**Лабораторная работа № 3**

**Работа с Entity Framework, Razor и Bootstrap. Добавление записей**

**Цель работы**

Освоить механизмы работы с Entity Framework для добавления данных, с HTML-помощниками и методами валидации модели.

**После выполнения ЛР студенты должны уметь**

1. Создавать представление с формой для добавления объектов.
2. Использовать HTML-помощников для создания элементов различных типов.
3. Создавать метод для обработки и проверки состояния модели.
4. Добавлять атрибуты для валидации свойств модели.

**Ход работы**

1. Открыть предыдущий проект ASP .NET MVC в среде Visual Studio.
2. Добавить метод для получения представления для добавления объекта
3. Добавить метод для обработки отправляемой пользователем формы
4. Создать представление для добавления нового объекта
5. Добавить HTML помощников разных типов (TextBox, CheckBox, RadioButton, DropDownList)
6. Добавить атрибуты для валидации модели и/или проверку модели в коде метода обработки

**Порядок выполнения работы**

* 1. **Базовые операции для работы с данными**

В рамках лабораторной работы вы продолжаете использовать предметную область, которую выбрали во время лабораторной №2.

***ВНИМАНИЕ!*** *В качестве источника данных настоятельно рекомендуется использовать базу данных, созданную в рамках выполнения ИДЗ по дисциплине «Технологии проектирования баз данных».*

В данных методических материалах рассматривается предметная область о проведении опросов, и используются ранее созданные сущности: [Участник опроса]–[Ответ участника на вопрос]–[Вопрос] (рисунок 1).

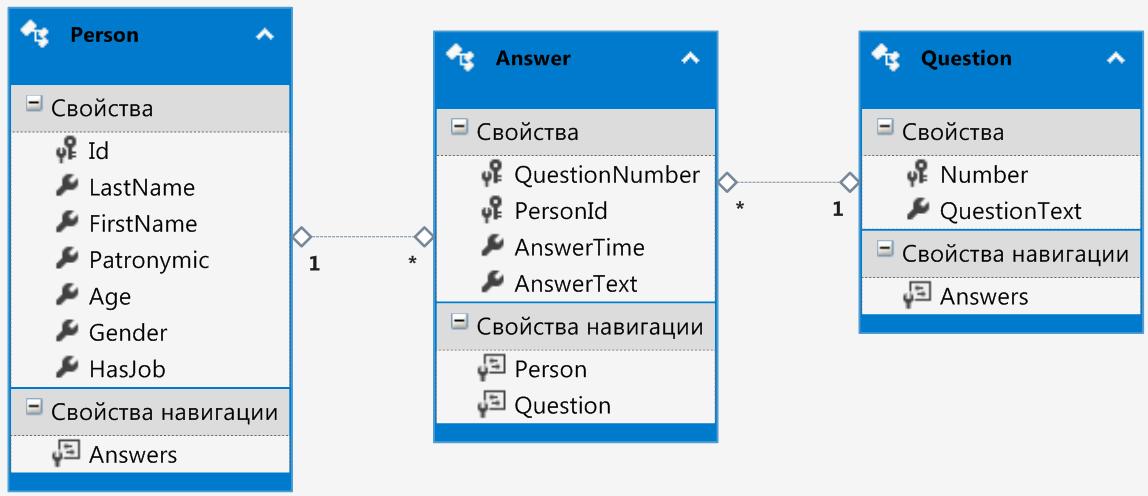


Рисунок 1 – Диаграмма используемой модели

С данной лабораторной начинается рассмотрение основных операций для работы с данными – операций CRUD:

* Create – создание;
* Read – чтение;
* Update – обновление;
* Delete – удаление.

Для предыдущей лабораторной вы так или иначе вручную заполняли таблицы своей базы данных, и применяли операции для чтения этих данных в приложении. В этой лабораторной работе будет рассмотрен способ добавления данных с использованием разрабатываемого вами веб-приложения.

В проекте используется Entity Framework, который, как упоминалось ранее, позволяет оперировать данными из базы в приложении. Обычно при разработке приложения создают уровень абстракции между контроллером и уровнем доступа к данным – например, применяют паттерн разработки «Репозиторий» и др. (см. ссылку №1 в списке источников) В сущности, репозиторий должен давать возможность работать с коллекциями объектов независимо от кода остального приложения. В целом Entity Framework уже предоставляет такой абстрактный контекст данных и коллекции, представленные классом DbSet (рисунок 2). Чтобы упростить лабораторные работы и сконцентрироваться на работе с самой технологией Entity Framework, отдельные репозитории не будут вводиться в проект.

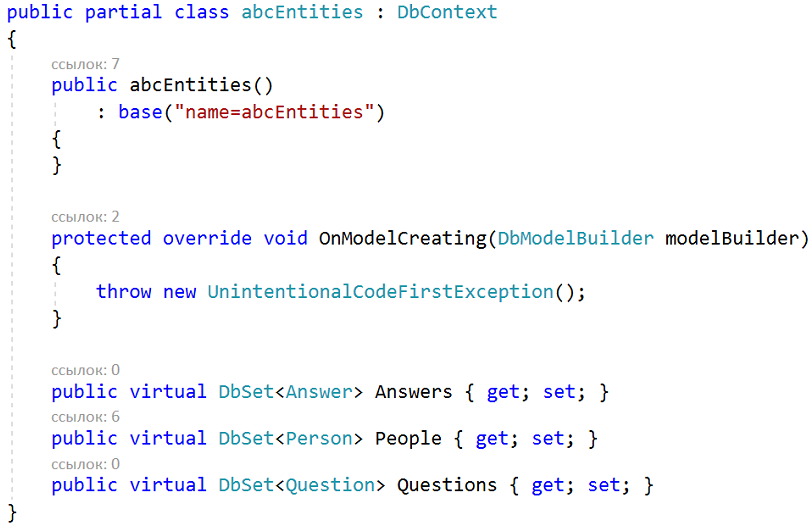


Рисунок 2 – Контекст данных, созданный при подходе Database First

Рассмотрим подробнее класс DbSet, методами которого предстоит воспользоваться в этой и следующих лабораторных работах. Если открыть ваш класс контекста данных (как на рисунке 2), навести курсор на класс DbSet и нажать правую кнопку мыши, то в выпадающем контекстом меню можно увидеть пункт «Перейти к определению» (рисунок 3).

