AJAX

Итак, мы наконец добрались до аякса (AJAX - эйждекс:)).

Какие модели взаимодействия сервера и клиента существуют?

- Обычный запрос - мы запрашиваем новую страницу через GET запрос. Пример - запрашиваем http://yandex.ru
- Postback - мы наживаем кнопку submit на сайте и форма отправляется на сервер для обработки. Пример: Классическая форма Login.
- Ajax - в коде JS мы запускаем функцию обращения к серверу в специальном формате (например, JSON). Мы передаем в специальной форме данные серверу и он нам отвечает какой-то порцией данных. Пример: работа корзины в магазине. Добавили в корзину и прозрачно отправляется запрос для записи данных в базу данных.
- Comet - технология, основанная на ajax. Создается виртуальный сеанс между сервером и клиентом и создается ощущение, что события по взаимодействию инициируются самим сервером (хотя по факту это не совсем так). Пример применения - чат (вам приходит в браузер сообщение от другого человека мгновенно. Инициация этого события идет от сервера, а не от клиента).

Как мы используем эти запросы:

- Если страница (данные) предназначена для продвижения - то обязательно ее показывать как обычный GET запрос (иначе поисковики до нее не доберутся, они не умеют выполнять ajax запросы и заполнять формы).
- Для любой интерактивности мы по возможности используем ajax - это более удобно, чем перезагрузка всей страницы через postback.
- Если необходимо взаимодействие пользователей в реальном времени между собой и с системой - то это comet.

Так получилось, что мы практически не используем классические формы.

Задание 1. Найти плюсы и минусы подходов: submit форм через ajax vs submit форм через postback.

Еще раз суть Ajax запросов: в JS-функции Вы можете сделать запрос на сервер. Сервер Вам что-то ответит в формате JSON и Вы этот ответ обработаете: либо вывод сообщения, либо сформируете какую-то разметку и вставите в DOM. При этом у пользователя страница не перезагружается, он не знает, что идет запрос на сервер.

Пример:

```
var params = {id:1, name: "elena"};
 $.ajax({
        type: 'POST',      // тип запроса, может быть GET
        url: '/serv/savename',        // URL на сервере,который будет обрабатывать наш запрос
        dataType: 'json',  // тип данных, передаваемых на сервер
        cache: false,    // надо ли кешировать результат сервера
        contentType: 'application/json; charset=utf-8',  // тип передаваемых данных - JSON (или принимаемых)
        data: JSON.stringify(params),    // какие параметры мы передаем серверу
        success: function (data, status) {    // если запрос прошел нормально, то вызов этой функции
                        var obj = data;
                        if (data.d != undefined) obj = data.d;
                        var response = eval('(' + obj + ')');
                        if (response.result) {
                            alert('Сохранено')
                        }else{
                            alert('Извините, произошла ошибка.');
                        }
        },
        error: function(er, info){   // если были ошибки - то вызовется эта функция.
        },
        complete: function (xhr, status) {
                // в любом случае вызовется (были ошибки или нет).
                //используйте чтобы создать progress bar.
        },
             beforeSend: function (xhr) {
                        xhr.setRequestHeader('Content-type', 'application/json; charset=utf-8');
        }
});
```

Сначала передаем параметры (params). Это JSON, который сериализуется в строку (JSON.stringify). В некоторых случаях сериализация не нужна.
Затем мы вызываем функцию $.ajax, которая на входе принимает JSON с параметрами.
Далее запрос выполняет сервер и выдает результат, который мы обрабатываем в функции success (в нашей реализации мы работаем с объектом response).

