Оглавление:

Введение

На современном этапе развития технологий сложно представить себе жизнь без Интернета. Сайты, размещенные в сети Интернет, позволяют быстро доносить информацию большому количеству лиц, привлекать новых участников, организовать обратную связь с посетителями и автоматизировать длительные по времени и регулярно повторяющиеся процессы. Благодаря таким сайтам, различные организации формируют свой образ далеко за пределами своего фактического местонахождения.

В качестве темы выпускной квалификационной работы была выбрана тема «Создание информационного сайта кафедры». Данная тема актуальна тем что, хоть у кафедры МО ЭВМ и есть сайт, в нем отсутствуют функции, которые позволяли бы выполнять регулярно повторяемые операции (такие как распределение нагрузки) в автоматическом режиме. Помимо этого, данный сайт устарел в моральном и техническом плане. В результате анализа данных проблем и было принято решение выбрать эту тему.

1. Цели и задачи

Целью данной выпускной квалификационной работы являлся процесс создания веб-сайта с использованием технологии, который будет соответствовать современным критериям сайтов учебных заведений. Сайт должен соответствовать следующим критериям:

- Иметь удобное представление информации
- Иметь удобное редактирование информации
- Быть быстро действенным
- Быть кроссплатформенным и кроссбраузерным
- Автоматизировать длительные по времени и регулярно проводимые операции

Достижение указанной цели осуществлялось путём решения следующих задач:

- Разработка структуры сайта и определение функциональных возможностей.
- Разработка структуры и создание базы данных.
- Разработка основных функций сайта: система авторизации, размещение новостей, вывод и редактирование информации о сотрудниках, автоматическое распределение учебной между преподавателями на основе учебного плана.
- Оформление и стилизация веб-сайта.

С учётом вышеперечисленных требований к функциональности сайта была составлена диаграмма вариантов использования сайта, изображенная на рисунке 1.

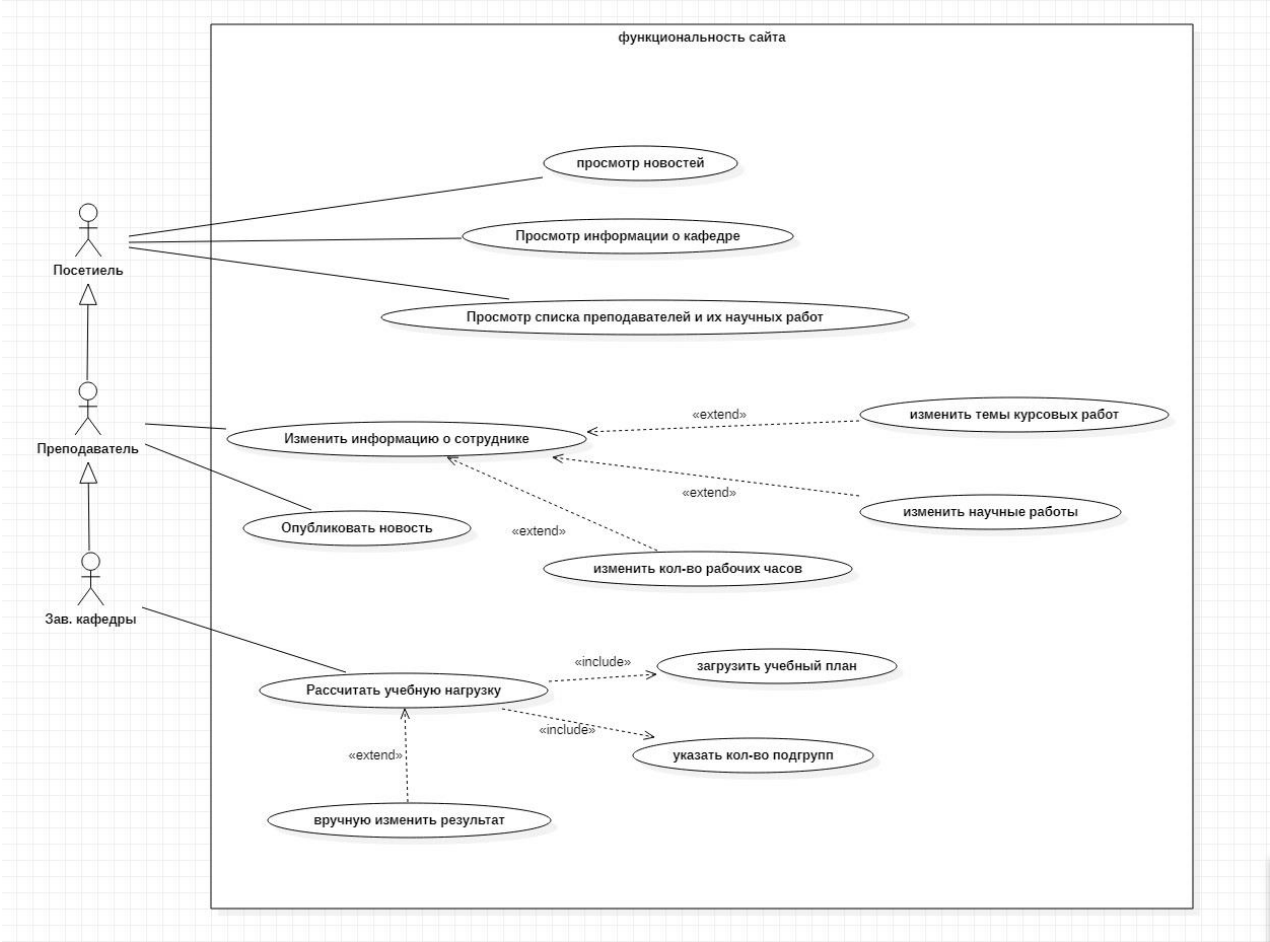


Рисунок 1 - диаграмма вариантов использования сайта

2. Описание средств разработки

Принимая во внимание все вышеперечисленные требования, было принято решение выбрать ASP.NET MVC 5 как основную технологию для разработки данного сайта. Начиная с 5 версии ASP.NET является кроссплатформенной технологией и может использоваться на операционных системах Windows и Linux.

Каждая технология, предназначенная для веб-разработки, имеет как достоинства, так и недостатки. Сравнивая ASP.NET с другими популярными технологиями веб-разработки, такими как Ruby-on-Rails, Node.js, PHP или Python, выделяется одно ключевое достоинство. Перечисленные средства являются интерпретируемыми, в то время как C# – компилируемый язык. Это предоставляет платформе .NET большое преимущество в скорости работы. Большая часть ошибок отлавливается разработчиком в момент компиляции, а все компоненты, не требуют интерпретатора, вместо этого работая с фреймворком, который, в свою очередь уже скомпилирован и вызывает напрямую функции операционной системы.

