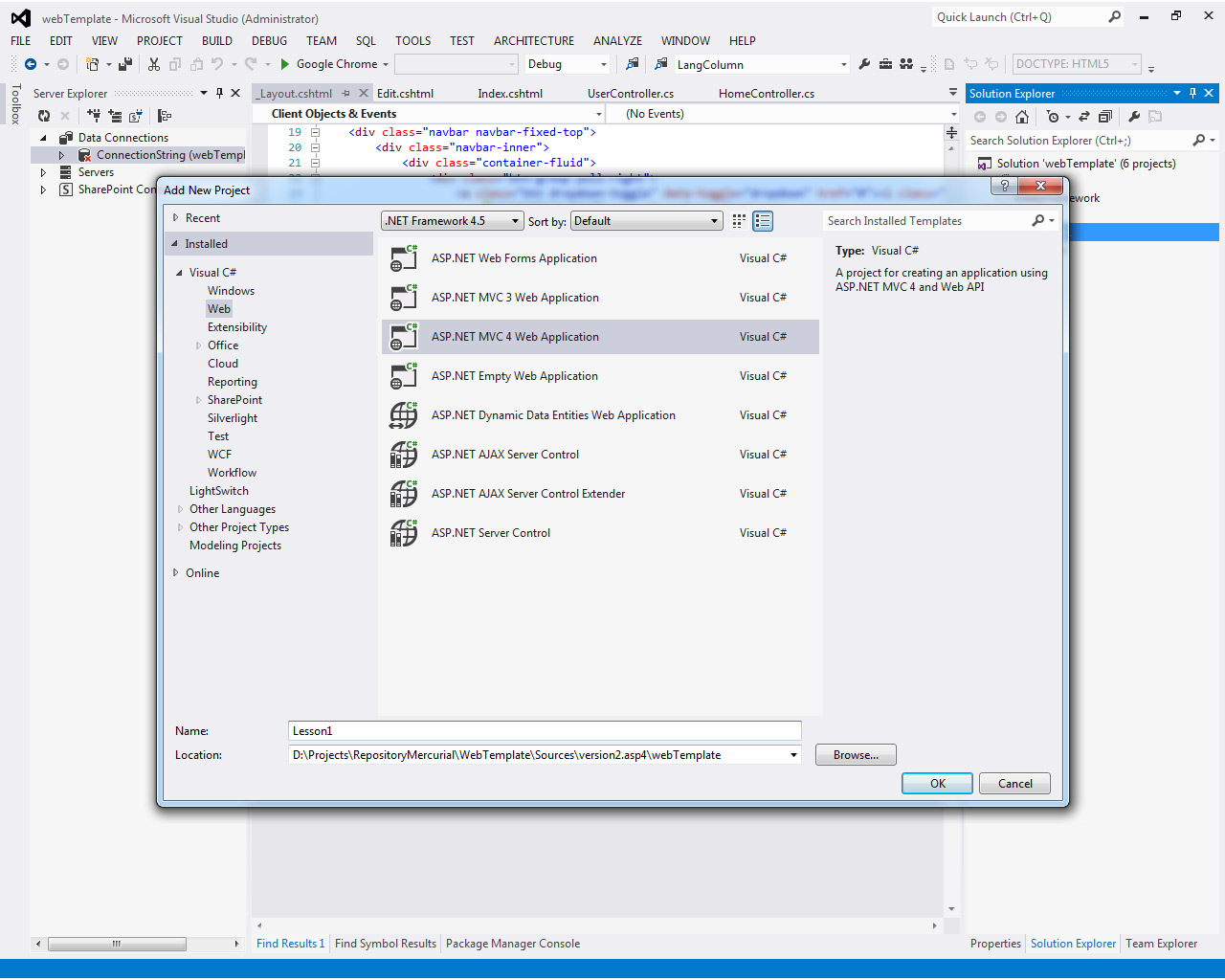
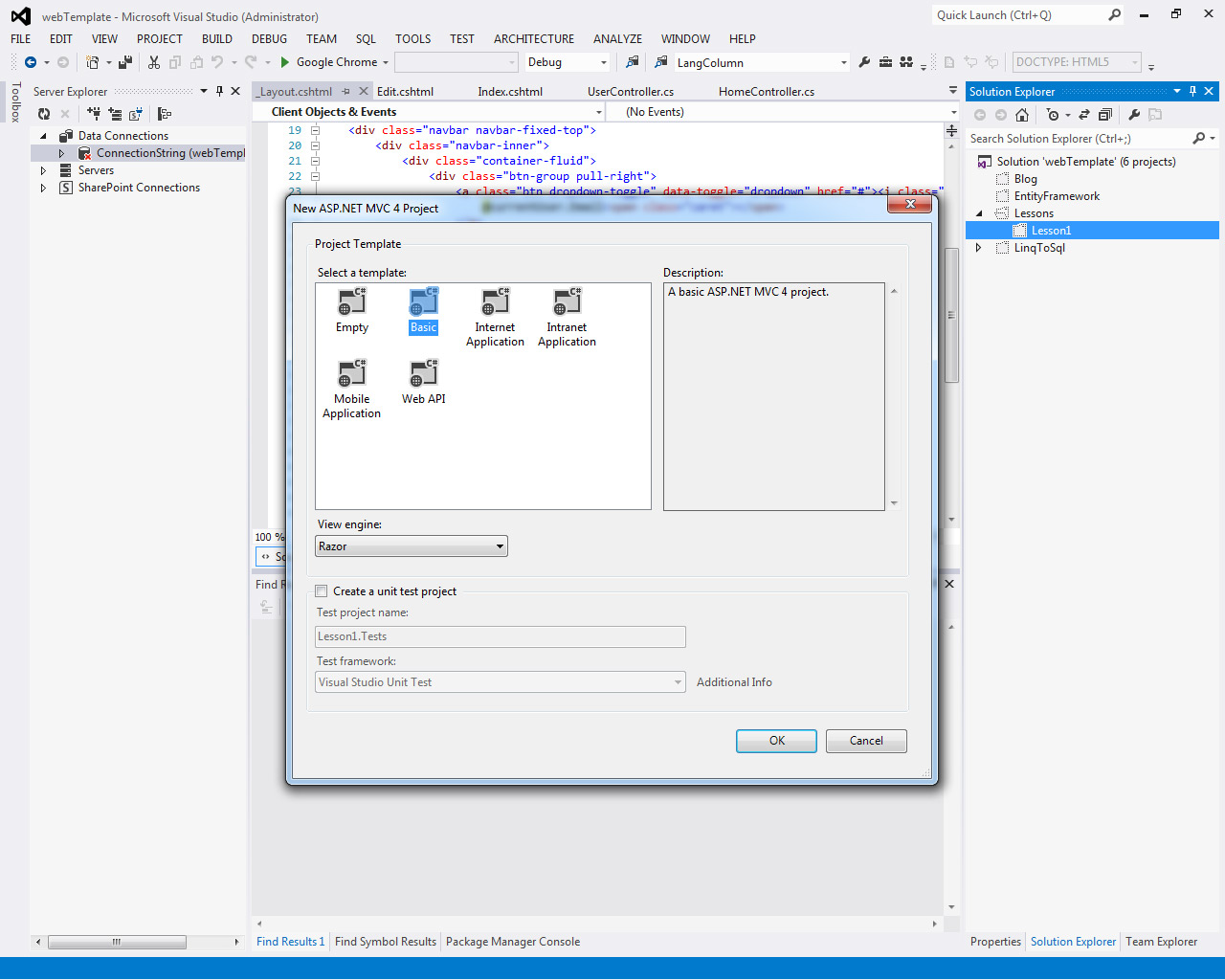
## Урок 1 – Чопочем или из чего что состоит.

