Разработка Web приложений с использованием ASP.NET MVC

СОДЕРЖАНИЕ

Лабораторная работа 1. Введение в разработку ASP.NET MVC Web Application

В этой работе вы создадите простой проект на основе контроллера и модели, изучите основные элементы веб-приложения. Более подробно язык программирования C# и шаблон MVC будет рассматриваться в следующих работах.

Упражнение 1. Создание веб-приложения ASP.NET MVC на основе контроллера

В этом упражнении вы создадите веб-приложение, содержащее только контроллер, который будет содержать метод действия передающий информацию в браузер для ее отображения.

1. Запустите Visual Studio и нажмите кнопку **Создание проекта**.
2. В диалоговом окне **New Project** (Создание проекта) выберите C#, Web и выберите тип проекта *Пустой ASP.NET Core* (как показано на рисунке 1.1).

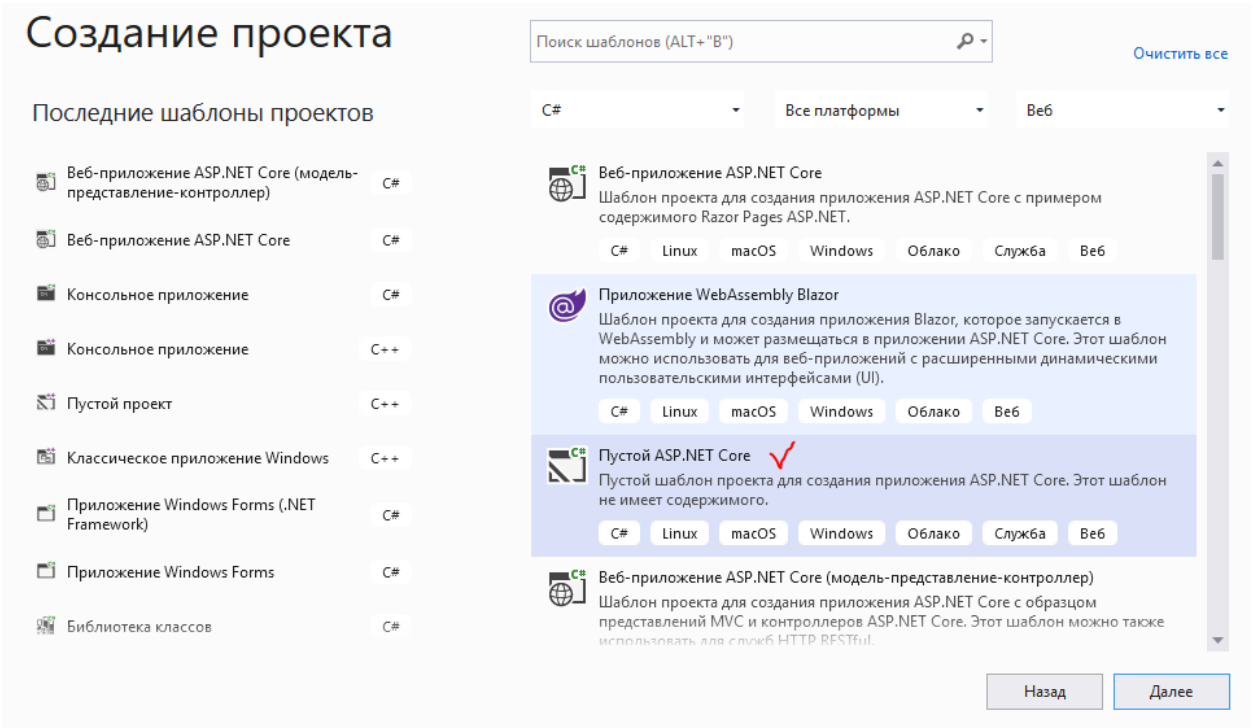


Рис.1.1 Создание проекта

3. Назовите новый проект **WebMVCR1,** оставьте расположение по умолчанию и нажмите кнопку *Далее*, чтобы продолжить.
4. В следующем окне **Дополнительные сведения** выберите *Целевую платформу* (укажите последнюю версию), оставьте настройки по

умолчанию (включенным флажок *Настроить для HTTPS* и выключенным флажок *Включить Docker*), нажмите кнопку *Создать*.

В результате будет создан проект с базовой структурой, но без папок и файлов, необходимых для создания MVC приложений.

5. В окне **Solution Explorer** (Обозреватель решений) изучите стандартную структуру проекта. Поскольку был выбран пустой шаблон проекта и приложение ничего не содержит, запускать приложение пока что бессмысленно.