Как упоминалось ранее - разрабатываемый продукт является не компилируемым. И хотя язык C# является компилируемым, сам проект ASP.NET MVC 5 не требует сборки. Вместо этого при необходимости компиляция всех файлов проходит в режиме реального времени, а изменения можно вносить в код даже во время работы приложения.

В итоге, проект, созданный на платформе ASP.NET MVC 5 будет обладать одновременно двумя преимуществами: первое - возможность разработки с использованием большого количества интерпретируемых платформ, второе - быстродействие и высокая производительность создаваемых сервисов.

В качестве основного паттерна приложения был выбран стандартный шаблон MVC (Model-View-Controller). Данный шаблон разделяет приложение

на три основных части: модель приложения, пользовательский интерфейс и взаимодействие с пользователем.

Схематичное изображение взаимодействия компонентов MVC представлено на Рисунке 2.1.

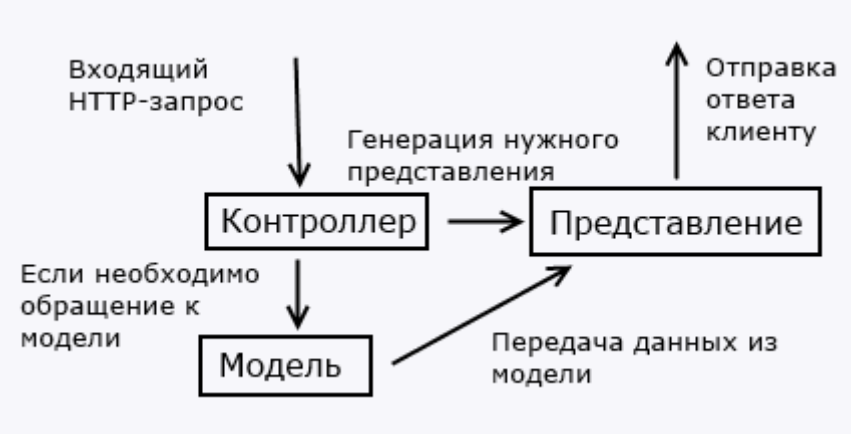


Рисунок 2 – Схема взаимодействия компонентов MVC

Контроллер является основным компонентом приложения. Данный класс связывает модель и представление. При получении введённых пользователем данных, он их обрабатывает согласно внутренней логике, а затем, при необходимости, обращается к модели и создает представление, соответствующее данному контроллеру.

Представление отвечает за визуальную часть приложения, оно выступает в качестве интерфейса, с которым взаимодействует пользователь, зашедший на сайт.

Моделью является набор классов, которые описывают логику используемых данных.

Выбор системы управления базой данных, создание структуры

Для хранения и систематизации данных на сайте используются базы данных. Для управления базой данных используется система управления базой данных – СУБД. На данный момент на рынке представлено большое количество различных СУБД, среди них – Postgre SQL, MS SQL, MySQL.

1. MySQL. На сегодняшний день является одной из самых распространённых СУБД. Как правило, используется при разработке веб сайтов и других веб приложений. Система является бесплатной и открытой, что является одним из значимых факторов ее популярности.

Преимуществам данной системы являются:

- СУБД легко установить. Как правило, она идет в комплекте с веб сервером, например как LAMP(Linux,Apache,MySQL,PHP). Программный продукт может управлять через консоль или графический интерфейс.
- СУБД практически полностью поддерживает язык SQL.
- MySQL может достаточно просто работать с большими объемами данных и при необходимости легко масштабироваться.
- Некоторые упрощения некоторых стандартов языка SQL позволяет ускорить работу СУБД.

К недостаткам использования MySQL можно отнести:

- Не полное соответствие стандартам языка SQL. Если до момента использования данной СУБД специалист работал с системой, использующей стандартный SQL, то при переходе на MySQL могут возникнуть некоторые трудности связанные с изучением специфики используемого языка.

- Проблемы с надежностью. MySQL из-за некоторых методов обработки информации в некоторых ситуациях уступает другим СУБД. Например, при кластерной обработке MySQL может организовать DDos атаку на собственную базу.

2. Postgre SQL

1) Из числа всех бесплатных СУБД является самым профессиональным решением. Postgre SQL бесплатная и открытая система. Но в отличии от MySQL старается полностью соответствовать стандартам языка SQL.

К достоинствам данной системы можно отнести:

- Postgre бесплатная система с открытым исходным кодом, которая соответствует стандартам SQL.
- Достаточно большое количество дополнений которые позволяют расширять функционал системы.[17]
- В отличии от других СУБД обеспечивает более высокий уровень целостности данных и транзакций.

К недостаткам можно отнести:

- При выполнении достаточно простых операций СУБД использует значительно больше ресурсов, чем ее конкуренты.
- Не смотря на достаточно большую базу пользователей, достаточно трудно найти хостинг, который будет поддерживать Postgre.

3. Microsoft SQL server

- Microsoft SQL server СУБД разработанная компанией Microsoft. Так же как MySQL и Postgre предназначена для управления

реляционными базами данных. Microsoft SQL server использует язык Transact-SQL. Данный язык соответствует стандартам языка SQL.

К преимуществам СУБД можно отнести:

- Шифрование всей хранимой информации в БД. Шифрование и расшифровка информации происходит непосредственно перед чтением или записью информации.
- Интуитивно понятный интерфейс.
- Высокая производительность СУБД.
- Легкая масштабируемость системы.

К недостаткам можно отнести:

- Ограниченный выбор платформ. СУБД будет работать только под управлением операционных систем семейства windows.
- Ограниченное число средств для разработки клиентских приложений. По умолчанию количество платформ для разработки клиентской части ограниченно. Другие СУБД поддерживают большее количество платформ.
- Microsoft SQL server является не бесплатным ПО. SQL server имеет несколько редакций которые отличаются предоставляемым функционалом и стоимостью.

[http://evansys.com/articles/perspektivy-i-tekhnologii-razvitiya-v-oblasti-tekhnicheskikh-nauk-sbornik-nauchnykh-trudov-po-itogam/sektsiya-20-informatsionnye-tekhnologii/sravnenie-sistem-upravleniya-bazami-dannykh/]

На основании проведенного анализа была выбрана Microsoft SQL в качестве основной СУБД, поскольку Microsoft SQL server является самым мощным решением из всех рассмотренных. По умолчанию данная СУБД обладает огромным функционалом. Продукт компании Microsoft позволяет

решать самые сложные задачи по хранению и обработке больших массивов данных.

Выбрав СУБД для использования, была разработана структура базы данных, изображенная на рисунке 3.