**Цель урока:** Изучить из чего состоит проект asp.net mvc (базовый), с чего всё начинается и как с этим работать. Изучение Nuget и Подключение протоколирования.

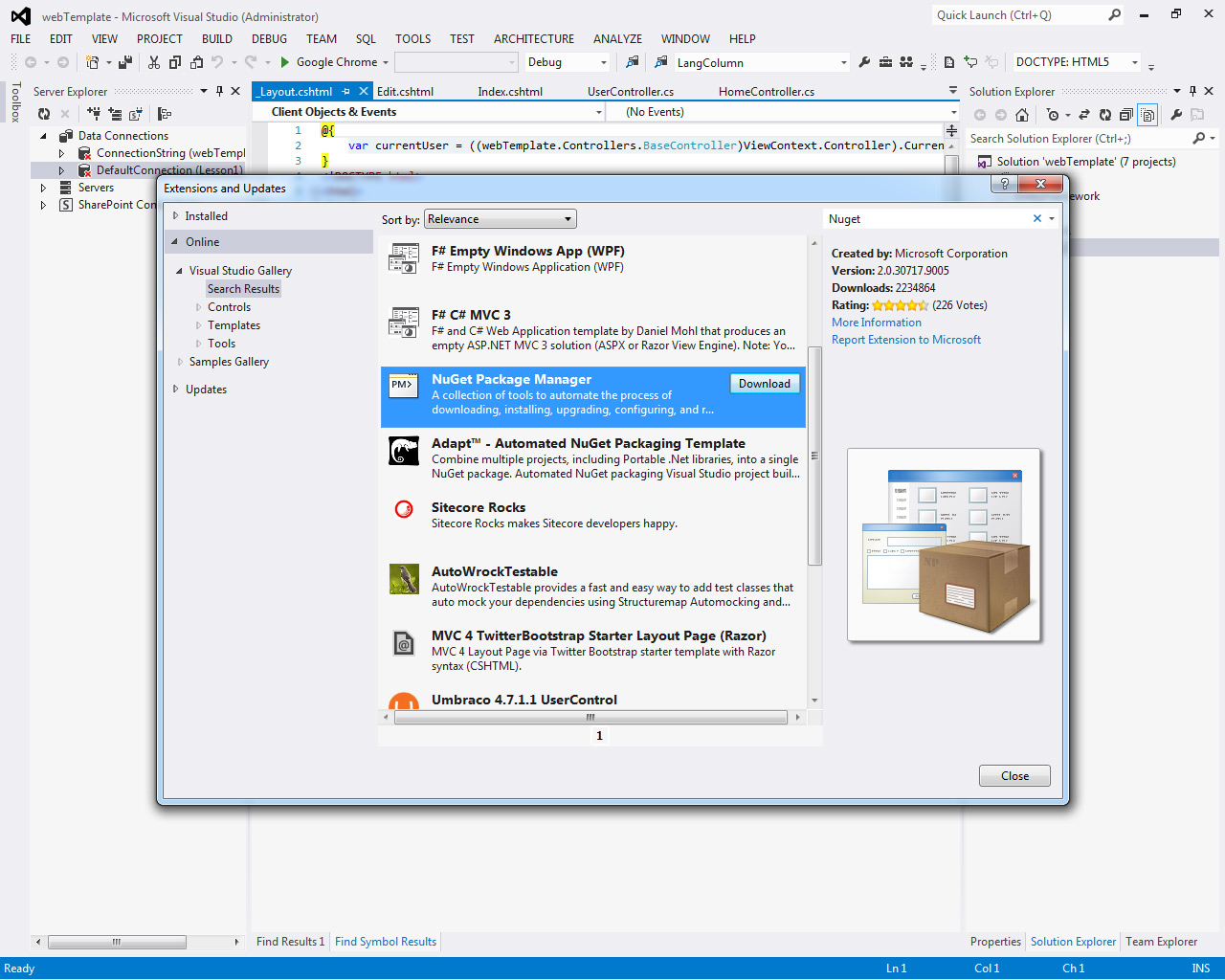
### Начало

Создадим приложение ASP.NET MVC 4 Web Application «Lesson1».

