создаем ссылку на действие, определенное в том же контроллере, то можем просто указать имя действия, например, Edit.

Когда надо указать ссылку на действие из другого контроллера, то в хелпере ActionLink в качестве третьего аргумента имя контроллера. Например, ссылка на действие List контроллера Book будет создаваться так:

```
@Html.ActionLink("Список книг", "List", "Book")
```

Кроме того, если у нас в некотором методе контроллера определено несколько параметров, то перегруженная версия хелпера ActionLink позволяет передать параметр объекта (обычно анонимный тип). Среда выполнения принимает свойства объекта и использует их для создания значений маршрутизации (имена свойств становятся именами параметров маршрута, а значения свойств представляют значения параметра маршрута):

Фрагмент представления Index.cshtml.

```
<td>
@Html.ActionLink("Edit", "Edit", new { id=item.EnrollmentID })
@Html.ActionLink("Details", "Details", new { id=item.EnrollmentID })
@Html.ActionLink("Delete", "Delete", new { id=item.EnrollmentID })
</td>
```

Первая ссылка *Edit* (см. рисунок 11) вызывает действие *Edit* контроллера *EnrollmentsController* с параметром *id*.

Действие `Edit` контроллера EnrollmentsController

```
// GET
public ActionResult Edit(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Enrollment enrollment = db.Enrollments.Find(id);
            if (enrollment == null)
            {
                return HttpNotFound();
            }
            ViewBag.CourseID = new SelectList(db.Courses, "CourseID", "Title",
enrollment.CourseID);
            ViewBag.StudentID = new SelectList(db.People, "ID", "LastName",
enrollment.StudentID);
            return View(enrollment);
        }
```

Конструктор SelectList (IEnumerable, String, String, Object) - инициализирует новый экземпляр класса SelectList, используя указанные элементы для списка, поле значений данных, поле текстовых данных и выбранное значение.

Параметры	Тип	Описание
items	System.Collections.IEnumerable	Элементы
dataValueField	System.String	Поле значений данных.
dataTextField	System.String	Поле текстовых данных.
selectedValue	System.Object	Выбранное значение.

Пример конструктора *SelectList* действия *Edit* контроллера EnrollmentsController:

```
ViewBag.CourseID = new SelectList(db.Courses, "CourseID", "Title", enrollment.CourseID);
ViewBag.StudentID = new SelectList(db.People, "ID", "LastName", enrollment.StudentID);
return View(enrollment);
```

Представление Edit.cshtml

```
@model WebApplication12.Enrollment

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Edit</title>
</head>
<body>
    @using (Html.BeginForm())
    {
        @Html.AntiForgeryToken()

        <div class="form-horizontal">
            <h4>Enrollment</h4>
            <hr />
            @Html.ValidationSummary(true, "", new { @class = "text-danger" })
            @Html.HiddenFor(model => model.EnrollmentID)

            <div class="form-group">
                @Html.LabelFor(model => model.CourseID, "CourseID", htmlAttributes: new { @class
= "control-label col-md-2" })
                <div class="col-md-10">
                    @Html.DropDownList("CourseID", null, htmlAttributes: new { @class = "form-
control" })
                    @Html.ValidationMessageFor(model => model.CourseID, "", new { @class = "text-
danger" })
                </div>
            </div>

            <div class="form-group">
                @Html.LabelFor(model => model.StudentID, "StudentID", htmlAttributes: new {
@class = "control-label col-md-2" })
```

```
                <div class="col-md-10">
                    @Html.DropDownList("StudentID", null, htmlAttributes: new { @class = "form-
control" })
                    @Html.ValidationMessageFor(model => model.StudentID, "", new { @class =
"text-danger" })
                </div>
            </div>

            <div class="form-group">
                @Html.LabelFor(model => model.Grade, htmlAttributes: new { @class = "control-
label col-md-2" })
                <div class="col-md-10">
                    @Html.EditorFor(model => model.Grade, new { htmlAttributes = new { @class =
"form-control" } })
                    @Html.ValidationMessageFor(model => model.Grade, "", new { @class = "text-
danger" })
                </div>
            </div>

            <div class="form-group">
                <div class="col-md-offset-2 col-md-10">
                    <input type="submit" value="Save" class="btn btn-default" />
                </div>
            </div>
        </div>
    }

    <div>
        @Html.ActionLink("Back to List", "Index")
    </div>
</body>
</html>
```

Соответственно в папке *Enrollments* генерируется следующая страница представления *Edit.cshtml* (рисунок 22).