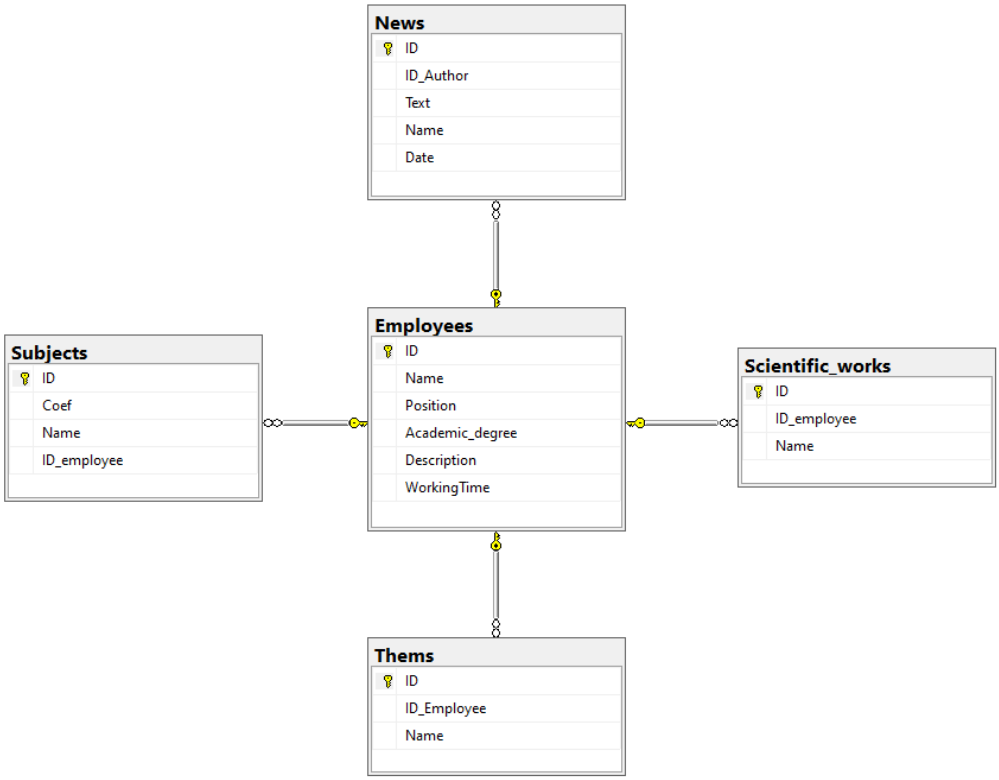


Рисунок 3 - структура базы данных

3. Реализация

3.1 Создание проекта

Перед началом реализации поставленной задачи необходимо создать проект. Встроенные шаблоны в среде разработки Visual Studio позволяют создать всю необходимую инфраструктуру проекта в автоматическом режиме. Для создания проекта MVC необходимо открыть Visual Studio и в меню File (Файл) выбрать пункт New Project... (Создать проект). В результате откроется диалоговое окно создания проекта, где необходимо перейти во вкладку Web и выбрать шаблон ASP.NET Web Application (.NET Framework). После указания названия проекта, необходимо перейти к окну выбора шаблона нового приложения (рисунок 3.2.1).

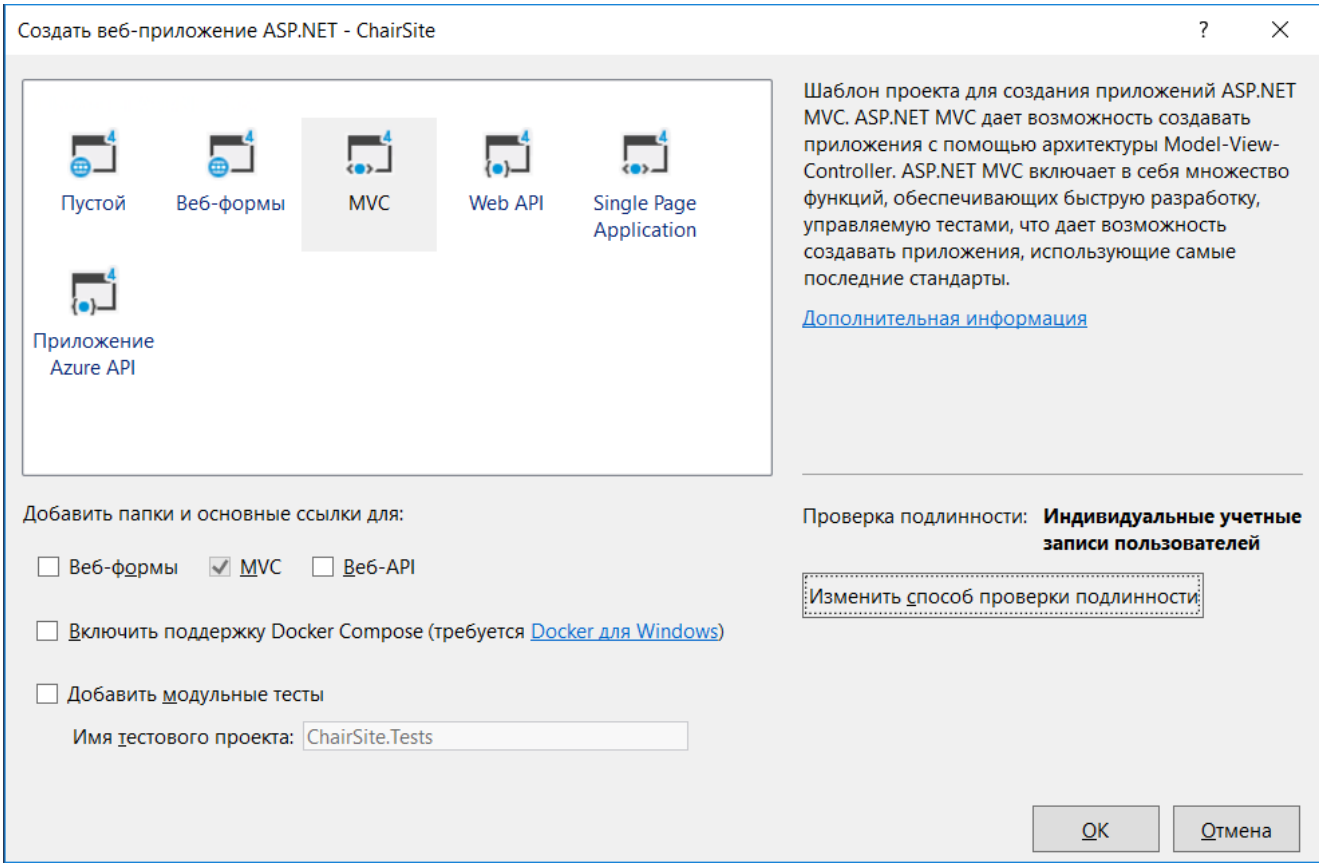


Рисунок 3.2.1

По умолчанию уже выбран шаблон MVC.