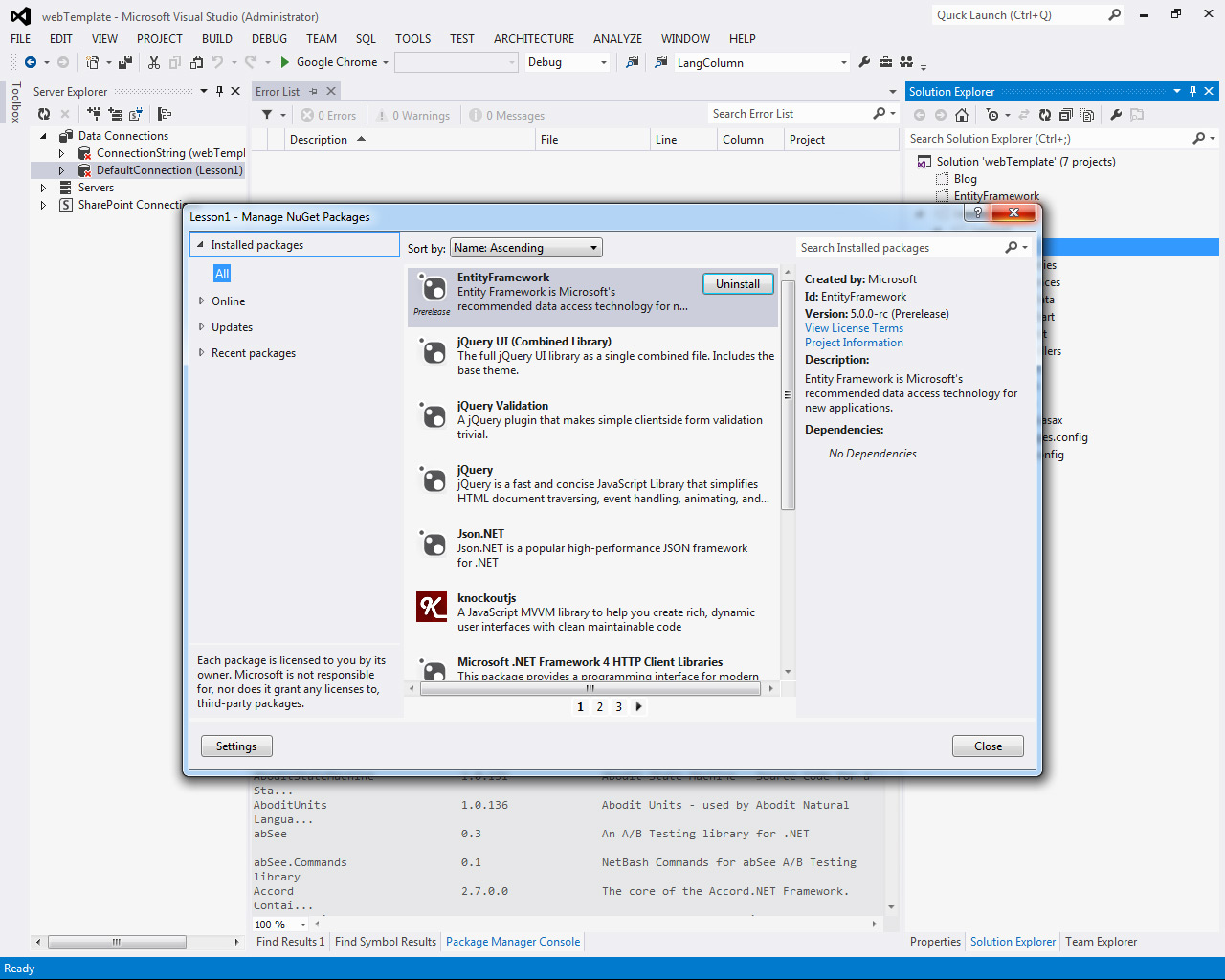




Не будем запускать приложение, а сразу установим (если до этого не сделали) NuGet расширение.



NuGet Package Manager – это расширение для Visual Studio, которое позволяет добавлять в существующие проекты модули, которые значительно упрощают работу. При создании «Basic» asp.net mvc 4 приложения в само приложение было добавлено изначально много модулей. Их список мы можем найти, кликнув в Manage NuGet Packages… в контекстном меню проекта

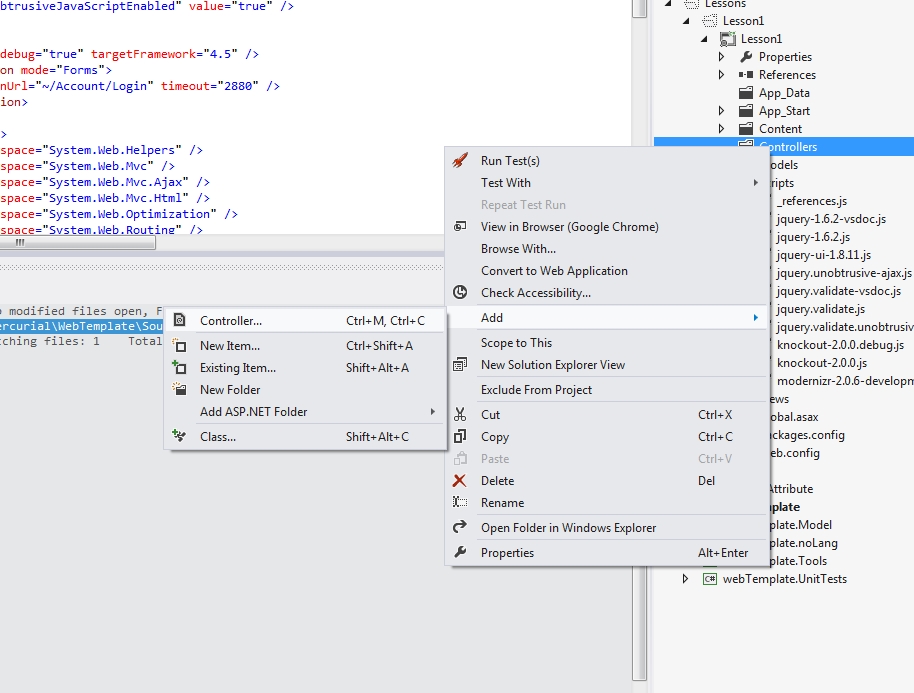


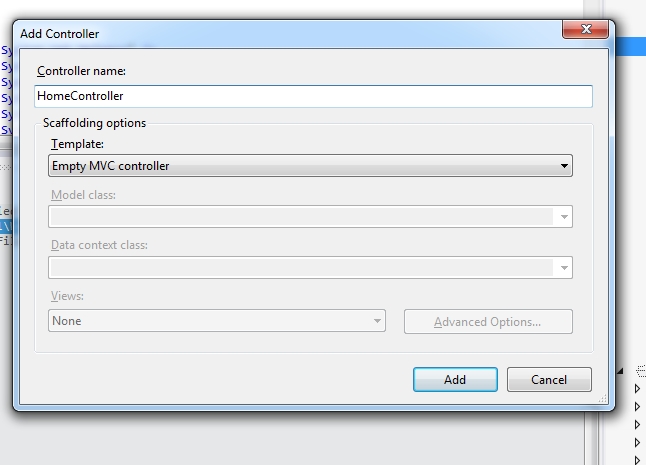
Из них по порядку:

* **Entity Framework** – обеспечивает работу с БД
* **jQuery (+ jQuery UI, jQuery Validation)** – популярный javascript framework (о нем потом еще пойдет речь).
* **Json.NET** – классы для работы с json-форматом данных.
* **knockoutjs** - не популярная javascript библиотека для работы с  Model View ViewModel архитектурой. (<http://knockoutjs.com/>)
* **Microsoft.Net Framework 4 Http Client Libraries** – программный интерфейс для работы с HttpContextом.
* **Microsoft ASP.NET MVC 4** – собственно классы паттерна проектирования MVC (то что есть основа всего этого)
* **Microsoft ASP.NET Razor 2** – view-движок. Есть еще ASPX и Spark – мы о них тоже будем говорить потом
* **Microsoft ASP.NET Universal Providers Core Libraries (Microsoft ASP.NET Universal Providers for LocalDB)**– предоставляет услуги для поддержки всех SQL Server 2005 (и более поздних) и для SQL Azure.
* **Microsoft ASP.NET Web API (Microsoft ASP.NET Web API Client Libraries, Microsoft ASP.NET Web API Core Libraries, Microsoft ASP.NET Web API Web Host)** – для создания REST приложений, работа с XML, JSON и всё такое
* **Microsoft ASP.NET Web Optimization Framework** – оптимизирует передачу данных, например минимизирует js-код
* **Microsoft ASP.NET WebPages 2** – набор классов для работы во View
* **Microsoft jQuery Unobtrusive Ajax (Microsoft jQuery Unobtrusive Validation)–** jQuery библиотека для поддержки ненавящевого ajax/валидации
* **Microsoft.Web.Infrastructure** – позволяет динамически регистрировать HTTP модули во время выполнения
* **Modernizr** – js-библиотека которая позволяет использовать html5 и css3 на старых браузерах
* **WebGrease** – позволяет минифицировать html, css, js.

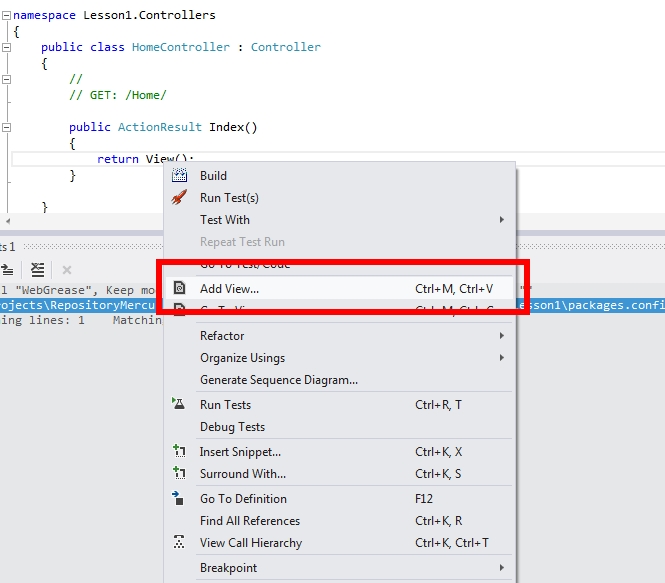
Ок, теперь когда мы так примерно прикинули из чего состоит наше приложение запустим его.

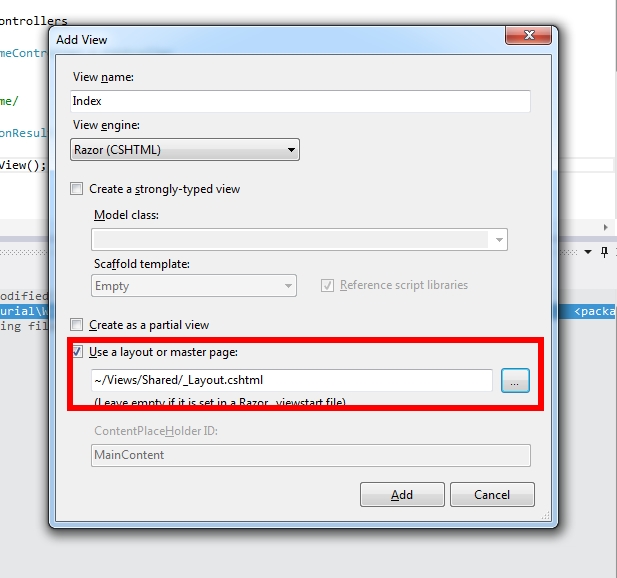
Для этого необходимо создать HomeController:





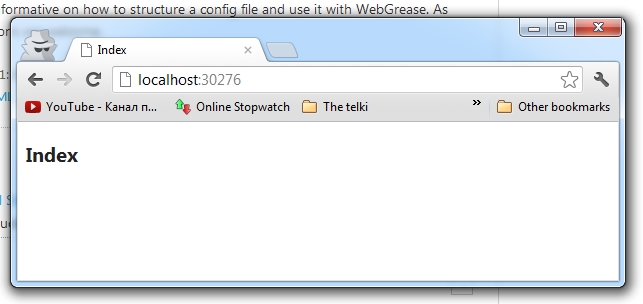
Для метода (действия) Index добавим View:





Собственно можно запускать.