Создание контролера

В архитектуре MVC входящие запросы обрабатываются контроллерами, которые являются классами (наследуются от Microsoft.AspNetCore.Mvc.Controller) и по соглашению размещаются в папке **Controllers** на корневом уровне проекта.

Для добавления пустого контроллера выполните следующие действия:

1. Добавьте в структуру проекта папку **Controllers**.
2. Добавьте папку **Controllers** добавьте новый контроллер используя команду *Добавить* в контекстном меню этой папки

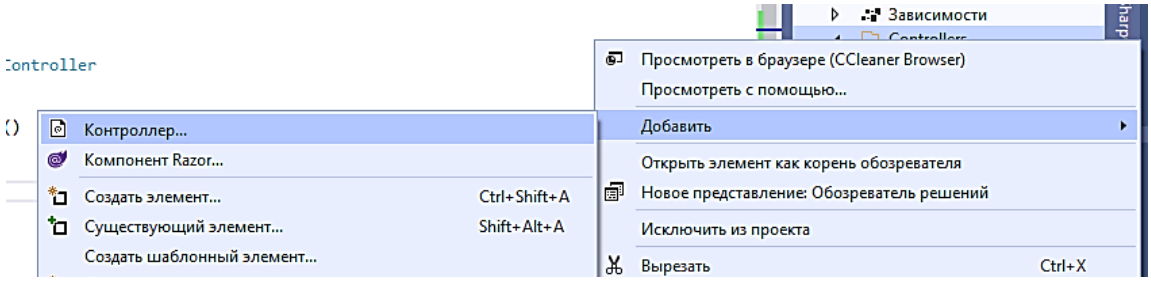


Рис. 1.2 Добавление контроллера

3. В открывшемся окне добавления шаблона выберите пустой контроллер (см. рис. 1.3) и добавьте его.

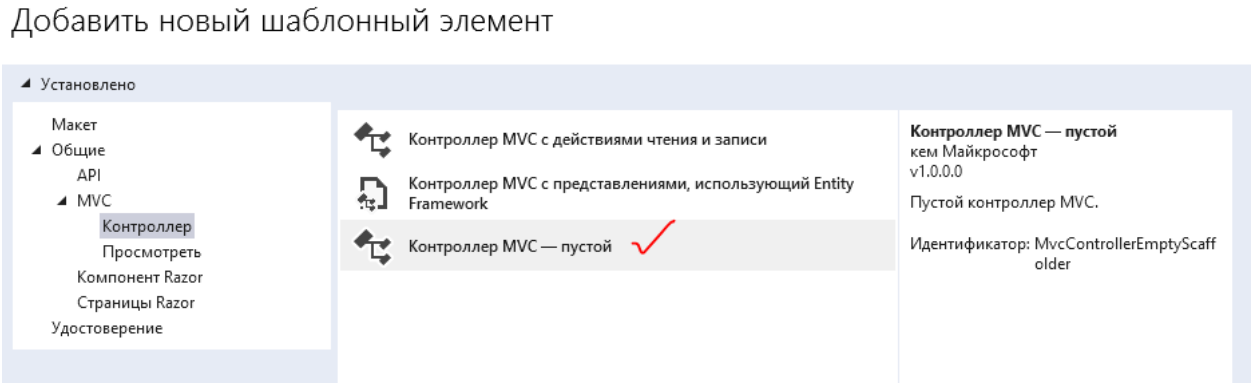


Рис. 1.3 Выбор шаблона контроллера

4. В окне Добавление нового элемента укажите имя контроллера *HomeController* (рис. 1.4) и нажмите кнопку добавления контроллера:

