**Лабораторная работа № 4**

**Работа с Entity Framework, Razor и Bootstrap. Редактирование и удаление записей. Частичные представления**

**Цель работы**

Освоить механизмы работы с Entity Framework для обновления и удаления данных, с HTML-помощниками для вызова частичных представлений и дочерних действий.

**После выполнения ЛР студенты должны уметь**

1. Создавать представление с формой для обновления объектов.
2. Создавать представление с формой для удаления объектов.
3. Использовать HTML-помощников для вызова элементов различных типов.
4. Создавать частичные представления и дочерние методы и вызывать их.

**Ход работы**

1. Открыть предыдущий проект ASP .NET MVC в среде Visual Studio.
2. Добавить метод для получения представления для редактирования объекта
3. Добавить метод для обработки отправляемой пользователем формы с отредактированным объектом
4. Добавить метод для получения представления для подтверждения удаления объекта
5. Добавить метод для обработки отправляемой пользователем формы для удаления объекта
6. Создать представления для страницы редактирования объекта и страницы для подтверждения удаления объекта
7. Добавить в класс свойство с типом DateTime (или использовать имеющееся) и форматировать его отображение в представлении
8. Создать частичное представление и/или дочернее действие для фрагмента кода представления и/или вывода дополнительной информации

**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).

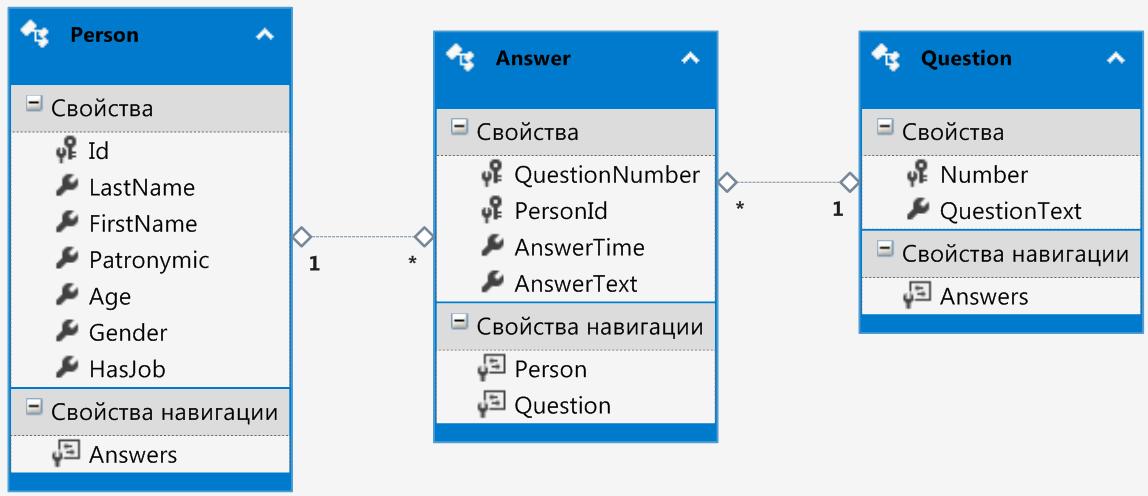


Рисунок 1 – Диаграмма используемой модели

В данной лабораторной будут рассмотрены операции по обновлению и удалению записей из наборов данных и соответствующих таблиц БД.

Если снова открыть файл метаданных класса DbSet<TEntity> (рисунок 2), то можно заметить, что в списке нет метода, который явно бы указывал на операцию по обновлению записи. Однако среди них (рисунок 3) есть методы с говорящими названиями Remove и RemoveRange, которые отвечают за удаление записей.

Пример использования этих методов и способа обновления записей будет приведен в следующих разделах. Подробнее узнать о методике обновления и удаления сущностей вы можете, ознакомившись с материалами лекции №5.

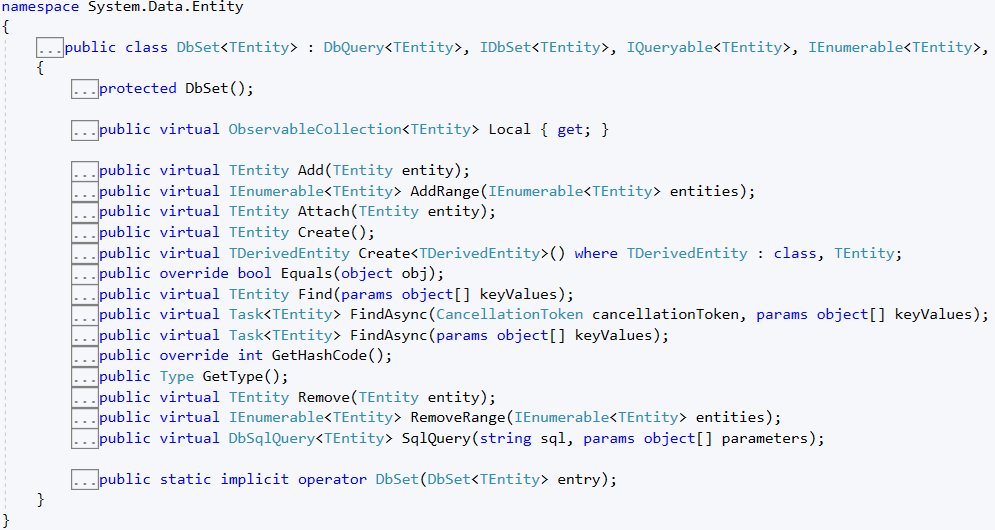


Рисунок 2 – Файл метаданных класса DbSet<TEntity>

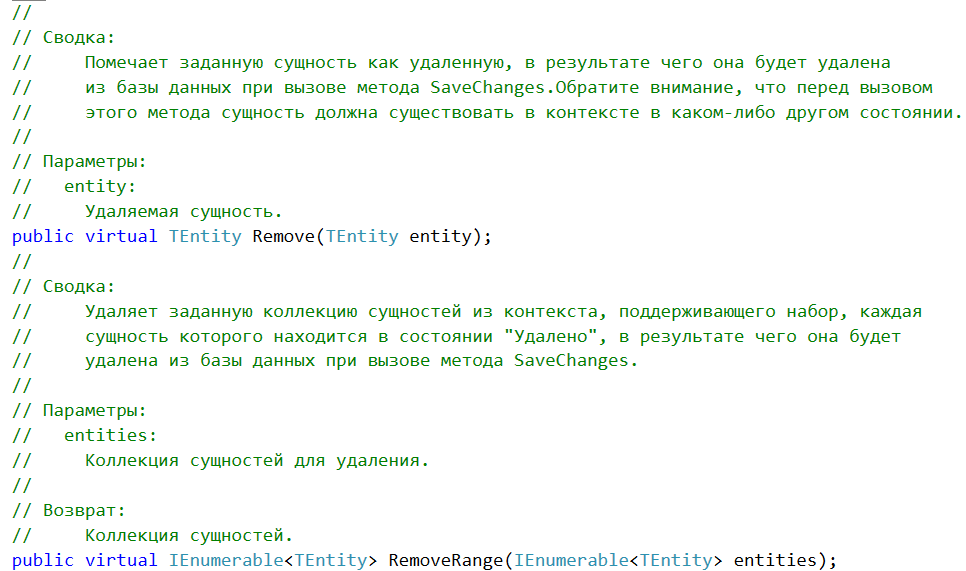


Рисунок 3 – Методы для удаления сущностей из набора данных, описанные в файле метаданных DbSet<TEntity>

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

Как и в случае с операцией добавления при редактировании необходимо реализовать в контроллере два метода, поскольку обращаться к нему мы будем дважды: сначала для получения страницы для редактирования сущности, а затем для отправки измененных данных (рисунок 4).

