Оглавление:

Введение

На современном этапе развития технологий сложно представить себе жизнь без Интернета. Бóльшую часть этой сети занимают веб-сайты. Веб-сайты используются во многих сферах жизни человека, они позволяют быстро находить информацию. В следствии этого можно сделать вывод о том, что средства и платформы для разработки веб-сайтов и приложений будут актуальным инструментом современного программиста еще не одно десятилетие. На данный момент существует большое множество различных технологий и их сочетаний. На фоне всех продуктов этой области выделяется технология ASP.NET поскольку позволяет использовать обширный набор элементов управления и библиотек классов, позволяющих быстрее разрабатывать приложения, а так же опирается на многоязыковые возможности .NET, что позволяет писать код страниц на VB.NET, Delphi.NET, C# и т.д. Данная квалификационная работа показывает возможности применения ASP.NET на примере создания веб-сайта кафедры «МО ЭВМ» Воронежского государственного университета.

Работа состоит из нескольких основных частей, включающих:

- теоретические данные
- планирование и подготовку к работе (разработка схемы сайта и схемы базы данных)
- установку и настройку компонентов, подключение базы данных с использованием ADO.NET
- поэтапный процесс написания основных компонентов веб-сайта
- внешнее оформление веб-сайта

1. Постановка задачи

В рамках этой выпускной квалификационной работы будет рассматриваться процесс создания веб-сайта с использованием технологии ASP.NET MVC 5. Результатом выполненной работы будет счиаться полностью функционирующий веб-сайт кафедры Математического Обеспечения ЭВМ, с собственной базой данных с возможностью авторизации и с разграничением ролей. Сайт должен содержать функционал, позволяющий зарегистрированному пользователю создавать/изменять/удалять информацию.

В ходе разработки будут решены следующие задачи:

— создание детального плана разработки проекта

— сконструирована база данных на базе MS SQL Server с использованием проектировщика баз данных Toad Data Modeler

— разработка основных компонентов веб-сайта

— использование системы авторизации, аутентификации и системы разграничения ролей под названием “ASP.NET Identity”

— стилизация веб-сайта с рассмотрением синтаксиса программирования Razor и использования библиотеки Bootstrap.

2. Теоретические сведения

2.1 Краткая характеристика

ASP.NET MVC Framework — фреймворк для создания веб-сайтов, который реализует принцип «Model-View-Controller».

Шаблон MVC, лежащий в основе новой платформы, подразумевает взаимодействие трех компонентов: контроллера (controller), модели (model) и представления (view). Что же представляют эти компоненты?

Контроллер (controller) представляет класс, с которого собственно и начинается работа приложения. Этот класс обеспечивает связь между моделью и представлением. Получая вводимые пользователем данные, контроллер исходя из внутренней логики при необходимости обращается к модели и генерирует соответствующее представление.

Представление (view) - это собственно визуальная часть или пользовательский интерфейс приложения - например, html-страница, через которую пользователь, зашедший на сайт, взаимодействует с веб-приложением.

Модель (model) представляет набор классов, описывающих логику используемых данных.

Общую схему взаимодействия упрощенно можно представить следующим образом:

Рисунок 2.1.1

2.2 История

В 1999 компанией Майкрософт было решено построить платформу с общими языками программирования Common Language Runtime (CLR) и на ее основе развить технологии, в нём, как и в Java, наличествовали программирование по принципам ООП, сборка мусора и другие возможности. Специалисты компании описывали это решение как «огромный риск», так как успех новой разработки был связан с успехом CLR, которая, находилась на ранней стадии разработки.

Компания Microsoft первоначально планировала назвать свой продукт ASP+ — как усовершенствование ASP. Но после создания платформы .NET ASP+ была переименована в ASP.NET и вошла в состав пакета среды разработки приложений Visual Studio.NET.

Непосредственно взаимодействуя с операционной системой, среда .NET Framework предоставляет интерфейс ASP-приложениям. Новая технология ASP.NET позволяет создавать приложения на нескольких языках программирования, например на Visual Basic .NET, C# и JScript .NET. Благодаря этому приложениям предоставляются возможности .NET, такие как работа в среде CLR, безопасность типов и наследование.

Технология ASP.NET является новой средой разработки Web-приложений. Технология ASP базировалась на использовании языков сценариев. В основу ASP.NET положена работа в среде CLR, что позволяет создавать Web-приложения на любом языке, поддерживаемом платформой .NET. Независимо от языка программирования, использованного при создании приложения ASP, его код компилируется в код на промежуточном языке IL. Это немаловажное преимущество, так как теперь возможности одного языка могут использоваться в другом языке без необходимости написания дополнительного кода. Таким образом достигается высокая степень повторного использования кода.

2.3 Особенности и принципы работы ASP.NET MVC 5

Поскольку ASP.NET основывается на Common Language Runtime (CLR), которая является основой всех приложений Microsoft .NET, разработчики могут писать код для ASP.NET, используя языки программирования, входящие в комплект .NET Framework (C#, Visual Basic.NET, J# и JScript .NET).

Для управления разметкой и вставками кода в представлении используется движок представлений. До версии MVC 5 использовались два движка: Web Forms и Razor.

Начиная с MVC 5 единственным движком, встроенным по умолчанию, является Razor. Движок WebForms использует файлы .aspx, а Razor — файлы .cshtml и .vbhtml для хранения кода представлений. Основой синтаксиса Razor является знак @, после которого осуществляется переход к коду на языках C#/VB.NET[26]. Также возможно и использование сторонних движков. Файлы представлений не являются стандартными статическими страницами с кодом html, а в процессе генерации контроллером ответа с использованием представлений компилируются в классы, из которых затем генерируется страница html.

При обработке запросов фреймворк ASP.NET MVC опирается на систему маршрутизации, которая сопоставляет все входящие запросы с определенными в системе маршрутами, которые указывают какой контроллер и метод должен обработать данный запрос. Встроенный маршрут по умолчанию предполагает трехзвенную структуру: контроллер/действие/параметр.

3. Практическая часть

3.1 Подготовка к работе и создание базы данных

Перед началом разработки необходим установить и настроить среду, в которой будет производиться создание веб-сайта. Visual Studio – одна из самых популярных на данный момент средств разработки, подходящая для этих целей.

Процесс установки и настройки Visual Studio, а также компонентов ASP.NET подробно описан на официальном сайте производителя: https://support.microsoft.com.

Проектировка базы данных будет производится в программе Toad Data Modeler 6.5. Размещение базы данных происходит на базе MS SQL Server. Установка и настройка данных программ обычно не вызывает трудностей.

Создаваемый сайт кафедры предполагает возможность просмотра и создания новостей кафедры, просмотр и изменение информации о преподавателях, об их курсовых темах и научных работах на уровне пользователя. С учетом данных требований была сформирована модель базы данных, изображенная на рисунке 3.1.1

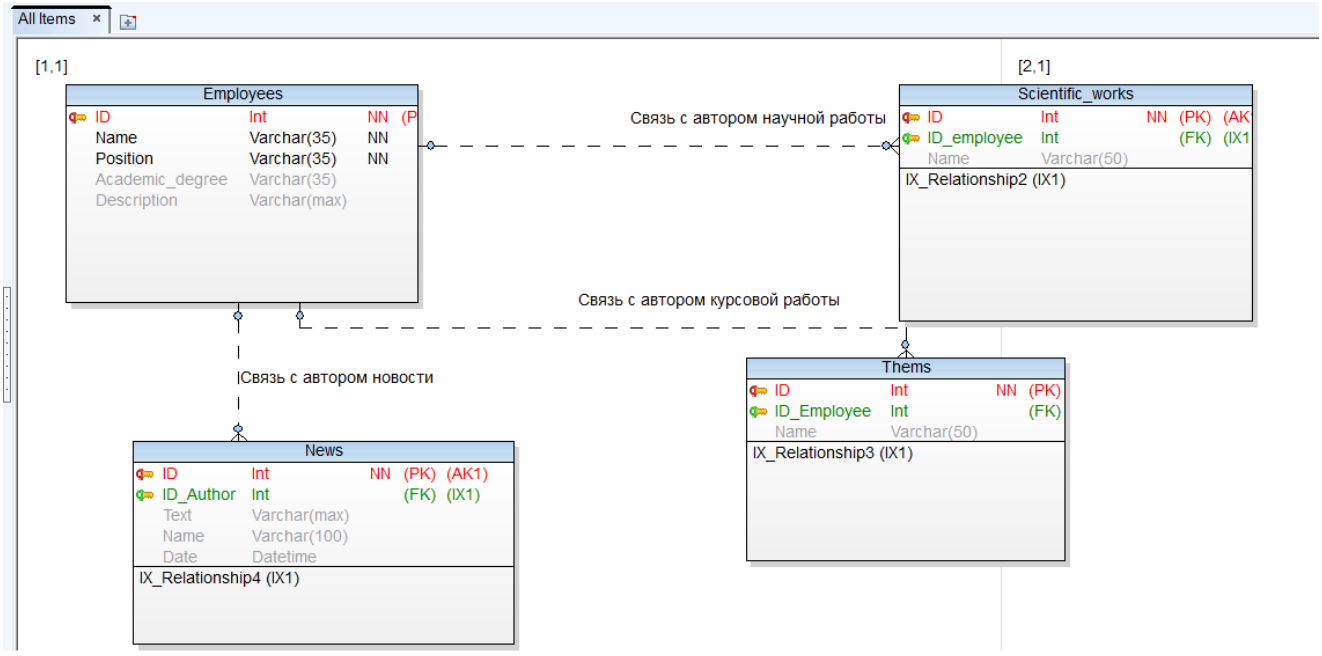

Рисунок 3.1.1

Далее следует генерация DDL скрипта, позволяющая автоматически создать базу данных в MS SQL Server

Для взаимодействия с базой данных будет использоваться MS SQL Server Management Studio. После создания базы данных и вставки DDL скрипта имеется готовая к работе база данных с настроенными связями (Рисунок 3.1.2).