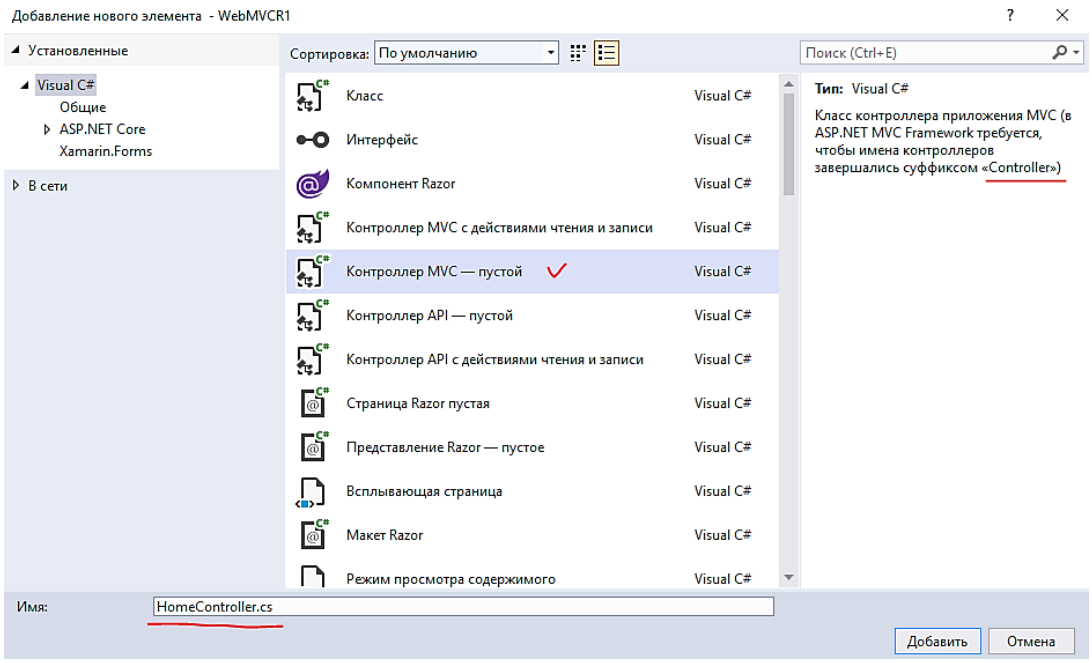


Рис. 1.4 Указание имени контроллера

Контроллер используется для определения и группировки набора действий, действие реализуется как метод контроллера, обрабатывающий запросы, то есть метод контроллера можно вызвать из Интернет через некоторый URL для выполнения действия. Запросы сопоставляются с действиями посредством маршрутизации.

5. Изучите код класса **HomeController**, обратите внимание на метод действия `Index()`.
6. Для тестирование контроллера закомментируйте метод `Index()` и добавьте другой вариант метода, который возвращает строку приветствия в зависимости от времени дня:

```
public string Index()
{
  int hour = DateTime.Now.Hour;
  string Greeting = hour < 12 ? "Доброе утро" : "Добрый
день";
  return Greeting;
}
```

Операция `x ? y : z` вычисляет значение `y`, если значение `x` равно `true`, и значение `z`, если значение `x` равно `false`.

Приложения ASP.NET Core (6.0 и выше), созданные на основе веб-шаблонов, содержат код запуска приложения в файле Program.cs. В этом файле настраиваются службы, необходимые приложению и определяется конвейер обработки запросов приложения определен как ряд компонентов ПО промежуточного слоя.

7. В файле Program.cs добавьте вызов

```
builder.Services.AddControllersWithViews();
var app = builder.Build();
```

Метод `AddControllersWithViews()` добавляет в коллекцию сервисов сервисы, которые необходимы для работы контроллеров MVC.

В ASP.NET Core MVC используется маршрутизация для сопоставления URL-адресов входящих запросов с действиями. Маршруты определяются в коде запуска (файл Program.cs). Они описывают то, как пути URL-адресов должны сопоставляться с действиями. С помощью маршрутов также формируются URL-адреса (для ссылок), отправляемые в ответах.

8. В файле Program.cs вызовите `MapControllers()` для регистрации маршрута контроллера (вызов метода `app.MapGet()` удалите):

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

Вышеуказанный вариант может заменить универсальный метод `MapDefaultControllerRoute()`.

9. Запустите проект, выбрав в меню **Debug** команду **Start Debugging**. Браузер должен отобразить результат метода действия `Index()` – строку приветствия.
10. Для тестирования маршрутизации введите в адресной строке браузера ссылки, направленные на метод `Index` контроллера **HomeController** и после каждой обновите страницу:

```
localhost:ваш_port/Home
localhost:ваш_port//Home/Index
```

11. Проверьте, что в обоих случаях браузер отображает ту же самую строку приветствия.

Это объясняется тем, что, когда браузер запрашивает адрес сайта, он получает выходные данные метода `Index` контроллера **HomeController**.

Упражнение 2. Передача параметров в контроллер

В этом упражнении вы узнаете, как передавать параметры контроллеру с помощью URL-адреса и строки запроса.

