Теория.

**Общие сведения**

Платформа ASP.NET MVC представляет собой фреймворк для создания сайтов и веб-приложений с помощью реализации паттерна MVC.

Концепция паттерна (шаблона) MVC (model - view - controller) предполагает разделение приложения на три компонента:

**Контроллер** (controller) представляет класс, обеспечивающий связь между пользователем и системой, представлением и хранилищем данных. Он получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления.

**Представление** (view) - это собственно визуальная часть или пользовательский интерфейс приложения. Как правило, html-страница, которую пользователь видит, зайдя на сайт.

**Модель** (model) представляет класс, описывающий логику используемых данных.

Общую схему взаимодействия этих компонентов можно представить следующим образом:



В этой схеме модель является независимым компонентом - любые изменения контроллера или представления не затрагивают модель. Контроллер и представление являются относительно независимыми компонентами, и нередко их можно изменять независимо друг от друга.

Благодаря этому реализуется концепция **разделение ответственности**, в связи с чем легче построить работу над отдельными компонентами. Кроме того, вследствие этого приложение обладает лучшей тестируемостью. И если нам, допустим, важна визуальная часть или фронтэнд, то мы можем тестировать представление независимо от контроллера. Либо мы можем сосредоточиться на бэкэнде и тестировать контроллер.

Конкретные реализации и определения данного паттерна могут отличаться, но в силу своей гибкости и простоты он стал очень популярным в последнее время, особенно в сфере веб-разработки.

Свою реализацию паттерна представляет платформа ASP.NET MVC. 2013 год ознаменовался выходом новой версии ASP.NET MVC - MVC 5, а также релизом Visual Studio 2013, которая предоставляет инструментарий для работы с MVC5.

Хотя во многих аспектах MVC 5 не слишком сильно будет отличаться от MVC 4, многое из одной версии вполне применимо к другой, но в то же время есть и существенные отличия:

* В MVC 5 изменилась концепция аутентификации и авторизации. Вместо SimpleMembershipProvider была внедрена система ASP.NET Identity, которая использует компоненты OWIN и Katana.
* Для создания адаптивного и расширяемого интерфейса в MVC 5 используется css-фреймворк Bootstrap
* Добавлены фильтры аутентификации, а также появилась функциональность переопределения фильтров
* В MVC 5 также добавлены атрибуты маршрутизации

Это наиболее важные нововведения в MVC 5. Кроме того, есть еще ряд менее значимых, например, использование по умолчанию Entity Framework 6, некоторые изменения при создании проекта (концепция One ASP.NET), дополнительные компоненты и т.д.

В любом случае все полученные при работе с MVC 4 навыки можно успешно применять при использовании MVC 5, учитывая, конечно, нововведения.

Для общего развития простенькие статьи на метаните:

<https://metanit.com/sharp/mvc5/1.1.php>

**Более подробно об конвеере обработки запроса (жизненный цикл запроса Asp.Net MVC)**

Простое и понятное изложение:

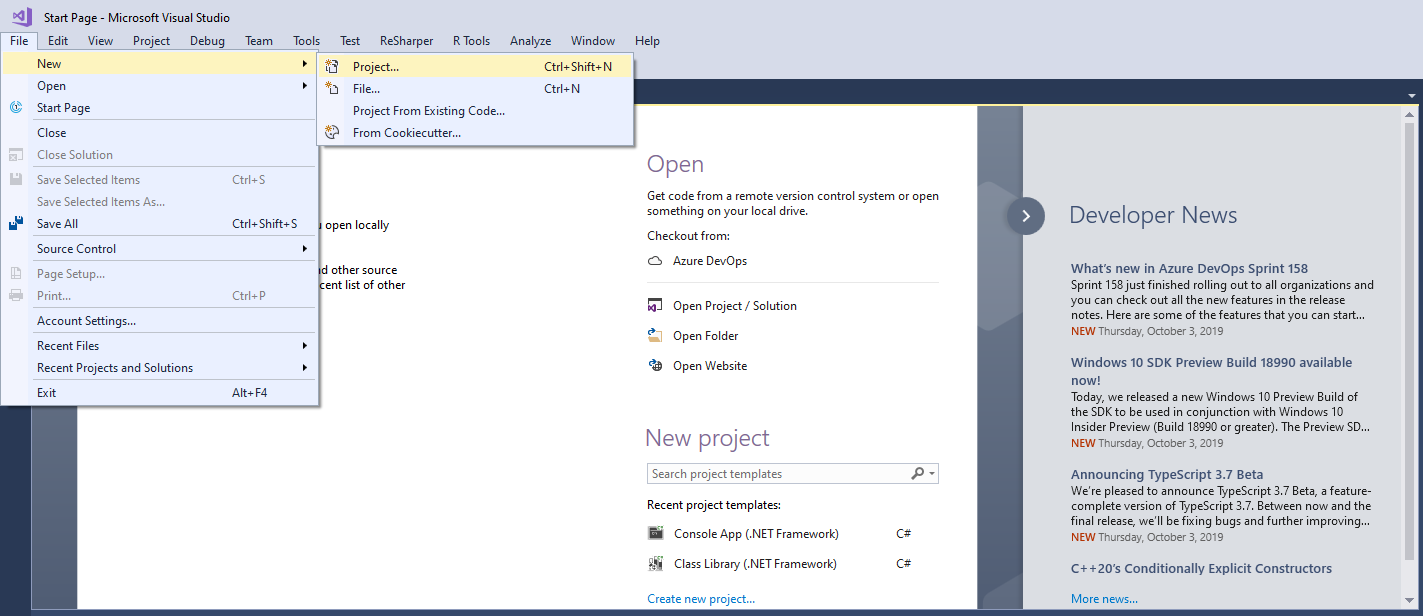
<https://www.dotnettricks.com/learn/mvc/aspnet-mvc-request-life-cycle>

Вот ещё одно изложение:

<https://d2funlife.com/asp-net-mvc-request-life-cycle>

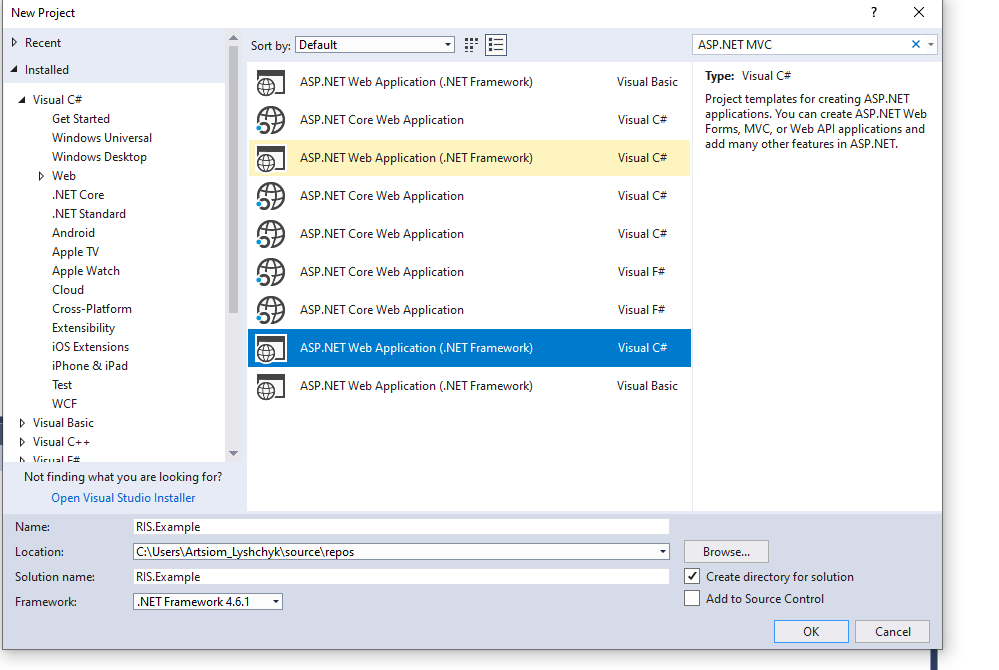
**Пример**

Для начала необходимо создать проект в среде разработки (желательно использовать VS 2017 или VS 2019 Community).