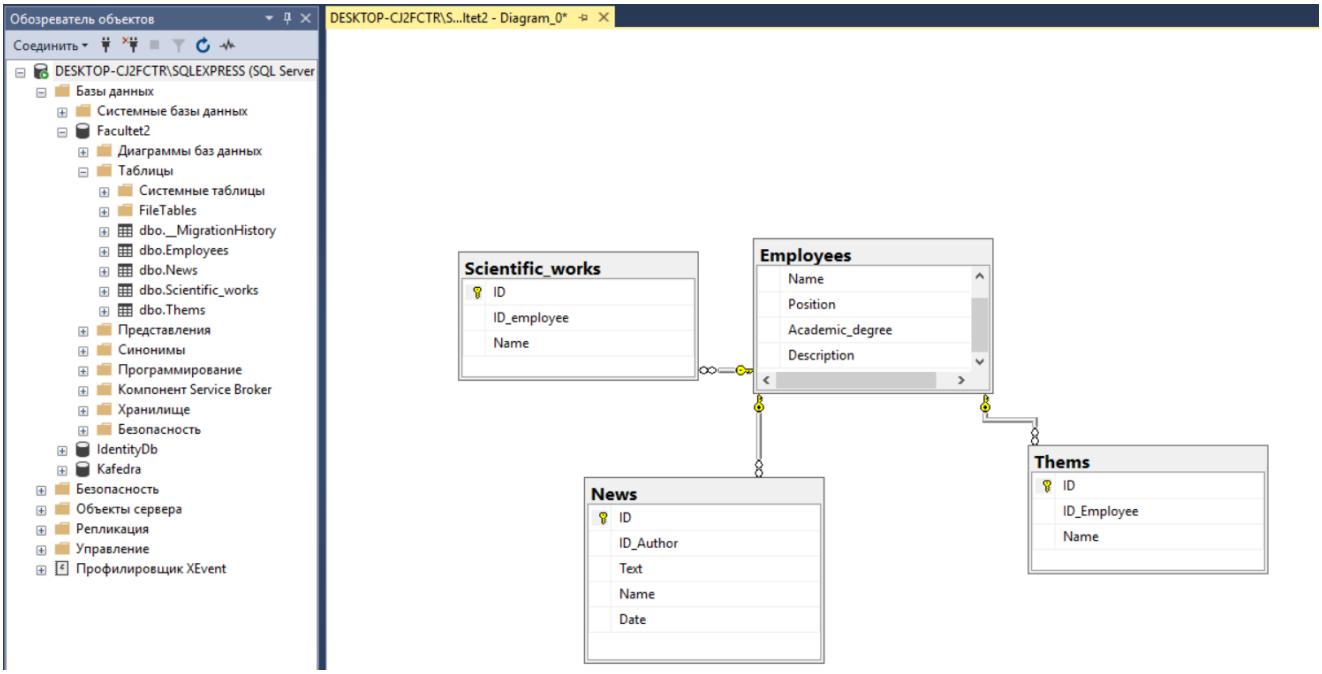

Рисунок 3.1.2

3.2 Начало работы с ASP.NET Framework и создание проекта

Откроем Visual Studio и в меню File (Файл) выберем пункт New Project... (Создать проект). Перед нами откроется диалоговое окно создания проекта, где нам надо перейти на подвкладку Web и выбрать шаблон ASP.NET Web Application (.NET Framework). Даём название проекту, нажмем ОК и перейдём к окну выбора шаблона нового приложения (рисунок 3.2.1).

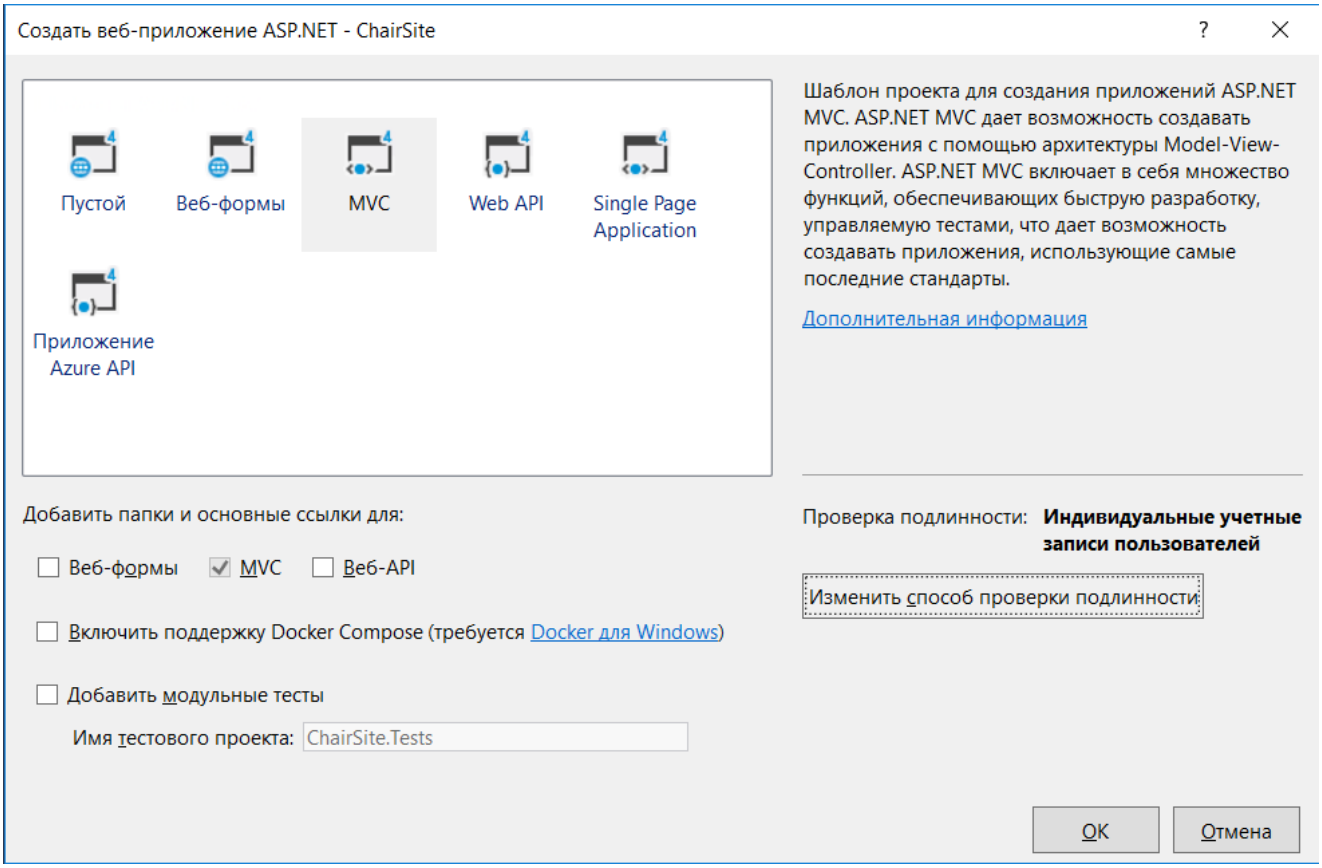

Рисунок 3.2.1

По умолчанию уже выбран шаблон MVC.

Кроме того, данное диалоговое окно позволяет задать опции тестирования.

Также нам доступен в правой части окна выбор механизма аутентификации в приложении (кнопка Change Authentication). По умолчанию установлен тип No Authentication, который подразумевает отсутствие какой-либо системы аутентификации. Но мы будем использовать метод проверки подлинности «**Индивидуальные учетные записи**

пользователей» позволяющий проходить авторизацию на сайте и хранить учетные записи в базе данных. Подробнее об авторизации будет рассказано в главе 4.4.

В конечном итоге мы получим проект, содержащий разветвленную структуру и имеющий некоторое наполнение по умолчанию.

Для генерации классов используем Entity Framework. В контекстном меню проекта добавим компонент под названием «Модель ADO.NET EDM». На следующем шаге мастера укажем «Code First из базы данных». После этого сконфигурируем подключение к ранее созданной базе данных. Укажем таблицы и представления, включаемые в модель. После подключения базы данных в обозревателе решений появиться четыре класса: **Employee.cs, News.cs, Scintific_works.cs, Thems.cs.** Таким образом, мы построили «мост» между БД и приложением. Созданные классы будем использовать в качестве моделей.

3.3 Разработка архитектуры проекта

Весь функционал проекта обеспечен благодаря следующей структуре, изображенной на рисунке 3.3.1

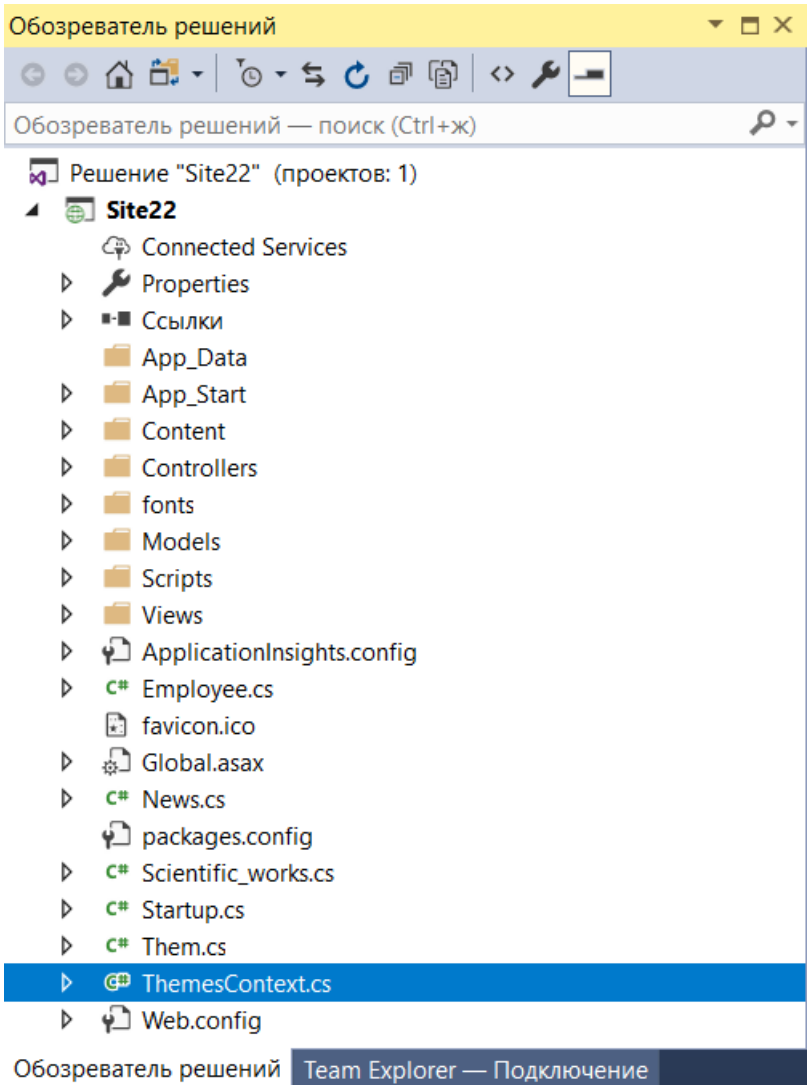

Рисунок 1.3.1

Краткая характеристика данных файлов и папок:

— **App_Data**: содержит файлы, ресурсы и базы данных, используемые приложением

— **App_Start**: хранит ряд статических файлов, которые содержат логику инициализации приложения при запуске

— **Content**: содержит вспомогательные файлы, которые не включают код на c# или javascript, и которые развертываются вместе с приложением, например, файлы стилей css

— **Controllers**: содержит файлы классов контроллеров. По умолчанию в эту папку добавляются два контроллера - HomeController и AccountController

— **fonts**: хранит дополнительные файлы шрифтов, используемых приложением

— **Models**: содержит файлы моделей. По умолчанию Visual Studio добавляет пару моделей, описывающих учетную запись и служащих для аутентификации пользователя

— **Scripts**: каталог со скриптами и библиотеками на языке javascript

— **Views**: здесь хранятся представления. Все представления группируются по папкам, каждая из которых соответствует одному контроллеру. После обработки запроса контроллер отправляет одно из этих представлений клиенту. Также здесь имеется каталог Shared, который содержит общие для всех представления

— **Global.asax**: файл, запускающийся при старте приложения и выполняющий начальную инициализацию. Как правило, здесь срабатывают методы классов, определенных в папке App_Start

— **packages.config**: файл, который содержит установленные в проект пакеты Nuget

— **Web.config**: файл конфигурации приложения

3.3.1 Контроллер и его методы

Так как с моделями и настройкой контекста данных мы закончили, то займемся другим компонентом приложения - контроллером.