Всё что мы увидим – это:



Почему контроллер надо было назвать именно Home, как это работает, более подробно мы будем изучать в следующих уроках. Так же мы изменим уже существующую структуру каталогов, чтобы нам потом проще работалось.

### Global.asax

А счас мы обратим внимание на файл Global.asax:

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

Класс MvcApplication наследует HttpApplication и содержит инструкции для инициализации приложения. Есть еще несколько событий, на которые можно добавить код. Рассмотрим их:

* **public void Init()** –приложение инициализируется или при первом вызове. Оно вызывается для всех экземпляров объекта HttpApplication.
* **public void Dispose()** – вызывается непосредственно перед уничтожением объекта HttpApplication. Это идеальное место для очистки ранее используемых ресурсов.
* **Application\_Error** – наступает, когда необработанное исключение случается в приложении.
* **Application\_Start** – наступает, когда первый экземпляр HttpApplication создается. Это позволяет создавать объекты доступные для всех экземпляров класса HttpApplication.
* **Application\_End** – наступает когда все созданные ранее экземпляры класса HttpApplication уничтожены. Это событие наступает только однажды в течении всего времени жизни приложения.
* **Application\_BeginRequest** – наступает когда приложение получает запрос. Первый раз это событие наступает для запроса страницы, когда пользователь вводит URL.
* **Application\_EndRequest** – Последнее событие которое наступает для запроса к приложению.
* **Application\_PreRequestHandlerExecute** – наступает прежде, чем ASP.NET запустит обработчик страницы или веб-службу.
* **Application\_PostRequestHandlerExecute** – наступает когда ASP.NET заканчивает обработку.
* **Applcation\_PreSendRequestHeaders** – наступает перед тем как ASP.NET посылает клиенту (браузеру) HTTP заголовки.
* **Application\_PreSendContent** – наступает перед тем как ASP.NET посылает клиенту (браузеру) HTTP содержимое.
* **Application\_AcquireRequestState** – наступает когда ASP.NET получает текущее состояние (состояние сессии) связанное с текущим запросом.
* **Application\_ReleaseRequestState** – наступает, когда ASP.NET завершает исполнение всех событий. В результате все модули сохраняют свои текущие состояния.
* **Application\_ResolveRequestCache** – наступает, когда ASP.NET выполняет запрос авторизации. Это позволяет модулям кеширования обработать запрос и обслужить из кэша, минуя обработчик выполнения.
* **Application\_UpdateRequestCache** – наступает, когда ASP.NET завершает выполнение обработчика, чтобы модули кеширования могли сохранить результат для использования в последующих ответах.
* **Application\_AuthenticateRequest** – наступает, когда модуль идентификации устанавливает личность текущего пользователя как действительные. На данный момент учетные данные пользователя уже проверены.
* **Application\_AuthorizeRequest** – наступает, когда модуль авторизации подтверждает, что пользователь может иметь доступ к ресурсам.
* **Session\_Start** – наступает, когда новый пользователь заходит на сайт.
* **Session\_End** – наступает, когда истекает время сессии пользователя, или он покидает сайт.

Ок, теперь чтобы воочию убедиться, что всё именно так и происходит, добавим протоколирование и сделаем это через добавление NLog модуля в NuGet

### Package Manager Console

В NuGet есть консоль для выполнения команд по установке\удалению\поиску модулей, и других вещей, типа скаффолдинга.

Для вывода всех установленных модулей пишем:

Get-Package

Для получения всех доступных к установке модулей:

Get-Package –ListAvailable

Для получения всех доступных модулей с названием NLog

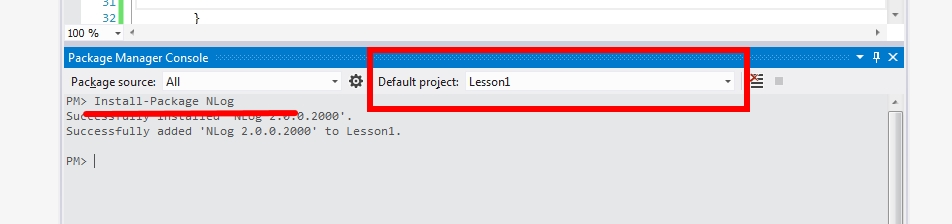
Get-Package -ListAvailable -Filter NLog

Или

Get-Package -ListAvailable | where {$\_.Id -match "NLog"} (это дольше)

Для установки модуля NLog необходимо вначале выбрать проект (если их в солюшене больше одного) и ввести команду:

Install-Package NLog



Файлы копируются в проект, добавляются ссылки на сборки и web.config может быть обновлен.