Кроме того, данное диалоговое окно позволяет задать опции тестирования.

Также в данном окне доступен выбор механизма аутентификации в приложении (кнопка Change Authentication). По умолчанию установлен тип No Authentication, который подразумевает отсутствие какой-либо системы аутентификации. Но для данной задачи будет использоваться метод проверки подлинности «**Индивидуальные учетные записи пользователей»** позволяющий проходить авторизацию на сайте и хранить учетные записи в базе данных. Выбор данной опции позволит сгенерировать и подключить в автоматическом режиме базу данных для авторизации, а также создаст контроллер, отвечающий за проверку подлинности введенных данных. В конечном итоге получен проект, содержащий разветвленную структуру и имеющий некоторое наполнение по умолчанию.

Для генерации классов модели используется Entity Framework. В контекстном меню проекта необходимо добавить компонент под названием «Модель ADO.NET EDM». На следующем шаге необходимо выбрать способ генерации. Поскольку база данных уже есть, то необходимо выбрать способ «Code First из базы данных». Данный способ генерации подразумевает автоматическую генерацию кода классов моделей по существующим таблицам в базе данных. После этого необходимо сконфигурировать подключение к ранее созданной базе данных. Для этого необходимо указать таблицы и представления, включаемые в модель. После подключения базы данных в обозревателе решений появиться пять классов: **Employee.cs, News.cs, Scintific_works.cs, Thems.cs, Subjects.cs.** Таким образом, был построен «мост» между БД и приложением. Созданные классы будут использоваться в качестве моделей.

В результате создания и правильной настройки проекта была создана вся необходимая инфраструктура в автоматическом режиме. Благодаря этому было сэкономлено большое количество времени и исключены вероятные ошибки при ручном создании структуры проекта.

Структура, полученная в результате проделанной работы изображена на рисунке 3.3.1

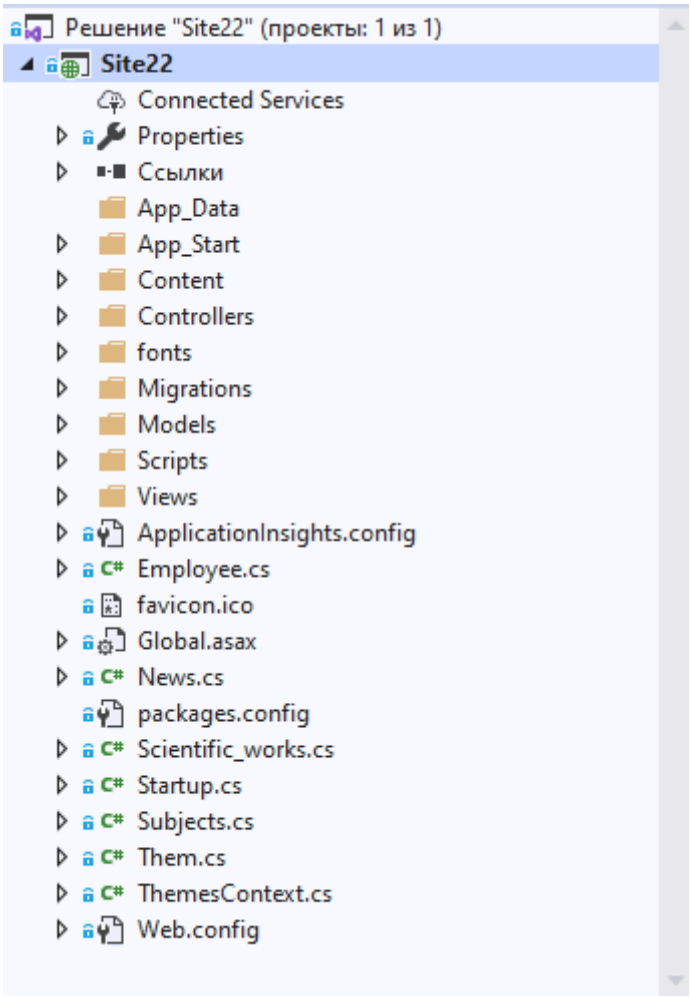


Рисунок 2.3.1

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

3.2 Контроллеры и их методы

Так как с моделями и настройкой контекста данных мы закончили, необходимо создать другой компонент приложения - контроллеры.

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.

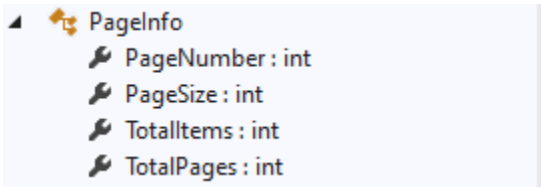
Для разработки требуемой функциональности необходимо было распределить функции на котроллеры: EmployeeController который будет полностью отвечать за работу с сотрудниками, NewsController который будет отвечать за показ, создание и удаление новостей, InfoController, который отвечает за показ общей информации о кафедре и SubjectController который будет отвечать за загрузку плана и распределение учебной нагрузки между преподавателями. Данное распределение улучшит читаемость и понимание кода.

Начинать разработку каждого контроллера необходимо 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 будет происходить взаимодействие с базой данных:

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

В контроллере EmployeeController был создан метод «FilteredBrowse», который позволяет выводить на страницу список преподавателей факультета. При этом в данном методе реализована фильтрация по должности и по учёной степени, а также решена проблема пагинации. В качестве входных данных используются названия должности и учёной степени, помимо этого в метод передается номер текущей страницы (по умолчанию – 1).

Поскольку пагинация страниц будет использоваться не один раз, для этой задачи была создана отдельная модель, параметрами которой являются номер текущей страницы, количество объектов на странице, общее количество объектов и общее количество страниц. Структура данного класса изображена на рисунке XXX



Также была создана модель страницы под названием «FilteredWorks» , в которой будут присутствовать условия фильтрации, выбранная должность и степень, а также информация для пагинации. Структура данного класса изображена на рисунке XXX

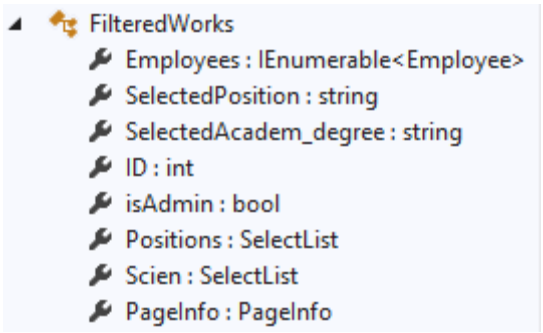


Рисунок 3 структура модели работников

Поскольку данный метод позволяет получить данные, целесообразно использовать для него запрос типа GET. Фреймворк ASP.NET MVC позволяет определить тип обрабатываемого запроса для действия, применив к нему соответствующий атрибут. В данном случае перед методом необходимо использовать атрибут [HttpGet].