При отправке GET-запроса значения передаются через строку запроса. Стандартный get-запрос принимает примерно следующую форму:

```
/метод?параметр1=значение1&параметр2=значение2
```

1. Откройте класс **HomeController**.
2. Измените метод `Index()`, добавив передаваемый ему строковый параметр, а в теле метода присвойте строковой переменной сообщение с передаваемым параметром:

```
public string Index(string hel)
{
    int hour = DateTime.Now.Hour;
```

```
            string Greeting = hour < 12 ? "Доброе утро" : "Добрый
день" + ", " + hel;
            return Greeting;
        }
```

3. Запустите приложение. Должна загрузиться страница `Home`.
4. Для проверки работы метода действия, принимающий параметр измените URL на следующий адрес (`port` – номер вашего порта) и учтите, что теперь имя контроллера – `MyController`:

```
localhost:port/My/Index?hel=Иван
```

Обновите страницу, должна появиться строка с переданным параметром.

Упражнение 3. Реализация взаимодействия контроллера и модели

Контроллер, реализованный в предыдущих упражнениях помимо обработки входных данных и вывода в браузер занимался и несвойственным ему действием – рассчитывал значение переменной, т.е. обрабатывал некую логику. В этом упражнении вы добавите в созданное веб-приложение, содержащее контроллер, модель, в которую будет перенесена логика приложения.

Реализация модели

Модель содержит данные и алгоритмы, которые будут определять логику приложения.

1. Добавьте в папку **Models** класс **ModelClass**.
2. Добавьте в класс статический метод и добавьте в его код расчета приветствия. В результате должно получиться следующее:

```
public class ModelClass
{
   public static string ModelHello()
   {
        int hour = DateTime.Now.Hour;
        string Greeting = hour < 12 ? "Доброе утро" : "Добрый
день";
        return Greeting;
    }
}
```

3. В методе контроллера `Index(string hel)` присвойте переменной строкового типа результат вызова метода модели `ModelHello()`:

```
public string Index(string hel)
{
 string Greeting = ModelClass.ModelHello() + ", " + hel;
   return Greeting;
}
```

4. Запустите приложение. Протестируйте работу метода действия, принимающий параметр, функциональность не должна измениться.

Лабораторная работа 3. Создание веб-приложения ASP.NET MVC

В этой работе вы добавите в проект WebMVCR1 представление, модель и реализуете взаимодействие с ними в рамках шаблона MVC.

Упражнение 1. Реализация взаимодействия контроллера и представления

Результатом всех предыдущих упражнений является текстовая строка. Для того чтобы создать на запрос браузера HTML ответ, необходимо создать представление.

Настройка контроллера

1. Снимите комментарий для метода `Index()`, возвращающий `IActionResult` и закомментируйте метод, возвращающий строку.
2. Измените тип возвращаемого параметра метода `Index()` на `ViewResult`:

```
public ViewResult Index()
{
    return View();
}
```

Работа контроллера состоит в том, чтобы создать некоторые данные и передать их представлению, которое отвечает за то, чтобы представить их в виде HTML.

Одним из способов передачи данных от контроллера к представлению является использование объекта **ViewBag**, который является членом базового класса **Controller**. **ViewBag** – это динамический объект, которому можно присвоить произвольные свойства, что делает эти значения доступными для любого представления, которое будет дальше их использовать.

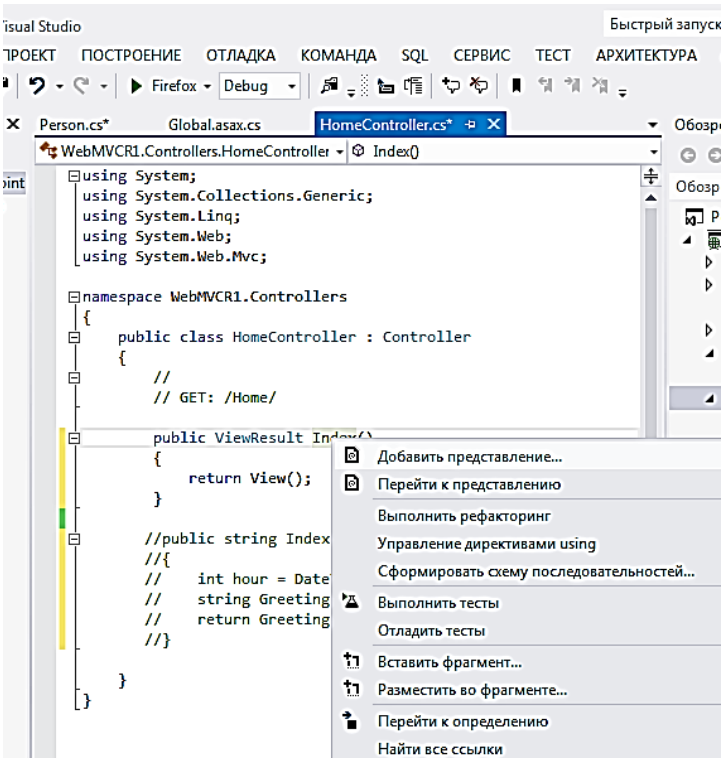
Другим способом является использование свойства **ViewData**, который предоставляет экземпляр класса **ViewDataDictionary**. Для передачи данных в представление нужно сначала добавить его в свойство контроллера **ViewData** в методе действия, который используется для отображения представления.

