ASP.NET - это средство создания динамических веб-страниц. Прежде чем страница будет отображена в браузере пользователю, движок ASP.NET обрабатывает входящий запрос, выполняет какую-то свою бизнес-логику, и затем выгружает на клиент сгенерированный HTML.

ASP.NET работает на веб-сервере IIS (Internet Information Service). Это несколько привязывает его к Windows-среде (т.е. хостинг или VPS сервер для ASP.NET нужен именно с ОС Windows Server). Хотя в новой версии MS сделали шаг в сторону IOS и Linux - ASP.NET MVC5 может работать и на этих платформах.

Есть 2 типа ASP.NET - MVC и Web Forms. Мы в дальнейшем будем использовать MVC. Вкратце рассмотрим также Web Forms, чтобы Вы могли поддерживать такие проекты.

Web Forms

Проект состоит из страниц (.aspx), которые могут быть вызваны пользователем. У каждой страницы есть Code Behind - aspx.cs - это файл кода, который содержит исполняемый код страницы. В этом файле Вы можете обрабатывать различные события страницы - нажатие кнопки, загрузка страницы (событие Page_Load), рендеринг страницы (Page_PreRender).

Разметка в ASPX может содержать специальные теги с префиксом ASP.NET - это встроенная функциональность от платформы ASP.NET. Любой серверный компонент имеет атрибут runat=”server”. Этот атрибут говорит системе, что для него нужно вести серверную обработку. Состояние элемента при перезагрузке страницы запоминается с специальное поле (ASP.NET ViewState). При большом количестве серверных элементов это поле становится очень большим - это один из основных минусов WebForms по сравнению с MVC.

Если Вы выбрали путь JS+AJAX, то не рекомендую ими пользоваться - лучше пишите свою HTML разметку и используйте свои JS компоненты. Руководствуйтесь следующим правилом: рендеринг данных для SEO должен производиться с помощью декларативной разметки <% %>, а все управляющие операции возложите на Ajax. Т.е. таким образом мы по минимуму используем инфраструктуру Web Forms. В дальнейшем это сильно облегчит миграцию ваших решений на другие платформы (если в этом будет необходимость, например, на ASP.NET MVC).

Про <% %>:
Это управляющие конструкции, которые позволяют писать C# код на странице ASPX.

Вместо использования ASP.NET контрола Repeater используйте следующий вариант:

```
<% foreach(var item in Items) { %>
        <p>
                <span data-itemID='<%= item.id %>'><%= item.name %></span>
        </p>
<% } %>
```

Items должна быть объявлена как public свойство класса страницы в соответствующем файле aspx.cs
Заметьте разницу между <% %> и <%= %>. Первый - это просто управляющая конструкция на странице (if, foreach, var i=0), а второй - это вывод чего-то на страницу.

Задание 1. Создайте страницу (перед этим создайте отдельный проект Web Forms) и выведите список из какой-либо таблицы Вашей базы с помощью <% %>.

Как общаться между страницами? Когда Вы делаете переходы между страницами, часто необходимо передавать какие-то значения.

Есть несколько способов это сделать. Их нужно использовать в уместном контексте. Давайте их рассмотрим.

1. Через строку URL
Пример:
/default.aspx?id=3&type=type1
Чтобы получить эти параметры на странице, используйте Page.Request[“id”]. Его имеет смысл использовать, когда Вы передаете общедоступные данные. Если есть какие-то секретные данные - Вы их записываете в хранилище (базу) с каким-то Guid и передаете в строке URL этот Guid. А на странице уже извлекаете Guid и, соответственно, данные по этому Guid.
2. Через Postback. Мы это не используем, но сказать о нем следует. При нажатии на серверную кнопку происходит обратная отправка на сервер. Страница автоматически загружает в контролы данные по их состоянию (по сути введенные значения на странице).
3. Через сессию. Тоже вариант (объект Session), но это нагружает сервер. Если в сессии хранится много данных, то все это нагружает память сервера. В данный момент мы не используем это хранилище.
4. Application. Если информация относится ко всему приложению в целом - то можно хранить в этом хранилище.
5. Профиль пользователя. Мы сделали для себя свой профиль, Вы можете попробовать использовать стандартный ASP.NET Profile. Он используется для того, чтобы хранить данные по конкретному пользователю.

6. Cookie пользователя. Помните, что cookie хранятся на клиенте пользователя и передаются с каждым запросом. Старайтесь не хранить там слишком много данных.

Какие объекты Вам надо изучить:
Request.Url - это все данные о текущем Url
HttpContext.Current - это все данные о текущем запросе
HttpContext.Current.Identity - данные о текущем юзере (авторизован ли он, его имя).
Page - это переменная, доступная на странице Page
Server (функции MapPath, HtmlDecode, HtmlEncode и др.) - все, что связано с сервером

Задание 2. Сделайте страницу, где будут выводиться данные по одной записи из таблицы, которую вы создали ранее. На таблице сделайте ссылку на эту страницу (передавайте параметр через URL).