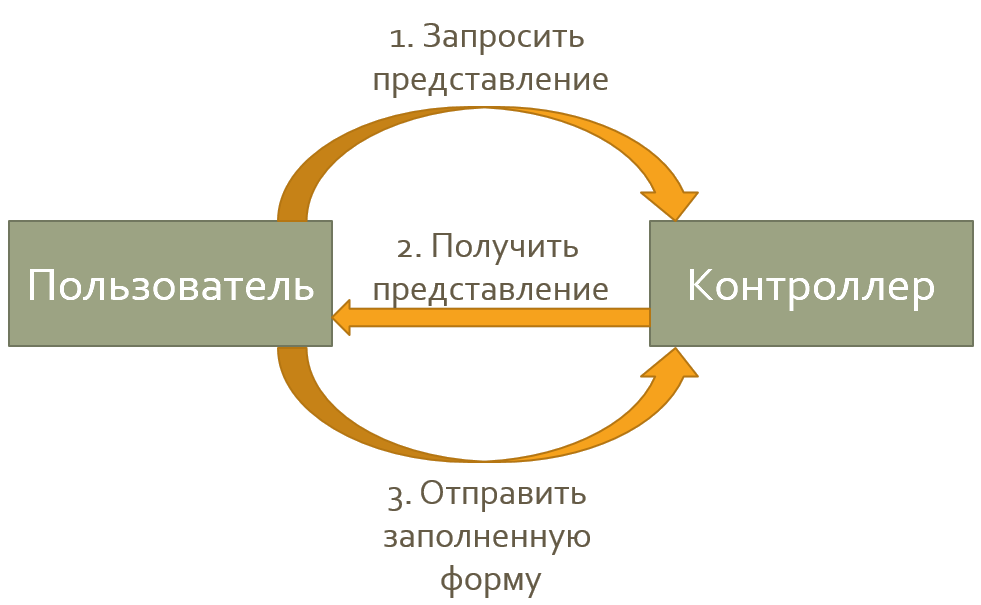


Рисунок 4 – Схема взаимодействия с приложением при создании и редактировании записей

Откройте ваш проект ASP .NET MVC в среде Visual Studio, найдите контроллер, в котором уже реализованы методы для добавления сущностей – методы для редактирования и удаления сущностей рекомендуется создавать в этом же контроллере.

Для GET метода для создания сущностей не требовалось параметров, а в теле метода просто возвращалось необходимое представление. В случае редактирования для метода потребуется параметр, с помощью которого можно определить, какую именно сущность предстоит редактировать. Чтобы получить сам экземпляр сущности, необходимо обратиться к контексту данных, а затем передать полученный объект представлению в качестве модели. Для удобства реализации правил валидации модели в рамках прошлой лабораторной был создан специальный класс модели представления PersonVM (рисунок 5). Для метода редактирования сущности будем также использовать этот класс.

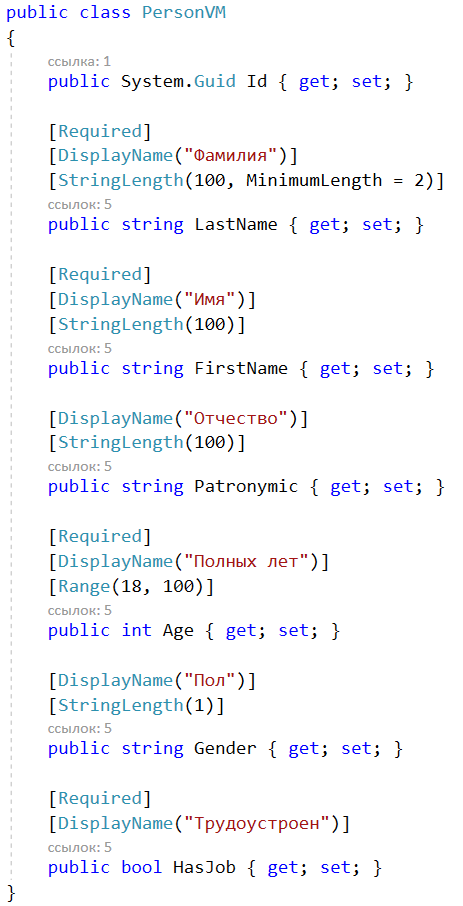


Рисунок 5 – Класс для модели представления

При помощи идентификатора мы получим из коллекции People объект класса Person, а в представление необходимо передать объекта класса PersonVM. Значит, среди прочих действий нужно преобразовать объект из одного класса в другой. На рисунке 6 представлено как в этом случае должен выглядеть GET метод для редактирования сущности.

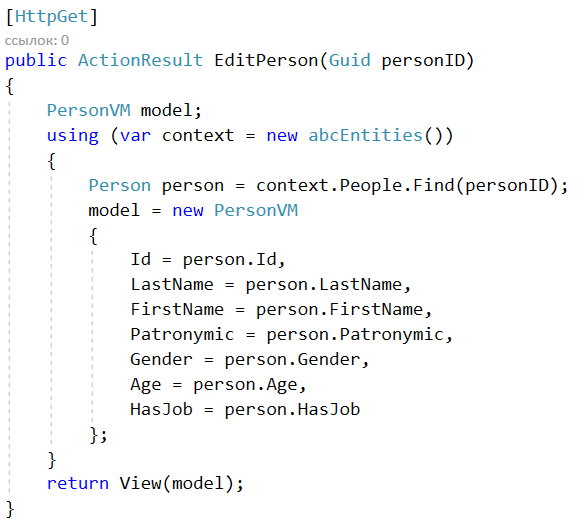


Рисунок 6 – GET метод для редактирования сущности

Создадим представление по шаблону Edit, как это показано на рисунке 7. Модель данных – класс PersonVM. В вашем случае это будет или класс той таблицы, для которой вы реализуете редактирование, или специально созданный класс модели представления.

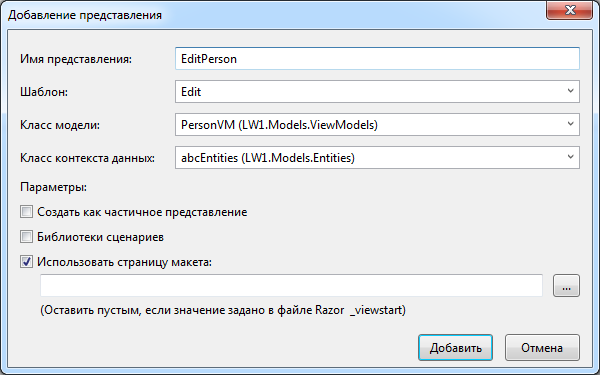


Рисунок 7 – Добавление представления

В целом представление для редактирования фактически идентично тому, которое вы создавали для добавления объекта, поскольку и в этом случае пользователю предстоит заполнять те же самые свойства объекта. Однако у этого представления есть важное отличие – наличие скрытого HTML-тега для идентификатора объекта (рисунок 8, строка 17).