Контроллер является центральным компонентом в архитектуре MVC. Контроллер получает ввод пользователя, обрабатывает его и посылает обратно результат обработки, например, в виде представления.

При использовании контроллеров существуют некоторые условности. Так, по соглашениям об именовании названия контроллеров должны оканчиваться на суффикс "Controller", остальная же часть до этого суффикса считается именем контроллера.

Чтобы обратиться контроллеру из веб-браузера, необходимо в адресной строке набрать адрес_сайта/Имя_контроллера/. Так, по запросу адрес_сайта/Home/ система маршрутизации по умолчанию вызовет метод Index контроллера HomeController для обработки входящего запроса. Если мы хотим отправить запрос к конкретному методу контроллера, то нужно указывать этот метод явно: адрес_сайта/Имя_контроллера/Метод_контроллера,

Для контроллеров предназначена папка Controllers. По умолчанию при создании проекта в нее добавляется контроллер HomeController, который практически не имеет никакой функциональности.

Прежде всего начнем разработку с подключения пространства имен моделей, даже несмотря на то, что он находятся в одном проекте, но в разных пространствах. Затем создается объект контекста данных, через который мы будем взаимодействовать с базой данных:

```
ThemesContext db = new ThemesContext();
```

Создадим метод «FilteredBrowse», который будет выводить на страницу список преподавателей факультета. При этом в данном методе нам необходимо будет реализовать фильтрацию по должности и по учёной степени, а также решить проблему пагинации. В качестве входных данных

будем использовать нужные нам названия должности и учёной степени, также будем передавать в метод номер текущей страницы (по умолчанию – 1)

Поскольку пагинация страниц будет использоваться не один раз, создадим для нее отдельную модель, в состав которой будет входить номер текущей страницы, количество объектов на странице, общее количество объектов и общее количество страниц.

Создадим модель страницы под названием «FilteredWorks» , в которой будут присутствовать условия фильтрации, выбранная должность и степень, а также информация для пагинации.

Для работы с SQL запросами здесь и далее будем использовать язык LINQ.

Создадим фильтрацию по должностям

```
if (!String.IsNullOrEmpty(Position) && !Position.Equals("Все"))
            { employees = employees.Where(p => p.Position == Position);}
```

Фильтрация по научной степени делается аналогично.

После расчета количества объектов на странице передаем всю необходимую информацию в модель, а после этого модель передаем в представление.

```
List<Employee> teachers = db.Employees.ToList();
int pageSize = 5; // количество объектов на страницу
IEnumerable<Employee> emoloyeesPerPages = employees.OrderBy(p => p.ID).Skip((page - 1) *
pageSize).Take(pageSize);
PageInfo pageInfo = new PageInfo { PageNumber = page, PageSize = pageSize, TotalItems =
employees.Count() };
FilteredWorks fw = new FilteredWorks
            {
                Employees = emoloyeesPerPages.ToList(),
                Positions = new SelectList(new List<string>() { "Все", "Секретарь",
"Зав. кафедры", "Заместитель зав. кафедры", "Преподаватель", "Доцент" }),
                Scien = new SelectList(new List<string>() { "Все", "Бакалавр",
"Магистр", "Кандидат наук", "Доктор наук" }),
                SelectedPosition = Position,
                isAdmin = IsAdmin(),
                SelectedAcadem_degree = Academ,
                PageInfo = pageInfo,
            };
            return View(fw);
```

Создадим метод «News». Структура данного метода аналогична предыдущему с той лишь разницей, что в нем мы не будем использовать

фильтрацию. Стоить упомянуть о том, что при добавлении новостей их первичный ключ будет увеличиваться, соответственно для того, чтобы на первых страницах выводились последние добавленные новости необходимо использовать сортировку по убыванию. Данный метод будет иметь следующую структуру:

```
  public ActionResult News(int? ID = 1, int page = 1)
        {
            IQueryable<News> news = db.News;
            News n = news.Where(p => p.ID == ID).FirstOrDefault();
            List<Employee> employees = db.Employees.Where(p => p.ID ==
n.ID_Author).ToList();
            Employee e = employees.FirstOrDefault();
            int pageSize = 2; // количество объектов на страницу
            IEnumerable<News> NewsPerPages = news.OrderByDescending(p =>
p.ID).Skip((page - 1) * pageSize).Take(pageSize);

            PageInfo pageInfo = new PageInfo { PageNumber = page, PageSize = pageSize,
TotalItems = news.Count() };

             NewsHelp nw = new NewsHelp
            {
                news = NewsPerPages,

                PageInfo = pageInfo

            };
            ViewBag.Author = e.Name;
            return View(nw);
        }
```

Следующим этапом следует создание метода «AboutEmployee», главной задачей которого является отображение подробной информации о сотруднике кафедры, в качестве входящего параметра принимается первичный ключ ID сотрудника. Запрос к базе данных на языке LINQ выглядит следующим образом:

```
Employee e = employees.Where(p => p.ID == ID).FirstOrDefault();
```

Поскольку у одного сотрудника может быть несколько курсовых тем или научных работ, будем использовать компонент ViewData. Пример использования данного компонента:

```
IList<Them> thems = db.Thems.Where(p => p.ID_Employee == ID).ToList();
ViewData["Thems"] = thems;
```

Следующим шагом является создание методов для создания новостей и для создания/удаления/изменения информации о сотрудниках. Поскольку

запросы бывают разных типов, например, GET и POST, фреймворк ASP.NET MVC позволяет определить тип обрабатываемого запроса для действия, применив к нему соответствующий атрибут: [HttpGet], [HttpPost], поэтому создание данных методов предполагает разделение создание двух методов, по одному для каждого типа запросов.

Метод «EditInfo» с атрибутом [HttpGet] выводит текущую информацию о пользователе по его ID:

```
Employee employee = db.Employees.Find(id);
if (employee != null)
            {
                return View(employee);
            }
```

Метод «EditInfo» с атрибутом [HttpPost] принимает в качестве входящего параметра объект модели сотрудника и сохраняет этот объект в базе данных:

```
[HttpPost]
public ActionResult EditInfo(Employee employee)
        {
            db.Entry(employee).State = EntityState.Modified;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
```

Метод «Create», предназначенный для добавления нового сотрудника в базу данных, создается по аналогии с предыдущим методом с разницей лишь в том, что в методе с атрибутом [HttpGet] мы создаем новый пустой объект и передаем его в представление для заполнения:

```
[HttpGet]
public ActionResult create()
        {
            Employee employee = new Employee();
            return View(employee);
        }
```

Метод «Delete», необходимый для удаления сотрудника из базы данных, во многом схож с предыдущими. Мы так же перед удалением выводим информацию о сотруднике, после этого удаляем его простым действием:

```
Employee b = db.Employees.Find(id);
            if (b != null)
            {
                db.Employees.Remove(b);
```

```
            db.SaveChanges();
        }
```

В методе «CreateNews», который предназначен для создания новостей нам необходимо узнать, какой из сотрудников авторизован на сайте на данный момент. Для этого будем использовать контекст «ApplicationUserManager», позволяющий работать с текущим авторизованным пользователем, а также контекст «ApplicationUser», необходимый для взаимодействия с базой данных, в которой хранятся индивидуальные учётные записи пользователей.

Таким образом метод примет следующий вид:

```
[HttpPost]
public ActionResult CreateNews(News news)
        {
            ApplicationUserManager userManager = HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

           ApplicationUser user = userManager.FindByName(User.Identity.Name);
            news.ID_Author = db.Employees.Where(m => m.Name == user.Email).Select(m => m.ID).Single();

            db.Entry(news).State = EntityState.Added;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
```

В результате проделанной работы мы получили контроллер «Home» наполненный необходимыми нам методами.

3.3.2 Создание представлений

Хотя работа приложения MVC управляется главным образом контроллерами, но непосредственно пользователю приложение доступно в виде представления, которое и формирует внешний вид приложения.

В ASP.NET MVC 5 представления - это файлы с расширением cshtml, которые содержат код пользовательского интерфейса в основном на языке html. Хотя представление содержит, главным образом, код html, оно не является html-страницей. При компиляции приложения на основе требуемого представления сначала генерируется класс на языке C#, а затем этот класс компилируется.

Все добавляемые представления, как правило, группируются по контроллерам в соответствующие папки в каталоге Views. Представления, которые относятся к методам контроллера Home, будут находиться в проекте в папке Views/Home. Однако при необходимости мы сами можем создать в каталоге Views папку с произвольным именем, где будем хранить дополнительные представления, необязательно связанные с определенными методами контроллера.

Чтобы произвести рендеринг представления в выходной поток, используется метод View(). Если в этот метод не передается имени представления, то по умолчанию приложение будет работать с тем представлением, имя которого совпадает с именем метода действия.

Стандартное представление очень похоже на обычную веб-страницу с кодом html. Однако оно также имеет вставки кода на C#, которые предваряются знаком @. Этот знак используется движком представлений Razor для перехода к коду на языке C#. Чтобы понять суть работы движка Razor и его синтаксиса, вначале посмотрим, что представляют из себя движки представлений.