Для работы с SQL запросами здесь и далее используется синтаксис LINQ.

Фильтрация по должности выглядит следующим образом:

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

Фильтрация по научной степени выглядит похожим образом:

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

После расчета количества объектов на странице необходимо сформировать модель, а также передать в нее всю необходимую информацию, после этого модель необходимо передать 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() };

var postions = db.Employees.Select(m => m.Position).Distinct().ToList();
postions.Add("Все");
var dergee = db.Employees.Select(m => m.Academic_degree).Distinct().ToList();
dergee.Add("Все");
FilteredWorks fw = new FilteredWorks
     {
         Employees = emoloyeesPerPages.ToList(),
         Positions = new SelectList(postions),
         Scien = new SelectList(dergee),
         SelectedPosition = Position,
         isAdmin = IsAdmin(),
         SelectedAcadem_degree = Academ,
         PageInfo = pageInfo,
      };
 return View(fw);
```

Далее был создан метод «EditInfo» для редактирования информации о сотруднике. Поскольку запрос к данному методу может быть двух типов: GET и POST, поэтому предполагается раздельное создание двух методов - по одному для каждого типа запросов.

Метод «EditInfo» с атрибутом [HttpGet] выводит текущую информацию о редактируемом 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, создан по аналогии с предыдущим. В методе с атрибутом [HttpGet] создается новый пустой объект и передается в представление для заполнения:

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

В методе с атрибутом [HttpPost] заполненная модель сохраняется в БД:

```
[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");
 }
```

Метод «Delete», необходимый для удаления сотрудника из базы данных, во многом схож с предыдущими. В этом методе так же перед удалением выводится информацию об удаляемом сотруднике, а после получения подтверждения происходит удаление из БД:

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

Следующим этапом был создан метод «AboutEmployee» с атрибутом [HttpGet], главной задачей которого является отображение подробной информации о сотруднике кафедры, в качестве входящего параметра принимается первичный ключ ID сотрудника. Запрос к базе данных на синтаксисе 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, для отображения списка этих работ используется 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: [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);
        }
```

Следующий контроллер, отвечающий за показ, создание и удаление новостей, является NewsController.

Был создан метод «News. 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 имеет 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);
        }
```

В методе «CreateNews», который предназначен для создания новостей необходимо узнать, какой из сотрудников авторизован на сайте на данный момент. Для этого используется 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 имеет следующий вид:

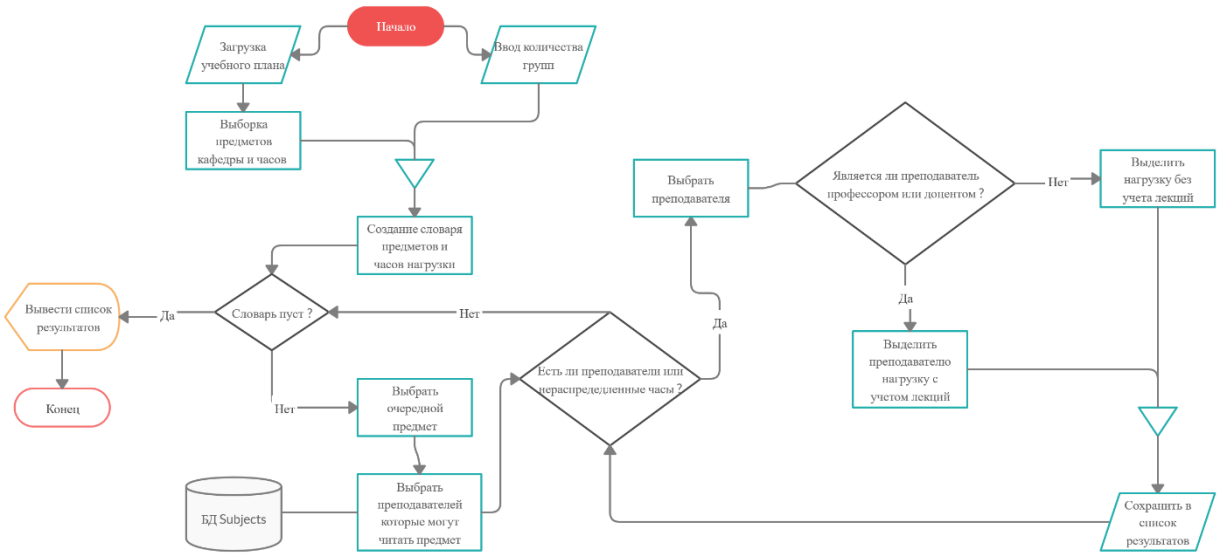
```
[HttpPost]
public ActionResult CreateNews(News news)
        {
```

```
                    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");
                }
```

Следующий рассматриваемый контроллер - InfoController. В данном контроллере реализованы методы About, Contacts, Index, History, отображающие статическую информацию о кафедре, ее истории и контактных данных.

Последний контроллер, который был создан – SubjectController, отвечающий за распределение учебной нагрузки. Метод под названием DownloadFile, который входит в состав этого контроллера, предназначен для отображения страницы загрузки файла и ввода данных о количестве подгрупп. Поскольку отображение не требует какой-либо логики – данный метод просто генерирует представление.

Следующий метод под названием Import предназначен для выборки данных из таблицы xls с последующим распределением нагрузки между преподавателями. Поскольку учебные планы имеют строгую структуру, то для выборки необходимых данных используется привязка к конкретным столбцам. Далее следует генерации списка, ключом которого является название предмета, а значением – структура, которая содержит информацию о количестве нагрузки в каждом семестре, в котором данный предмет читается. Пока данный словарь не пуст – происходит распределение нагрузки между преподавателями. Алгоритм работы показан на рисунке Ч


Начало
Загрузка учебного плана
Ввод количества групп
Выборка предметов кафедры и часов
Создание словаря предметов и часов нагрузки
Словарь пуст ?
Да
Нет
Вывести список результатов
Конец
Нет
Выбрать очередной предмет
БД Subjects
Выбрать преподавателей которые могут читать предмет
Есть ли преподаватели или нераспределенные часы ?
Да
Выбрать преподавателя
Является ли преподаватель профессором или доцентом ?
Нет
Выделить нагрузку без учета лекций
Да
Выделить преподавателю нагрузку с учетом лекций
Сохранить в список результатов

3.3 Представления

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#.

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.

С начала было созданно представление для контроллера «FilteredWorks», который отображает сотрудников кафедры, с возможностью фильтрации. В начале файла необходимо указать модель. Делается это следующим образом:

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

Далее необходимо получить от пользователя условия фильтрации. Для этого созданы два выпадающих меню, а идет их передача в контроллер для данного представления. В результате получена 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 на страницу выводится список преподавателей, подходящих по данному фильтру. Для этого используется таблицу. Для отображения элементов модели (преподавателей) используется цикл foreach. Вывод самой информации происходит с помощью встроенных функций Html.DisplayFor. Следует упомянуть что в 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 добавить 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, то необходимо 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>
```

Далее следует создание представлений предназначенных для создания/изменения/удаления информации. Первое из них - «EditInfo», позволяющее редактировать информацию о сотруднике.

