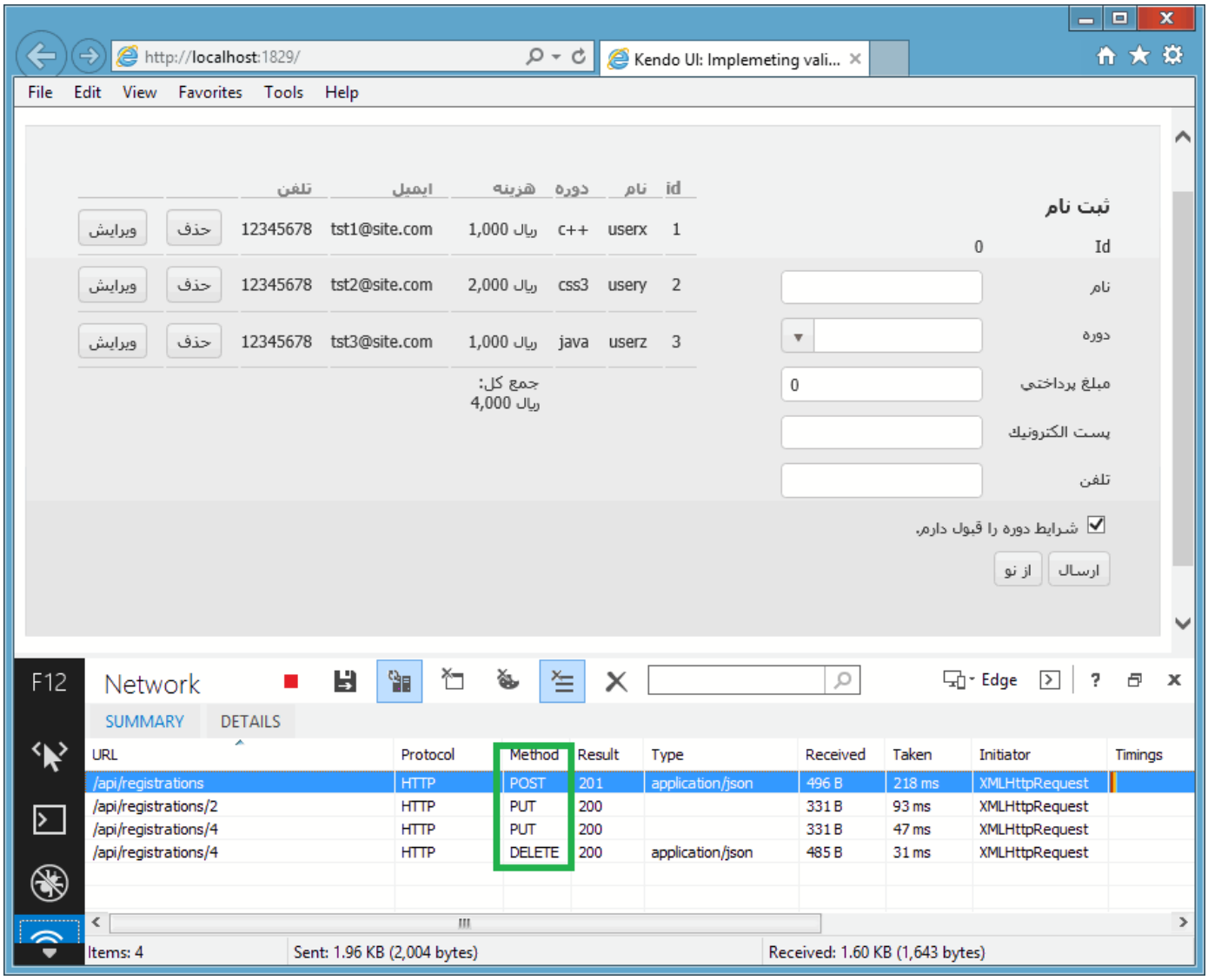
عنوان: Kendo UI MVVM
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۲۴ ۱۵:۵۵
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, MVVM, ASP.NET Web API, jQuery, KnockoutJS, Kendo UI

**پیشنیازها**

- « استفاده از Kendo UI templates »
- « اعتبار سنجی ورودی‌های کاربر در Kendo UI »
- « فعال سازی عملیات CRUD در Kendo UI Grid » جهت آشنایی با نحوه‌ی تعریف DataSource ایی که می‌تواند اطلاعات را ثبت، حذف و یا ویرایش کند.

در این مطلب قصد داریم به یک چنین صفحه‌ای برسیم که در آن در ابتدای نمایش، لیست ثبت نام‌های موجود، از سرور دریافت و توسط یک Kendo UI template نمایش داده می‌شود. سپس امکان ویرایش و حذف هر ردیف، وجود خواهد داشت، به همراه امکان افزودن ردیف‌های جدید. در این بین مدیریت نمایش لیست ثبت نام‌ها توسط امکانات binding توکار فریم ورک MVVM مخصوص Kendo UI صورت خواهد گرفت. همچنین کلیه اعمال مرتبط با هر ردیف نیز توسط data binding دو طرفه مدیریت خواهد شد.