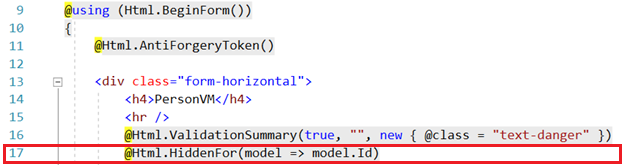


Рисунок 8 – Фрагмент кода представления для редактирования объекта

При создании нового объекта от пользователя не требуется самостоятельно заполнять идентификатор, так как это свойство обычно заполняется или на уровне приложения или самой базы данных. В случае же редактирования нам необходимо однозначно определить, какой объект будет изменен. Для этого, как было указано выше, в GET метод передается идентификатор. Когда пользователь заполнит элементы HTML формы и нажмет кнопку submit, в POST метод будет передан объект класса PersonVM с заполненными свойствами. При этом у объекта будут заполнены те свойства, которые были приведены в виде input тегов в рамках формы представления. В теле POST метода снова необходимо найти объект в контексте данных, чтобы заменить хранимые значения на новые. Чтобы однозначно найти объект, нужен идентификатор. Если в рамках формы в представлении нет элемента для свойства идентификатора, то это значение будет утеряно (рисунок 9).

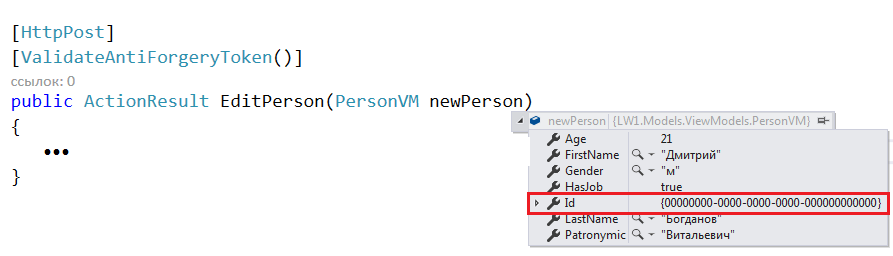


Рисунок 9 – Значение идентификатора при отсутствии соответствующего HTML-тега на форме

Значит, для идентификатора необходимо также указать HTML-тег input, но при этом пользователь не должен редактировать это значение. Поскольку такой суррогатный ключ в качестве идентификатора не несет какой-либо полезной информации для пользователя, то лучше его вообще не отображать. Для этого и используется выражение **@Html.HiddenFor(model => model.Id)**, которое создает разметку, представленную на рисунке 10.



Рисунок 10 – HTML-тег для скрытого свойства

Таким образом, свойство вносится в форму, не отображается пользователю, но передается вместе с объектом в POST метод (рисунок 11).

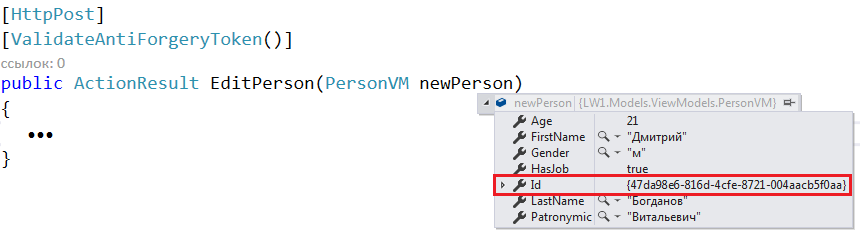


Рисунок 11 – Значение идентификатора при наличии соответствующего HTML-тега на форме

В POST метод передается объект класса PersonVM, а в наборе данных контекста ожидается объект класса Person. Значит, после проверки корректности модели необходимо преобразовать объект в нужный класс, чтобы обновить запись в источнике данных. Тогда POST метод может выглядеть, как представлено на рисунке 12.



Рисунок 12 – POST метод для обновления объекта

Как и в случае с методом добавления, если вы использовали динамический словарь ViewBag, то его значения нужно восстановить перед возвращением представления в случае неудачного завершения проверки корректности модели (рисунок 12, комментарий).

* 1. **Работа со свойствами типа DateTime**

Вероятнее всего, у вашего класса есть свойство с типом DateTime. Поскольку отображение таких свойств имеет некоторые особенности, рассмотрим эту ситуацию отдельно.

В базе данных у ваших атрибутов могут быть типы datetime, date, time, но при отображении в свойства класса на языке C# все эти типы будут представлены классом DateTime и выражены как дата и время суток.

Для примера добавим в модель представления PersonVM новые свойства для этих трех случаев (рисунок 13).

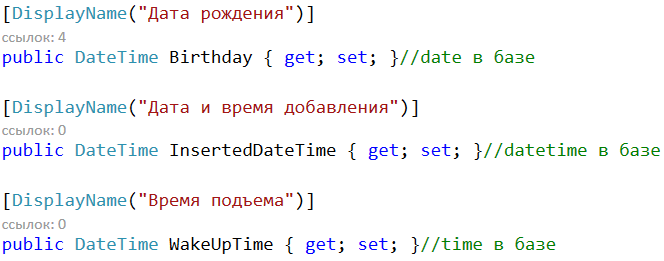


Рисунок 13 – Фрагмент кода модели представления с новыми свойствами

Если создать автоматически сгенерированное представление, то для этих свойств будет прописано выражение @Html.EditorFor, а преобразованные HTML-теги будут выглядеть, как представлено на рисунке 14.

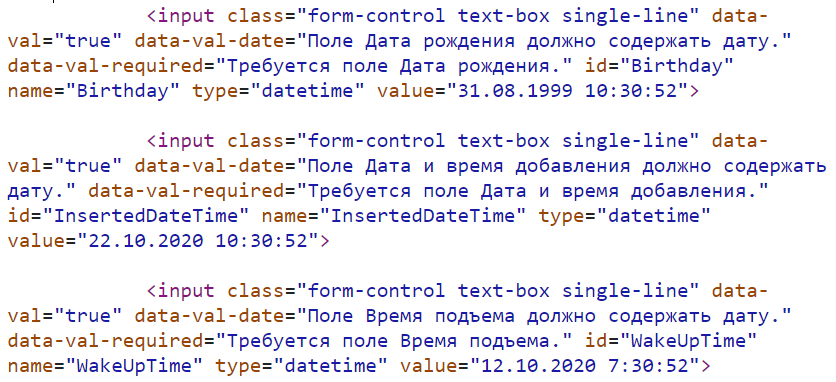


Рисунок 14 – Сгенерированные HTML-теги для новых свойств

На рисунке 15 продемонстрировано, как эти свойства будут выглядеть на самой странице. Во всех трех случаях отображается и дата, и время.