3. В новом методе реализуйте функциональность приветствия, результат проверки времени дня присвойте свойству динамического объекта **ViewBag** (имя свойства может быть произвольным, так как свойство не существует до того момента, пока ему не будет присвоено значение), а свойству присвойте **ViewData** произвольное значение:

```
public ViewResult Index()
{
    int hour = DateTime.Now.Hour;
    ViewBag.Greeting = hour < 12 ? "Доброе утро" : "Добрый день";
    ViewData["Mes"] = "хорошего настроения";
    return View();
}
```

Создание представления

1. Для добавления представления для метода действия `Index` в контекстном меню этого метода (см. рис. 3.1) выберите команду **Add View** (Добавить представление):



Список команд открывается после щелчка провой кнопки мыши по методу Index

Рис. 3.1 Команды добавления представления

2. В окне добавления нового шаблонного элемента укажите *Представление Razor – пустое* и далее – **Добавить**:

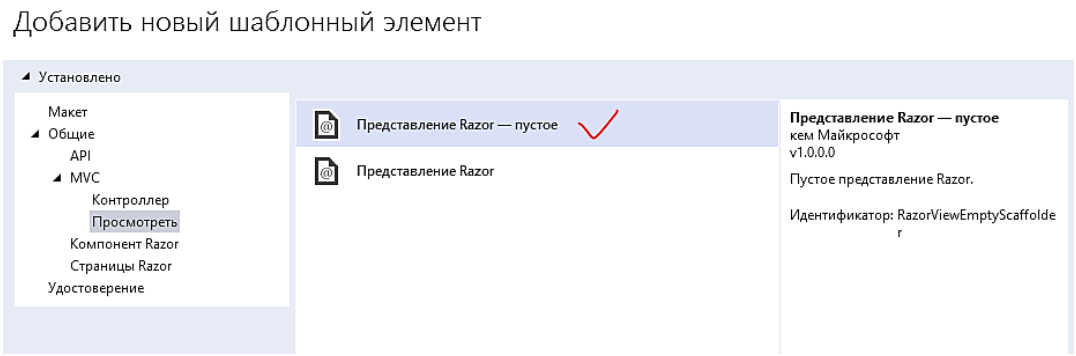


Рис. 3.2 Добавление шаблонного элемента

3. В окне *Добавление нового элемента* оставьте имя – **Index** и далее – **Добавить**.