Компоненты

На мой взгляд, главной фишкой Web Forms по сравнению с MVC являются компоненты. По сути, это законченный кусок кода, который можно вставить на страницу. Здесь идет полная аналогия со страницами (только расширение ASCX). Попробуйте создать простейший компонент и внедрить на страницу. У компонентов могут быть свойства (public) и их можно будет устанавливать в разметке компонента на странице.

```
public string ItemID {set; get; }
```

```
<uc1:Comp ID="comp1" runat="server" Title='11111' ItemID='2' />
```

Примечание. Добавлять компоненты на страницу лучше через Design Mode.

Еще важный момент - изучите под отладкой как происходят события в странице. Добавьте все обработчики страницы (google:asp.net page lifecycle) и поставьте прерывания (слева от строки появится красная точка). Также добавьте кнопку и ее обработчик. Посмотрите порядок этих событий в процессе отладки (F10 -следующий шаг, F5 - запустить / отпустить отладку).

Мастер-страница

Существует такое понятие, как мастер-страница. Это страница, которая является шаблоном для других страниц. Она содержит общие элементы для страниц. Обычно страницы сайта имеют общие блоки: шапка, подвал, меню, поиск и т.д. Все это оформляется с помощью мастер страниц.

Задание 3. Google: using asp.net master pages - создайте одну мастер страницу с каким-то текстом и для нее создайте 2 страницы. Попробуйте сделать вложенные мастер-страницы.

Задание 4. Изучите класс Membership и как создавать/подключать базу управления пользователями для Membership.

ASP.NET MVC

Дальше мы работаем только с MVC. У нас есть довольно много проектов на Web Forms (более того, у нас была разработана CMS под Web Forms), но все-таки будущее за MVC, поэтому изучаем ее.

Общее устройство MVC

3 слоя - Модель, Представление, Контроллер

Все начинается с Контроллера - он принимает запрос и через Модель извлекает данные, далее он передает эти данные в определенном виде на Представление, которое формирует вывод в браузер. Все. Можно заканчивать главу. Дальше только детали.

В чем удобство такой модели? Ваше приложение организационно разбивается на 3 слоя. Каждый слой в общем случае можно делать отдельно. Кто-то делает представление, кто-то занимается моделью. Важно только определить интерфейсы взаимодействия - т.е. форматы-интерфейсы между слоями. Между Моделью и Контроллером - это сигнатуры функций вызова. Между Контроллером и Представлением - формат выдачи данных (по сути определения классов Модели).

Следующий момент - MVC упрощает сопровождение. Каждый слой отвечает за свою функцию. Паттерн задает определенный способ разработки, что уменьшает некоторые негативные моменты вроде спагетти кода и т.д.

MVC позволяет внедрить в проект автоматизированное тестирование. Мы уже практически созрели для внедрения тестов в наши проекты. Модульные тесты можно внедрять для тестирования работы контроллеров и модели. Обычно здесь никак не обойтись без контейнера IOC и Mock объектов. Еще раз повторюсь, что в данный момент мы не делаем автоматизированные тесты.

Обычно имеет смысл делать в таком порядке:

- создаем BLL и DAL (можно просто заглушки). Мы определяем вид модели.

- создаем структуру сайта - через контроллеры и их методы. Также мы создаем default Представления. На этом этапе можно по сути полностью воссоздать основные методы в Контроллерах.
- один программист начинает делать начинку BLL, другой создает View.

Контроллер
Контроллер - это специальный класс. В нем есть действия (Action)- это специальные методы в классе, которые вызываются через URL

Home/Index - вызываем контроллер Home и у него действие Index. У действия могут быть параметры. Причем модель MVC может самостоятельно обратывать входящие переменные и строить из них объекты для передачи на вход действиям.

Построение URL строится на базе использования **Маршрутизации** - т.е. способа определения Контроллера и Действия по URL, а также обратный процес - построение URL по параметрам Контроллер, Действия, переменные.

В целом вы можете не трогать сначала Маршрутизацию (т.е. не менять настроек в RouteConfig, который вызывается в Global.asax). MVC сделан таким образом, что вы можете его использовать по очень простому пути с постепенным усложнением (пока просто используйте по умолчанию реализации основных механизмов).

У Действий и Контроллеров есть **Фильтры** - они задают способ взаимодействия с данным Действием.

Например [NonAction] - указывает что этот метод не является действием, [Roles...] - указывает какие роли могут использовать этот Метод. [Authorized] - указывает, что данный метод может быть вызван только авторизованным пользователем. [HttpPost] - метод можно вызвать только при Post (полезно указывать для Ajax запросов, которые обычно идут через Post).