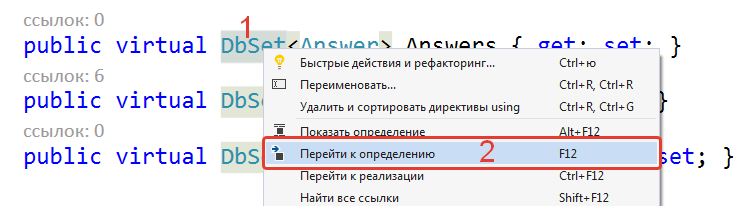


Рисунок 3 – Контекстное меню для выбранного участка кода

Если вы выберете данный пункт, то в новой вкладке откроется файл метаданных класса DbSet<TEntity>, где вы увидите методы этого класса и их описание. На рисунке 4 представлен фрагмент этого файла, где приведено описание методов Add и AddRange для добавления сущностей. Вы можете обращаться и к другим классам и объектам, чтобы переходить к их определению. Определение класса даст вам представление о его содержимом, а определение объекта покажет место его объявления.

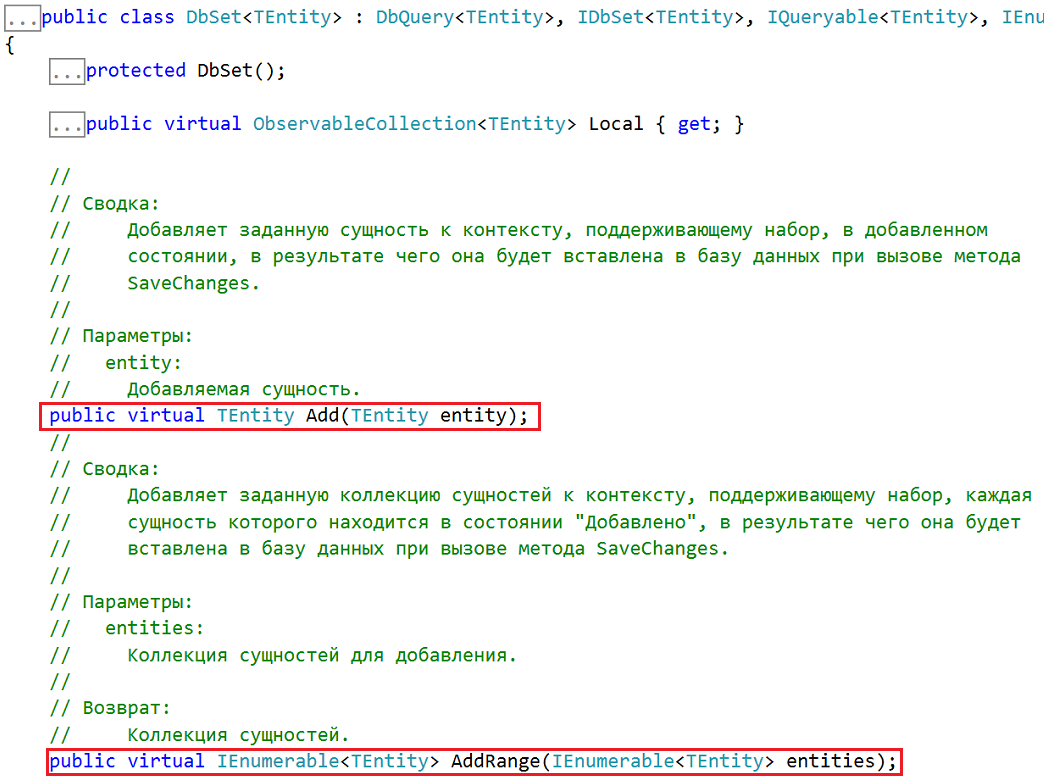


Рисунок 4 – Фрагмент файла метаданных класса DbSet<TEntity>

* 1. **Создание метода и представления для добавления записи**

Предполагается, что вы продолжаете работать с выбранной для прошлой лабораторной предметной областью, так что к этому моменту у вас в проект добавлена модель данных, и создан контроллер, где реализованы методы для получения списка объектов и одного объекта по его идентификатору. Откройте ваш проект ASP .NET MVC в среде Visual Studio. Для удобства настройки переходов между страницами можете продолжать работать с уже созданным вами ранее контроллером.

Как реализовать добавление новой записи в таблицу базы данных? Для начала определитесь, с какой таблицей вы собираетесь работать. В рамках примера для лабораторной был реализован метод для получения списка участников опроса, и создано соответствующее представление, так что продолжим работать с этой таблицей и подготовим все необходимое для добавления записей в таблицу Person.

Для получения списка объектов или одного объекта по идентификатору нам было достаточно создать по одному методу в контроллере и соответствующему представлению. При реализации добавления (и редактирования) записи процесс несколько меняется – вы не только запрашиваете у контроллера представление для работы, вам также необходимо передать обратно внесенные данные, чтобы их можно было сохранить (рисунок 5).

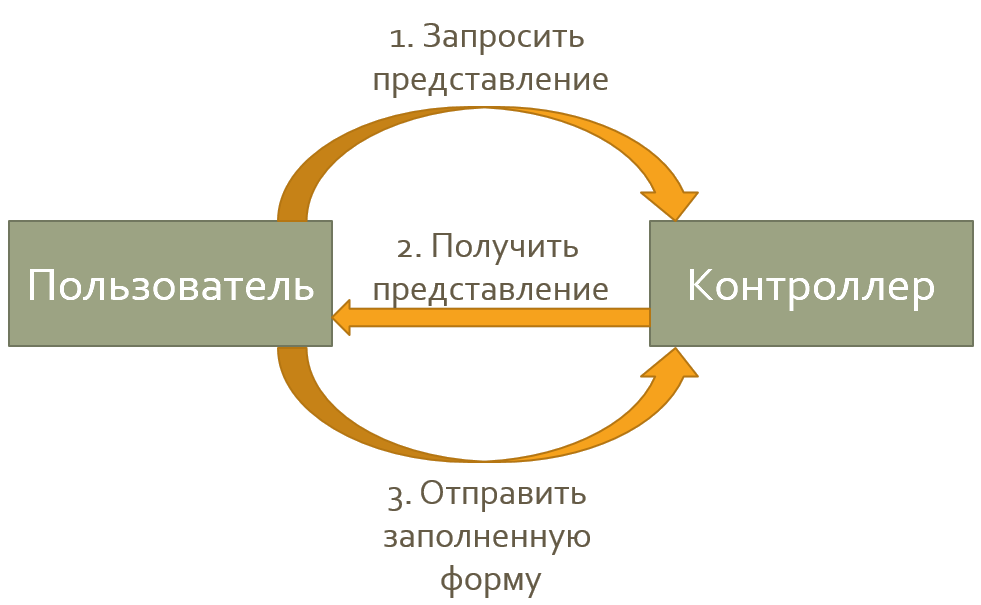


Рисунок 5 – Схема взаимодействия с приложением при создании и редактировании записей

Далее необходимо создать метод, который будет возвращать представление с полями для добавления нового объекта (рисунок 6).