При вызове метода View контроллер не производит рендеринг представления и не генерирует разметку html. Контроллер только готовит данные и выбирает, какое представление надо возвратить в качестве объекта ViewResult. Затем уже объект ViewResult обращается к движку представления для рендеринга представления в выходной результат.

Если ранее предыдущие версии ASP.NET MVC и Visual Studio по умолчанию поддерживали два движка представлений - движок Web Forms и движок Razor, то сейчас Razor в силу своей простоты и легкости стал единственным движком по умолчанию. Использование Razor позволило уменьшить синтаксис при вызове кода C#, сделать сам код более "чистым".

Здесь важно понимать, что Razor - это не какой-то новый язык, это лишь способ рендеринга представлений, который имеет определенный синтаксис для перехода от разметки html к коду C#.

Использование синтаксиса Razor характеризуется тем, что перед выражением кода стоит знак @, после которого осуществляется переход к коду C#.

Приступим к созданию представлений для ранее написанных контроллеров.

Для создания представления достаточно кликнуть по имени контроллера и в появившемся контекстном меню выбрать пункт «Добавить представление».

Шаблон создания представления имеет целый ряд гибких настроек, но мы будем использовать пустой шаблон.

Для начала создадим представление для контроллера «FilteredWorks», который отображает нам сотрудников кафедры, с возможность фильтрации.

Для начала укажем какая модель будет использоваться в представлении. Делается это следующим образом:

```
@model Site22.Models.FilteredWorks
```

Далее нам необходимо получить от пользователя условия фильтрации. Для этого создадим два выпадающих меню, а затем передадим их в контроллер для данного представления. В результате получим следующую разметку:

```
<form method="get">
    <div class="form-inline">
        <label class="control-label"> Ученая степень: </label>
        @Html.DropDownList("Academ", Model.Scien as SelectList, htmlAttributes: new {
@class = "form-control" })
        <label class="control-label">Дожность: </label>
        @Html.DropDownList("Position", Model.Positions as SelectList, htmlAttributes:
new { @class = "form-control" })
        <input type="submit" value="Фильтр" class="btn btn-default" />

    </div>
</form>
```

После получения информации от пользователя выведем на страницу список преподавателей, подходящих по данному фильтру. Для этого используем таблицу. Для отображения элементов модели (преподавателей) используется цикл foreach. Вывод самой информации происходит с помощью шаблонных хелперов Html.DisplayFor. Следует упомянуть что в нашем приложении по умолчанию используется css-фреймворк для создания адаптивных веб-приложений «Bootsrap». Данный фреймворк позволяет разделить страницу на 12 условных единиц и указать сколько из этих единиц будет занимать блок на разных разрешениях монитора. Например, строка *col-xs-12 col-md-8* означает то, что на экранах с средним разрешением блок будет занимать 8 условных единиц места, а на экранах с низким разрешением – 12. Таким образом страница будет содержать следующую разметку:

```
<div class="col-xs-12 col-md-8 ">
    <br>
    <table class="table">
        <tr>
            <th> Преподаватель </th>
            <th> Должность </th>
            <th> Учёная степень </th>
```

```
        </tr>

        @foreach (var item in Model.Employees)
        {
            <tr>
                <td>    @Html.DisplayFor(modelItem => item.Name         </td>
                <td>    @Html.DisplayFor(modelItem => item.Position)    </td>
                <td>    @Html.DisplayFor(modelItem => item.Academic_degree) </td>
                <td><p><a href="/home/aboutemployee/@item.ID">Подробнее</a></p></td>

            </tr>
        }
```

В конце страницы необходимо добавим пагинацию:

```
<div class="btn-group">
        @Html.PageLinks(Model.PageInfo, x => Url.Action("FilteredBrowse",
        new { page = x, Academ= Model.SelectedAcadem_degree,
            Position = Model.SelectedPosition
             }))
    </div>
```

Представление для отображения новостей делается аналогичным образом за исключением того, что там не применяется фильтрация.

Представление для отображения подробной информации о сотруднике имеет схожую структуру, но поскольку у сотрудника может быть несколько тем курсовых работ или научных работ, мы будем использовать для них отдельный список:

```
        <td><p>Темы курсовых работ: </p></td>
                    <td>
                        <p>
                            @foreach (var std in ViewData["Thems"] as IList<Them>)
                            {
                                <li>
                                    @std.Name
                                </li>
                            }
                        </p>
                    </td>
        <td><p>Научные работы: </p></td>
             <td>
                <p>
                  @foreach (var std in ViewData["Scient"] as IList<Scientific_works>)
                   {
                      <li>
```

```
                    @std.Name
                </li>
            }
            </p>
        </td>
```

Перейдем к созданию представлений предназначенных для создания/изменения/удаления информации. Начнем с представления «EditInfo», позволяющего редактировать информацию о сотруднике.

Структура данного преставления схожа с структурами ранее рассмотренных преставлений. При создании данного представления будем использовать строго типизированный хелпер Html.EditorFor. Он Создает элемент разметки, который доступен для редактирования, для указанного свойства модели: Html.Editor("Name"). Пример:

```
<tr>
        <td><p>ФИО: </p></td>
        <td><p>@Html.EditorFor(model => model.Name)</p></td>
</tr>
```

Для отправки данных контроллеру – создадим кнопку отправки, а также добавим кнопку отмены изменений:

```
<input type="submit" value="Сохранить" />
@Html.ActionLink("Отменить изменения и вернуться на главную", "Index")
```

Представления «Create», «Delete», «CreateNews», предназначенные для добавления/удаления сотрудника и создание новой новости делаются аналогично.

Представления главной страницы, истории факультета и контактов будут статическими, поэтому заполним их как обычные html страницы.

Создадим мастер-страницу для удобной навигации по сайту.

Мастер-страницы используются для создания единообразного, унифицированного вида сайта. По сути мастер-страницы - это те же самые представления, но позволяющие включать в себя другие представления.

Например, можно определить на мастер-странице общие для всех остальных представлений элементы, а также подключить общие стили и скрипты.

В итоге нам не придется на каждом отдельном представлении прописывать путь к файлам стилей, а потом при необходимости его изменять.

По умолчанию при создании нового проекта ASP.NET MVC 5 в проект уже добавляется мастер-страница под названием _Layout.chtml, которую можно найти в каталоге Views/Shared. На первый взгляд это обычное представление за одним исключением: здесь используется метод @RenderBody(), который является заместителем и на место которого потом будут подставляться другие представления, использующие данную мастер-страницу. В итоге мы сможем легко установить для представлений веб-приложения единообразный стиль.

С помощью Html разметки создадим название кафедры, разместим логотип факультета. Для корректировки расположения объектов используем css, для этого в файле /contecnt/site.css добавим все необходимые изменения стиля.

Добавим навигационную панель с помощью класса navbar. С помощью класса .navbar-inverse сделаем навигационную панель темного цвета. Добавим пункты меню с помощью класса .nav и .navbar-nav. После этого добавим форму авторизации. По умолчанию она находится в правом углу, не будем это изменять.

В результате навигационная панель примет вид:

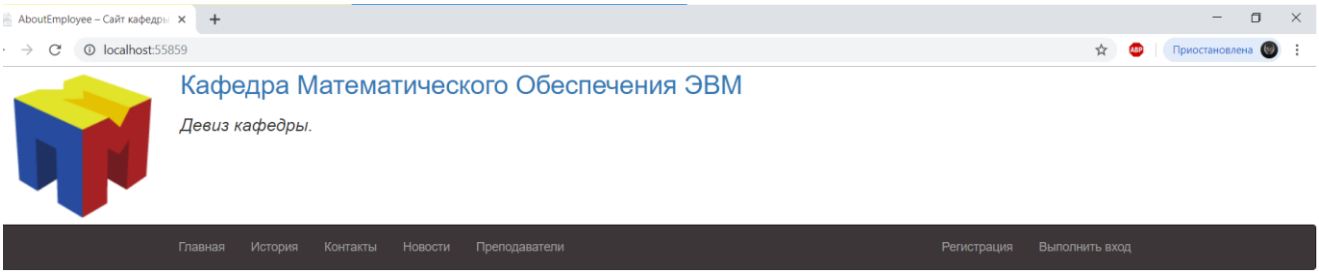

Рисунок 3.3.2.1

В ходе проделанной работы сайт получил окончательный внешний вид.

3.4 Аутентификация и авторизация

Большую роль в веб-приложениях играют механизмы авторизации и аутентификации. Они позволяют разграничить доступ для различных групп пользователей, а также идентифицировать пользователей.

Аутентификация - это процесс идентификации пользователя, то есть грубо говоря мы узнаем, что за пользователь посетил веб-приложение. А авторизация уже представляет процесс определения прав, которые могут быть даны аутентифицированному пользователю, его возможностей по доступу к ресурсам веб-приложения.

При выполнении работы воспользуемся встроенной системой авторизации и аутентификации в .NET приложениях под названием ASP.NET Identity. Ранее в главе 4.2 при создании решения мы выбрали метод проверки пользователей «**Индивидуальные учетные записи пользователей»** . Благодаря этому выбору созданный проект уже по умолчанию имеет всю необходимую для авторизации инфраструктуру: модели, контроллеры, представления. Если мы заглянем в узел References (Библиотеки), то увидим там ряд ключевых библиотек, которые и содержит необходимые для авторизации и аутентификации классы:

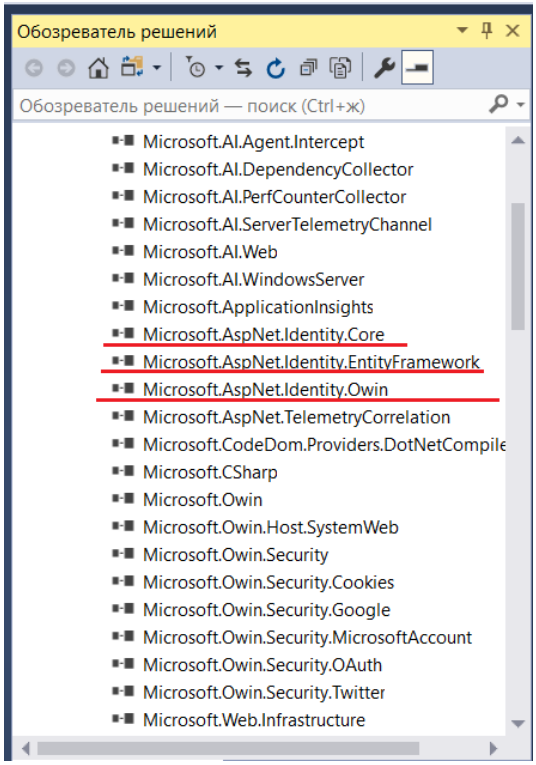

Рисунок 3.4.1

Ключевыми объектами в Asp.Net Identity являются пользователи и роли. Вся функциональность по созданию, удалению пользователей, взаимодействию с хранилищем пользователей хранится в классе **UserManager**. Для работы с ролями и их управлением в AspNet Identity определен класс **RoleManager**.