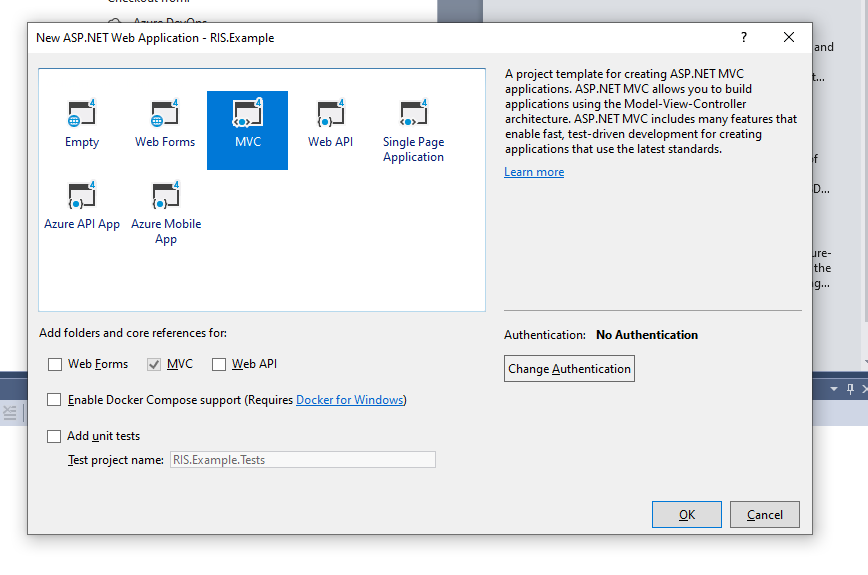


Для этого выбираем File -> New -> Project…

Далее в поиске вводим «ASP.NET MVC» и выбираем ASP.NET Web Application C#. Вводим желаемое имя проекта и папку, где он прорастёт (Location).



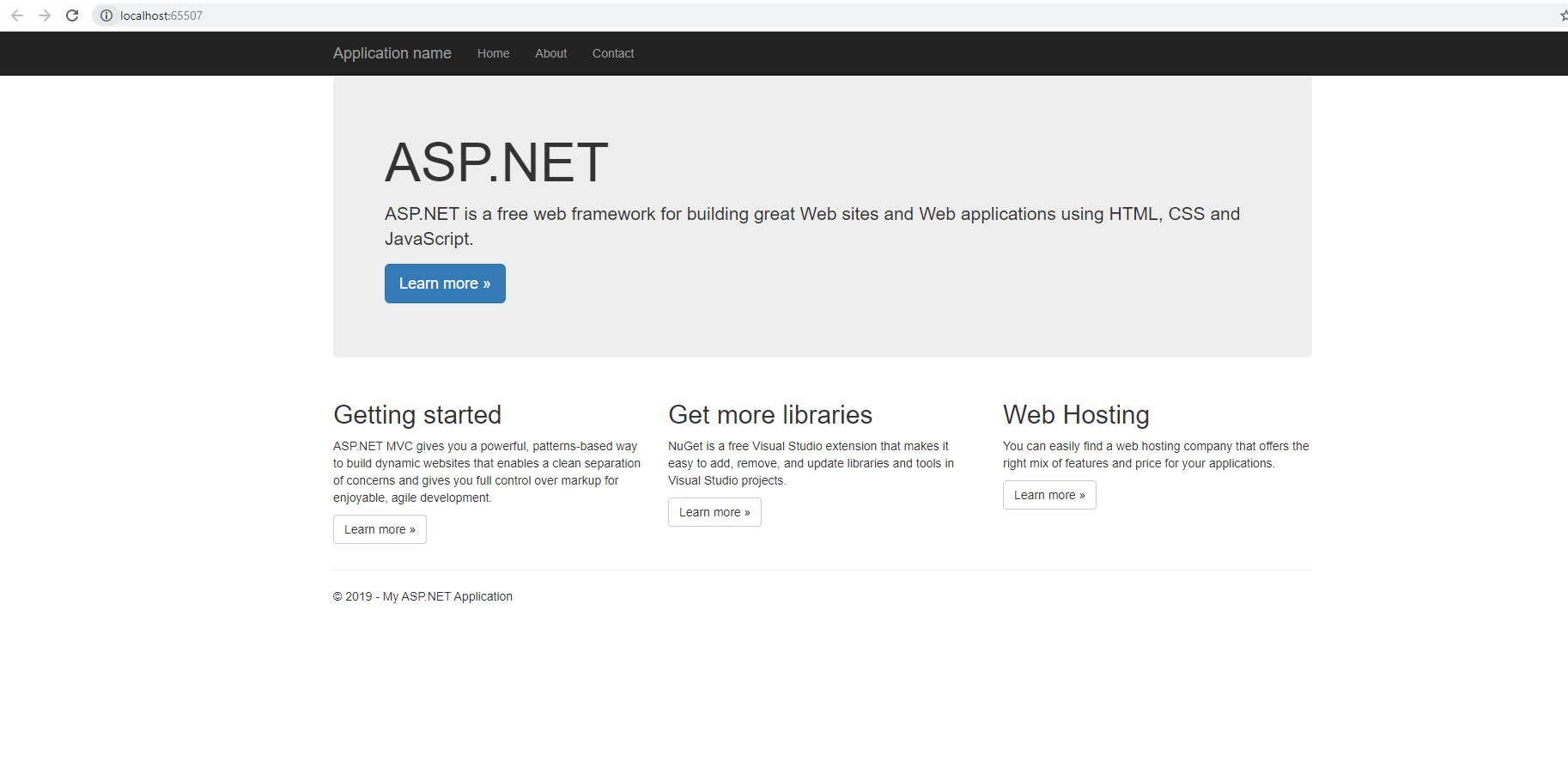
Далее, среда разработки предложит выбрать шаблон. Выбираем MVC, всё остальное оставляем без изменений.



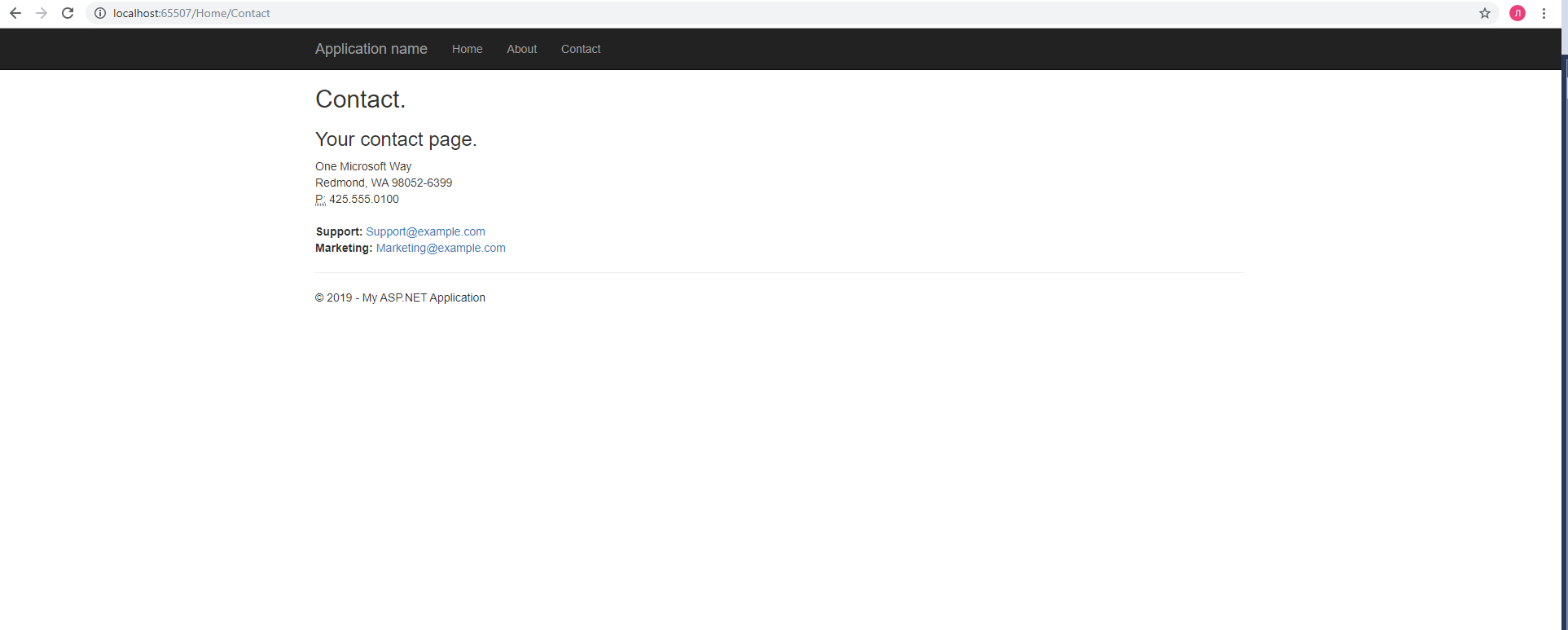
1. Проверка работоспособности проекта

Первоначальный шаблон ASP.NET MVC приложения позволяет сразу же его запустить и посмотреть, как он выглядит.