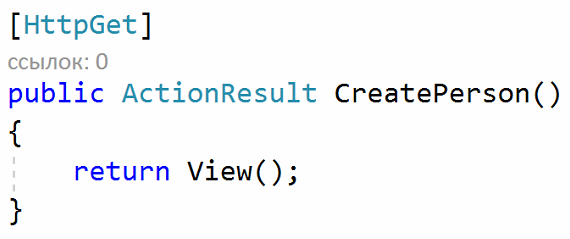


Рисунок 6 – Метод, возвращающий представление

Для созданного метода необходимо добавить представление. Стоит отметить, что ASP.NET поддерживает такую функциональность, как формирование шаблонов кода контроллера и представлений.

В случае с представлениями, это позволяет автоматически формировать разметку, в зависимости от класса решаемой задачи на будущей форме. Создадим представление по шаблону Create, как это показано на рисунке 7. Модель данных – класс Person. В вашем случае это будет класс той таблицы, для которой вы реализуете добавление.

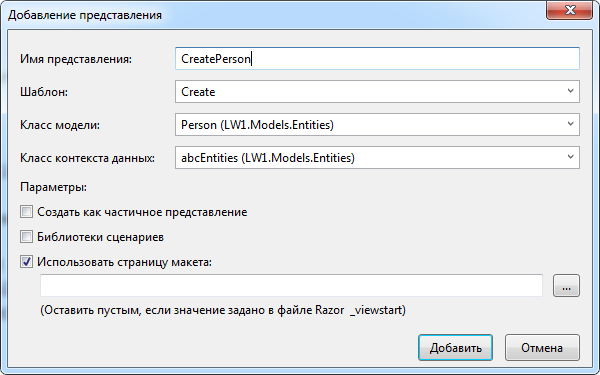


Рисунок 7 – Добавление представления

На рисунке 8 представлен фрагмент html-разметки, сгенерированной автоматически при помощи выбранного шаблона.



Рисунок 8 – Фрагмент представления для добавления объекта

В процессе работы над лабораторной работой данный скрипт будет изменен.

Для того, чтобы иметь возможность отобразить форму для добавления нового объекта, необходимо добавить кнопку «Добавить» на форму со списком объектов, для этого в коде основного представления перед скриптом, формирующим список объектов, нужно вставить скрипт, генерирующий кнопку на форме. Здесь необходимо будет указать метод контроллера, в который будет осуществлен переход при нажатии кнопки. Сделать это можно двумя способами, представленными на рисунке 9.

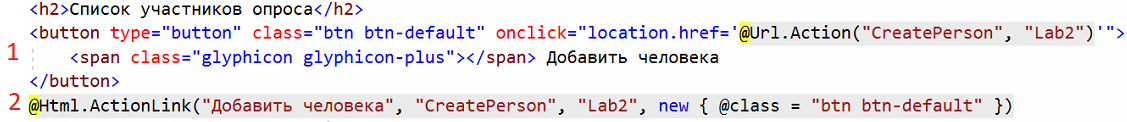


Рисунок 9 – Код добавления кнопок

В первом случае кнопка создается при помощи тега <button>. Данный тег предназначен для создания кнопок на форме и имеет множество атрибутов, например, как здесь, такие атрибуты, как type, class, onclick.

Атрибут type отвечает за тип кнопки или, иными словами, за ее поведение на форме. По умолчанию данный атрибут принимает значение “submit”, это означает, что при нажатии на нее произойдет отправка формы (отправка считанных с нее данных на контроллер).

Значение type = “button” означает, что кнопка не будет отправлять форму, а будет просто выполнять какие-то определенные действия при нажатии.

И как раз-таки за то действие, которое будет выполнено при нажатии кнопки, отвечает атрибут onclick. Здесь данный атрибут принимает значение ссылки, по которой будет осуществлен переход по нажатию кнопки (ссылка прописана в виде Url-адреса, где метод Action принимает два параметра – имя метода контроллера и имя самого контроллера).

Во втором же случае кнопка создается при помощи HTML-помощника HTML.ActionLink, что, по сути своей, является обычной ссылкой и принимает в данном случае четыре параметра – отображаемый текст, имя метода контроллера, имя самого контроллера, html-атрибуты. Здесь в html-атрибутах можно заметить тот же класс Bootstrap «btn btn-default», что и в теге <button> в первом случае. Поэтому стилизация кнопок в итоге выглядит одинаково, за исключением того, что в первом случае в тег <button> встроен тег <span> со стилизацией «glyphicon glyphicon-plus» (знак плюса).

В результате добавления кнопок в разметку представления, они должны отобразиться на форме со списком объектов в следующем виде (рисунок 10):

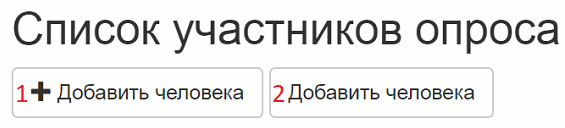


Рисунок 10 – Кнопки в представлении

По функционалу данные кнопки не отличаются друг от друга.

Далее, так как контроллер ожидает данных от представления (рисунок 5), в классе контроллера необходимо создать метод, который будет принимать объект.

Объект – экземпляр класса Person, название метода для удобства оставим идентичным с тем, что использовали для получения представления, то есть CreatePerson. Однако нужно обозначить, что к этому методу можно обратиться только за тем, чтобы передать объект, а не для получения исходного представления. Для этого укажем атрибут HttpPost, поскольку HTTP-метод POST предназначен для отправки данных на сервер.