Помимо этого, была создана и присоединена база данных для хранения учётных записей пользователей, имеющая следующую структуру:

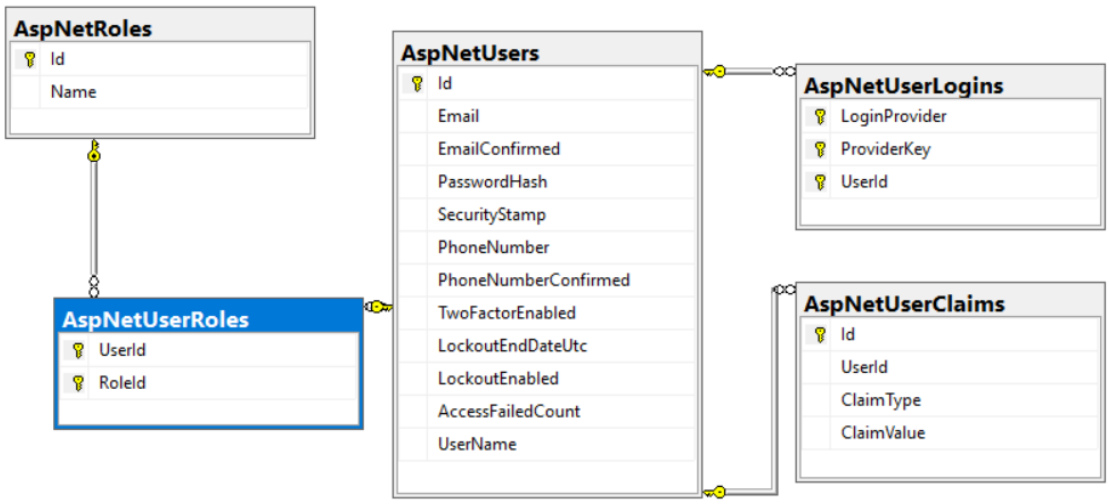

Рисунок 3.4.2

Рассмотрим таблицы, с которыми мы будем работать чаще всего:

— **AspNetRoles**: содержит определения ролей

— **AspNetUserRoles**: таблица, устанавливающая для пользователей определенные роли

— **AspNetUsers**: собственно, таблица пользователей. Если мы ее откроем, то увидим данные зарегистрированного пользователя

Поскольку в нашем проекте предполагается ручное создание пользователей и назначение ролей, добавим в таблицу **AspNetRoles** две роли: «**admin**» и «**user**».

В таблицу «**AspNetUsers**» добавим пользователей. Это также можно сделать, запустив сайт и вызвав метод «Регистрация», находящемся на мастер-странице.

В таблице «**AspNetUserRoles»** назначим роли созданным пользователям.

Роли позволяют создать группы пользователей с определенными правами и в зависимости от принадлежности к той или иной группе, разграничить доступ к ресурсам приложения. Для того что бы ограничить доступ к методу контроллера достаточно перед этим методом поставить атрибут проверки роли, например,:

```
[Authorize(Roles = "admin")]
```

При этом имя роли должно совпадать с одной из ролей, находящихся в таблице **«AspNetRoles»**. Ограничим использование методов «Create», «EditInfo» и «Delete» предназначенные для добавления нового сотрудника, редактирования информации о сотруднике и удаления сотрудника соответственно. Для этого перед методами укажем допустимую роль «admin». Метод создания новости «CreateNews» оставим доступным для пользователей с ролью «user». Остальные действия остаются по умолчанию общедоступными. В результате мы реализовали систему авторизации и разграничения ролей.

3.5 Тестирование и результат

Одно из преимуществ разработки на платформе ASP.NET MVC предоставляют богатые возможности по тестированию веб-приложения. Можно самим выполнять тестирование тех или иных моментов вручную, а можно использовать специальные небольшие программы, которые называются юнит-тесты.

Юнит-тесты позволяют быстро и автоматически протестировать отдельные участки кода независимо от остальной части программы. При надлежащем составлении юнит-тесты вполне могут покрыть большую часть кода приложения.

Большинство юнит-тестов так или иначе имеют ряд следующих признаков:

Тестирование небольших участков кода ("юнитов")

При создании юнит-тестов выбираются небольшие участки кода, которые надо протестировать. Как правило, тестируемый участок должен быть меньше класса, а в большинстве случаев тестируется отдельный метод класса. Упор на небольшие участки позволяет довольно быстро писать простенькие тесты.

Однажды написанный код нередко читают многократно, поэтому важно писать понятный код. Особенно это важно в юнит-тестах, где в случае неудачи при тестировании разработчик должен быстро прочитать исходный код и понять в чем проблема и как ее исправить. А использование небольших участков кода значительно упрощает подобную работу.

Тестирование в изоляции от остального кода

При тестировании важно изолировать тестируемый код от остальной программы, с которой он взаимодействует, чтобы потом четко определить возможность ошибок именно в этом изолированном коде. Что упрощает и повышает контроль над отдельными компонентами программы.

Тестирование только общедоступных конечных точек

Всего лишь небольшие изменения в классе могут привести к неудаче многих юнит-тестов, поскольку реализация используемого класса изменилась. Поэтому при написании юнит-тестов ограничиваются только общедоступными конечными точками, что позволяет изолировать юнит-тесты от многих деталей внутренней реализации компонента. В итоге уменьшается вероятность, что изменения в классах могут привести к провалу юнит-тестов.

Посмотрим на примере, как создавать юнит-тесты. По умолчанию при создании проекта в любой версии Visual Studio для создаваемого проекта веб-приложения нам уже предлагается включить дополнительный проект с тестами с помощью опции **Add unit tests**, но мы можем добавить тесты к уже существующему проекту, для этого необходимо добавить в решение новый тип проекта Unit Test Project.

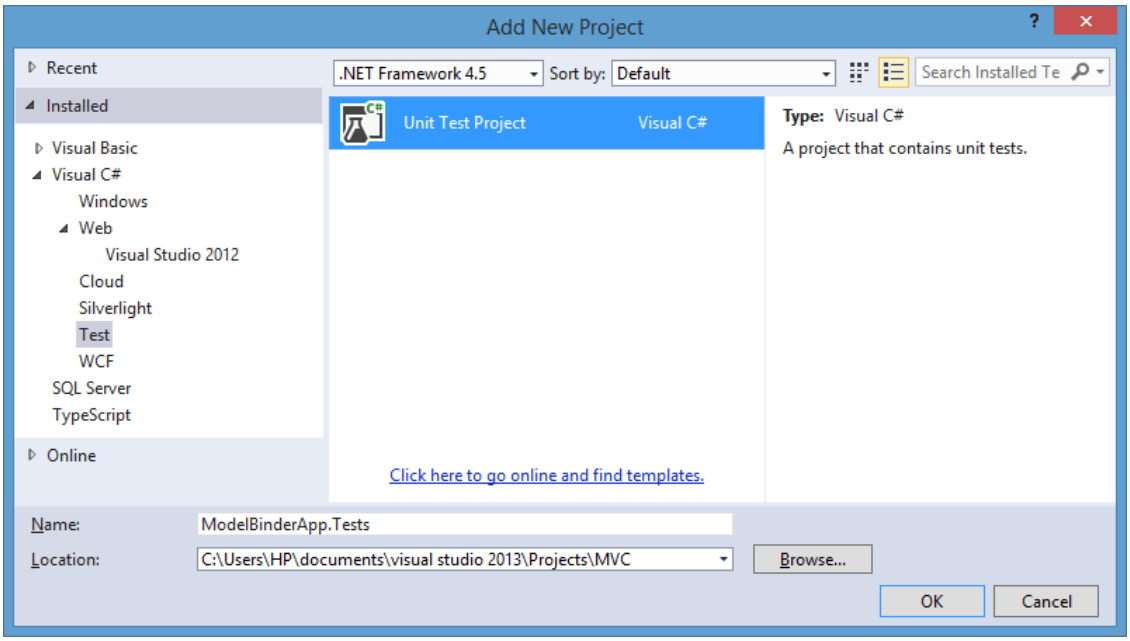

Рисунок 3.5.1

При одновременном создании обоих проектов по умолчанию тестовый проект уже содержит ряд тестов. При одновременном создании обоих проектов по умолчанию тестовый проект уже содержит ряд тестов. В частности для контроллера **HomeController** в проекте тестов будет создан класс HomeControllerTest. Этот простейший стандартный тест не охватывает

всех возможных ошибок, например, были ли сгенерировано нужное представление. Он просто призван дать начальное понимание работы тестов.

Теперь запустим тест на выполнение. Для этого перейдем в окно **Test Explorer** и нажмем в нем на кнопку **Run All.** Если все нормально, то обозреватель тестов сигнализирует нам зеленым цветом, что все тесты успешно пройдены:

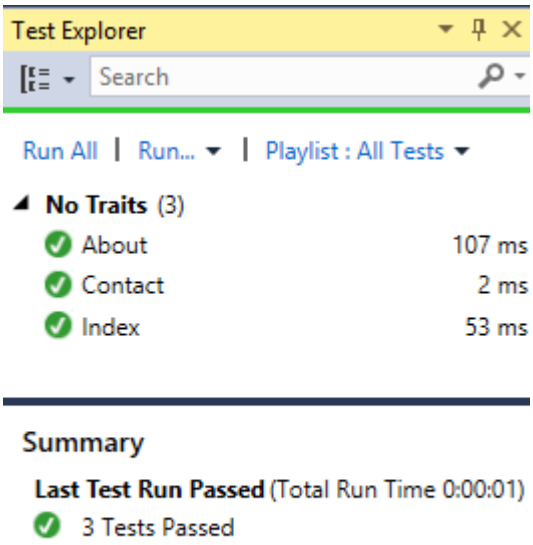

Рисунок 3.5.2

Результат

В результате проделанной работы получен полностью функционирующий веб-сайт кафедры «МО ЭВМ»:

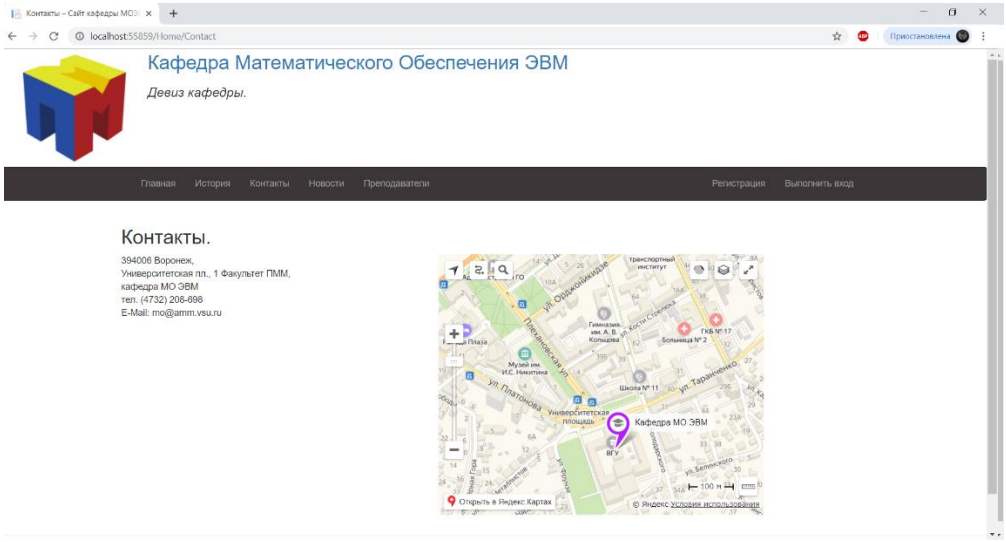

Рисунок 1

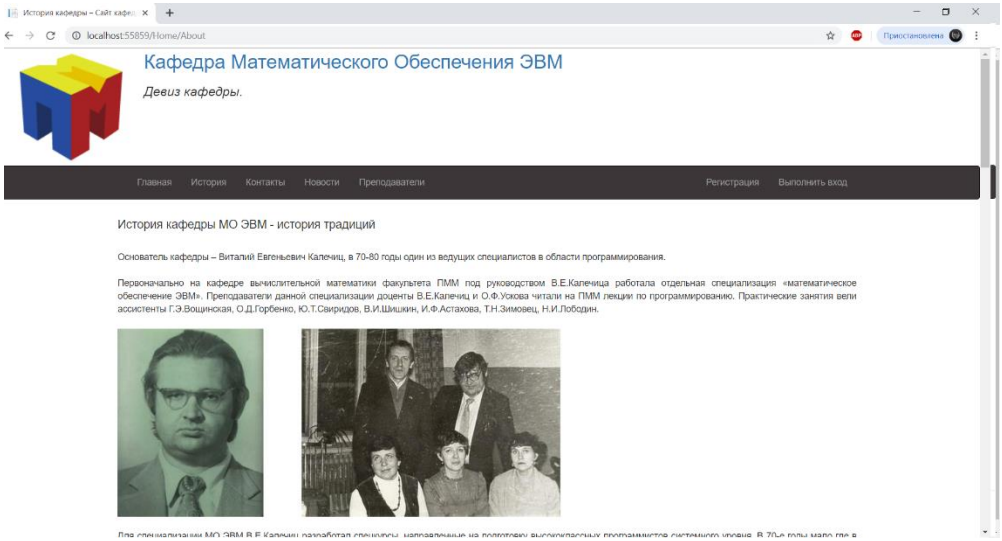

Рисунок 2

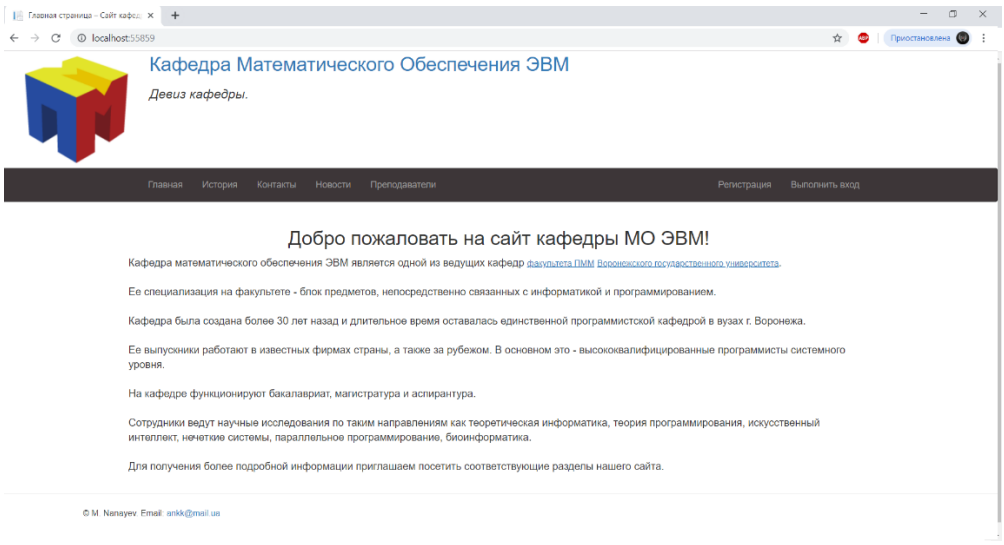

Рисунок 3

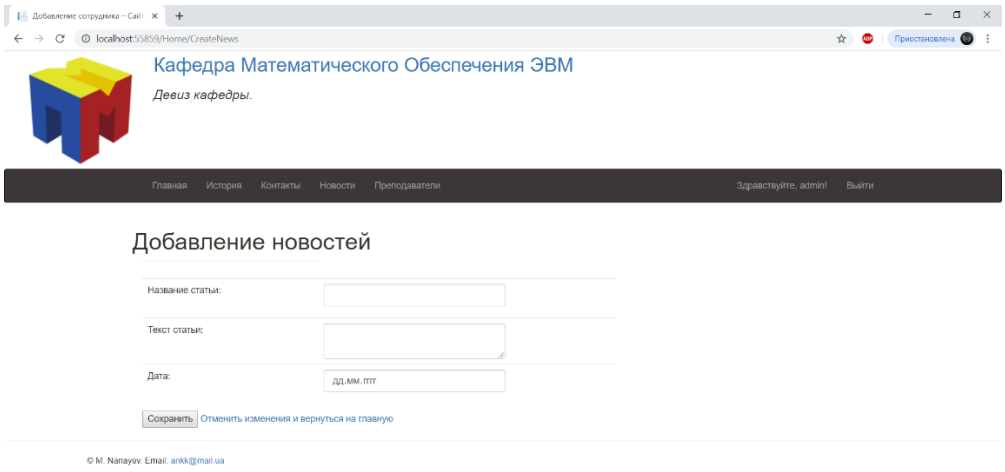

Рисунок 2

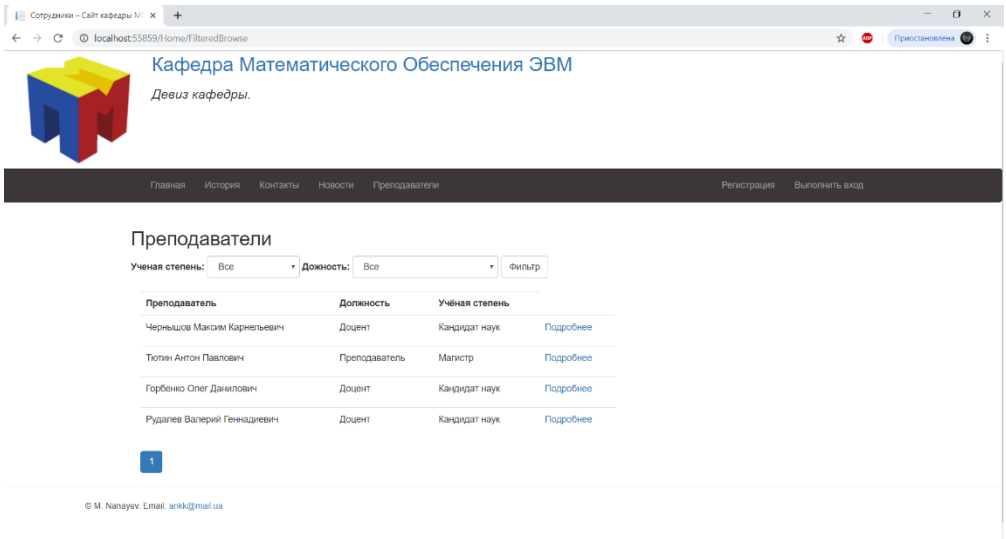

Рисунок 3

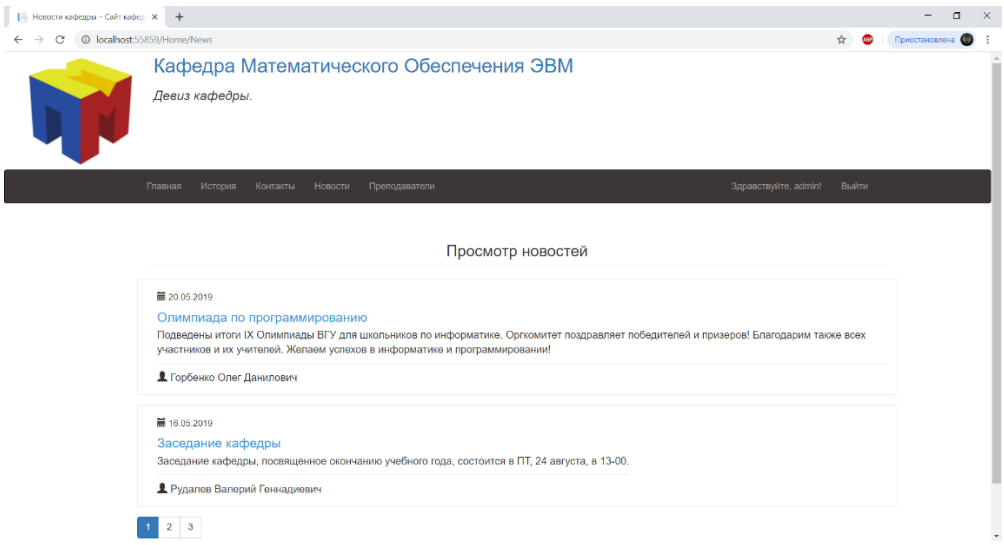
Кафедра Математического Обеспечения ЭВМ
Девиз кафедры.
Главная
История
Контакты
Новости
Преподаватели
Здравствуйте, admin!
Выйти
Просмотр новостей
20.05.2019
Олимпиада по программированию
Подведены итоги IX Олимпиады ВГУ для школьников по информатике. Оргкомитет поздравляет победителей и призеров! Благодарим также всех участников и их учителей. Желаем успехов в информатике и программировании!
Горбенко Олег Данилович
18.05.2019
Заседание кафедры
Заседание кафедры, посвященное окончанию учебного года, состоится в ПТ, 24 августа, в 13-00.
Рудаков Валерий Геннадиевич
1
2
3

Рисунок 4

Заключение и выводы

В результате проделанной работы была достигнута поставленная цель, а именно был создан сайт полностью функционирующий сайт кафедры с собственной базой данных.