С помощью комбинации клавиш CTRL + F5 запустим приложение. Браузер откроется автоматически.



Можно покликать по ссылкам в меню навигации. (пример Contact)



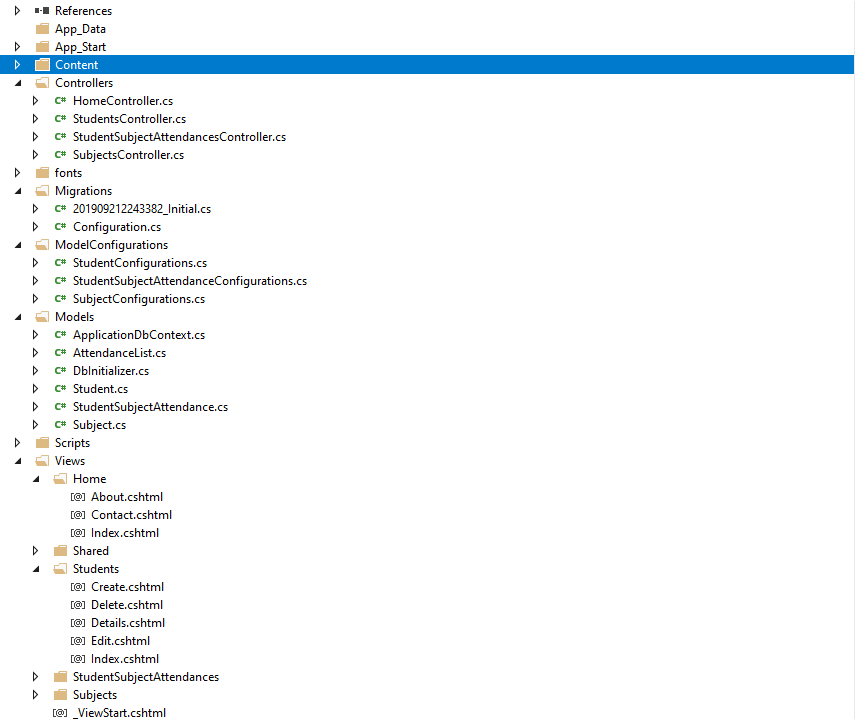
1. Построение моделей базы данных.

Для построения базы данных мы будем использовать самый популярный ORM фреймворк – EntityFramework.

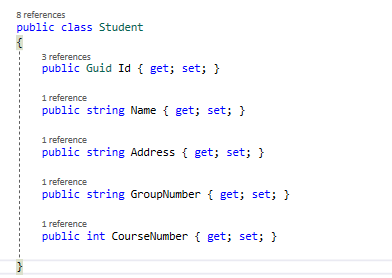
Сразу уточним, что в проекте не будет использоваться многослойная архитектура со «швами» между слоями или микросервисы. Всё будет создаваться в одном проекте. Цель лабораторной работы – демонстрация технологий и общее понимание возможностей всех используемых фреймворков.

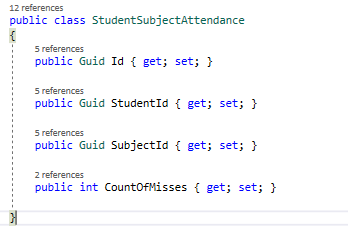
Создадим необходимые модели базы данных в соответствии с предметной областью задачи. В нашем случае – учёт посещаемости студентами занятий.

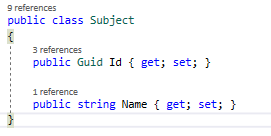
Структура проекта:



В папке Models создадим новые классы





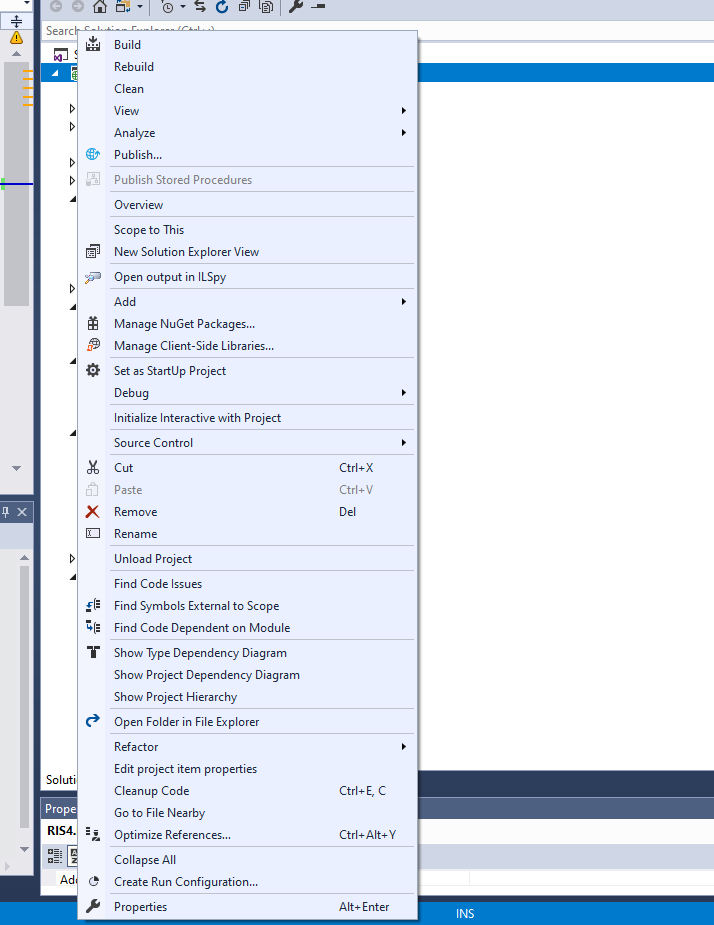


Имеются 3 сущности – Студент (Student), Предмет (Subject), Посещаемость (StudentSubjectAttendance).

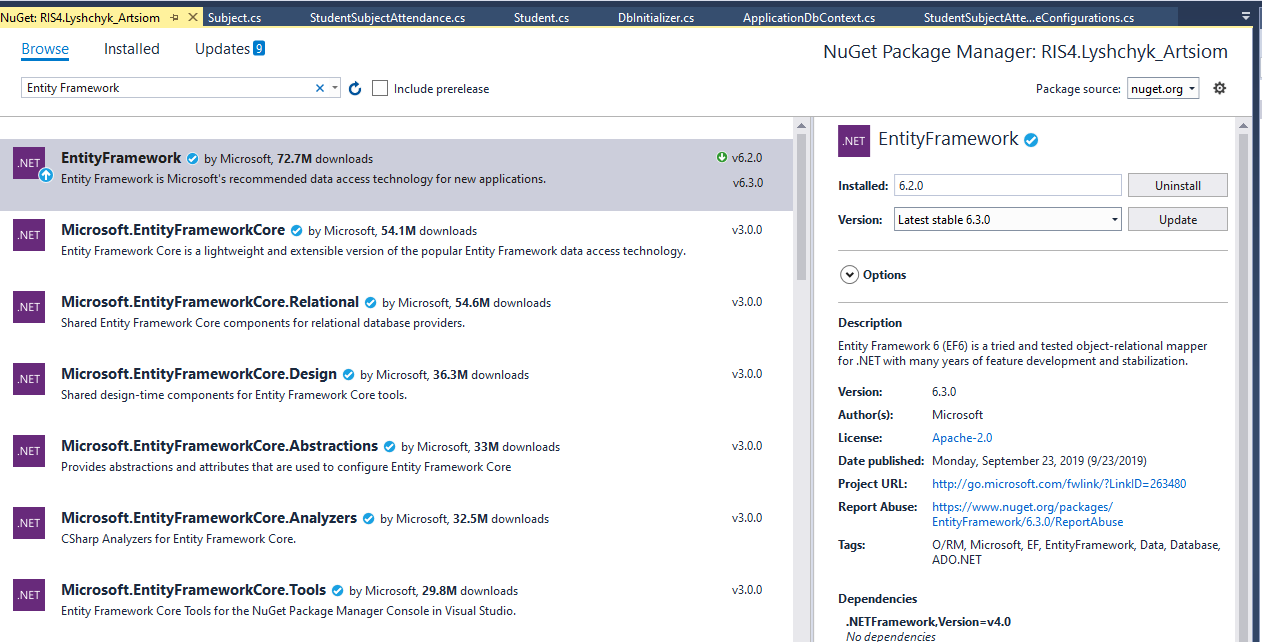
Представленные сущности являются пустышками. Они не несут никакой дополнительной логики, кроме как логики структурированного хранения данных в программе с соответствием таблицам в базе данных (1 сущность – 1 таблица). Их ещё называют DTOs – Data Transfer Objects.