Структура данного преставления схожа с структурами ранее рассмотренных преставлений. При создании данного представления используется функция Html.EditorFor. Она создает элемент разметки, который доступен для редактирования, для указанного свойства модели, например, имени. Пример:

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

Для отправки данных контроллеру – была создана кнопка отправки, а также создана кнопка 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 заполнены 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:

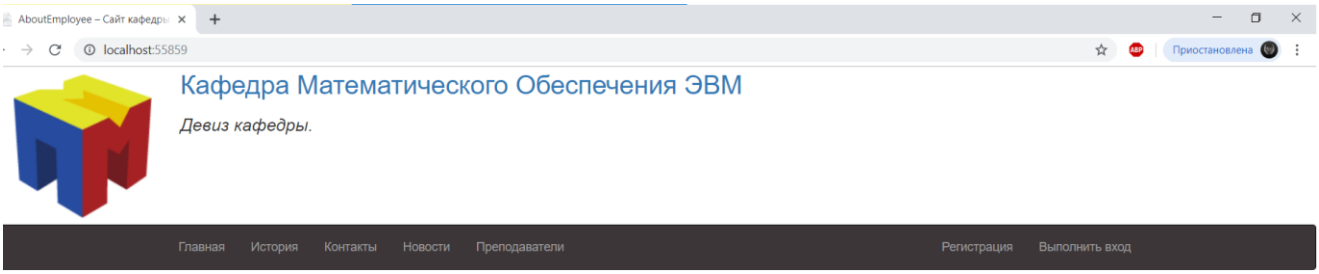


Рисунок 3.3.2.1

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:

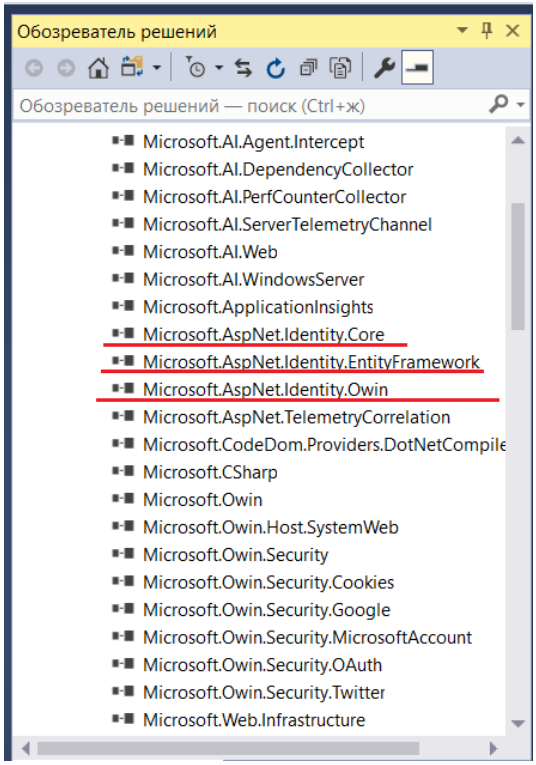


Рисунок 3.4.1

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:

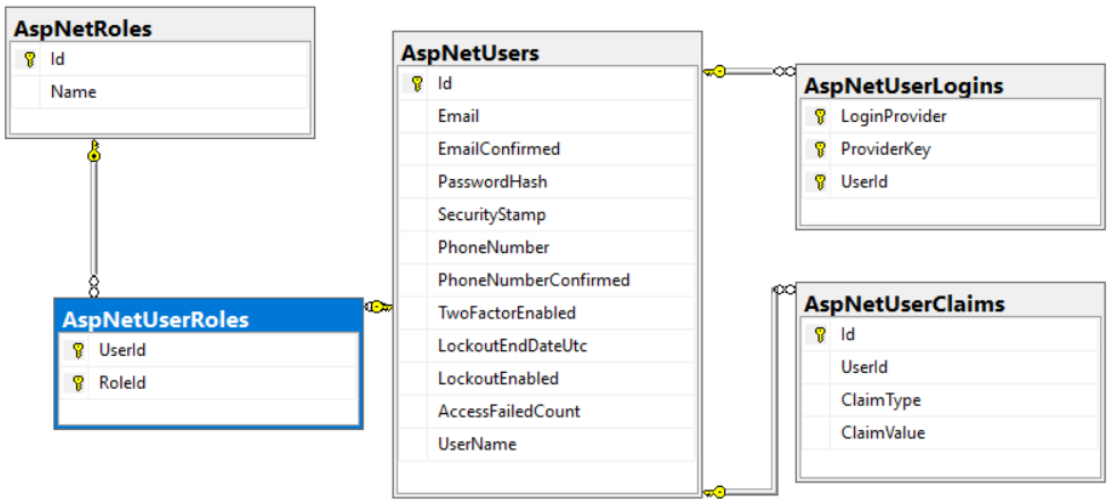


Рисунок 3.4.2

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.