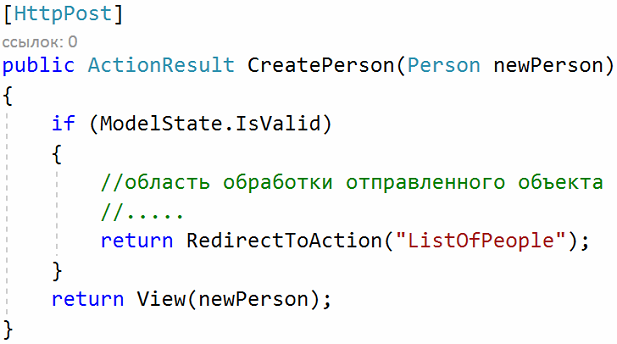


Рисунок 11 – Заготовка метода для получения добавляемого объекта

Это не окончательный вид метода.

* 1. **Проектирование веб-формы**

Вернемся к структуре сгенерированного кода представления (рисунок 8), где можно заметить конструкцию using (Html.BeginForm). Данный помощник создает открывающий тег <form> и закрывающий тег </form> – формирует на вашей странице форму. Веб-формы (или HTML-формы) – один из основных элементов взаимодействия между пользователем и приложением, который позволяет пользователю ввести данные, которые затем отправляются на сервер для их дальнейшей обработки (см. ссылку №2 в списке источников). Форма является элементом-контейнером, но при этом содержит и собственные атрибуты, определяющие ее поведение. Основными атрибутами являются action и method. Атрибут action определяет адрес, куда должны быть переданы данные после отправки формы. Атрибут method указывает, какой HTTP-метод будет использован при передаче данных (это может быть «Get» или «Post»). При использовании метода Post данные из формы включаются в тело HTTP-запроса. Если при определении формы указать метод Get, то данные из формы добавляются к URI атрибута action, их разделяет символ '?', и полученный URI посылается на сервер. Предпочтительно использовать метод Post, особенно при отправке больших объемов данных и паролей.

Больше информации об атрибутах формы вы можете прочитать по ссылке №3 в списке источников.

В нашем примере помощник Html.BeginForm() вызывается без аргументов, а значит атрибуты action и method будут заполнены по умолчанию: у method будет значение Post, а у action будет название метода, для которого создавалось представление. В данном примере в контроллере Lab2 были созданы Get и Post методы с названием CreatePerson (рисунки 6 и 11), и для этого Get метода было добавлено представление, так что значение атрибута action будет составлено из этих названий. Преобразование значения помощника в HTML код представлено на рисунке 12.



Рисунок 12 – Объявление формы в HTML коде страницы CreatePerson

У Html.BeginForm есть несколько перегрузок. В одной из таких перегрузок можно явно указать название метода, куда должны передаваться данные при отправке, название контроллера, где этот метод находится, и тип HTTP-запроса (рисунок 13).



Рисунок 13 – Объявление формы с указанием основных атрибутов

После объявления формы (рисунок 8) можно увидеть очередной помощник – Html.AntiForgeryToken, который помогает предотвратить так вид атаки, как межсайтовая подделка запроса (Сross Site Request Forgery, CSRF), производя верификацию токенов при обращении к методу действия. Для корректной работы необходимо также дописать соответствующий атрибут методу, который принимает данные при отправке формы (рисунок 14).

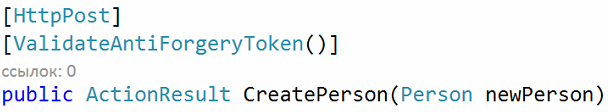


Рисунок 14 – Дополнительный атрибут для метода CreatePerson

Следом внутри объявленной формы можно заметить помощника Html.ValidationSummary, который добавляет список ошибок проверки достоверности модели. Проверка достоверности или валидация модели происходит в методе, указанном в атрибуте action у формы. В данном примере это метод CreatePerson (рисунок 11). Подробнее о проверке подлинности модели, ошибках уровня модели и ошибках уровня свойств можно прочитать по ссылке №4 в списке источников.

Затем (рисунок 8) начинается разметка для заполнения свойств модели. На рисунке 15 приведен HTML код разметки для свойства «LastName». Подобным образом формируется разметка и для остальных свойств.

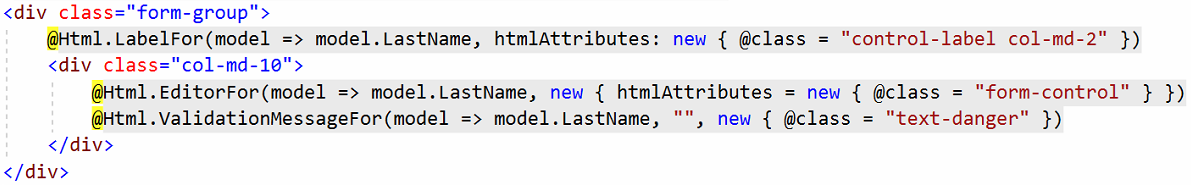


Рисунок 15 – Код добавления поля для ввода фамилии

HTML-помощник LabelFor предназначен для указания подписи к полю на форме. Приведенный код отобразит подпись к полю для фамилии наименованием указанного свойства (то есть, поле будет подписано как LastName). Для переопределения этой подписи вручную (например, чтобы поле было подписано как «Фамилия»), имеется возможность дописать еще один параметр в помощник LabelFor (рисунок 16).

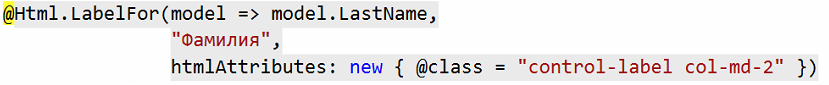


Рисунок 16 – Код создания label с собственной подписью

Помощник EditorFor визуализирует редактор для указанного свойства модели, подбирая HTML-элемент в соответствие с типом и метаданными свойства. Так для свойства string LastName отобразится поле для ввода строк, а для свойства int Age – поле, принимающее только числовые значения (рисунок 17). Для многих свойств такой подход может быть достаточным, но поскольку EditorFor – обобщенный шаблонный метод, для него доступно меньше параметров для редактирования отображения.