Подключим EntityFramework к приложению.

Для этого кликаем правой кнопкой мыши по проекту



Далее выбираем Manage NuGet Packages.



В появившемся окне вводим в поиск Entity Framework и нажимаем справа кнопку Install (т.к. в примере он уже установлен, предлагается его удалить (Uninstall)).

Отлично. Мы установили Entity Framework в наш проект.

Перед работой с Entity Framework предполагается ознакомиться с материалом:

1. <https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx>

Вот его краткое описание на метаните.

**Entity Framework** представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работает с объектами.

Первая версия Entity Framework - 1.0 вышла еще в 2008 году и представляла очень ограниченную функциональность, базовую поддержку ORM (object-relational mapping - отображения данных на реальные объекты) и один единственный подход к взаимодействию с бд - Database First. С выходом версии 4.0 в 2010 году многое изменилось - с этого времени Entity Framework стал рекомендуемой технологией для доступа к данным, а в сам фреймворк были введены новые возможности взаимодействия с бд - подходы Model First и Code First.

Дополнительные улучшения функционала последовали с выходом версии 5.0 в 2012 году. И наконец, в 2013 году был выпущен Entity Framework 6.0, обладающий возможностью асинхронного доступа к данным.

Центральной концепцией Entity Framework является понятие **сущности** или entity. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст, вес. Свойства необязательно представляют простые данные типа int, но и могут представлять более комплексные структуры данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют **ключами**.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой Entity Framework является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем не только извлекать определенные строки, хранящие объекты, из бд, но и получать объекты, связанные различными ассоциативными связями.

Другим ключевым понятием является **Entity Data Model**. Эта модель сопоставляет классы сущностей с реальными таблицами в БД.

Entity Data Model состоит из трех уровней: концептуального, уровень хранилища и уровень сопоставления (маппинга).

На концептуальном уровне происходит определение классов сущностей, используемых в приложении.

Уровень хранилища определяет таблицы, столбцы, отношения между таблицами и типы данных, с которыми сопоставляется используемая база данных.

Уровень сопоставления (маппинга) служит посредником между предыдущими двумя, определяя сопоставление между свойствами класса сущности и столбцами таблиц.

Таким образом, мы можем через классы, определенные в приложении, взаимодействовать с таблицами из базы данных.

**Способы взаимодействия с БД**

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

* **Database first**: Entity Framework создает набор классов, которые отражают модель конкретной базы данных
* **Model first**: сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере.
* **Code first**: разработчик создает класс модели данных, которые будут храниться в бд, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы

В нашем проекте будет использоваться подход Code-First.

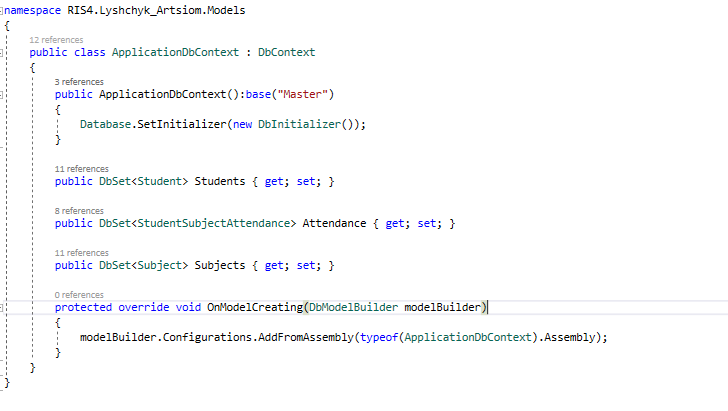
Основу функциональности Entity Framework составляют классы, находящиеся в пространстве имен *System.Data.Entity*. Среди всего набора классов этого пространства имен следует выделить следующие:

* **DbContext**: определяет контекст данных, используемый для взаимодействия с базой данных.
* **DbModelBuilder**: сопоставляет классы на языке C# с сущностями в базе данных.
* **DbSet/DbSet<TEntity>**: представляет набор сущностей, хранящихся в базе данных

В любом приложении, работающим с БД через Entity Framework, нам нужен будет контекст (класс производный от DbContext) и набор данных DbSet, через который мы сможем взаимодействовать с таблицами из БД. В данном случае таким контекстом является класс ApplicationDbContext.

В конструкторе этого класса вызывается конструктор базового класса, в который передается строка "DbConnection" - это имя будущей строки подключения к базе данных. В принципе мы можем не использовать конструктор, тогда в этом случае строка подключения носила бы имя самого класса контекста данных.

Создадим класс ApplicationDbContext



DbSet’ы – это коллекции, к которым мы будем обращаться для выполнения операций над данными.

Не обращайте внимание на DbInitializer и переопределённый метод OnModelCreating (об этом чуть позже).

Сейчас необходимо указать строку подключения к базе данных в конфигурации приложения.

**Для этого желательно бы меть Microsoft SQL Server Management Studio**.

1. **Строка** **подключения:**
2. Обращаемся к структуре проекта и находим файл Web.config

Web.config – файл, содержащий конфигурацию текущего web-приложения.

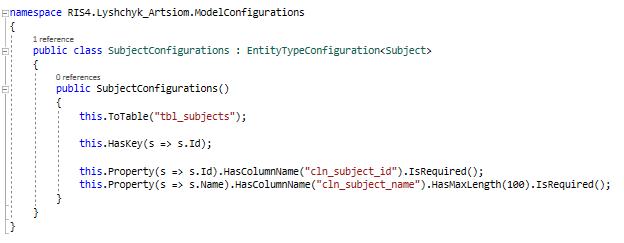


1. Находим секцию **<connectionStrings>** и в ней дописываем строку подключения, как указано на рисунке.
   1. DataSource – это название Sql Server’а, который на ней установлен (например DESKTOP-asdasd123123\ms\_local\_db). Это необходимо смотреть непосредственно в MS Sql Server Management Studio.
   2. Name – псевдоним строки подключения, который мы будем
2. Открываем MS SQL Management Studio и создаём соответствующую базу данных на указанном в строке подключения сервере. Так же необходимо передать как строку в конструктор базового класса, от которого наследуется наш ApplicationDbContext, как предста влено на рисунке «псевдоним» строки подключения в конфигурационном файле. Больше ничего делать не нужно.

Далее создадим базу данных на основе подхода Code First.

Для каждой сущности создадим конфигурационный класс (поместим в папку ModelConfigurations).