Рисунок 15 – Отображение свойств для даты и времени, сгенерированное по умолчанию

Отредактировать только одно выражение @Html.EditorFor и указать желаемое форматирование для этих свойств не выйдет, поэтому для редактирования непосредственно HTML тега нужно искать альтернативный способ.

### **3.1 Использование выражения TextBoxFor**

Параметры для форматирования есть в выражении @Html.TextBoxFor(expression, **string format**, object htmlAttributes). Тогда код представления будет выглядеть, как представлено на рисунке 16.

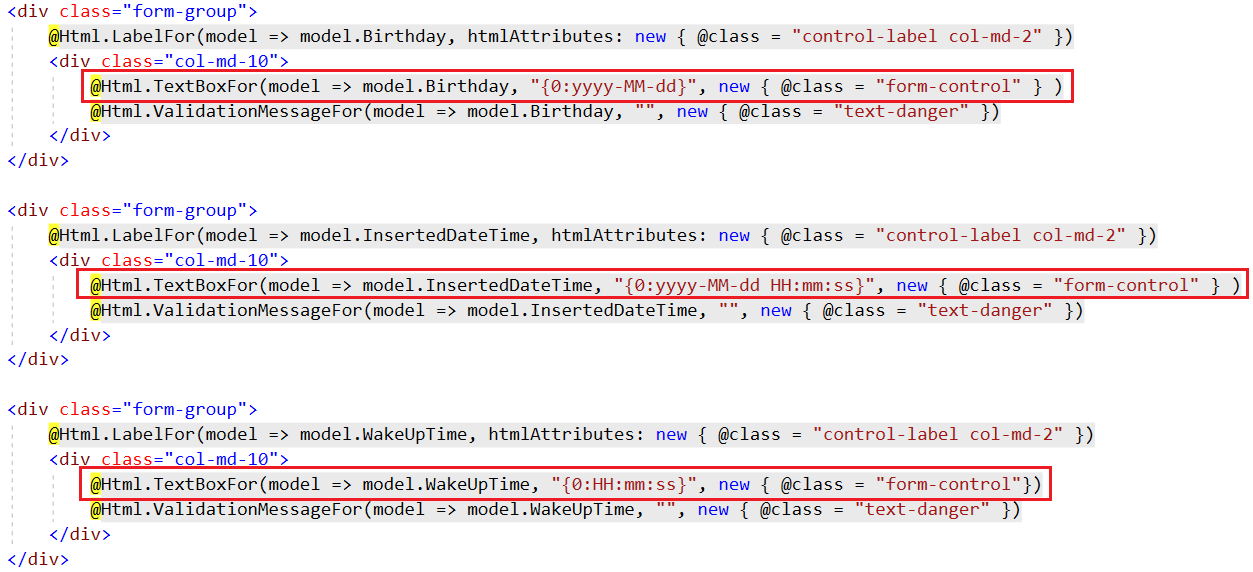


Рисунок 16 – Фрагмент кода представления: указание форматирования в параметрах TextBoxFor

Кроме того, в параметре HTML атрибутов (где сейчас указан класс фреймворка Bootstrap) можно уточнить тип элемента, чтобы в сгенерированном элементе использовать специальный интерфейс date picker или time picker.

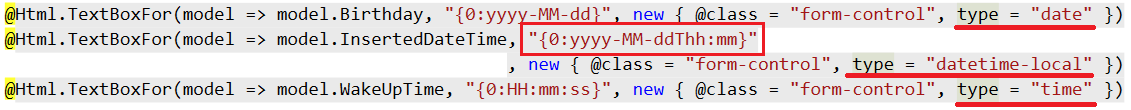
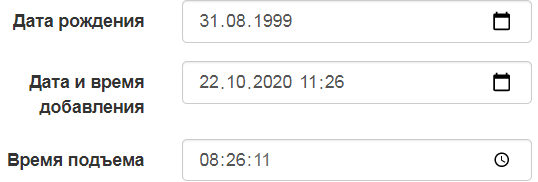
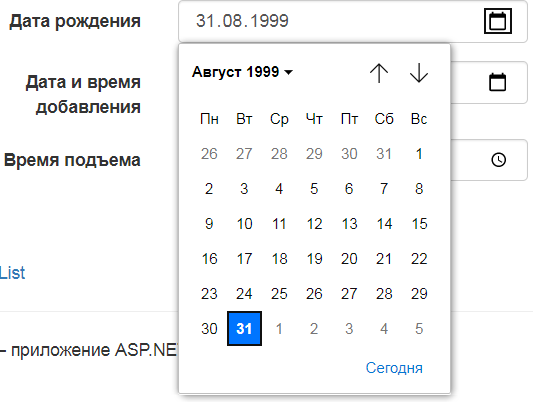
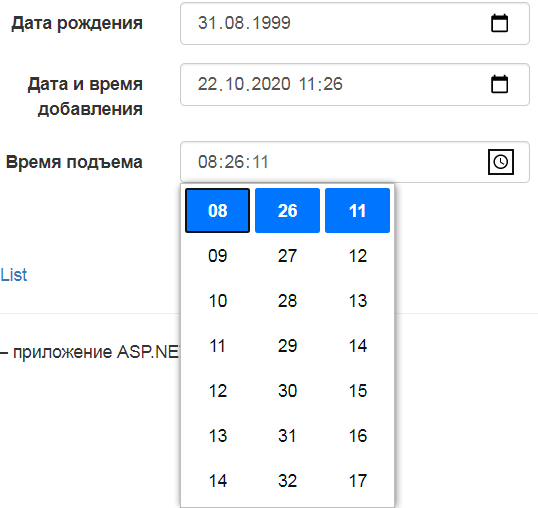


Рисунок 17 – Измененный код представления элементов с указанием их типов для тега input

Обратите внимание, что для элемента, где должны отображаться и дата, и время, название типа выглядит как datetime-local, и строка форматирования отличается от той, что использовалась для этого свойства ранее (см. ссылку №1 в списке источников). На рисунке 18 продемонстрировано, как будут отображены эти элементы, если в представлении использовать код с рисунка 17, и как выглядит интерфейс date picker и time picker.







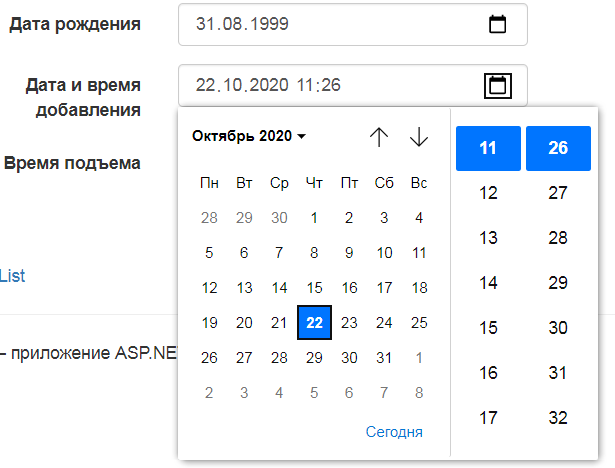


Рисунок 18 – Обновленные элементы представления и работа интерфейсов date picker и time picker

В данном случае вы контролируете отображение элемента в конкретно заданном представлении. Если вам понадобится реализовать редактирование этих же свойств в другом представлении, то настройку отображения будет необходимо прописывать заново.