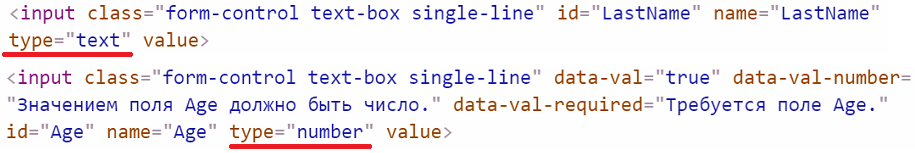


Рисунок 17 – HTML код для свойств разных типов

Помощник Html.ValidationMessageFor также отображает сообщения об ошибках проверки достоверности модели, но не весь список, как ValidationSummary, а ошибки конкретного свойства, к которому он относится.

Как можно заметить, у всех помощников, оканчивающихся на «For», первым параметром идет лямбда-выражение, где указывается свойство модели, для которого этот помощник создается. Такие помощники называются строго типизированными.

Рассмотрим примеры для других типов свойств модели. У персоны имеется свойство Gender (пол) с типом string. Сейчас это реализовано как поле для ввода слова, но возможна и реализация в более привычном виде – в виде радио-кнопок (radiobuttons). Пример того, как это будет выглядеть представлен на рисунке 18.

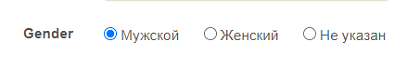


Рисунок 18 – Радиокнопки выбора пола персоны

В коде представления это будет выглядеть так, как показано на рисунке 19.

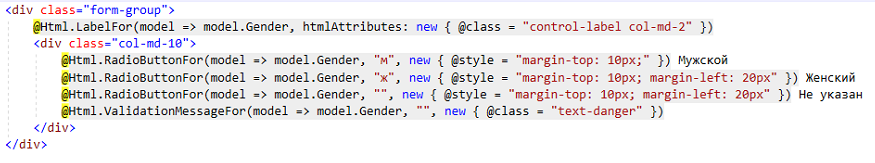


Рисунок 19 – Фрагмент кода представления для добавления радиокнопок

Здесь первым параметром является свойство модели, вторым – значение, которое примет данное свойство при выборе этой радиокнопки.

Задачу выбора пола можно реализовать и в виде выпадающего списка с элементами.

Стоит отметить, что каждый раз ставить в соответствие реальному значению свойства какое-то собственное описание (как, например, «м» – Мужской) неудобно, это заставляет тратить время на выполнение лишней работы. Этот процесс можно автоматизировать на контроллере.

Для этого необходимо создать метод для получения списка возможных значений пола персоны в классе контроллера и осуществить его вызов (рисунок 20).

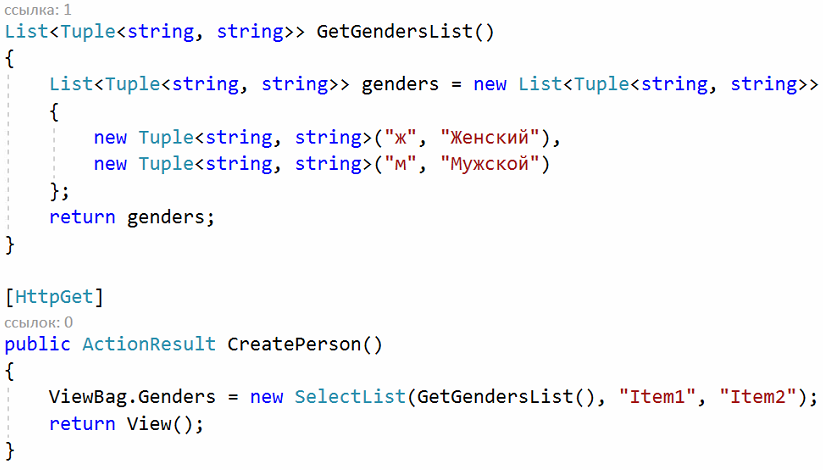


Рисунок 20 – Реализация и вызов метода формирования значений пола персоны

В приведенном коде метода реализуется создание списка объектов класса Tuple.

Класс Tuple предоставляет возможность работать с несколькими объектами различных типов как с единым целым. Результирующий набор объектов называется “кортеж”. В данном случае обращение к элементам кортежа осуществляется при помощи обозначений «Item1» (обозначение пола как в базе) и «Item2» (то наименование, которое будет поставлено в соответствие).

Теперь в представление передается ViewBag и вместо TextBoxFor или EditorFor указывается DropDownListFor (выпадающий список), как это показано на рисунке 21.

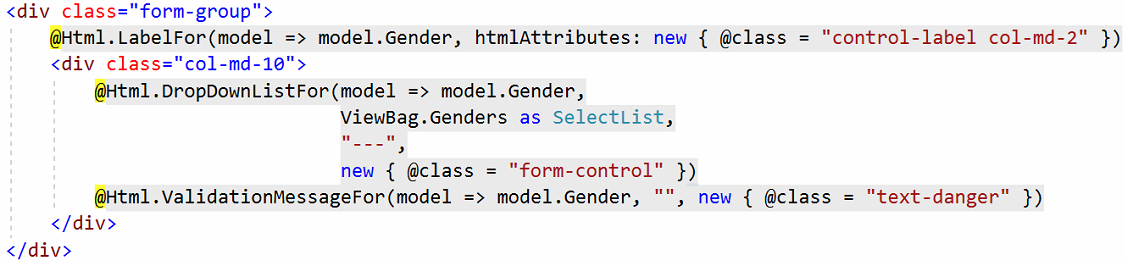


Рисунок 21 – Фрагмент кода представления для добавления выпадающего списка

В методе контроллера, который получает данные от представления, нужно снова инициализировать список для того, чтобы восстановить его на форме (рисунок 22).