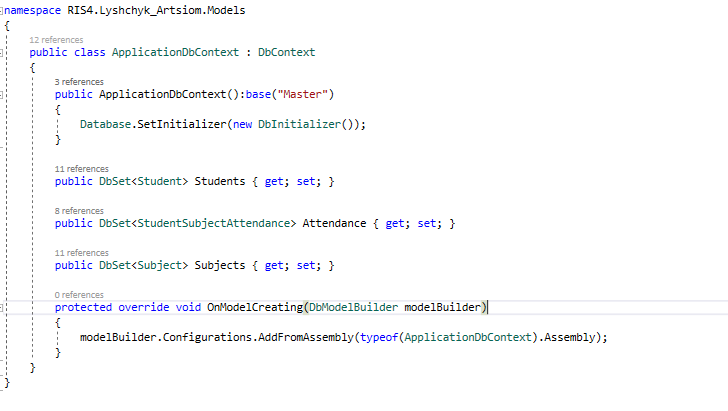




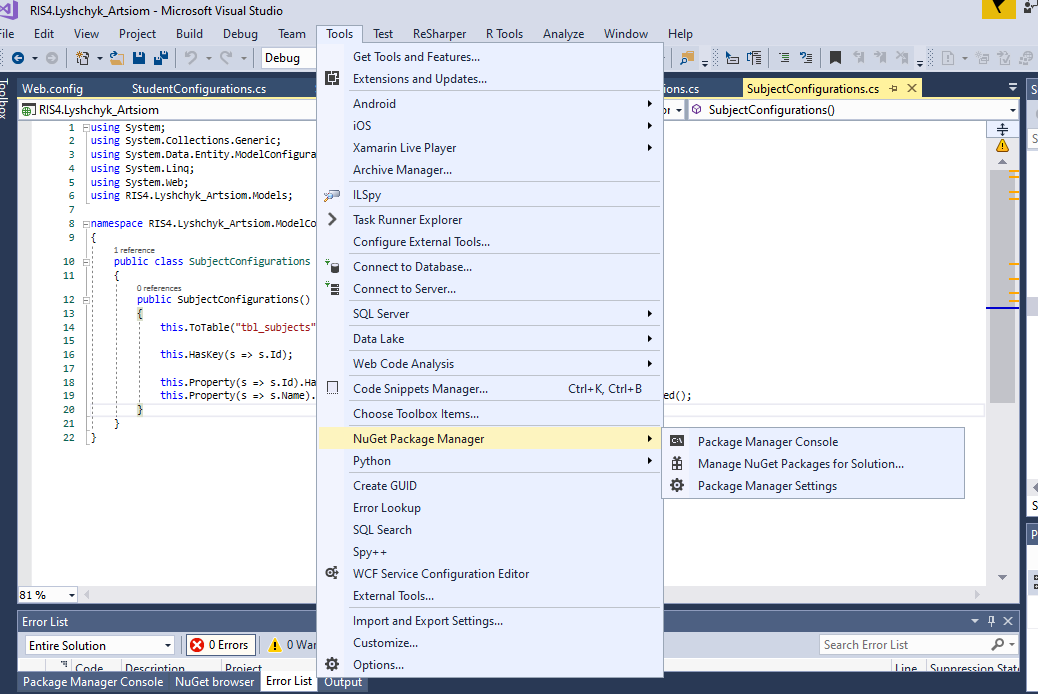


Далее, посмотрим ещё раз на класс ApplicationDbContext.

В методе OnModelCreating мы «соберём» все конфигурации из сборки, в которой находится наш контекст.

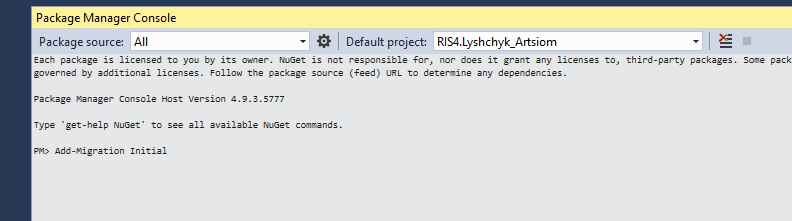


Создадим базу данных. Для этого заходим в пункт Tools -> Nuget Package Manager

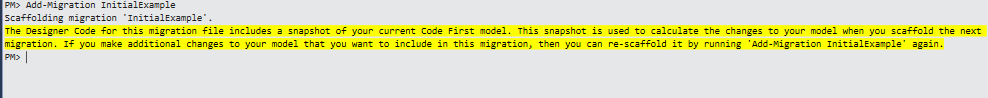


Внизу видим Package Manager Console

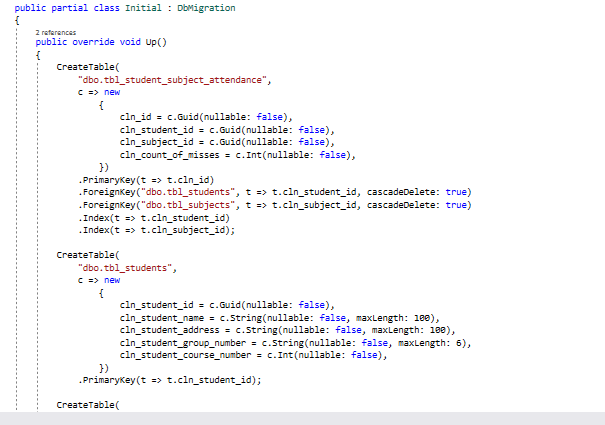
Открываем её и вводим



Автоматически откроется класс созданной миграцией в созданной папке Migrations

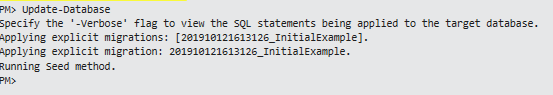


Вы увидите что-то вроде

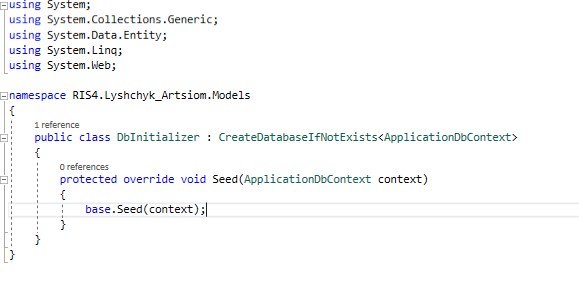


Это скрипто-подобный код создания/изменения базы данных на определённом этапе. Своеобразные коммиты системы контроля версий. По этим миграциям можно откатываться назад, вперёд, так же возможно изменения их кода (это уже advance фича, т.к. необходимо полное понимание работы кода и понимания структуры базы данных).

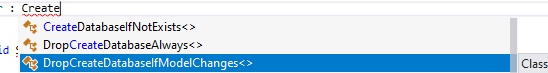
Далее вводим в ту же консоль команду **Update-Database** и видим приблизительно следующее



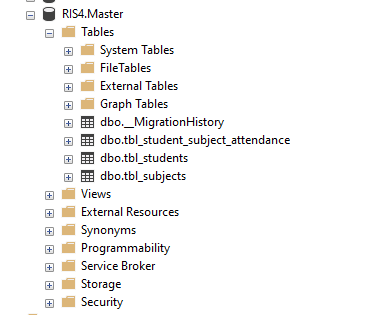
Обратим внимание на строку Running seed method. Entity Framework предоставляет возможность создания так называемых DbInitializer’ов. Если обратить внимание на класс ApplicationDbContext мы найдём создание объекта DbInitializer. (создан вручную).



Здесь, используя переменную типа ApplicationDbContext (которая хранит ссылку на созданный в памяти объект DbContext или производных этого типа, в нашем случае ApplicationDbContext) context мы можем получить доступ к DbSet для каждой «таблицы» и вставлять данные для начальной инициализации ( очень полезная штука, если необходимо часто тестировать приложение на начальной стадии разработки или при «демо» приложения заказчику, и для таких случаев из коробки можно получить типизированные классы с говорящим названием CreateDatabaseIfNotExists, что означает создание базы данных, если она не существует и запуск «seed» метода). Так же имеются следующие возможные опции



Можно зайти в MS SQl Server и проверить наличие созданных таблиц (так же можно использовать встроенный в VisualStudio SQL Server Object Explorer).