### **3.2 Использование атрибутов свойств класса и выражения EditorFor**

Чтобы не проделывать одну и ту же работу несколько раз и не отслеживать, что правила отображения совпадают во всех представлениях, можно вынести правила отображения в атрибуты свойств, которые задаются в объявлении класса (рисунок 19).

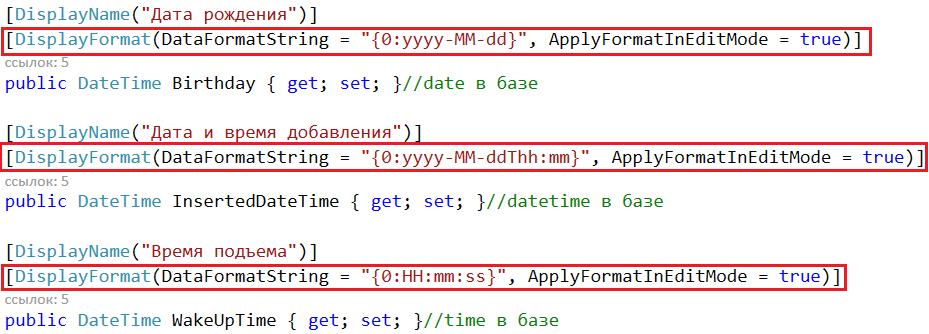


Рисунок 19 – Фрагмент класса PersonVM с объявлением свойств

Однако конструкция TextBoxFor при конвертации в HTML код игнорирует атрибут DisplayFormat (см. ссылку №2 в списке источников). Чтобы атрибут был воспринят и корректно использован, в представлении необходимо использовать конструкцию EditorFor, где в атрибутах HTML можно также указать тип, чтобы использовать интерфейс date picker или time picker (рисунок 20). В итоге на странице элементы будут отображены идентично тому, как продемонстрировано на рисунке 18.

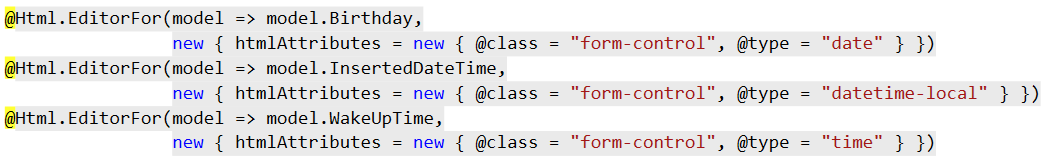


Рисунок 20 – Отображение форматированных свойств с помощью EditorFor

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

Для реализации удаления, с одной стороны, достаточно одного метода с идентификатором записи в качестве параметра, поскольку от пользователя ничего дополнительно заполнять не требуется. С другой стороны, операция удаления довольно серьезная и необратимая, так что позволять пользователю выполнять ее по одному нажатию может быть опрометчиво. Поэтому реализуем два метода: один вернет представление со страницей подтверждения выбора (рисунок 21), а второй непосредственно удалит выбранную запись (рисунок 22).

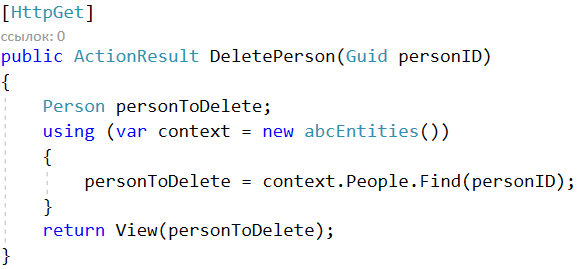


Рисунок 21 – Метод для возвращения страницы подтверждения удаления

Первый метод имеет тип GET, принимает в качестве параметра идентификатор, ищет в контексте данных необходимую запись и передает ее в качестве модели в представление. Как и в случае с редактированием, здесь можно воспользоваться не исходным классом, а создать экземпляр класса модели представления, чтобы автоматически применились атрибуты для отображения названий свойств.

Второй метод имеет тип POST и также принимает в качестве параметра идентификатор объекта. В рамках одного контроллера не может быть двух методов с одинаковым названием и набором параметров одного типа, поэтому просто создать метод [HttpGet] public ActionResult DeletePerson(Guid personID) нельзя. Поэтому при объявлении метода необходимо придумать для него новое название. Однако при этом можно указать в атрибуте альтернативное имя, по которому метод и будет вызываться (рисунок 22, атрибут ActionName).

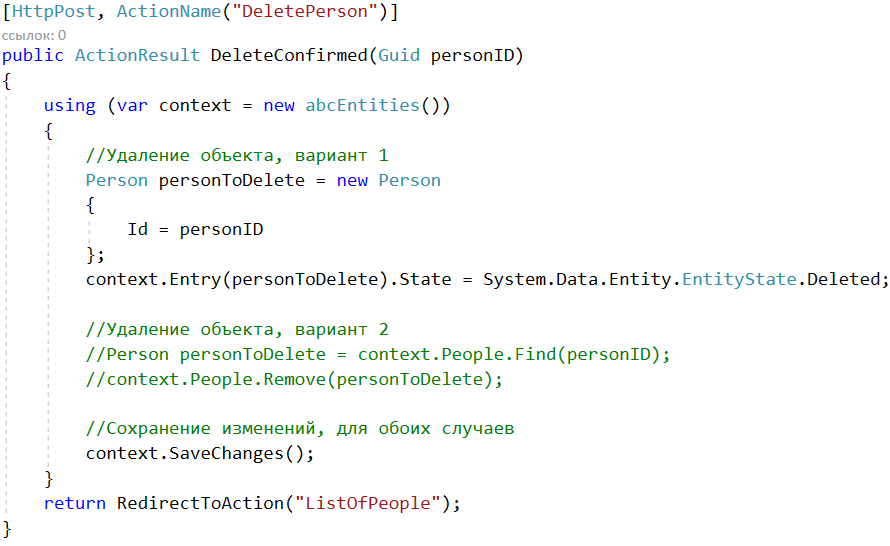


Рисунок 22 – Реализация POST метода удаления записи

В методе приведены два варианта реализации удаления записи. Подробнее о разнице между этими методами вы можете узнать из лекции №5.

Следующим шагом будет добавление представления (рисунок 23).

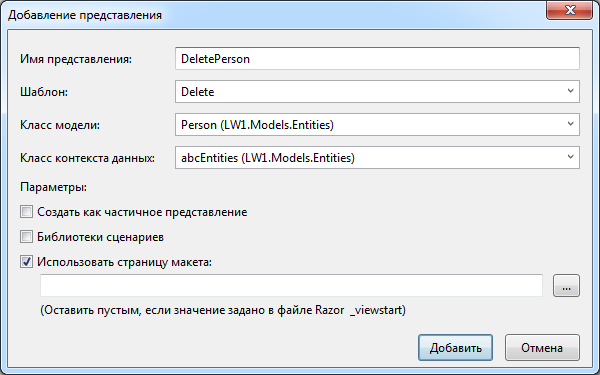


Рисунок 23 – Добавление представления

По умолчанию в данном представлении отображаются все значения свойств выбранного объекта и сообщение, уточняющее, действительно ли пользователь хочет удалить эту запись. В целом это очень похоже на представление типа Details, однако, в этом представлении формируются форма и кнопка для ее отправки, благодаря чему и может быть вызван POST метод непосредственно для удаления записи (рисунок 24).