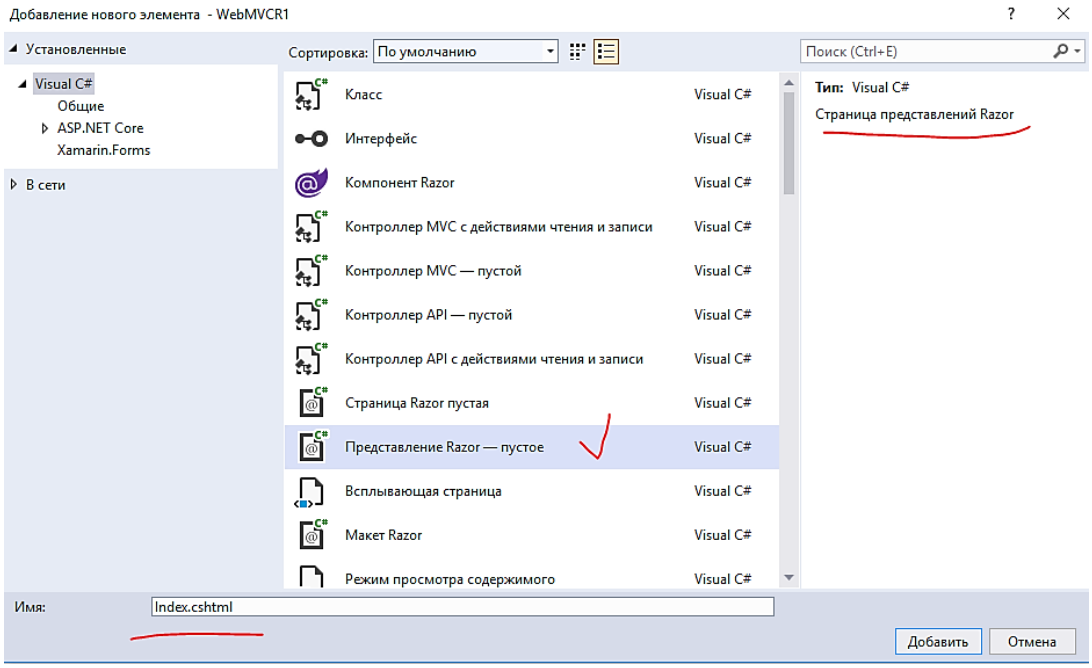


Рис. 3.3 Добавление нового элемента

4. Проверьте, что в структуре проекта автоматически появилась новая папка **Views** с вложенной папкой **Home** и файлом Index.cshtml.
5. После добавления представления откроется файл Index.cshtml для редактирования. Добавьте тег `<div>` и текст для вывода:

```
<div>
    @ViewBag.Greeting, спасибо, что зашли и @ViewData["Mes"]
</div>
```

6. Запустите приложение. Обработка выражения обозначает вставку значения, которое динамически присвоилось `ViewBag.Greeting` и `ViewData["Mes"]` метода действия в представление, в итоге должна открыться страница с указанным текстом.

Упражнение 2. Реализация взаимодействия модели, представления и контроллера в шаблоне MVC

В этом упражнении вы добавите основные элементы шаблона MVC – модель и представление, а также настроите контроллер для взаимодействия с ними в веб-приложение ASP.NET MVC.

Реализация модели

Данные, с которыми будет работать клиент, должны быть представлены моделями. Для этого упражнения модель будет определяться классом, описывающим персону с двумя свойствами – имя и фамилия.

1. Добавьте в папку **Models** класс **Person**, который будет реализовывать модель данных о человеке.
2. Добавьте в класс поля, описывающие имя и фамилию персоны, а также переопределите метод `ToString()`:

```
    public class Person
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }

        public override string ToString() => FirstName + " " +
LastName;
    }
```

3. Постройте приложение.

Связь главной формы и формы ввода данных

Ввод имени персоны будет выполняться на другой форме, на которую можно будет перейти из главной формы.

1. Для реализации возможности перехода на форму ввода (она будет создана позднее) с помощью метода `ActionLink` (принимающего два параметра: первый – текст для отображения в ссылке, например, "Введите имя", а второй – выполняемое действие `InputData`, когда пользователь нажимает на ссылку) добавьте ссылку на нее из представления `Index.cshtml`:

```
<div>
    @ViewBag.Greeting, спасибо, что зашли и @ViewData["Mes"]
    <p>@Html.ActionLink("Введите свои данные", "InputData")</p>
</div>
```

2. Запустите приложение. Проверьте отображение ссылки.

Настройка контроллера

3. В файл контроллера HomeController добавьте метод действия `InputData`, которому будет соответствовать адрес формы:

```
public ViewResult InputData()
{
    return View();
}
```

Добавление строго типизированного представления

Для метода действия `InputData` необходимо добавить строго типизированное представление, которое будет обрабатывать определенный тип (модель).

1. Для добавления представления для метода действия `InputData` в контекстном меню этого метода выберите команду **Add View** (Добавить представление).
2. В окне (см. рис. 3.2) добавления нового шаблонного элемента укажите *Представление Razor* и далее – **Добавить.**
3. В окне добавления представления (см. рис. 3.4):
    a. оставьте имя – **InputData**, в списке *Шаблон* укажите наиболее подходящий шаблон для создания персоны – **Create**.