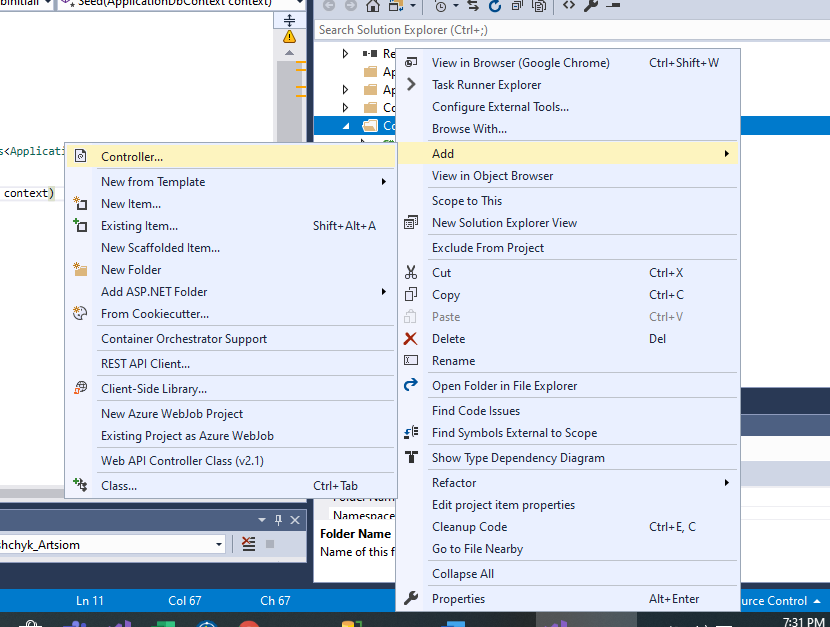


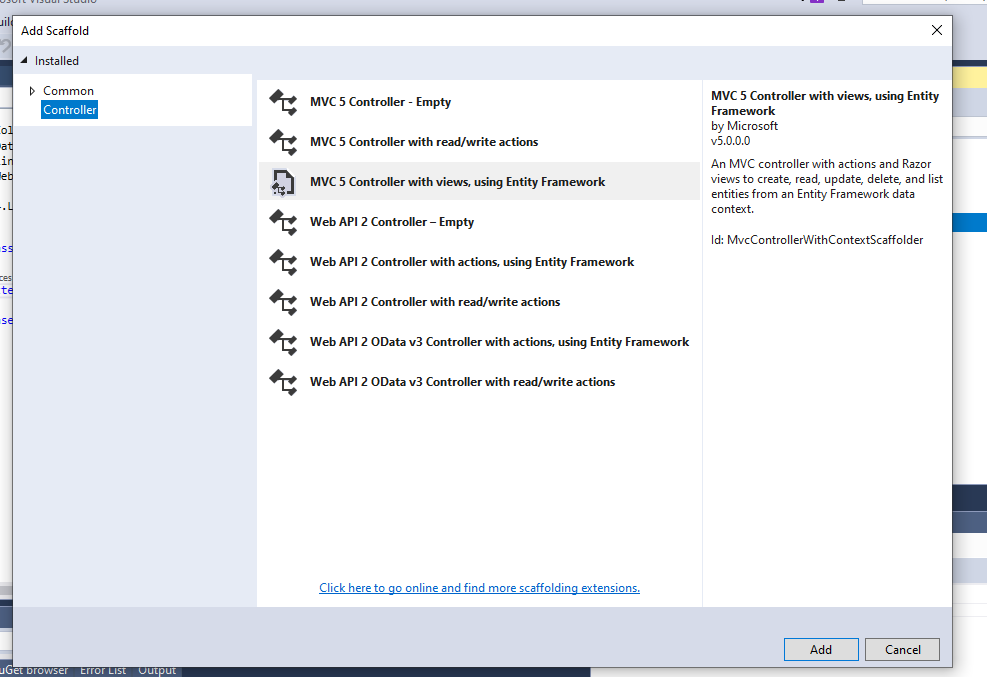
Далее с помощью встроенного инструмента Scaffolding сгенерируем CRUD страницы для имеющихся сущностей

Небольшая ремарка: этот метод подходит исключительно для учебных целей. Работу контроллеров можно подправить, добавив дополнительный слой абстракции в виде сервисов.

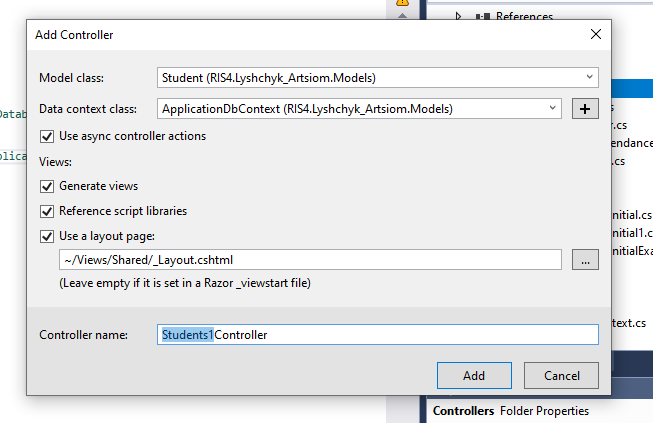
Делаем следующее

ПКМ по папке Controllers ->

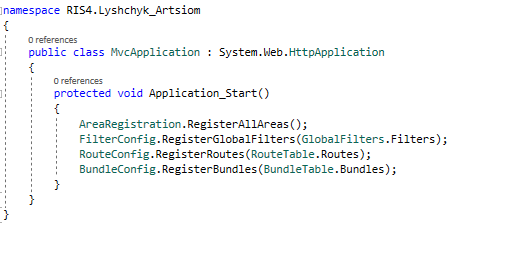




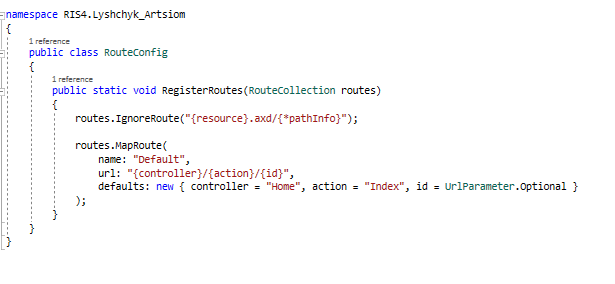
Выбираем выделенный шаблон на рисунке. Далее указываем Model class, т.е. класс нашей сущности.



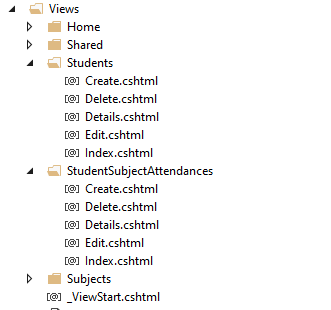
Т.к. в этом случае контроллер уже существует, предлагается название Students1Controller. Не нужно менять название контроллеров, т.к. в таком случае придётся изменять конфигурацию маршрутов (Route Config). Кстати, она представлена в файле Global.asax. Этот файл содержит точку входа (Entry Point) приложения.



А сама конфигурация происходит в классе Там же происходит конфигурация фильтров и бандлов (подтягиваемых зависимостей (скрипт-файлов и т.п.)).



И так, после создания для всех моделей, мы можем увидеть следующую иерархию Views.