Это основные и самые популярные фильтры. Вы также можете создавать свои более сложные фильтры (например, действие можно вызывать только в определенное время дня). На мой взгляд, на первых порах можно обходиться простой бизнес логикой внутри самих методов. Также помните, что фильтры могут привести в неявным проблемам производительности (если содержат тяжелую бизнес-логику).

Контроллер выдает на выходе объект ActionResult - это может быть View (представление), JSON объект и др менее распространенные типы. Опять же здесь все можно расширять, например выдавать Excel объект.
Обычно используется либо View, либо JSON, либо Redirect. Для View в параметре указывается модель, на основании данных которой будет строиться представление. Второй способ общения View с Controller - это коллекция ViewBag - вы можете что угодно передавать дополнительно через эту коллекцию (ее поля создаются динамически).

Общий процесс построения контроллера:

- обработка входных данных (если это необходимо)
- вызов метода модели
- подготавливаем результат к выводу (во View в виде Model и дополнительных данных во ViewBag).

Т.е. методы контроллера в идеале должны иметь примитивную структуру и не иметь никакой бизнес-логики в себе.

Модель
Мы уже по сути обсуждали Модель в предыдущей главе - это классы данных (сущности), а также классы Менеджеров, которые содержат всю бизнес-логику приложения.

Куда отнести подготовку формата данных - к контроллерам или к модели? Очень часто в ajax запросах данные должны быть сформированы в некий JSON. На мой взгляд, лучше это делать в контроллерах. Модель должна просто отвечать за бизнес-логику и давать общие данные, а сам контроллер готовит упаковку этих данных для вывода в представления. Это более универсальный метод, т.к. одна и та же функция бизнес-логики может использоваться в различных контекстах и требовать различного вывода данных.

В классическом MVC проекте есть также такое понятие как аннотация (напр. [Required]) атрибутов - они в дальнейшем используются чтобы построить формы с помощью специальных методов. Мы стараемся не использовать эти возможности, т.к. работаем в основном на ajax формах, т.е. без использования некоторых встроенных возможностей фреймворка ASP.NET MVC.

Представление
На входе у представления - Модель и дополнительные данные по ViewBag. Представление выводит в браузер верстку и стили.

Старайтесь не использовать методы Модели прямо в Представлении. Представление - это всего лишь рендеринг данных, которые мы получили от Контроллера.

Представления бывают частичные (partial) и обычные. Частичные - это представление можно использовать как мини компонент (т.е. вставлять в другие представления).

Layout - это аналог мастер страницы в MVC - это общий шаблон страницы, который вы можете применить к Представлению.
Для этого нужно всего лишь установить в начале представления
Laout = “путь к layout”;

Обычно для вывода кода используется Razor синтаксис.

```
@{
это блок C# кода для инициализации каких-то переменных или вычислений (но без бизнес логики приложения - только логика относительно вывода данных, например разбиение на 4 колонки и т.д.)
}
```

Это цикл

```
@foreach(var item in Items){
        <li>@item.name</li>
}

@if(a ==0){
        @Html.Raw(a)     - используем функцию чтобы просто вывести текст без обработки.
}
```

Как работать с моделью. Сначала определяете ее тип (т.е. если есть модель, то представление становится строго типизированным).

```
@{
        model List<Products>
}
```

Далее вызываем ее как @Model

Задание 5. Изучите методы класса Html - он содержит много вспомогательных функций для работы с разметкой.

Примечание. Мы не рассматриваем здесь некоторые крупные возможности вроде Html.BeginForm или Html.CheckBoxFor, которые позволяют создать быструю разметку. Обычно мы создаем все свои сложные компоненты/формы на JS и jQuery. Вы сможете узнать о них подробнее в ресурсах, которые указаны ниже.

Задание 6. Разработайте 1 страницу для своего приложения - т.е. обращаемся по адресу, запрашиваем выполнение операции в Модели, и выводим представление на основании данных Модели.

Задание 7. Прочитайте про секции Web.config
https://ru.wikipedia.org/wiki/Web.config

Что почитать (либо поищите что-то более новое) -
http://www.williamspublishing.com/Books/978-5-8459-1609-9.html
http://www.williamspublishing.com/Books/978-5-8459-1867-3.html

Также почитайте эту краткую книгу на английском (там все понятно, даже если совсем немного знаете язык)
https://drive.google.com/a/rudensoft.ru/file/d/0B8N7eBTTS7MbazFVSWRBeU9hUXc/view

Пожалуй, это все, что я хотел рассказать про базис ASP.NET. Нам большая часть инфраструктуры ASP.NET не нужна. Наша задача на этом уровне - просто получить данные от DAL и сформировать базовую верстку для дальнейшей стилизации средствами CSS и ее обработку с помощью JS/jQuery.

В следующей главе мы будем рассматривать, как создавать HTML + CSS верстку.