Для удаления из проекта модуля необходимо, чтобы он не был связан с другими модулями. Удаляем так:

Uninstall-Package NLog

### NLog

После установки пользуемся документацией на NLog (<http://nlog-project.org/wiki/Tutorial>) – и добавляем в web.config:

<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<targets>

<target name="logfile" xsi:type="File" fileName="C://file.txt" />

</targets>

<rules>

<logger name="\*" minlevel="Info" writeTo="logfile" />

</rules>

</nlog>

Мы ее потом исправим, а пока проверим как есть.

Добавим в код:

protected void Application\_Start()

{

**logger.Info("Application Start");**

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

**public void Init()**

**{**

**logger.Info("Application Init");**

**}**

**public void Dispose()**

**{**

**logger.Info("Application Dispose");**

**}**

**protected void Application\_Error()**

**{**

**logger.Info("Application Error");**

**}**

**protected void Application\_End()**

**{**

**logger.Info("Application End");**

**}**

**...**

Запустим и откроем файл C://file.txt, завершим приложение (Stop). Мы увидим какие события наступили:

2012-09-18 19:18:11.5668|INFO|Lesson1.MvcApplication|Application Start

2012-09-18 19:18:13.7319|INFO|Lesson1.MvcApplication|Application Init

2012-09-18 19:18:14.2709|INFO|Lesson1.MvcApplication|Application Init

2012-09-18 19:18:14.2769|INFO|Lesson1.MvcApplication|Application BeginRequest

2012-09-18 19:18:14.3579|INFO|Lesson1.MvcApplication|Application AuthenticateRequest

2012-09-18 19:18:14.3579|INFO|Lesson1.MvcApplication|Application AuthorizeRequest

2012-09-18 19:18:14.3579|INFO|Lesson1.MvcApplication|Application ResolveRequestCache

2012-09-18 19:18:14.3989|INFO|Lesson1.MvcApplication|Session Start

2012-09-18 19:18:14.3989|INFO|Lesson1.MvcApplication|Application AcquireRequestState

2012-09-18 19:18:14.3989|INFO|Lesson1.MvcApplication|Application PreRequestHandlerExecute

2012-09-18 19:18:15.9580|INFO|Lesson1.MvcApplication|Application PreRequestHandlerExecute

2012-09-18 19:18:15.9580|INFO|Lesson1.MvcApplication|Application ReleaseRequestState

2012-09-18 19:18:15.9580|INFO|Lesson1.MvcApplication|Application UpdateRequestCache

2012-09-18 19:18:15.9580|INFO|Lesson1.MvcApplication|Application EndRequest

2012-09-18 19:18:15.9580|INFO|Lesson1.MvcApplication|Application PreSendRequestHeaders

2012-09-18 19:18:35.6061|INFO|Lesson1.MvcApplication|Session End

2012-09-18 19:18:38.0833|INFO|Lesson1.MvcApplication|Application Dispose

2012-09-18 19:18:38.0833|INFO|Lesson1.MvcApplication|Application End

2012-09-18 19:18:39.1383|INFO|Lesson1.MvcApplication|Application Dispose

В Application\_Start выполняется 4 строки регистрации Area (область), Filter (фильтров), Bundle (пучок), Route (маршрутов).

Подробности по инициализации Filter, Вundle и Route находятся в папке App\_Start.

### WebActivator

WebActivator – это модуль, который позволяет запустить код до самого первого старта App\_Start. Это может быть необходимо для того, чтобы, например, создать тестовую БД перед запуском (потом сделаем).

Установим:

Install-Package WebActivator

Добавим класс в App\_Start папку:

[assembly: WebActivator.PreApplicationStartMethod(typeof(PreStartApp), "Start")]

namespace Lesson1.App\_Start

{

public static class PreStartApp

{

private static NLog.Logger logger = NLog.LogManager.GetCurrentClassLogger();

/// <summary>

/// Метод запускается один раз перед стартом приложения

/// </summary>

public static void Start()

{

logger.Info("Application PreStart");

}

}

}

В файле логов увидим что строка Application PreStart исполняется раньше Application Start:

2012-09-19 10:29:01.3950|INFO|Lesson1.App\_Start.PreStartApp|Application PreStart

2012-09-19 10:29:01.6290|INFO|Lesson1.MvcApplication|Application Start

Напоследок сделаем запись логов более осмысленными в файлы разложенные по разным категориям и пишущие в папку Contents а не в корень C:

Создадим 4 файла отдельно для trace (трассировки), debug (отладки), info (информации), error (ошибки). Определим место записи:  **/Contents/logs/[текущая дата]** Перепишем конфигурацию:

<nlog autoReload="true" xmlns="http://www.nlog-project.org/schemas/NLog.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<variable name="logDirectory" value="${basedir}/Content/logs/${shortdate}" />

<targets>

<target name="fileLogTrace" xsi:type="File" fileName="${logDirectory}/trace.txt" />

<target name="fileLogDebug" xsi:type="File" fileName="${logDirectory}/debug.txt" />

<target name="fileLogInfo" xsi:type="File" fileName="${logDirectory}/info.txt" />