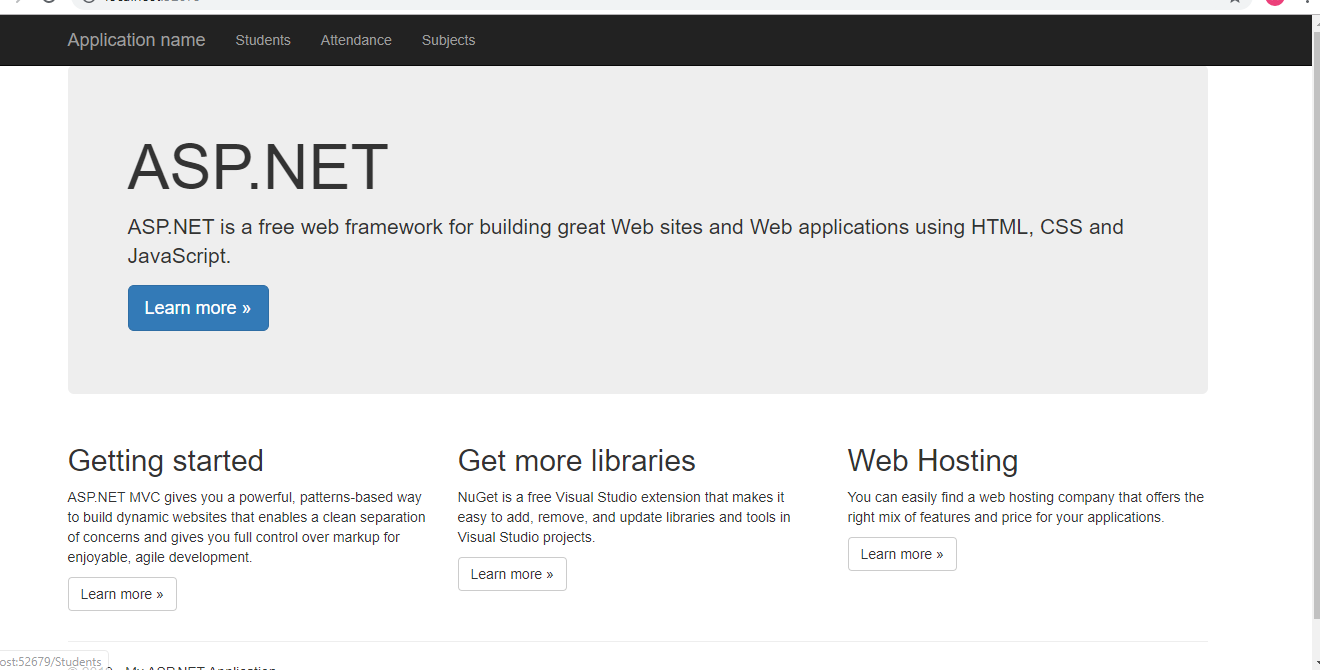


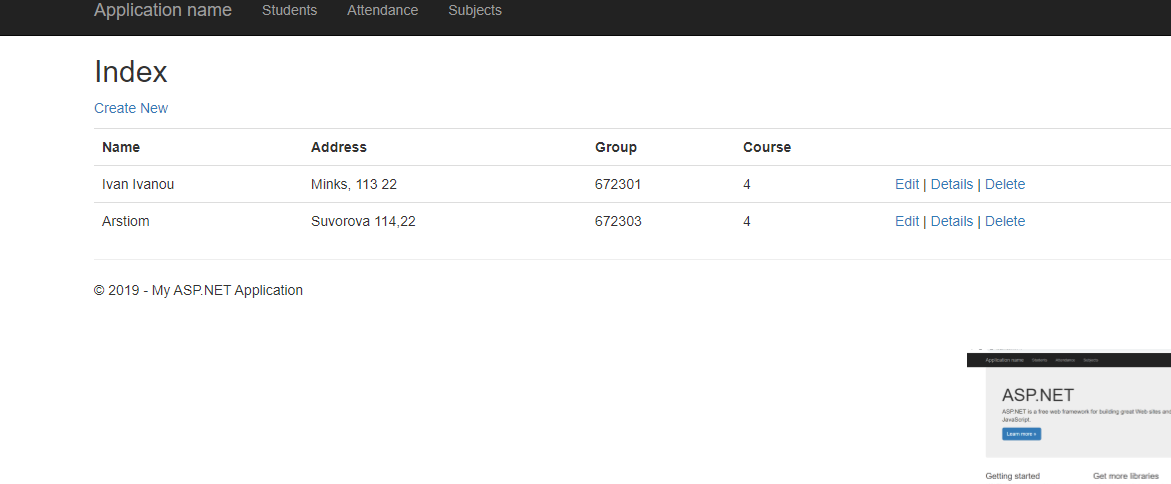
Сделаем следующее – обратимся к Views->Shared->Layout.

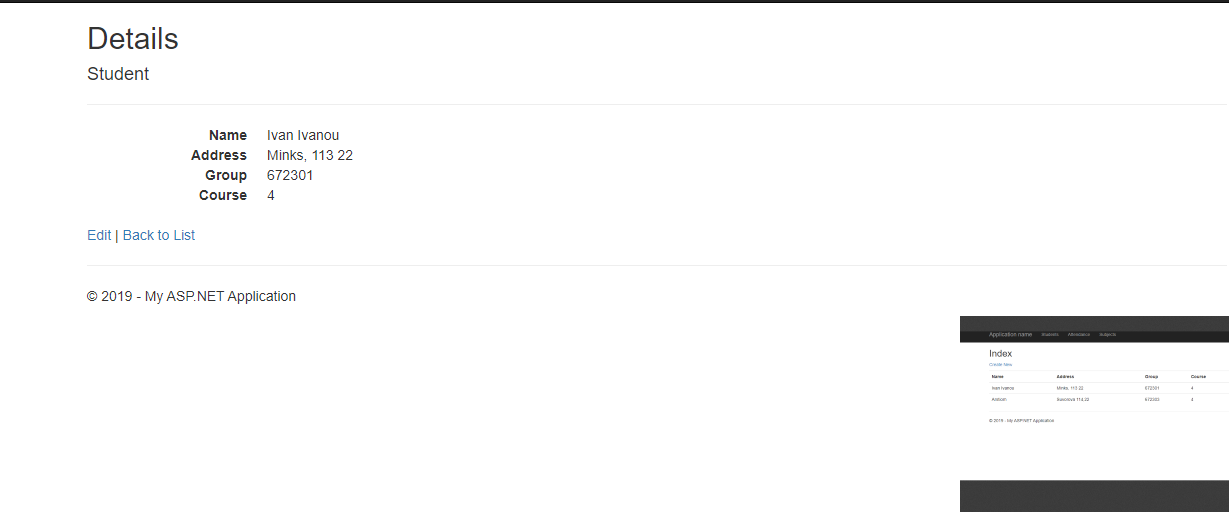
И изменим имеющиеся линки в навигационной панели на такие, как показано на рисунке.

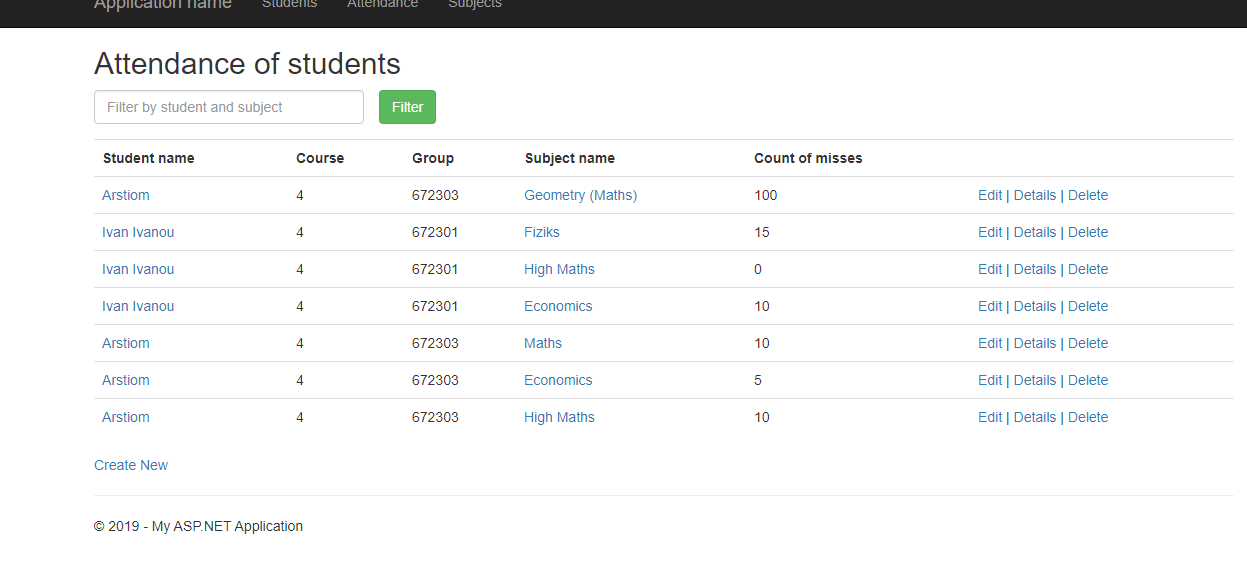


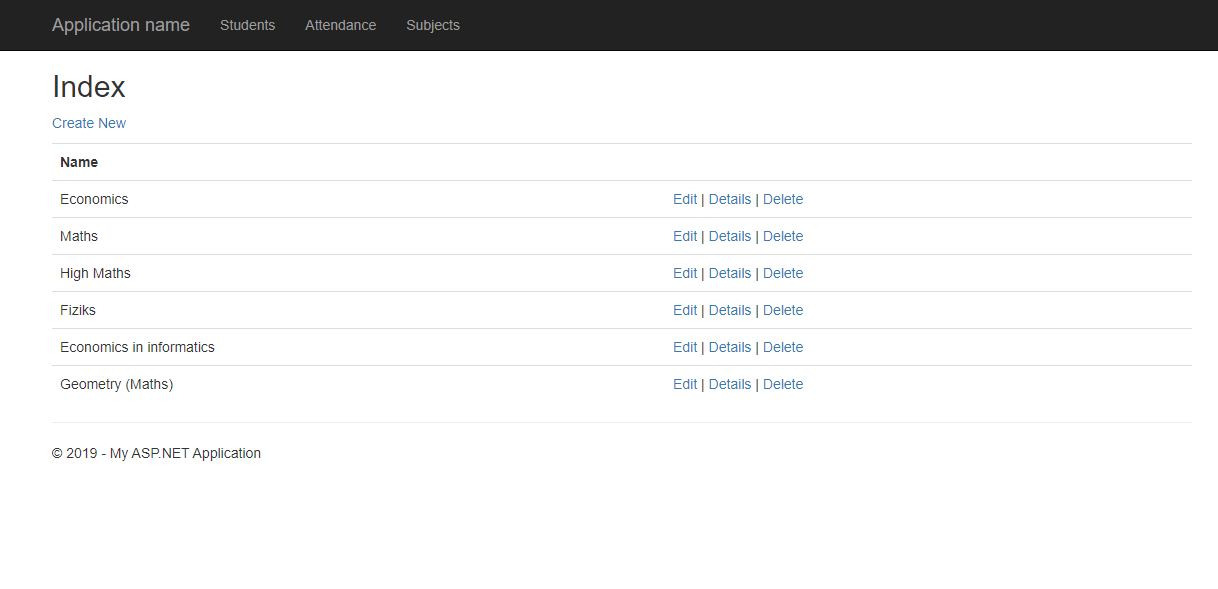
Запустим приложение, и посёрфим по линкам.