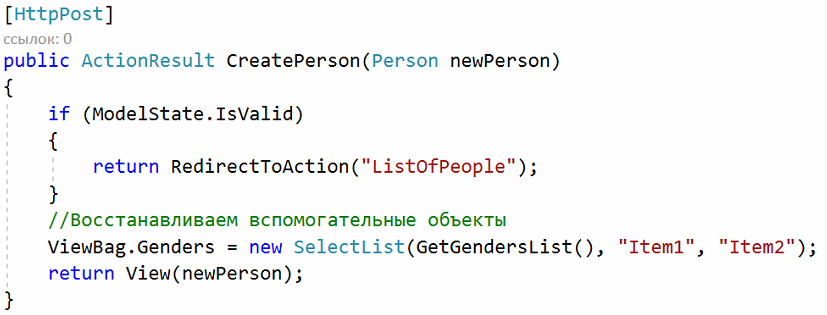


Рисунок 22 – Код метода, принимающего данные представления

Выпадающий список на форме выглядит следующим образом (рисунок 23):

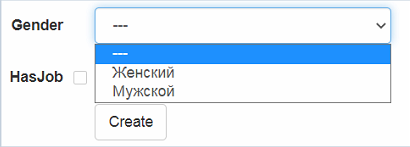


Рисунок 23 – Выпадающий список

* 1. **Обновление правил валидации и метода для добавления записи**

Вернемся в контроллер к методу, который должен получать отправляемый в форме объект. Для проверки достоверности нужно обратиться к свойству IsValid словаря состояния модели – ModelState (рисунок 24). Среди значений словаря будут те свойства модели, которые были указаны в веб-форме.

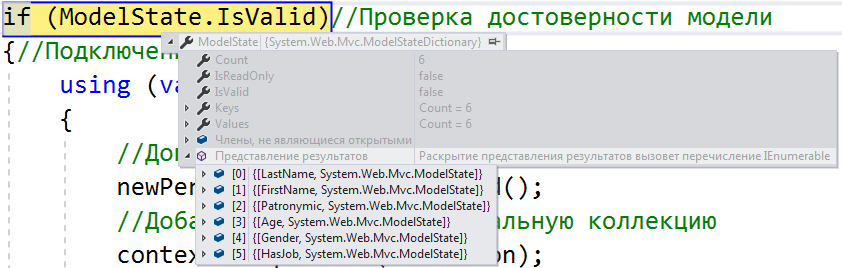


Рисунок 24 – Словарь состояния модели

Проверка свойств модели основывается на проверке типов свойств и проверке правил, заданных атрибутами у свойств. Поскольку наша модель – отображение таблицы из БД, созданное при подходе Database First, в классе модели нет атрибутов с правилами проверки, так что сейчас вся проверка основана на валидации типов. Поэтому для данного примера ошибка будет выявлена только для свойства int Age.

Изменим в представлении параметр у помощника валидации так, чтобы у нас отображались ошибки всех уровней (рисунок 25).



Рисунок 25 – Измененный параметр помощника

Дополним правила проверки. Для этого откроем класс модели (в данном примере класс Person), который был автоматически сгенерирован по таблице БД, и допишем несколько атрибутов для свойств (рисунок 26).

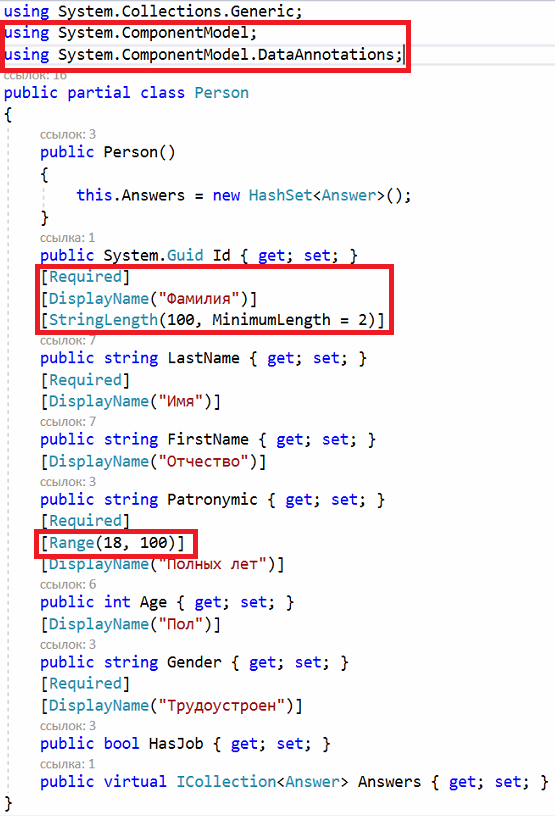


Рисунок 26 – Обновленный класс модели

Для применения атрибутов необходимо указать, что в коде будут также использоваться пространства имен System.ComponentModel и System.ComponentModel.DataAnnotations.

Во-первых, зададим отображение названий свойств на русском языке. Для этого используем атрибут DisplayName, в параметрах которого укажем необходимое наименование). Во-вторых, укажем атрибут Required для тех свойств, которые обязательны для заполнения, исходя из структуры соответствующей таблицы. Теперь пустые строки будут считаться недопустимым значением. На рисунке 27 отображено представление после попытки отправить незаполненную форму. Наименования свойств отображаются на русском языке, помощники валидации указывают текст ошибок.

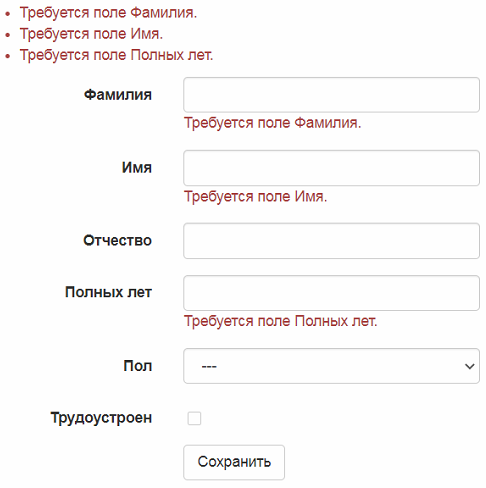


Рисунок 27 – Отображение ошибок уровня свойств модели

Атрибут Range задает ограничения на числовое диапазон, а атрибут StringLength – минимально и максимально допустимую длину строки знаков. Проверка этих правил начнется, когда будет удовлетворено требование атрибута Required (рисунок 28).