<target name="fileLogErrors" xsi:type="File" fileName="${logDirectory}/errors.txt" />

</targets>

<rules>

<logger name="\*" level="Trace" writeTo="fileLogTrace" />

<logger name="\*" level="Debug" writeTo="fileLogDebug" />

<logger name="\*" level="Info" writeTo="fileLogInfo" />

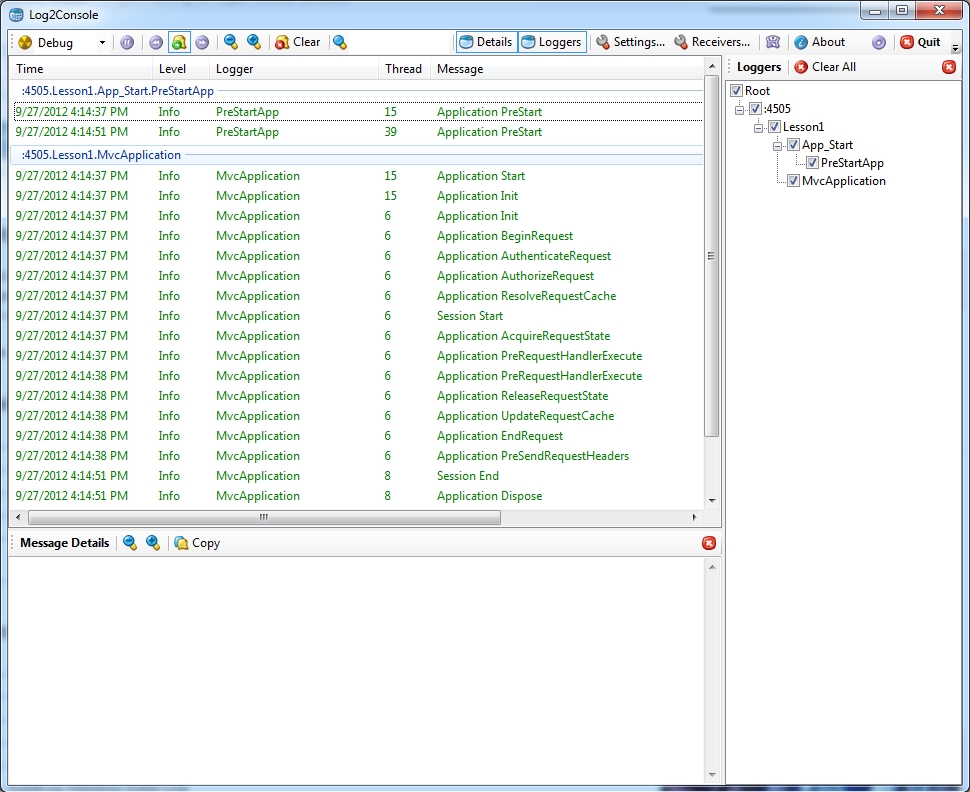
<logger name="\*" minlevel="Warn" writeTo="fileLogErrors" />

</rules>

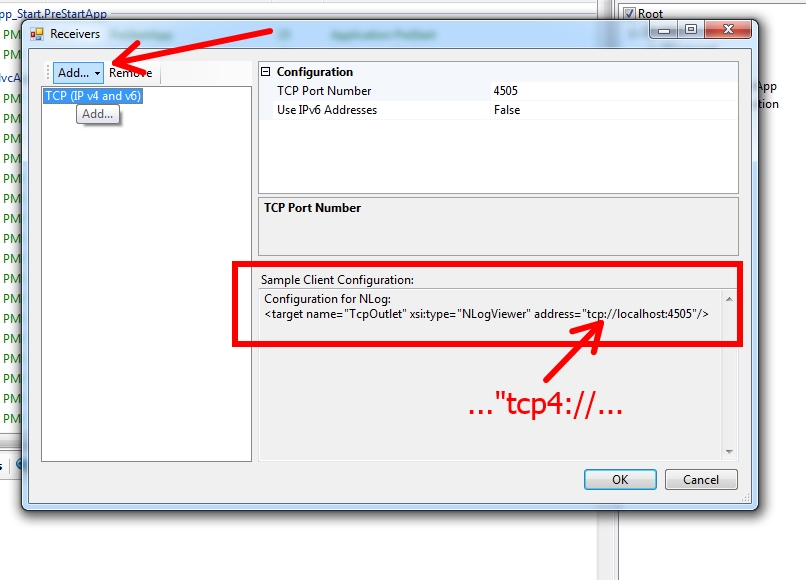
</nlog>

### Log2Console

Для NLog есть еще классная программа Log2Console, которая позволяет получать логи прямо в окно программы.



Запускаем программу и настраиваем приемщик:



В Web.config пишем:

<target name="TcpOutlet" xsi:type="NLogViewer" address="tcp4://localhost:4505"/>

Обращаю внимание, что писать надо **address=”tcp4://…”,** а не **address=”tcp://…”**