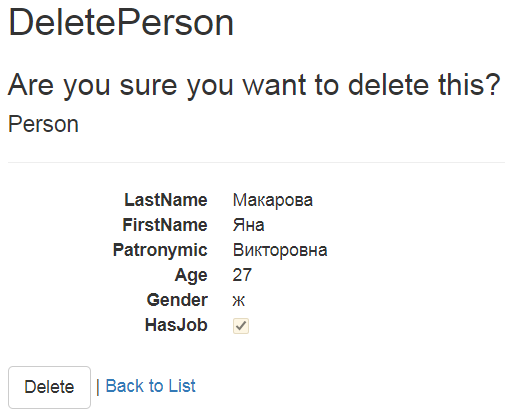


Рисунок 24 – Страница для подтверждения удаления, созданная по умолчанию

Форма задается с указанием метода и передаваемого идентификатора (рисунок 25).



Рисунок 25 – Форма для передачи идентификатора в метод подтверждения удаления

На основной странице, где выводится список элементов добавьте ссылки на методы для редактирования и удаления (рисунки 26 и 27).



Рисунок 26 – Фрагмент представление со списком элементов, тело таблицы с добавленными ячейками

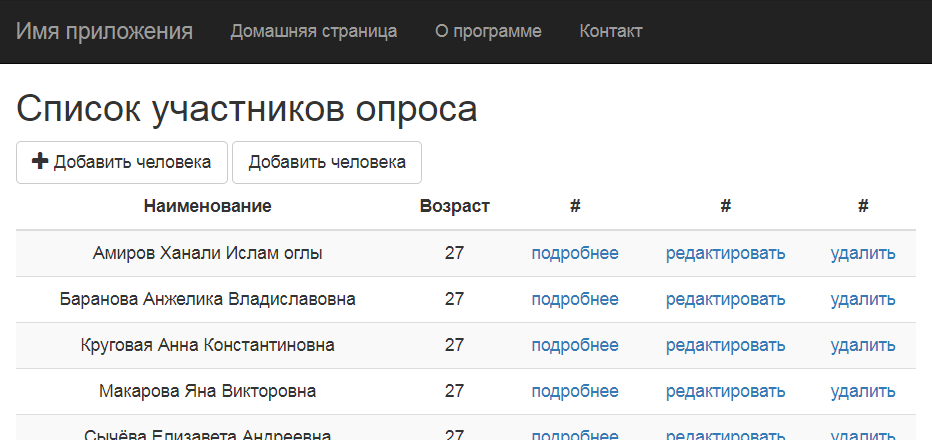


Рисунок 27 – Страница обновленного представления

* 1. **Частичные представления (Partial Views)**

Снова взгляните на рисунок 27. Можно заметить, что на странице есть шапка со ссылками «Домашняя страница», «О программе», «Контакт», а также таблица, ссылки для работы с отдельными элементами, кнопки. Если посмотреть сгенерированный код страницы, то он будет представлять собой единый файл с HTML разметкой. Однако в самом приложении это хранится не так. Код для шапки страницы прописан не в том же представлении, что и таблица, а в так называемом файле макета – \_Layout.cshtml из папки Views/Shared. Таким образом, итоговые страницы на самом деле могут генерироваться с использованием нескольких файлов. Это относится не только к самой HTML разметке, но и подключаемым файлам .css и .js.

Раз окончательный вид страницы собирается из нескольких файлов, то ничего не мешает еще разделить разметку. Например, если одни и те же фрагменты разметки требуются в нескольких разных местах приложения, вместо дублирования содержимого можно применять частичные представления. Это отдельные файлы представлений, которые содержат фрагменты разметки, и могут быть включены в другие представления.

Для примера сначала изменим способ отображения элементов в представлении со списком людей. Сейчас список отображается в виде таблицы, где в теле таблицы прописан цикл (рисунок 28).

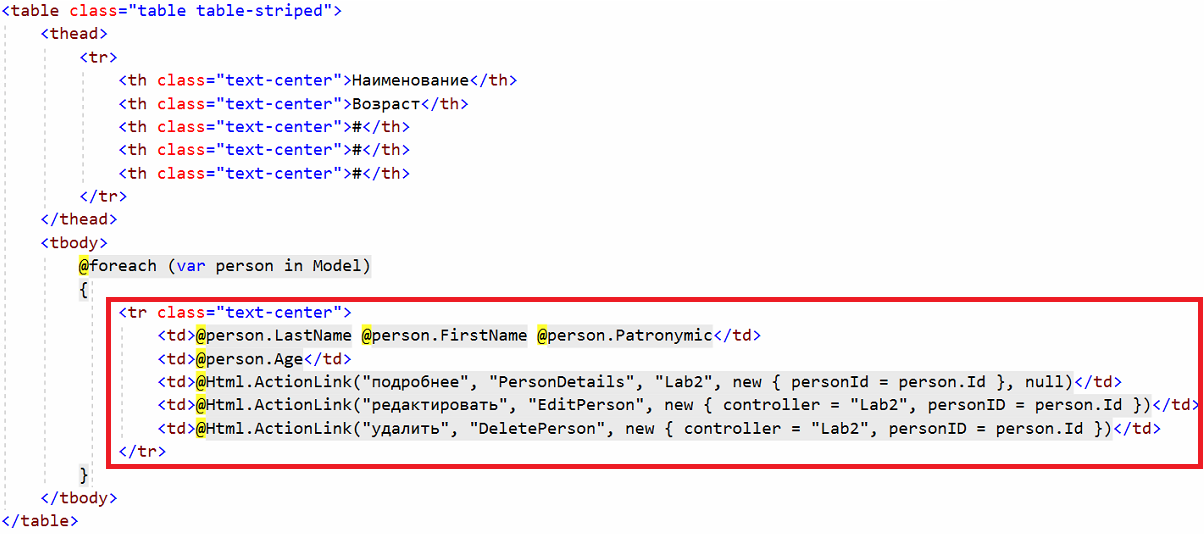
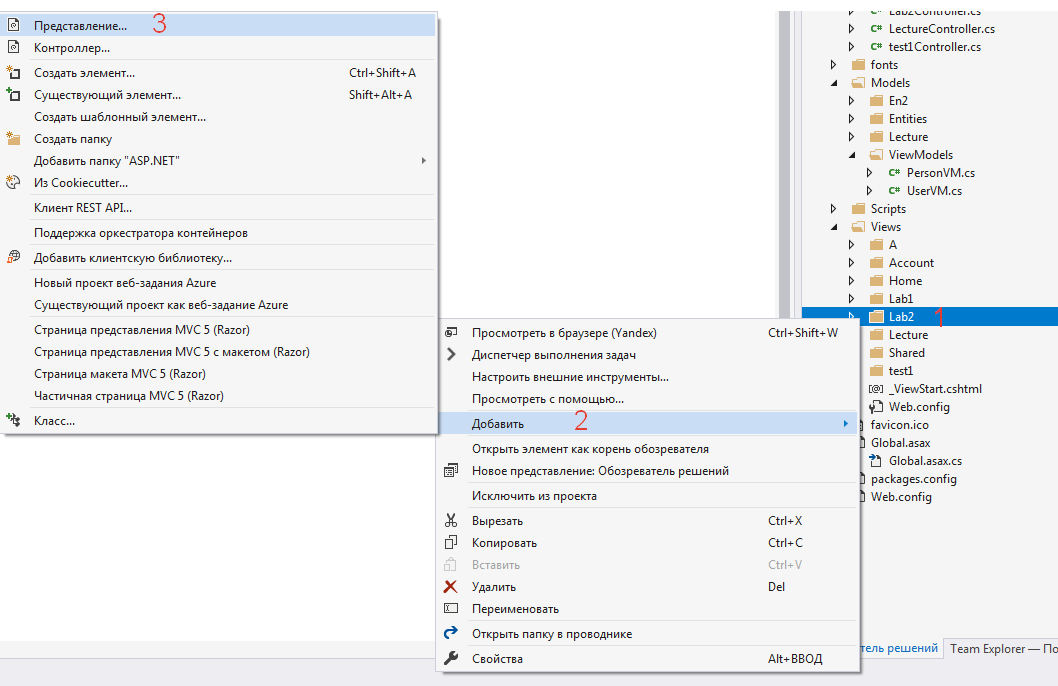