Во время выполнения данной выпускной квалификационной работы были получены базовые навыки моделирования и создания базы данных, а так же клиентской и аппаратной части веб-сайта на базе фреймворка ASP.NET MVC 5

В перспективе усовершенствование и добавление нового с целью преобразования в расширенный инструмент автоматизации работы кафедры, с функциями учета успеваемости, инфографики, контроля за нагрузкой

Список литературы

1. Рудалёв В.Г. «Материал к курсу Интернет-технологии»
2. А. Фримен. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов: Пер. с англ. / А. Фримен. -5-е изд. –М : «ВИЛЬЯМС», 2014. -736с.
3. ASP.NET MVC 5 | Полное руководство. –URL: https://metanit.com/sharp/mvc5/
4. Учебник по Bootstrap | ИТ Шеф. –URL: https://itchief.ru/bootstrap/
5. ASP.NET MVC. –URL: https://habr.com/ru/post/175999/

Приложение 1.

Листинг контроллера HomeConroller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Data.Entity;
using Site22.Models;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using System.Data.Entity;

namespace Site22.Controllers
{
    public class HomeController : Controller

    {
        ThemesContext db = new ThemesContext(); // Основная бд
        ApplicationDbContext db1 = new ApplicationDbContext(); // бд авторизации

        public ActionResult Index()
        {
            return View();
        }

        [Authorize(Roles = "admin")]
        public ActionResult Admin()
        {
            return View();
        }


        [Authorize(Roles = "admin")]
        [HttpGet]
        public ActionResult EditInfo(int? id = 1) //Редактирование информации о
сотруднике
        {
            if (id == null)
            {
                return HttpNotFound();
            }
            Employee employee = db.Employees.Find(id);

            if (employee != null)
            {
                return View(employee);
            }
            return HttpNotFound();
        }
        [Authorize(Roles = "admin")]
        [HttpPost]
        public ActionResult EditInfo(Employee employee)
        {
            db.Entry(employee).State = EntityState.Modified;


            db.SaveChanges();
            return RedirectToAction("Index");
        }
        //добавление нового сотрудника в базу данных
        [Authorize(Roles = "admin")]
        [HttpGet]
```

```
        public ActionResult create()
        {
            Employee employee = new Employee();
            return View(employee);
        }
        [Authorize(Roles = "admin")]
        [HttpPost]
        public ActionResult create(Employee employee)
        {
            int maxID = db.Employees.Select(m => m.ID).Max();
            employee.ID = maxID + 1;
            db.Entry(employee).State = EntityState.Added;
            db.SaveChanges();

            return RedirectToAction("Index");
        }

        //Удаление сотрудника из базы данных
        [Authorize(Roles = "admin")]
        public ActionResult Delete(int id)
        {
            Employee b = db.Employees.Find(id);
            if (b != null)
            {
                db.Employees.Remove(b);
                db.SaveChanges();
            }
            return RedirectToAction("Index");
        }
        [Authorize(Roles = "admin")]
        [HttpGet]
        public ActionResult Delete(int? ID = 0)
        {
            Employee employee = db.Employees.Find(ID);
            if (employee != null)
                return View(employee);
            else
                return HttpNotFound();
        }

        //создание новой новости
        [HttpGet]
        [Authorize(Roles = "user")]
        public ActionResult CreateNews()
        {
            return View();
        }

        [Authorize(Roles = "user")]
        [HttpPost]
        public ActionResult CreateNews(News news)
        {
            ApplicationUserManager userManager =
HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();
            ApplicationUser user = userManager.FindByName(User.Identity.Name);
//User.Identity.Name - имя текущего пользователя
            int maxID = db.News.Select(m => m.ID).Max();
            news.ID = maxID + 1;
            news.ID_Author = db.Employees.Where(m => m.Name == user.Email).Select(m =>
m.ID).Single();
            db.Entry(news).State = EntityState.Added;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
```

```
        public ActionResult About()
        {
            return View();
        }

        public ActionResult Contact()
        {
            return View();
        }
        //подробная инф. о сотруднике
        [HttpGet]
        public ActionResult AboutEmployee(int? ID = 2)
        {
            IQueryable<Employee> employees = db.Employees;
            Employee e = employees.Where(p => p.ID == ID).FirstOrDefault();

            IList<Them> thems = db.Thems.Where(p => p.ID_Employee == ID).ToList();
            IList<Scientific_works> Scient_w = db.Scientific_works.Where(p =>
p.ID_employee == ID).ToList();

            ViewBag.ID = ID;
            ViewBag.Name = e.Name;

            if (Scient_w == null || Scient_w.Count == 0)
                Scient_w.Add(new Scientific_works() { Name = "Работ нет" });

            ViewData["Scient"] = Scient_w;
            if (thems == null || thems.Count == 0)
                thems.Add(new Them() { Name = "Тем курсовых нет" });
            ViewData["Thems"] = thems;
            ViewBag.Academ = e.Academic_degree;
            ViewBag.Position = e.Position;
            ViewBag.Description = e.Description;

            return View();
        }
        //вывод новостей
        [HttpGet]
        public ActionResult News(int? ID = 1, int page = 1)
        {

            IQueryable<News> news = db.News;
            News n = news.Where(p => p.ID == ID).FirstOrDefault();
            List<Employee> employees = db.Employees.Where(p => p.ID ==
n.ID_Author).ToList();
            Employee e = employees.FirstOrDefault();
            int pageSize = 2; // количество объектов на страницу
            IEnumerable<News> NewsPerPages = news.OrderByDescending(p =>
p.ID).Skip((page - 1) * pageSize).Take(pageSize);
            PageInfo pageInfo = new PageInfo { PageNumber = page, PageSize = pageSize,
TotalItems = news.Count() };
            NewsHelp nw = new NewsHelp
            {
                news = NewsPerPages,

                PageInfo = pageInfo

            };
            ViewBag.Author = e.Name;


            return View(nw);
        }
```

```
        //вывод сотрудников
        public ActionResult FilteredBrowse(string Position, string Academ, int page = 1)
        {
            IQueryable<Employee> employees = db.Employees;
            if (!String.IsNullOrEmpty(Academ) && !Academ.Equals("Все"))
            {
                employees = employees.Where(p => p.Academic_degree == Academ);
            }
            if (!String.IsNullOrEmpty(Position) && !Position.Equals("Все"))
            {
                employees = employees.Where(p => p.Position == Position);
            }

            List<Employee> teachers = db.Employees.ToList();
            int pageSize = 5; // количество объектов на страницу
            IEnumerable<Employee> emoloyeesPerPages = employees.OrderBy(p =>
p.ID).Skip((page - 1) * pageSize).Take(pageSize);
            PageInfo pageInfo = new PageInfo { PageNumber = page, PageSize = pageSize,
TotalItems = employees.Count() };

            FilteredWorks fw = new FilteredWorks
            {
                Employees = emoloyeesPerPages.ToList(),
                Positions = new SelectList(new List<string>() { "Все", "Секретарь",
"Зав. кафедры", "Заместитель зав. кафедры", "Преподаватель", "Доцент" }),
                Scien = new SelectList(new List<string>() { "Все", "Бакалавр",
"Магистр", "Кандидат наук", "Доктор наук" }),
                SelectedPosition = Position,
                isAdmin = IsAdmin(),
                SelectedAcadem_degree = Academ,
                PageInfo = pageInfo,
            };
            return View(fw);
        }

        //вспомогательная функция
        private bool IsAdmin()
        {
            return User.IsInRole("admin");
        }
    }
}
```

Приложение 2.

Листинг PageInfo.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Data.Entity;
namespace Site22.Models
{
    public class PageInfo
    {

        public int PageNumber { get; set; } // номер текущей страницы
        public int PageSize { get; set; } // кол-во объектов на странице
        public int TotalItems { get; set; } // всего объектов
        public int TotalPages  // всего страниц
        {
            get { return (int)Math.Ceiling((decimal)TotalItems / PageSize); }
        }


    }
    public class FilteredWorks
    {        // Список тем
        public IEnumerable<Employee> Employees { get; set; }        // Выбранные условия
фильтра
        public string SelectedPosition { get; set; }
        public string SelectedAcadem_degree { get; set; }
        public int ID { get; set; }
        public bool isAdmin { get; set; }

        // Элементы формы

        public SelectList Positions { get; set; }
        public SelectList Scien { get; set; }
        // Инфа для пагинации
        public PageInfo PageInfo { get; set; }
    }
    public class NewsHelp
    {
        public IEnumerable<News> news { get; set; }
        // Инфа для пагинации
        public PageInfo PageInfo { get; set; }
    }
}
```

Приложение 3.

Листинги представлений.

News.cshtml

```
@model Site22.Models.NewsHelp
@using Site22.Models
@{ ViewBag.Title = "Новости кафедры"; Layout = "~/Views/Shared/_Layout.cshtml"; }


<html lang="ru">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap -->
    <link href="/examples/vendors/bootstrap-3.3.7/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>


        <div class="col-xs-12 col-md-8">
            <div class="container">
                <h1 class="h3 text-center page-header">Просмотр новостей</h1>
            </div>
            @foreach (var item in Model.news)
            {
                <div class="container">
                    <div class="row">

                        <div class="wp-block property list">
                            <div class="wp-block-body">
                                @*<div class="wp-block-img">
                                        <a href="#">
                                            <img src="/examples/images/image1-800-
530.jpg" alt="">
                                        </a>
                                    </div>*@
                                <div class="wp-block-content">
                                    <small>
                                        <span class="glyphicon glyphicon-calendar" aria-
hidden="true"></span> @Html.DisplayFor(modelItem => item.Date)
                                    </small>
                                    <h4 class="content-title">@Html.DisplayFor(modelItem
=> item.Name)</h4>
                                    <p class="description">@Html.DisplayFor(modelItem =>
item.Text)</p>



                                    <span class="pull-md-right">
                                        <span class="capacity">
                                            <i class="fa fa-user"></i>  <span
class="glyphicon glyphicon-user"> </span>  @Html.DisplayFor(modelItem =>
item.Employee.Name)
                                        </span>
                                    </span>
                                </div>
                            </div>

                        </div>
```

```
                    </div>
                </div>

            }
            <div class="btn-group">
                @Html.PageLinks(Model.PageInfo, x => Url.Action("News",
                new { page = x
                     }))
            </div>
        </div>


        <!-- jQuery -->
        <script src="/examples/vendors/jquery/jquery-3.2.1.min.js"></script>
        <!-- Bootstrap -->
        <script src="/examples/vendors/bootstrap-3.3.7/js/bootstrap.min.js"></script>

</body>
</html>
```