b. в списке *Класс модели* выберите класс **Person (WebMVCR1.Models)**

c. В разделе *Параметры* снимите все флажки

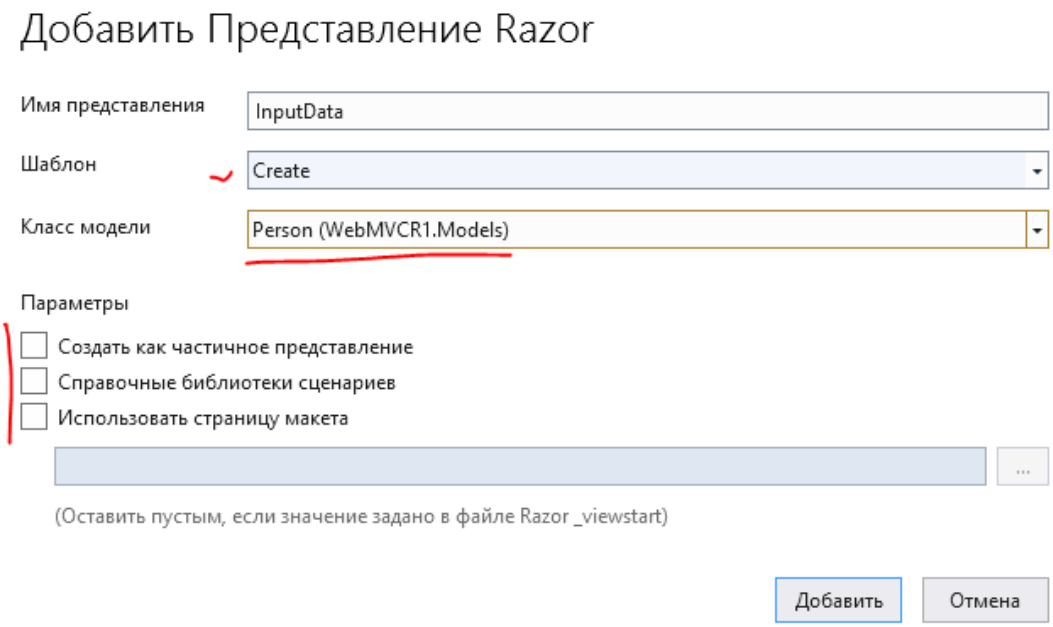


Рис. 3.4 Добавление строго типизированного представления

После добавления представления откроется файл InputData.cshtml для редактирования.

4. Изучите содержимое файла. Обратите внимание на первую строку

```
@model WebMVCR1.Models.Person
```

Razor-оператор `@model` определяет тип модели, с которым связано представление.

Далее представлена разметка страницы и реализация тела на основе tag-хелперов. Вспомогательные функции тегов позволяют серверному коду участвовать в создании и отображении HTML-элементов в файлах Razor – большинство встроенных вспомогательных функций тегов работают со стандартными HTML-элементами и предоставляют для них атрибуты на стороне сервера, элемент <input> содержит атрибут asp-for – этот атрибут извлекает имя свойства указанной модели в готовый для просмотра HTML-код.

Применение tag-хелперов, а также и html-хелперов будет рассмотрено далее в руководстве, в этом упражнении вы реализуете HTML-разметку самостоятельно.

5. Удалите сгенерированный код разметки в теге `<body>` и оставьте только заголовок:

```
<body>
    <h4>Person</h4>

</body>
```

Создание формы для ввода данных

1. В файле InputData.cshtml внутри тега `<body>` добавьте текст для ввода данных с помощью запроса `POST` (`<form method="post")` и реализуйте передачу с запросом `post` данных о модели с помощью полей ввода `<input type="text" name="FirstName"/>` и `<input type="text" name="LastName"/>` (их свойства `name` соответствуют именам свойств модели **Person** и при нажатии кнопки и отправки запроса передаются значения этих полей):

```
<h4>Person</h4>
<div>
    <form method="post">
          <table>
            <tr>
                <td><p>Введите имя:</p></td>
                <td><input type="text" name="FirstName" /> </td>
            </tr>
            <tr>
                <td><p>Введите фамилию:</p></td>
                <td><input type="text" name="LastName" /> </td>
            </tr>
            <tr><td><input type="submit" value="Отправить" />
</td><td></td></tr>
          </table>
    </form>
</div>
```

2. Запустите приложение. Проверьте работу ссылки – должна открыться страница с предложением ввода данных.

Обработка данных на форме

Для получения и обработки отправленных данных формы применяются два метода действия, которые вызываются одним и тем же URL, но MVC гарантирует, что будет вызван соответствующий метод в зависимости от того, какой был запрос – GET или POST:

- метод, который отвечает на HTTP GET запросы*:* `GET` запрос является тем, с чем браузер имеет дело после каждого клика по ссылке. Этот вариант действий будет отвечать за отображение начальной пустой формы, когда кто-то первый раз посетит `/Home/InputData`.
- метод, который отвечает на HTTP POST запросы. Запросы по умолчанию, формы отправляются браузером как `POST` запросы. Этот вариант действий будет отвечать за получение отправленных данных и решать, что с ними делать.

1. В класс `HomeController` импортируйте пространство имен

```
using WebMVCR1.Models;
```

2. Добавьте атрибут `HttpGet` для существующего метода действия `InputData`, это означает, что данный метод должен использоваться только для GET запросов:

```
[HttpGet]
public ViewResult InputData()
{
    return View();
}
```

3. Добавьте перегруженную версию метода действия `InputData`, который принимает параметр `Person` и применяет атрибут `HttpPost`, это означает, что новый метод будет иметь дело с POST запросами:

```
[HttpPost]
public ViewResult InputData(Person p)
{
    return View("Hello", p);
}
```

Данный вариант перегруженного метода действий `InputData` показывает, как можно указать MVC обрабатывать конкретное представление, а не представление по умолчанию, в ответ на запрос. Этот вызов метода `View` говорит MVC найти и обработать представление, которое называется `Hello` и передать представлению объект `p`.