Рисунок 28 – Отображение элементов до изменений

Вынесем и перепишем часть в красной рамке в частичное представление, и в цикле будем вызывать только его. Процесс добавления частичного представления можно увидеть на рисунке 29.



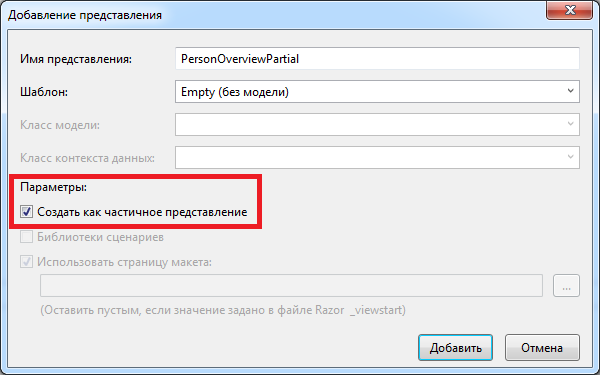


Рисунок 29 – Создание частичного представления

После создания по шаблону Empty частичное представление будет выглядеть как пустой файл. В исходном представлении моделью являлась коллекция объектов класса Person. В теле таблицы в цикле идет обращение к конкретному элементу коллекции. Если мы выносим эту часть в отдельное частичное представление, то в нем качестве модели должна быть уже не коллекция, а сам класс Person. Кроме того, в теле цикла очередной элемент сохранялся в переменной person, но в частичном представлении такой переменной нет. Однако мы будем передавать в это частичное представление объект в качестве модели, значит, обратиться к нему можно по имени Model. Вместо таблицы будем использовать для отображения блоки div и классы фреймворка Bootstrap. Тогда частичное представление будет выглядеть, как продемонстрировано на рисунке 30.

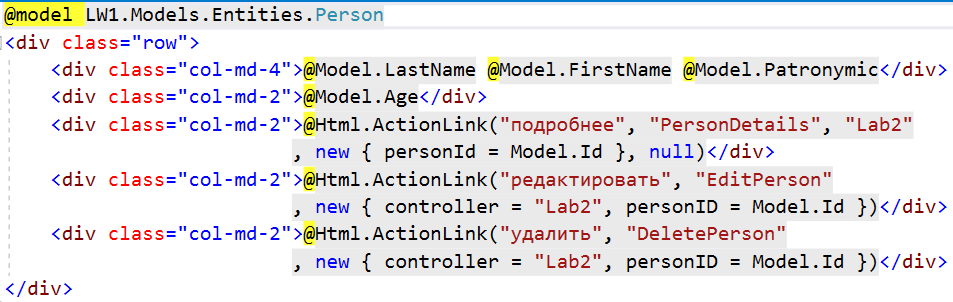


Рисунок 30 – Код частичного представления

Чтобы вызвать частичное представление, нужно использовать выражение @Html.Partial или @Html.RenderPartial. Пример использования этих выражений приведен на рисунке 31.

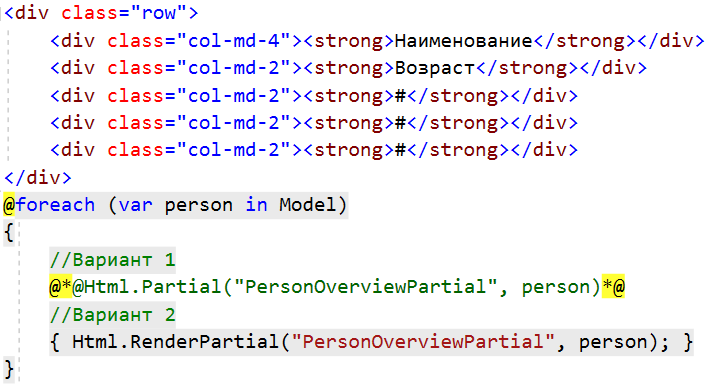


Рисунок 31 – Фрагмент основного представления, вызов частичного представления

Оба метода принимают в качестве параметров имя частичного представления и объект модели (если такой вообще имеется). Html.Partial возвращает HTML код представления в виде строки, а Html.RenderPartial напрямую пишет вывод в выходной поток.

В итоге сгенерированная страница выглядит следующим образом:

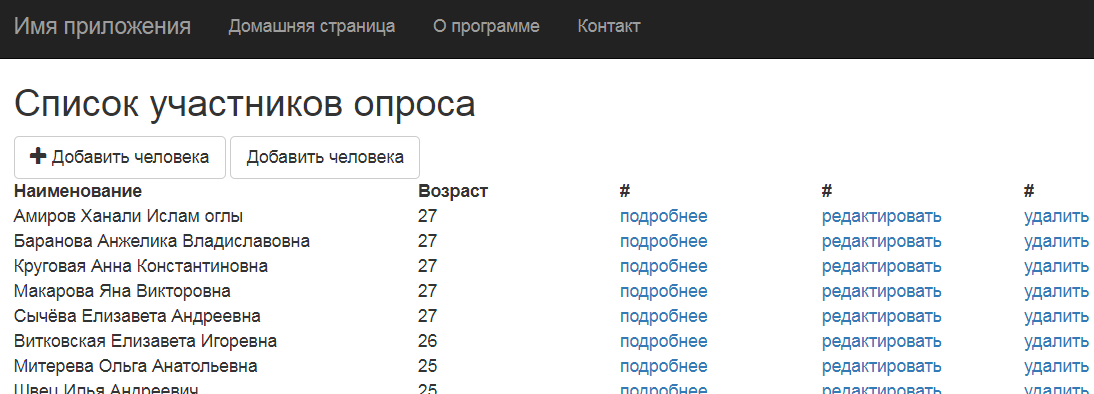


Рисунок 32 – Страница после использования частичного представления

Как можно заметить, для вызова таких частичных представлений не требуются соответствующие им методы в контроллере, поскольку необходимые данные они получают из основного представления.