FilteredBrowse.cshtml:

```
@model Site22.Models.FilteredWorks
@using Site22.Models

@{ ViewBag.Title = "Сотрудники"; Layout = "~/Views/Shared/_Layout.cshtml"; }
<h2>Преподаватели</h2>
<form method="get">
    <div class="form-inline">
        <label class="control-label"> Ученая степень: </label>
        @Html.DropDownList("Academ", Model.Scien as SelectList, htmlAttributes: new {
@class = "form-control" })
        <label class="control-label">Дожность: </label>
        @Html.DropDownList("Position", Model.Positions as SelectList, htmlAttributes:
new { @class = "form-control" })
        <input type="submit" value="Фильтр" class="btn btn-default" />

    </div>
</form>

<div class="col-xs-12 col-md-8 ">
    <br>
    <table class="table">
        <tr>
            <th> Преподаватель </th>
            <th> Должность </th>
            <th> Учёная степень </th>
        </tr>


        @foreach (var item in Model.Employees)
        {
            <tr>
                <td>    @Html.DisplayFor(modelItem => item.Name)         </td>
                <td>    @Html.DisplayFor(modelItem => item.Position)    </td>
                <td>    @Html.DisplayFor(modelItem => item.Academic_degree) </td>
                <td><p><a href="/home/aboutemployee/@item.ID">Подробнее</a></p></td>
                @if (Model.isAdmin)
                {
                    <td><p><a href="/home/EditInfo/@item.ID">Изменить</a></p></td>
                    <td><p><a href="/home/Delete/@item.ID">Удалить</a></p></td>
                }

            </tr>
        }
    </table>
```

```
    @if (Model.isAdmin)
    {
        <a class="btn btn-default" href="/home/Create/" role="button">Добавить
сотрудника</a>
    }

    <div class="btn-group">
        @Html.PageLinks(Model.PageInfo, x => Url.Action("FilteredBrowse",
        new { page = x, Academ= Model.SelectedAcadem_degree,
            Position = Model.SelectedPosition
             }))
    </div>
</div>
```

EditInfo.cshtml:

```
@model Employee

@{ ViewBag.Title = "Изменение информации о сотруднике";
    Layout = "~/Views/Shared/_Layout.cshtml"; }
@using (Html.BeginForm())
{

<h1> Редактировать информацию о сотруднике</h1>
    <div class="col-md-8">
        <table class="table">
            <tr>


                <td><p> @Html.HiddenFor(model => model.ID, htmlAttributes: new { @class
= "control-label col-md-2" })</p></td>
            </tr>
            <tr>

                <td><p>ФИО: </p></td>
                <td><p>@Html.EditorFor(model => model.Name, new { htmlAttributes = new {
@class = "form-control" } })</p></td>
            </tr>
            <tr>
                <td>Должность: </td>
                <td> @Html.EditorFor(model => model.Position, new { htmlAttributes = new
{ @class = "form-control" } }) </td>
            </tr>
            <tr>
                <td>Ученая степень: </td>
                <td> @Html.EditorFor(model => model.Academic_degree, new {
htmlAttributes = new { @class = "form-control" } }) </td>
            </tr>
            <tr>
                <td>Информация: </td>
                <td> @Html.EditorFor(model => model.Description, new { htmlAttributes =
new { @class = "form-control" } }) </td>
            </tr>


        </table>
        <input type="submit" value="Сохранить" />
        @Html.ActionLink("Отменить изменения и вернуться на главную", "Index")
    </div>
}
```

Delete.cshtml:

```
@model Employee


@{ ViewBag.Title = "Добавление сотрудника";
    Layout = "~/Views/Shared/_Layout.cshtml"; }
```

```
@using (Html.BeginForm())
{

    <h1> Удаление сотрудника </h1>
    <div class="col-md-8">

        <table class="table">

            <tr>


                <td><p> @Html.HiddenFor(model => model.ID, htmlAttributes: new { @class
= "control-label col-md-2" })</p></td>
            </tr>
            <tr>

                <td><p>ФИО: </p></td>
                <td><p>@Html.DisplayFor(model => model.Name, new { htmlAttributes = new
{ @class = "form-control" } })</p></td>
            </tr>
            <tr>
                <td>Должность: </td>
                <td> @Html.DisplayFor(model => model.Position, new { htmlAttributes =
new { @class = "form-control" } }) </td>
            </tr>
            <tr>
                <td>Ученая степень: </td>
                <td> @Html.DisplayFor(model => model.Academic_degree, new {
htmlAttributes = new { @class = "form-control" } }) </td>
            </tr>
            <tr>
                <td>Информация: </td>
                <td> @Html.DisplayFor(model => model.Description, new { htmlAttributes =
new { @class = "form-control" } }) </td>
            </tr>


        </table>
        <input type="submit" value="Удалить" />
        @Html.ActionLink("Отменить изменения и вернуться к списку", "Index")
    </div>
}
```

CreateNews.cshtml:

```
@model News


@{ ViewBag.Title = "Добавление сотрудника";
    Layout = "~/Views/Shared/_Layout.cshtml"; }

@using (Html.BeginForm())
{

    <h1> Добавление новостей</h1>
    <div class="col-md-8">

        <table class="table">

            <tr>


                <td><p> @Html.HiddenFor(model => model.ID, htmlAttributes: new { @class
= "control-label col-md-2" })</p></td>
            </tr>
            <tr>
```

```
                <td><p>Название статьи: </p></td>
                <td><p>@Html.EditorFor(model => model.Name, new { htmlAttributes = new {
@class = "form-control" } })</p></td>
            </tr>
            <tr>
                <td>Текст статьи: </td>
                <td> @Html.EditorFor(model => model.Text, new { htmlAttributes = new {
@class = "form-control" } }) </td>
            </tr>
            <tr>
                <td>Дата: </td>
                <td> @Html.EditorFor(model => model.Date, new { htmlAttributes = new {
@class = "form-control" } }) </td>
            </tr>


        </table>
        <input type="submit" value="Сохранить" />
        @Html.ActionLink("Отменить изменения и вернуться на главную", "Index")
    </div>
}
```

Create.cshtml:

```
@model Employee


@{ ViewBag.Title = "Добавление сотрудника";
    Layout = "~/Views/Shared/_Layout.cshtml"; }

@using (Html.BeginForm())
{

    <h1> Добавление нового сотрудника </h1>
    <div class="col-md-8">

        <table class="table">

            <tr>


                <td><p> @Html.HiddenFor(model => model.ID, htmlAttributes: new { @class
= "control-label col-md-2" })</p></td>
            </tr>
            <tr>

                <td><p>ФИО: </p></td>
                <td><p>@Html.EditorFor(model => model.Name, new { htmlAttributes = new {
@class = "form-control" } })</p></td>
            </tr>
            <tr>
                <td>Должность: </td>
                <td> @Html.EditorFor(model => model.Position, new { htmlAttributes = new
{ @class = "form-control" } }) </td>
            </tr>
            <tr>
                <td>Ученая степень: </td>
                <td> @Html.EditorFor(model => model.Academic_degree, new {
htmlAttributes = new { @class = "form-control" } }) </td>
            </tr>
            <tr>
                <td>Информация: </td>
                <td> @Html.EditorFor(model => model.Description, new { htmlAttributes =
new { @class = "form-control" } }) </td>
            </tr>
```

```
        </table>
        <input type="submit" value="Сохранить" />
        @Html.ActionLink("Отменить изменения и вернуться к списку", "Index")
    </div>
}
```

AboutEmployee.cshtml:

```
@{ ViewBag.Title = "Информация о сотрудниках";
    Layout = "~/Views/Shared/_Layout.cshtml"; }
<h4>Выбрано:</h4>
<form method="post" action="">
    <input type="hidden" value="@ViewBag.ID" name="EmplyeeId" />
   <div class="col-md-8">
       <table class="table">
           <tr>
               <td><p>Преподаватель: </p></td>
               <td><p>@ViewBag.Name</p></td>
           </tr>
           <tr>
               <td>Должность: </td>
               <td> @ViewBag.Position </td>
           </tr>
           <tr>
               <td>Ученая степень: </td>
               <td> @ViewBag.Academ </td>
           </tr>
           <tr>
               <td><p>Темы курсовых работ: </p></td>
               <td>
                   <p>
                       @foreach (var std in ViewData["Thems"] as IList<Them>)
                       {
                           <li>
                               @std.Name
                           </li>
                       }
                   </p>
               </td>
           </tr>
           <tr>
               <td><p>Научные работы: </p></td>
               <td>
                   <p>
                       @foreach (var std in ViewData["Scient"] as
IList<Scientific_works>)
                       {
                           <li>
                               @std.Name
                           </li>
                       }
                   </p>
               </td>
           </tr>
           <tr>
               <td>Информация: </td>
               <td> @ViewBag.Description </td>
           </tr>


       </table>
       </div>
</form>
```

_Layout.cshtml:

```
<!DOCTYPE html>
```

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title – Сайт кафедры МОЭВМ</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")

</head>
<body>


    <a href="/home" class="custom-logo-link" rel="home" itemprop="url">
        <img src="~/Content/logo.png" class="custom-logo" alt="Кафедра МОЭВМ"
itemprop="logo">
    </a>
    <div id="site-identity">
        <p class="site-title"><a href="/home/index" rel="home">Кафедра Математического
Обеспечения ЭВМ</a></p>

        <p class="site-description">Девиз кафедры.</p>
    </div>

    <div class="navbar navbar-inverse" style="background-color: #3D3638;">
        <div class="container">
            <div class="navbar-header">



            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav navbar-left">
                    <li>@Html.ActionLink("Главная", "Index", "Home")</li>
                    <li>@Html.ActionLink("История", "About", "Home")</li>
                    <li>@Html.ActionLink("Контакты", "Contact", "Home")</li>
                    <li>@Html.ActionLink("Новости", "News", "Home")</li>
                    <li>@Html.ActionLink("Преподаватели", "FilteredBrowse", "Home")</li>

                </ul>
                @Html.Partial("_LoginPartial")
            </div>
        </div>
    </div>

    <div class="container body-content">
        @RenderBody()


    </div>

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>

</html>
<hr />
<footer>
    <div class="col-md-offset-1">
    <p><font size="2">&copy; M. Nanayev. Email: <a
href="mailto:ankk@mail.ua">ankk@mail.ua</a> </font></p>
        </div>
</footer>
```