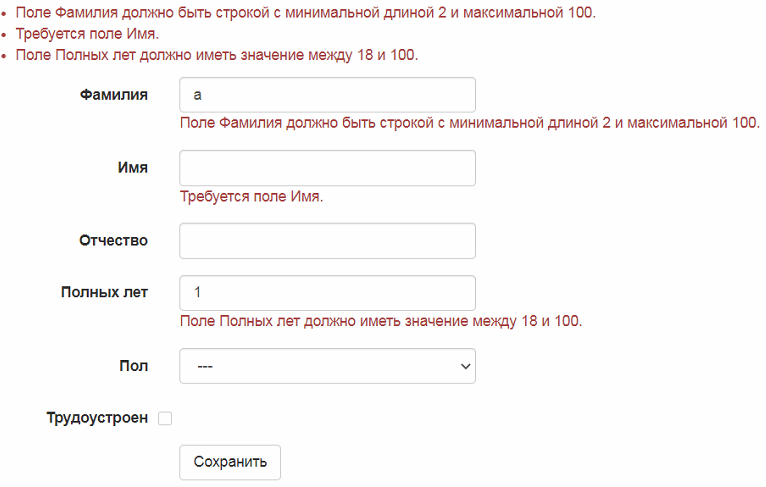


Рисунок 28 – Отображение ошибок, сгенерированных атрибутами Range, StringLength и Required

Кроме того, дополнительную проверку можно осуществлять в самом коде метода, и там же добавлять ошибки свойств и ошибки уровня модели. Подробнее об этом вы можете прочитать по ссылке №5 в списке источников.

Если проверка достоверности пройдена успешно, то далее можно провести оставшиеся необходимые манипуляции над объектом, подключиться к контексту данных (здесь – abcEntities), добавить объект в нужный набор данных (context.People) и сохранить изменения (context.SaveChanges). Обновленный код метода представлен на рисунке 29.

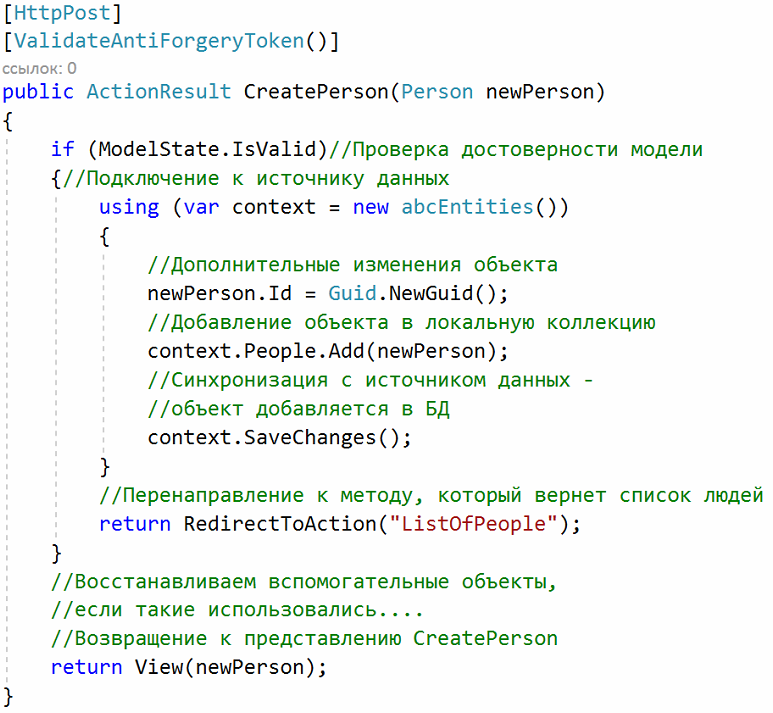


Рисунок 29 – Обновленный метод

* 1. **Добавление класса модели представления (ViewModel)**

Использование атрибутов – удобный механизм для проверки состояния модели. Однако, поскольку исходные классы являются автоматически сгенерированными, при обновлении модели данных все внесенные изменения исчезнут. Чтобы избежать таких потерь, можно отказаться от использования исходного класса. Для этого создадим новый класс, который будем использовать в качестве модели в представлении.

Создадим в папке Models еще одну папку с названием ViewModels (рисунок 30).

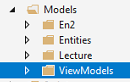


Рисунок 30 – Создание новой папки

Теперь создадим в этой папке новый класс (рисунок 31). Для этого нажмите на папку правой кнопкой мыши, выберите пункт «Добавить», а затем – «Создать элемент».

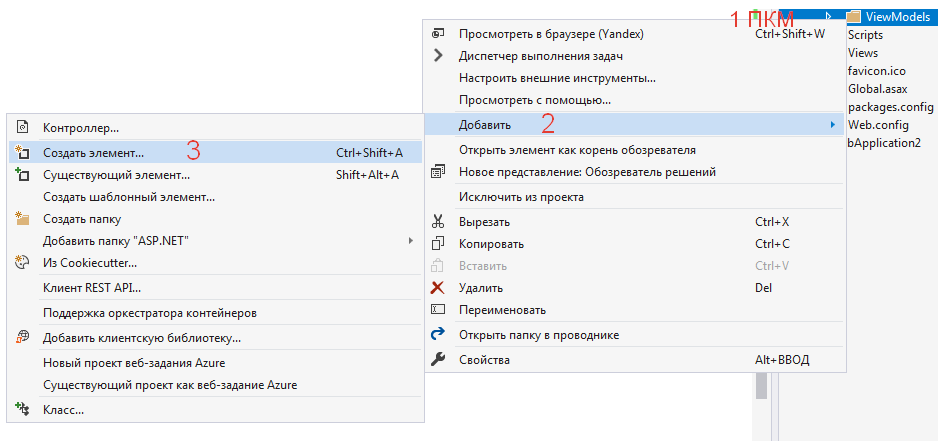


Рисунок 31 – Создание нового элемента в папке ViewModels

В открывшемся окне выберите пункт «Класс» и дайте название новому классу. Для упрощения ориентирования в коде рекомендуется в названии использовать название исходного класса, с которым вы собираетесь работать (в данном случае Person), и дописать к нему суффикс VM – ViewModel.