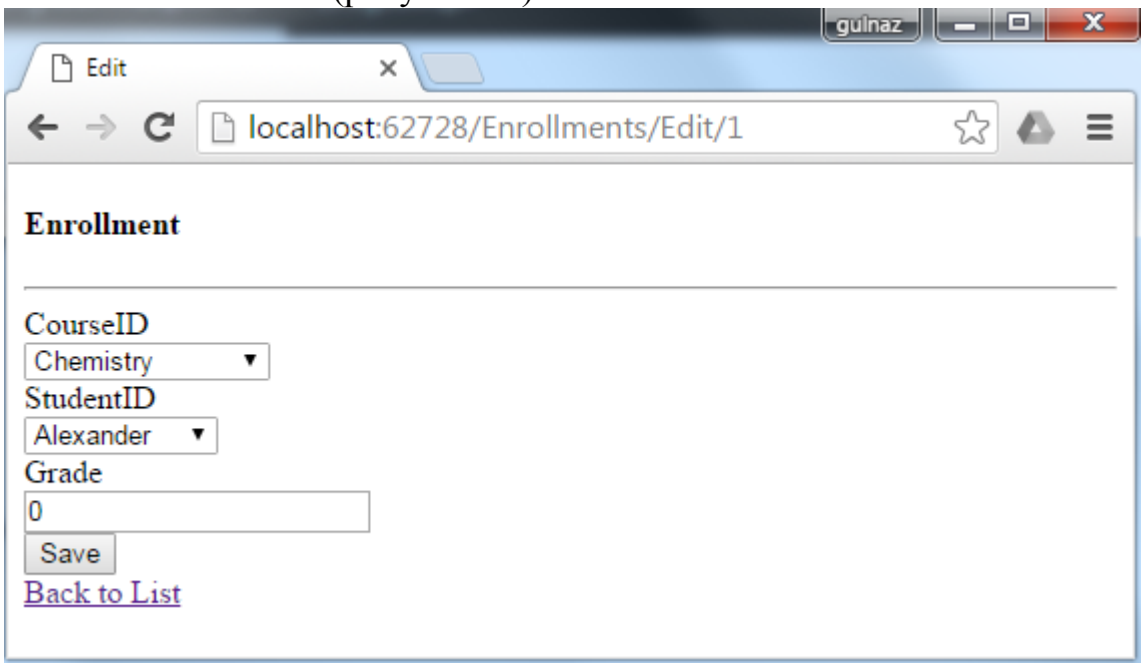


Рис 22

В представлении *Edit.cshtml* ключевое поле *EnrollmentID* передается как скрытое поле.

```
@Html.HiddenFor(model => model.EnrollmentID)
```

Метод DropDownList в представлений *Edit (файл Edit.cshtml)* - возвращает элемент select с единственным выбором, используя указанные вспомогательный метод HTML, имя поля формы, элементы списка и атрибуты HTML.

Синтаксис

```
        public static MvcHtmlString DropDownList(this HtmlHelper htmlHelper, string name,
IEnumerable<SelectListItem> selectList, object htmlAttributes);
```

Шаблонные хелперы

Кроме базовых html-хелперов, генерирующих определенные элементы разметки html, фреймворк

ASP.NET MVC также имеет **шаблонные (или шаблонизированные) хелперы**. В отличие от рассмотренных в прошлой главе html-хелперов они не генерируют определенный элемент html. Шаблонные хелперы смотрят на свойство модели и генерируют тот элемент html, который наиболее подходит данному свойству, исходя из его типа и метаданных.

В ASP.NET MVC имеются следующие шаблонные хелперы:

- **Display.** Создает элемент разметки для отображения значения указанного свойства модели: Html.Display("Name")
- **DisplayFor**

Строго типизированный аналог хелпера Display: Html.DisplayFor(m => m.Name)

- **Editor.** Создает элемент разметки для редактирования указанного свойства модели: Html.Editor("Name")
- **EditorFor.** Строго типизированный аналог хелпера Editor: Html.EditorFor(m => m.Name)
- **DisplayText.** Создает выражение для указанного свойства модели в виде простой строки: Html.DisplayText("Name")
- **DisplayTextFor.** Строго типизированный аналог хелпера DisplayText: Html.DisplayTextFor(m => m.Name)

Дополнительно смотри:
https://msdn.microsoft.com/ru-ru/library/system.web.mvc.html.displaynameextensions(v=vs.118).aspx

По нажатию кнопки «*Save*» по методу post передается перегруженному методу *Edit* следующие параметры: *EnrollmentID, CourseID, StudentID,Grade.*

```
public ActionResult Edit([Bind(Include = "EnrollmentID,CourseID,StudentID,Grade")] Enrollment
enrollment)
        {
            if (ModelState.IsValid)
            {
                db.Entry(enrollment).State = EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            ViewBag.CourseID = new SelectList(db.Courses, "CourseID", "Title",
enrollment.CourseID);
```

```
            ViewBag.StudentID = new SelectList(db.People, "ID", "LastName",
enrollment.StudentID);
            return View(enrollment);
        }
```

Задание

Добавить страницы по реализации следующих операций: удаления, добавления, обновление записей.