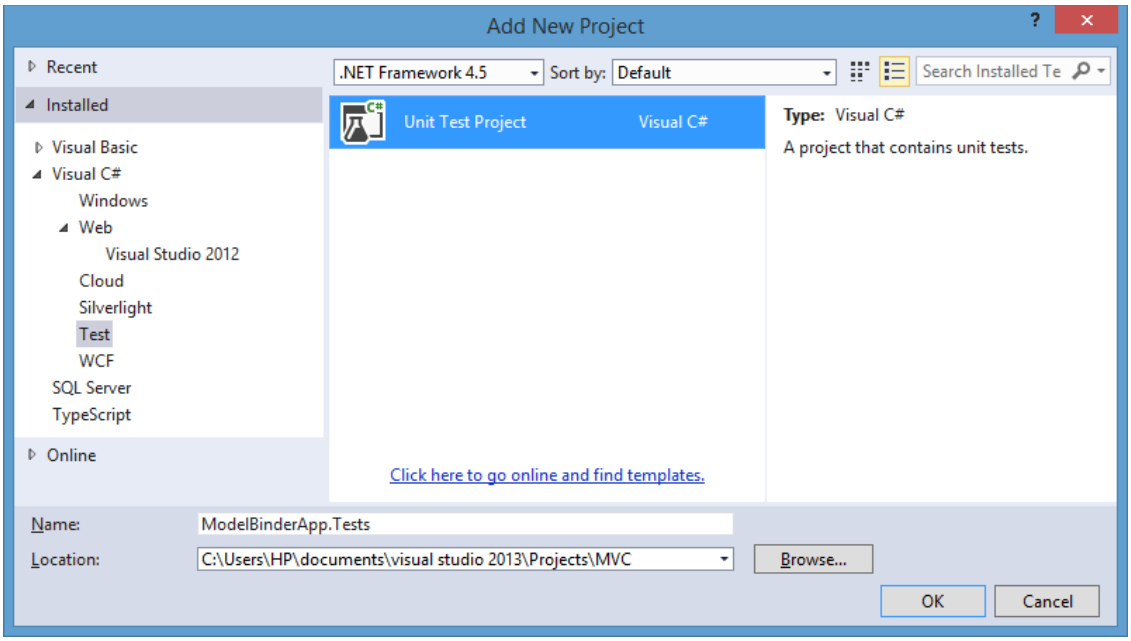


Рисунок 3.5.1

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:

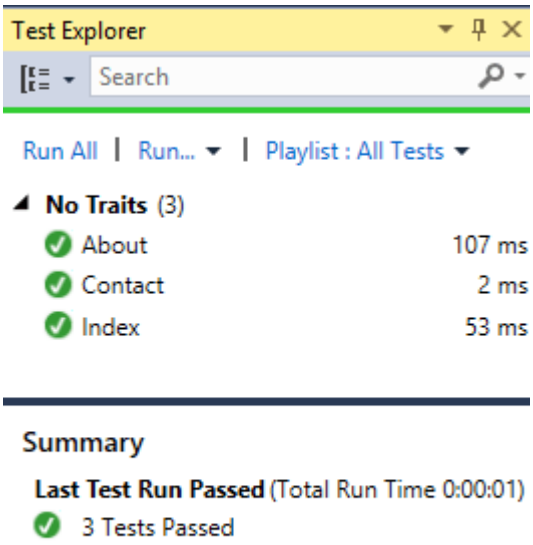


Рисунок 3.5.2

Результат

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

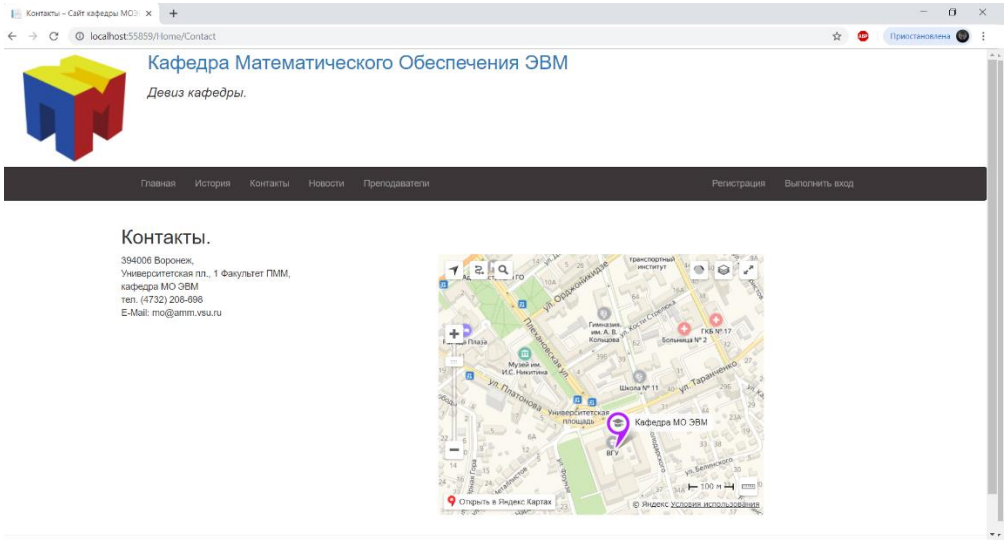


Рисунок 1

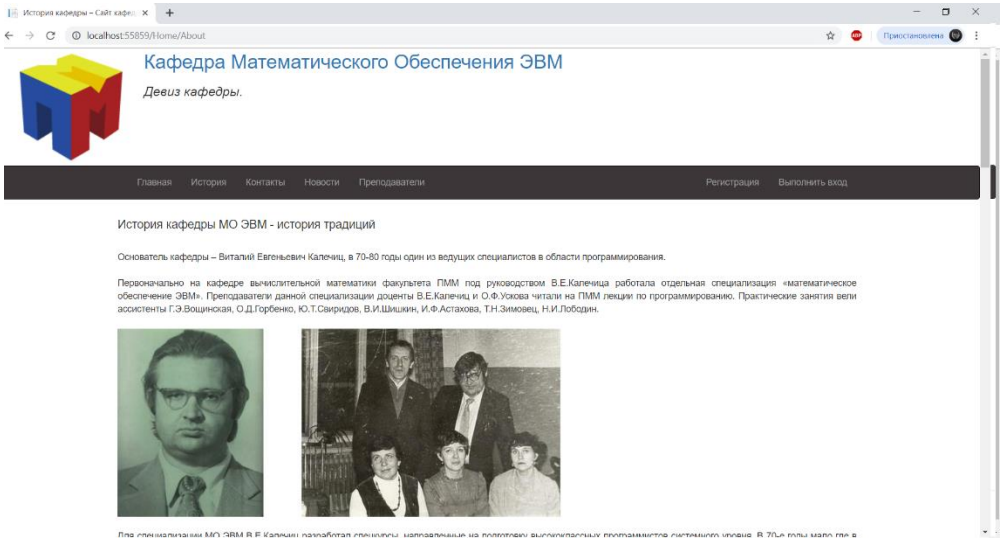


Рисунок 2

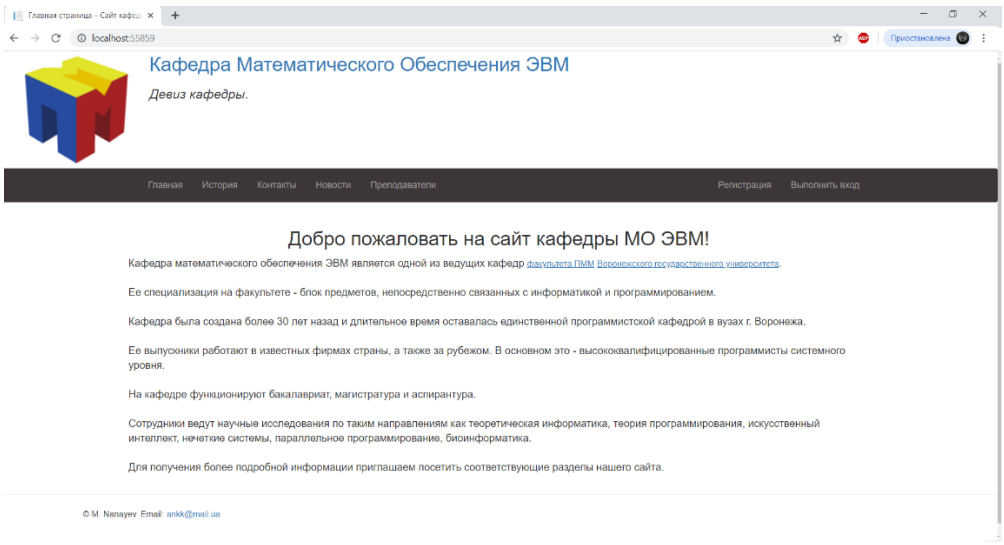


Рисунок 3

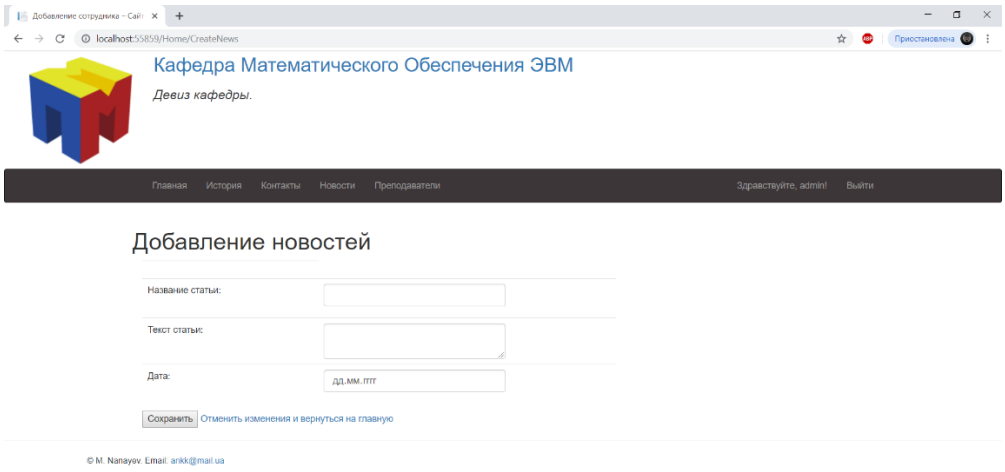


Рисунок 4

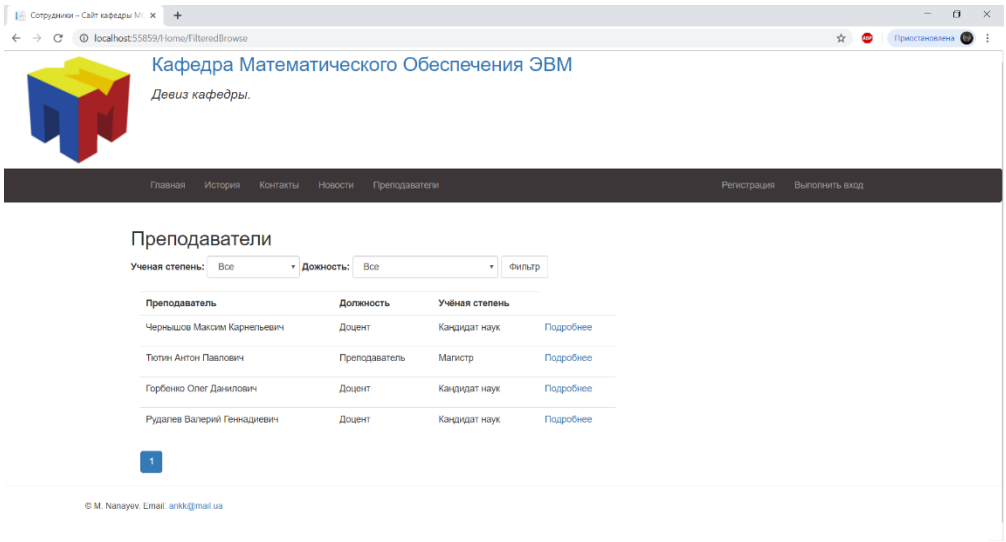


Рисунок 5

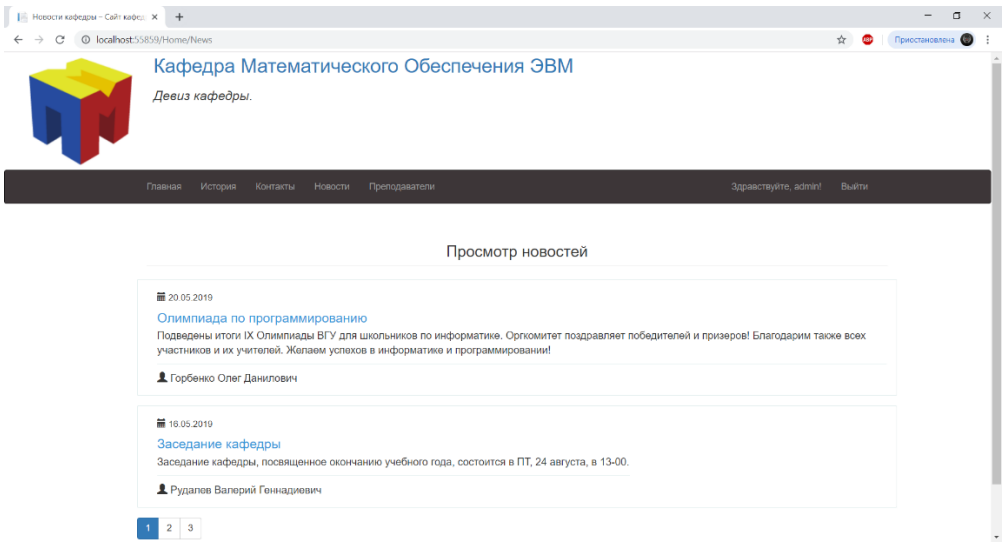

Кафедра Математического Обеспечения ЭВМ
Девиз кафедры.
Главная
История
Контакты
Новости
Преподаватели
Здравствуйте, admin!
Выйти
Просмотр новостей
20.05.2019
Олимпиада по программированию
Подведены итоги IX Олимпиады ВГУ для школьников по информатике. Оргкомитет поздравляет победителей и призеров! Благодарим 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 6

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 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.

Список использованных источников:

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>
```