**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 ActionResult 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):

// В контекстном меню метода – это значит, нужно навести мышку на public ViewResult Index() (прямо в коде!!!), нажать ПКМ и выбрать все что показано на рисунке.



*Рис. 3.1 Команды добавления представления*

2. В окне добавления представления оставьте имя – **Index**, в списке шаблона укажите **Empty** (без модели) и снимите флажок *Use a layout page* (Использовать страницу макета):



*Рис. 3.2 Добавление представления*

// Получилось.

3. После добавления представления откроется файл Index.cshtml для редактирования. Внутри тега <div> добавьте текст для вывода:

<div>

@ViewBag.Greeting, спасибо, что зашли и @ViewData["Mes"]

</div>

4. 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.

// Результат:

Добрый день, спасибо, что зашли и хорошего настроения

***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()

{

string s = FirstName + " " + LastName;

return s;

}

}

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.

// Результат

Добрый день, спасибо, что зашли и хорошего настроения

[Введите свои данные](https://localhost:44360/My/InputData)

**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).

// В контекстном меню метода – это значит, нужно навести мышку на public ViewResult InputData() (прямо в коде!!!), нажать ПКМ и выбрать все что показано на рисунке.

2. В окне добавления представления (см. рис. 3.3) оставьте имя – **InputData**, в списке *Шаблон* укажите **Empty**, в списке *Класс модели* выберите класс **Person (WebMVCR1.Models)** и снимите флажок *Use a layout page* (Использовать страницу макета):



*Рис. 3.3 Добавление строго типизированного представления*

После добавления представления откроется файл InputData.cshtml для редактирования.

3. Обратите внимание на первую строку

@model WebMVCR1.Models.Person

Razor-оператор @model определяет тип модели, с которым связано представление. В этом упражнении будет применяться обычная разметка для ввода данных. Позднее будут рассмотрены специальные элементы (html-хелперы) для ввода информации.

// +

**Создание формы для ввода данных**

1. В файле InputData.cshtml внутри тега <div> 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):

<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.

// Результат. Появились формы для ввода имени и фамилии

Введите имя:



Введите фамилию:



**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 данные и решать, что с ними делать.

1. В класс HomeController импортируйте пространство имен

using WebMVCR1.Models;

// оно уже было автоматически импортировано

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 представление Hello – для этого щелкните правой кнопкой мыши внутри одного из методов HomeController //Наверное, имеется ввиду **любой** метод в этом классе // и выберите команду **Add View** (Добавить представление) (см. рис. 3.4):

// по аналогии с примерами создания представления выше

a. Установите имя представления – Hello.

b. В списке **Шаблон** укажите Empty.

c. В списке **Класс модели** выберите класс Person (WebMVCR1.Models).

d. Снимите флажок *Use a layout page* (Использовать страницу макета).



*Рис. 3.4 Добавление представления для ответа*

После добавления представления откроется файл Hello.cshtml для редактирования.

5. Внутри тега <div> добавьте текст для вывода данных свойства модели – имени и ссылку на главную страницу:

<div>

<h1>Здравствуйте, @Model.ToString()!</h1>

<p>

@Html.ActionLink("На главную", "Index")

</p>

</div>

6. Запустите и протестируйте работу приложения.

// Условно выполнено. Есть вопросы.

***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 представление ListPerson – для этого щелкните правой кнопкой мыши внутри метода OutputData() и выберите команду **Add View** (Добавить представление) (см. рис. 3.5):

a. Установите имя представления на ListPerson,

b. В списке **Шаблон** укажите Empty.

c. В списке **Класс модели** выберите класс PersonRepository (WebMVCR1.Models)

d. Снимите флажок *Use a layout page* (Использовать страницу макета).



*Рис. 3.5 Добавление представления для вывода данных*

После добавления представления откроется файл ListPerson.cshtml для редактирования.

2. Внутри тега <div> добавьте текст для вывода данных свойства модели – имени и 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>

3. 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>

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

// Работает. Работают и переходы. При повторном запуске сохранются предыдущие персонажи