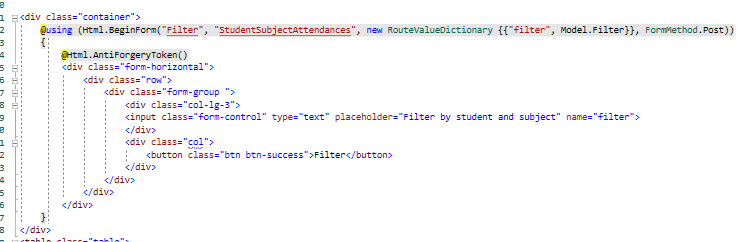




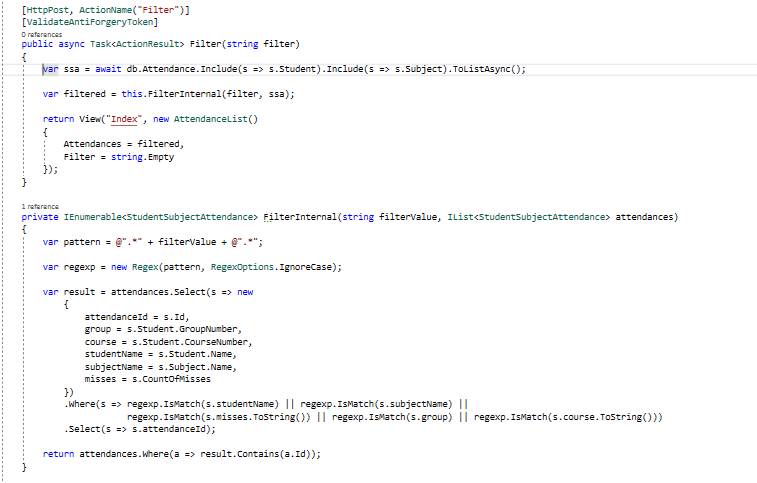




На страницу Attendance of students добавим такой же фильтр, как показано на рисунке.

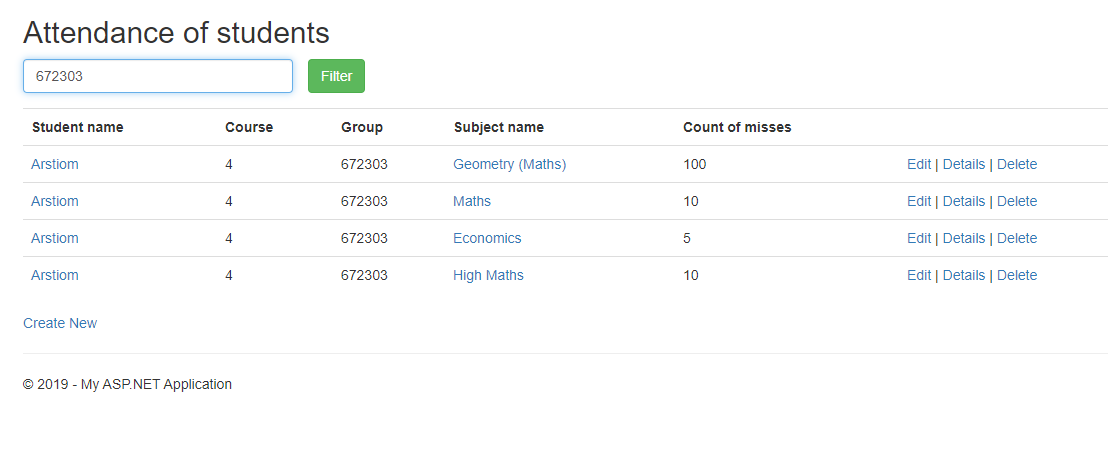


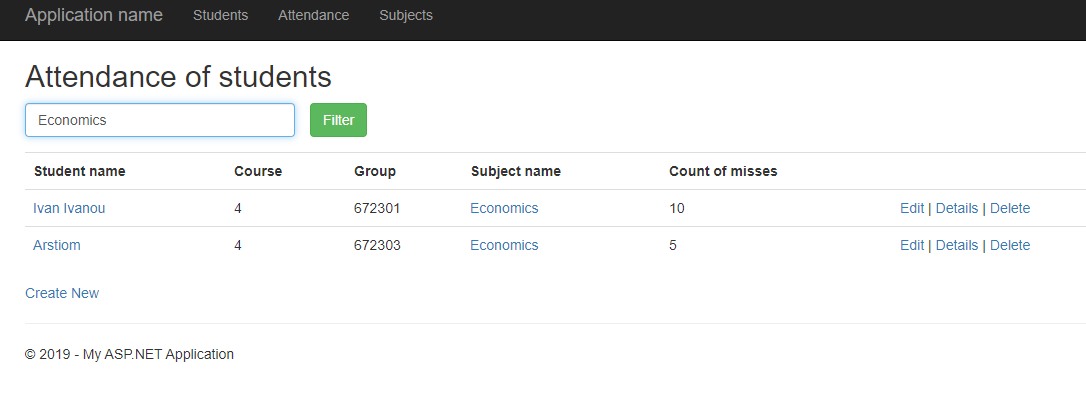
Это html-код будущего фильтра. Далее добавим в контроллер этой страницы указанный метод с аннотацией post (можно использовать GET, т.к. всё равно используется url сегменты маршрутизации для передачи параметров фильтру).

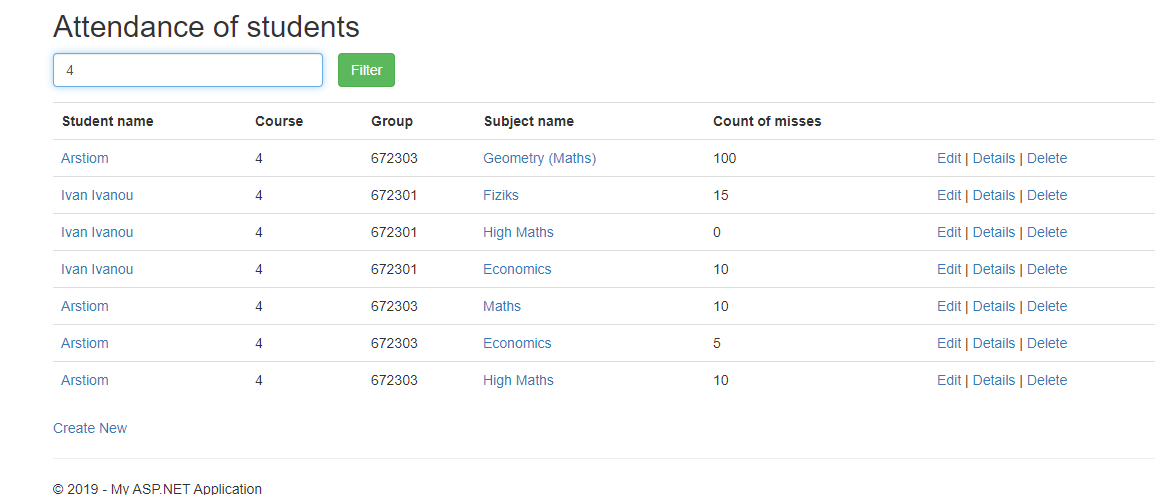


Фильтр будет работать по всем полям.

Проверим его работу.







Так же добавим линки для Student name и Subject name для более удобной навигации.

Для этого во Views-StudentSubjectAttendances-> Index изменим следующее



Проверим работоспособность нового действия.