Входной параметр может быть сколь угодно сложным. Это может быть JSON с вложенными объектами и массивами. В таких случаях Вам придется на стороне сервера правильно обрабатывать такие параметры через приведения к Dictionary<string, object> (если это был объект) и List<string> (если это массив).

```
var params = {
        ar: [1,2,3,4],   // (будет как List<int>)
        catID: 10,  //int
        product: { id:4, name: 'Prod 1'}  // принимаем как объект, а потом приводим к  Dictionary<string, object>
};
```

Давайте посмотрим, что мы должны сделать на стороне сервера.
Создаем в контроллере такой метод:

```
    public ActionResul savename(int id, string name)
    {
        var res = false;
        var mng = new DBprovider();
        res = mng.SaveName(id, name);
        var json = JsonConvert.SerializeObject(new
                    {
                        result=res
                    });
        return Content(json, "application/json");

    }
```

Как видите, в MVC все довольно просто. Правда Вам придется немного потрудиться, чтобы передавать сложные объекты (Вы увидите это решение в Учебном проекте). В этом случае потребуется более сложная обработка данные через Request.InputStream.

Заметьте, что имя метода, названия и тип параметров полностью должны совпадать.
В функции мы делаем некоторые действия и выдаем результат в виде JSON объекта (здесь используется анонимные классы для удобства и сериализация через JsonConvert.SerializeObject).

В целом ничего сложного - просто попробуйте сами.

Теперь давайте разберем пример, как работать с массивом и созданием верстки в AJAX-запросе. Допустим, нам надо вывести категорию товаров:

```
var params = {catID: 2};
 $.ajax({
                    type: 'POST',
                    url: '/serv/getproducts',
                    dataType: 'json',
                    cache: false,
                    contentType: 'application/json; charset=utf-8',
                    data: JSON.stringify(params),
                    success: function (data, status) {
                        var obj = data;
                        if (data.d != undefined) obj = data.d;
                        var response = eval('(' + obj + ')');
                        if (response.result) {
                      var s = [];
```

```
                $.each(response.items, function(i, item){
                        var t = "<div class='prItem' data-itemID='"+item.id+"'>"+
                                "<span class='prName'>"+item.name+"</span>"+
                                "<a class='rpAddToCart'>Купить</a>"
                        "</div>";
                        s.push(t);
                });
                $('.prContainer').html(s.join(""));

                  }else{
                      alert('Извините, произошла ошибка.');
                  }
              },
            error: function(er, info){
            },
                complete: function (xhr, status) {
                },
                beforeSend: function (xhr) {
                    xhr.setRequestHeader('Content-type', 'application/json; charset=utf-8');
                }
            });
```

Серверная часть тоже довольно простая:

```
    public ActionResult getproducts(int catID)
    {
        var res = false;
        var mng = new DBprovider();
        var items  = mng.GetProducts(catID);
        var json = JsonConvert.SerializeObject(new
             {
                result = res,
                        items = items.Select(new x=>{x.id, name = x.name ?? ""}) // берем проекцию объекта
             });
             return Content(json, "application/json");

    }
```

Еще один совет - лучше оберните свою функцию AJAX в другую функцию (т.е. вызывайте не напрямую, а через свою специальную функцию). У нас она выглядит следующим образом:

```
ajaxSend: function (url, data, callback, noProgressBar) {
        var params = JSON.stringify(data);
        $.ajax({
            type: 'POST',
            url: url,
            dataType: 'json',
```

```
            cache: false,
            contentType: 'application/json; charset=utf-8',
            data: params,
            success: function (data, status) {
                var response = data;
                if (data.d != undefined) response = data.d;
                if (callback) callback(response);
            },
            complete: function () {
                if (!noProgressBar) as.sys.closeProgressBar();
            },
            beforeSend: function (xhr) {
                xhr.setRequestHeader('Content-type', 'application/json; charset=utf-8');
                if (!noProgressBar) as.sys.openProgressBar();
            }
        });
    },
```

Ее вызов:

```
as.sys.ajaxSend('/serv/createcustomer', params, function (data) {
                    data = eval('(' + data + ')');
                    if (data.result) {
                    } else {
                    }
                });
```

Это позволяет внедрять общие моменты для всех ajax запросов (progress bar, логирование операций и т.д.) и уменьшает количество кода - его становится проще читать.