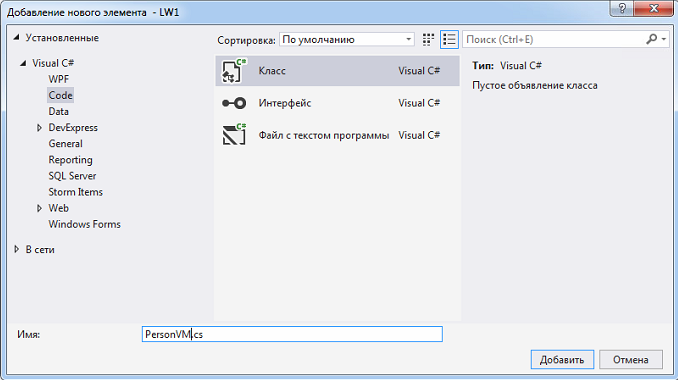


Рисунок 32 – Создание нового класса

Поскольку данный класс планируется использовать для добавления нового объекта и для редактирования существующих, продублируем свойства, которые будут затронуты при этих действиях. В данном случае это все свойства, кроме коллекции ICollection<Answer> Answers. Получившийся код класса представлен на рисунке 33.

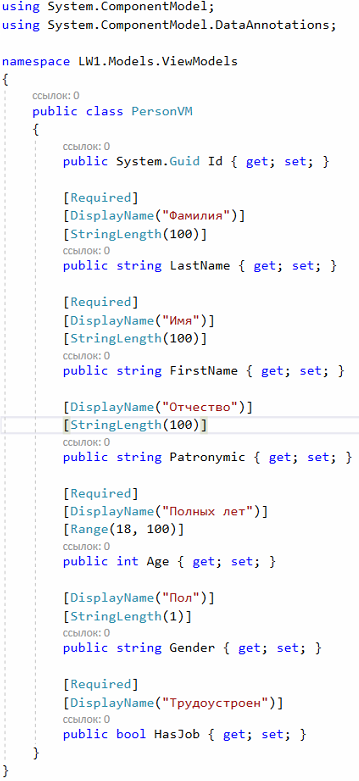


Рисунок 33 – Код класса представления

Теперь осталось заменить название исходного класса на название нового класса в представлении (рисунок 34) и в методе контроллера (рисунок 35). Обратите внимание, что в контроллере будет необходимо указать новое пространство имен, к которому относится ваша модель.



Рисунок 34 – Замена модели в представлении

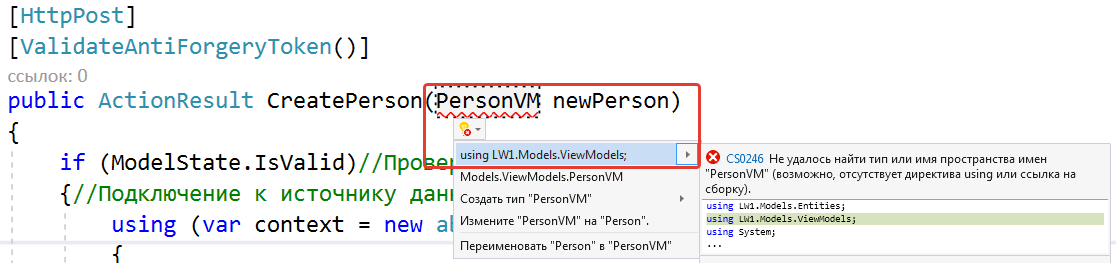


Рисунок 35 – Замена модели в коде метода контроллера

Названия свойств в обоих классах идентичны, поэтому в представлении никаких дополнительных замен проводить не придется. Однако для набора данных в контексте этот класс совершенно иной, и добавить объект этого класса не получится. Для корректной работы необходимо создать объект нужного класса и скопировать значения из полученного объекта класса модели представления (рисунок 36).

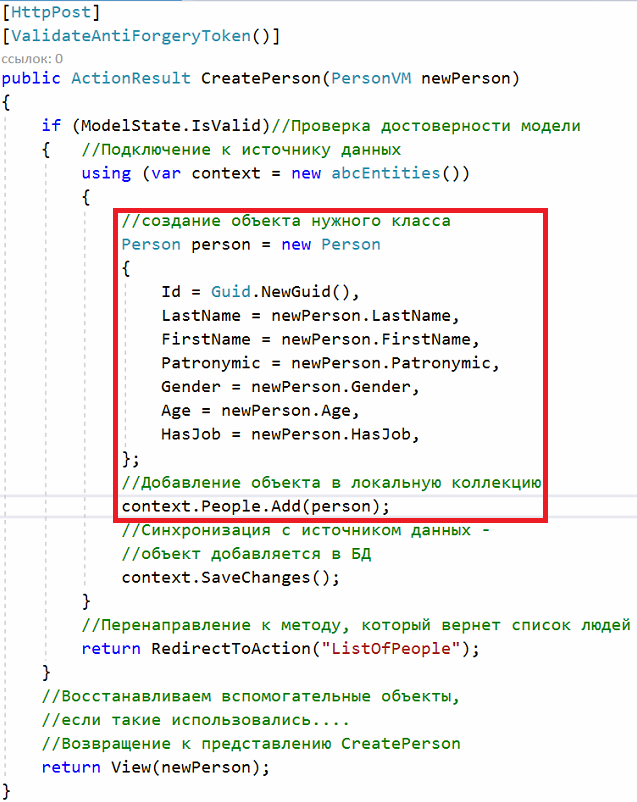


Рисунок 36 – Обновленный метод

**Оформление отчетов**

Отчет о выполнении лабораторной работы оформляется в печатном виде, должен включать титульный лист и состоять из следующих структурных элементов и разделов:

– цель работы;

– постановка задачи;

– порядок выполняемых действий с описанием;

– выводы по работе.

**Использованные источники и материалы для самостоятельного изучения**

1. Repository (Репозиторий) [Электронный ресурс] / Справочник «Паттерны проектирования» – Режим доступа: http://design-pattern.ru/patterns/repository.html
2. Ваша первая HTML форма [Электронный ресурс] / MDN Web Docs – Режим доступа: https://developer.mozilla.org/ru/docs/Learn/HTML/Forms/Ваша\_первая\_HTML\_форма
3. Элемент HTML form [Электронный ресурс] / MDN Web Docs – Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTML/Element/form
4. Проверка достоверности модели [Электронный ресурс] / Professor Web – Режим доступа: https://professorweb.ru/my/ASP\_NET/mvc/level7/7\_4.php
5. Валидация модели в контроллере [Электронный ресурс] / METANIT.COM – Режим доступа: https://metanit.com/sharp/mvc/7.4.php