4. Создайте пустое представление `Hello` – для этого щелкните правой кнопкой мыши внутри одного из методов `HomeController` и выберите команду **Add View** (**Добавить представление**):
    - Выберите именно пустое представление (подключение модели выполним самостоятельно)
    - Имя представления – `Hello`

После добавления представления откроется файл Hello.cshtml для редактирования.

5. Добавьте в начало страницы ссылку на используемую модель:

```
@model WebMVCR1.Models.Person
```

6. Вставьте текст для вывода данных свойства модели – имени, а также ссылку на главную страницу:

```
<div>
    <h1>Здравствуйте, @Model.ToString()!</h1>
    <p>
        @Html.ActionLink("На главную", "Index")
    </p>
</div>
```

7. Запустите и протестируйте работу приложения – после ввода данных должна открыться страница приветствия.

Упражнение 3. Реализация модели-репозитория

В этом упражнении вы создадите модель – репозиторий для хранения коллекции персон, настроите контроллер и создадите представление для вывода списка персон.

Добавление модели - репозитория

1. Добавьте в папку **Models** класс **PersonRepository**, который будет реализовывать модель коллекции.
2. Добавьте в класс поле, описывающие коллекцию – список персон:

```
public class PersonRepository
{
  private List<Person> persons = new List<Person>();
```

3. Далее в классе укажите два свойства только для чтения – первое должно возвращать количество персон, второе – саму коллекцию:

```
public int NumberOfPerson
{
    get
    {
        return persons.Count();
    }
}

public IEnumerable<Person> GetAllResponses
{
    get
    {
        return persons;
    }
}
```

4. Для возможности добавления персон в коллекцию реализуйте в классе советующий метод:

```
public void AddResponse(Person pers)
{
    persons.Add(pers);
}
```

5. Постройте приложение.

Настройка контроллера

1. В поле класса контроллера **HomeController** добавьте код создания статического объекта репозитория:

```
private static PersonRepository db = new PersonRepository();
```

2. В перегруженную версию метода действия `InputData`, который принимает параметр `Person` и применяет атрибут `HttpPost` укажите код добавления объекта `Person` в коллекцию `db` (выделено жирным шрифтом):

```
[HttpPost]
public ViewResult InputData(Person p)
{
    db.AddResponse(p);
    return View("Hello", p);
}
```

3. Добавьте новый метод действия, реализующий получение коллекции и информации о ее размере в динамический объект `ViewBag`:

```
public ViewResult OutputData()
{
    ViewBag.Pers = db.GetAllResponses;
    ViewBag.Count = db.NumberOfPerson;
    return View("ListPerson");
}
```

В данном случае предполагается вывод списка персон на другую страницу.

Добавление строго типизированного представления для отображения коллекции

1. Создайте пустое представление `ListPerson` – для этого щелкните правой кнопкой мыши внутри метода `OutputData()` и выберите команду **Add View (Добавить представление)**:
   - Выберите именно пустое представление (подключение модели выполним самостоятельно)
   - Имя представления – `ListPerson`,

После добавления представления откроется файл ListPerson.cshtml для редактирования.

2. Добавьте директиву для подключения используемой модели, обратите внимание, что это `PersonRepository`:

```
@model WebMVCR1.Models.PersonRepository
```

3. Внутри тега `<div>` добавьте текст для вывода данных свойства модели – имени, а также ссылку на главную страницу:

```
<div>
 <h3>Список участников</h3>
    <p>Персон всего @ViewBag.Count</p>

            <ul>
                @foreach (var c in ViewBag.pers)
                {
                    <li>@c.ToString()</li>
              }
            </ul>
    <p>@Html.ActionLink("На главную", "Index")</p>
</div>
```

4. Для возможности просмотра списка персон (перехода на страницу после заполнения коллекции) добавьте в файл Index.cshtml соответствующую ссылку:

```
<p>
  @Html.ActionLink("Посмотрите, кто уже здесь", "OutputData")
 </p>
```

5. Постройте приложение. Запустите и протестируйте его работу. Введите данные о нескольких человек и просмотрите их список, перейдя по ссылке, добавленной в предыдущем пункте.