Важный момент - как искать ошибки в AJAX запросах (вы с этим встретитесь не раз, т.к. здесь обычно может быть много нюансов)

- ставим alert или console - до и после (перед вызовом и в success)
- смотрим F12 / console в Chrome - нет ли ошибок там?
- ставим breakpoint на сервере - срабатывает ли он - пошаговое прохождение
- если параметров много - то уменьшаем количество параметров передачи. Возможно где-то не совпадает тип или наименование переменной.

Задание 2. А теперь задачи:

1) сделать вывод каталога через ajax и кнопку покупки товара
2) сделать связь комбиков(выпадающих списков) страна, город. Город сначала не показывается и заполняется автоматически при выборе страны.

3) Сделать JS компонент корзины - он подгружает состояние корзины с сервера + есть внешний метод для обновления состояния корзины (его вы будете вызывать, когда будете добавлять что-то в корзину).

Примечание: при создании компонентов обязательно используйте описанный ранее шаблон JS-компонента и приведенную в этой главе обертку для вызова AJAX.

На самом деле Вы уже можете создавать довольно сложные приложения. Однако здесь нужна постоянная практика. Постарайтесь за следующий месяц написать как можно больше AJAX функциональности в купе с построением верстки через JS. Создавайте свои компоненты и делайте их универсальными.

Bootstrap

Переходим к Bootstrap. На самом деле здесь особо писать нечего - надо от корки до корки (вернее от тега до тега) изучать сайт http://getbootstrap.com/ или его русскоязычный аналог http://bootstrap-3.ru/. Обязательно постарайтесь хотя бы бегло его прочитать перед началом создания верстки при помощи Bootstrap. Здесь лишь вкратце опишем, что Вам может дать Bootstrap.

Есть 2 основные версии - 2.3.2 и 3. Они довольно сильно отличаются. Документация по 2.3.2 - http://getbootstrap.com/2.3.2/

Bootstrap значительно упрощает верстку, причем и адаптивную(в зависимости от разрешения экрана пользователя). Вам просто необходимо устанавливать правильные классы для элементов.

Если вы собираетесь использовать Bootstrap в своих проектах - очень рекомендую сразу придерживаться шаблона, указанного здесь - http://getbootstrap.com/getting-started/. Это избавит вас от возможных проблем с адаптивность и другими тонкими моментами.

Также в Bootstrap есть хороший набор иконок (вы просто указывате и он превращается в иконку).

Примечание. Для иконок очень рекомендую использовать иконки от http://fontawesome.io/icons/

Есть стили для большинства элементов (формы, кнопки, таблицы, панели, сообщения, badge и другие).

Вы можете создавать свои темы или использовать готовые (цвета, шрифты и другое).

Есть довольно большой набор компонентов JS - вкладки, аккордеон, диалогое окно, подсказки и др.).

Также Вы сможете найти довольно много компонентов, сделанных под Bootstrap (boostrap master, bootstrap datetimepicker и другие).

Основные классы, которые очень часто используются:
btn, table, alert, panel, nav, col-md, form-control

Задание 3. Изучите ресурс http://startbootstrap.com/bootstrap-resources/ и выпишите, что Вам показалось наиболее интересным в плане функционала. Приведите пример с использованием 2-3 вариантов на своих страницах.

В общем пробуйте, изучайте примеры с сайта Bootstrap. Попробуйте адаптировать в свои решения верстку Bootstrapи его компоненты.

Задание Изучите частичный аналог jQuery UI (http://jqueryui.com/). Сравните их. Приведите пример с использованием 2-3 вариантов на своих страницах.

Итак, мы подошли к концу курса. Однако для Вас все только начинается. Самое главное здесь не теория, а практика. Полученные знания помогут Вам обойти некоторые распространенные проблемы, однако Вы всегда должны быть готовы решать новые проблемы (разработка без решения новых и сложных проблем - утопия, именно поэтому так важно для начала научиться пользоваться поисковиком).

Следующий шаг - это самостоятельное создание своего проекта или дальнейшее участие в рамках нашего Учебного проекта.