* 1. **Дочерние действия**

Как было описано в предыдущем разделе, одни и же фрагменты разметки могут быть использованы в разных частях приложения, поэтому их можно вынести в частичное представление. То же самое касается логики действий в контроллере. В некоторых случаях повторяющиеся части можно вынести в так называемые дочерние действия – методы, вызываемые внутри представления (как основного, так и дочернего).

Создадим дочернее действие, которое принимает в качестве параметра идентификатор человека и возвращает количество вопросов, на которые этот человек ответил (рисунок 33).

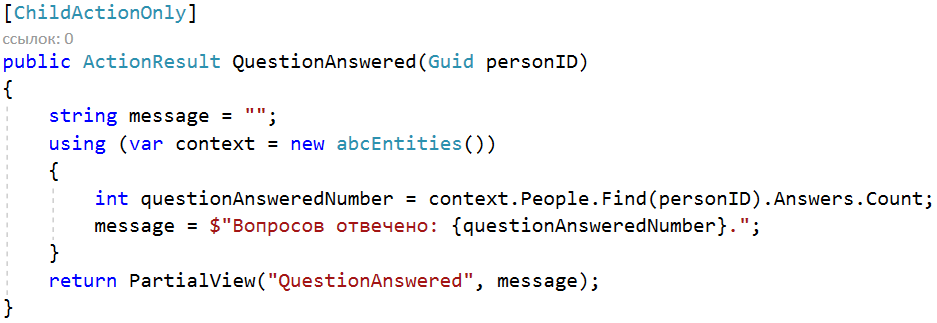


Рисунок 33 – Дочернее действие

Атрибут ChildActionOnly гарантирует, что метод действия может быть вызван только как дочерний метод в рамках представления. Метод действия не обязан иметь этот атрибут, чтобы использоваться в качестве дочернего действия, но предпочтительнее применять его, чтобы воспрепятствовать вызову таких методов действий в виде результата пользовательского запроса (см. ссылку №3 в списке источников).

Метод возвращает частичное представление, о чем говорит строка return PartialView(…). В качестве модели в представление передается объекта типа String. Поскольку PartialView с единственным параметром ожидает или название представления (строка), или объект модели, то запись **return PartialView(message)** в данном случае будет ошибочно воспринята как вызов частичного представления, название которого указано в переменной message. Для устранения этой неопределенности в методе указываем и название представление (совпадает с именем метода), и передаваемый объект.

Создадим частичное представление для отображения полученной строки (рисунок 34).

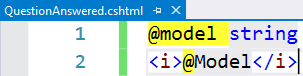


Рисунок 34 – Частичное представление для дочернего действия

Для вызова дочерних действий используются методы Html.Action и Html.RenderAction. Как и в случае с Partial и RenderPartial, Action возвращает результат в виде строки HTML, а RenderAction встраивает результат в выходное поток. Вызовем дочернее действие в ранее созданном частичном представлении PersonOverviewPartial (рисунок 35).

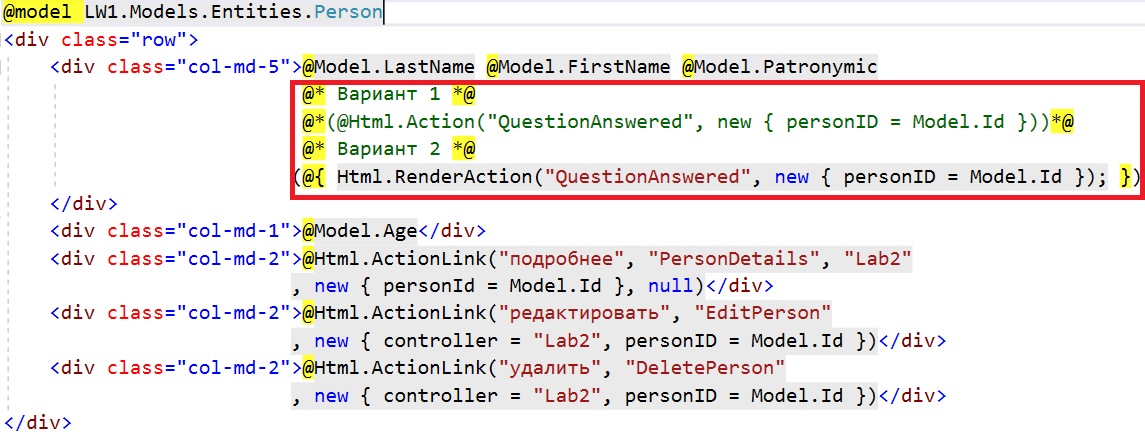


Рисунок 35 – Частичное представление с вызовом дочернего действия

После этого страница целиком будет выглядеть следующим образом:

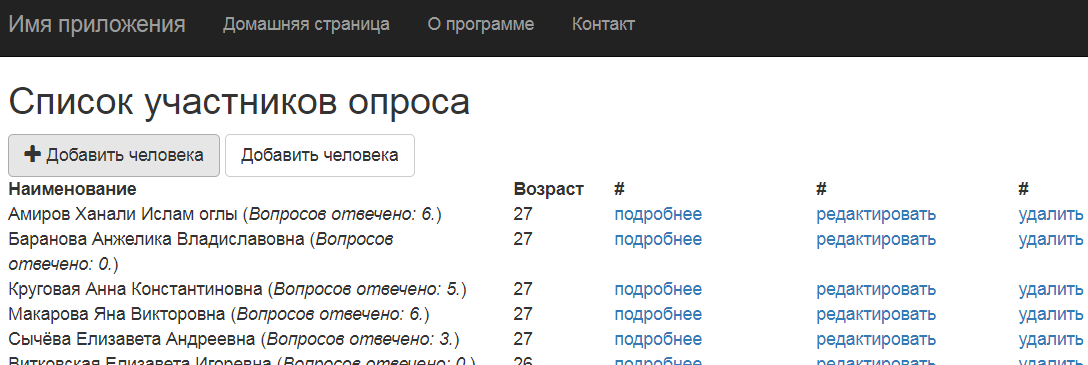


Рисунок 36 – Обновленная страница представления

**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. datetime-local [Электронный ресурс] / MDN Web Docs – Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTML/Element/input/datetime-local
2. DisplayFormat for TextBoxFor in MVC [Электронный ресурс] / tack Overflow – Режим доступа: https://stackoverflow.com/questions/14700873/displayformat-for-textboxfor-in-mvc
3. Дочерние действия [Электронный ресурс] / Professor Web – Режим доступа: https://professorweb.ru/my/ASP\_NET/mvc/level4/4\_14.php