**Kendo UI MVVM**

الگوی MVVM یا Model-View-ViewModel که برای اولین بار جهت کاربردهای WPF و Silverlight معرفی شد، برای ساده سازی اتصال تغییرات کنترل‌های برنامه به خواص ViewModel یک View کاربرد دارد. برای مثال با تغییر عنصر انتخابی یک DropDownList در یک View، بلافاصله خاصیت متصل به آن که در ViewModel برنامه تعریف شده‌است، مقدار دهی و به روز خواهد شد. هدف نهایی آن نیز جدا سازی منطق کدهای UI، از کدهای جاوا اسکریپتی سمت کاربر است. برای این منظور کتابخانه‌هایی مانند Knockout.js به صورت اختصاصی برای این کار تهیه شده‌اند؛ اما Kendo UI نیز جهت یکپارچگی هرچه تمامتر اجزای آن، دارای یک فریم ورک MVVM توکار نیز می‌باشد. طراحی آن نیز بسیار شبیه به Knockout.js است؛ اما با سازگاری 100 درصد با کل مجموعه.

پیاده سازی الگوی MVVM از 4 قسمت تشکیل می‌شود:

- Model که بیانگر خواص متناظر با اشیاء رابط کاربری است.
- View همان رابط کاربری است که به کاربر نمایش داده می‌شود.
- ViewModel واسطی است بین Model و View. کار آن انتقال داده‌ها و رویدادها از View به مدل است و در حالت binding دوطرفه، عکس آن نیز صحیح می‌باشد.
- Declarative data binding جهت رهایی برنامه نویس‌ها از نوشتن کدهای هماهنگ سازی اطلاعات المان‌های View و خواص ViewModel کاربرد دارد.

در ادامه این اجزا را با پیاده سازی مثالی که در ابتدای بحث مطرح شد، دنبال می‌کنیم.

**تعریف Model و ViewModel**

در سمت سرور، مدل ثبت نام برنامه چنین شکلی را دارد:

```
namespace KendoUI07.Models
{
    public class Registration
    {
        public int Id { set; get; }
        public string UserName { set; get; }
        public string CourseName { set; get; }
        public int Credit { set; get; }
        public string Email { set; get; }
        public string Tel { set; get; }
    }
}
```

در سمت کاربر، این مدل را به نحو ذیل می‌توان تعریف کرد:

```
<script type="text/javascript">
        $(function () {
            var model = kendo.data.Model.define({
                id: "Id",
                fields: {
                    Id: { type: 'number' }, // leave this set to 0 or undefined, so Kendo knows it is
new.
                    UserName: { type: 'string' },
                    CourseName: { type: 'string' },
                    Credit: { type: 'number' },
                    Email: { type: 'string' },
                    Tel: { type: 'string' }
                }
            });
        });
    </script>
```

و ViewModel برنامه در ساده‌ترین شکل آن اکنون چنین تعریفی را خواهد یافت:

```
<script type="text/javascript">
        $(function () {
            var viewModel = kendo.observable({
```

```
                accepted: false,
                course: new model()
            });
        });
    </script>
```

یک viewModel در Kendo UI به صورت یک observable object تعریف می‌شود که می‌تواند دارای تعدادی خاصیت و متد دلخواه باشد. هر خاصیت آن به یک عنصر HTML متصل خواهد شد. در اینجا این اتصال دو طرفه است؛ به این معنا که تغییرات UI به خواص viewModel و برعکس منتقل و منعکس می‌شوند.

### اتصال ViewModel به View برنامه

تعریف فرم ثبت نام را در اینجا ملاحظه می‌کنید. فیلدهای مختلف آن بر اساس نکات اعتبارسنجی HTML 5 با ویژگی‌های خاص آن، مزین شده‌اند. جزئیات آن‌را در مطلب « اعتبار سنجی ورودی‌های کاربر در Kendo UI » پیشتر بررسی کرده‌ایم.
اگر به تعریف هر فیلد دقت کنید، ویژگی data-bind جدیدی را هم ملاحظه خواهید کرد:

```
<div id="coursesSection" class="k-rtl k-header">
        <div class="box-col">
            <form id="myForm" data-role="validator" novalidate="novalidate">
                <h3>ثبت نام</h3>
                <ul>
                    <li>
                        <label for="Id">Id</label>
                        <span id="Id" data-bind="text:course.Id"></span>
                    </li>
                    <li>
                        <label for="UserName">نام</label>
                        <input type="text" id="UserName" name="UserName" class="k-textbox"
                               data-bind="value:course.UserName"
                               required />
                    </li>
                    <li>
                        <label for="CourseName">دوره</label>
                        <input type="text" dir="ltr" id="CourseName" name="CourseName" required
                               data-bind="value:course.CourseName" />
                        <span class="k-invalid-msg" data-for="CourseName"></span>
                    </li>
                    <li>
                        <label for="Credit">مبلغ پرداختی</label>
                        <input id="Credit" name="Credit" type="number" min="1000" max="6000"
                               required data-max-msg="6000 و 1000 عددی بین" dir="ltr"
                               data-bind="value:course.Credit"
                               class="k-textbox k-input" />
                        <span class="k-invalid-msg" data-for="Credit"></span>
                    </li>
                    <li>
                        <label for="Email">پست الکترونیک</label>
                        <input type="email" id="Email" dir="ltr" name="Email"
                               data-bind="value:course.Email"
                               required class="k-textbox" />
                    </li>
                    <li>
                        <label for="Tel">تلفن</label>
                        <input type="tel" id="Tel" name="Tel" dir="ltr" pattern="\d{8}"
                               required class="k-textbox"
                               data-bind="value:course.Tel"
                               data-pattern-msg="8 رقم" />
                    </li>
                    <li>
                        <input type="checkbox" name="Accept"
                               data-bind="checked:accepted"
                               required />
                        شرایط دوره را قبول دارم.
                        <span class="k-invalid-msg" data-for="Accept"></span>
                    </li>
                    <li>
                        <button class="k-button"
                                data-bind="enabled: accepted, click: doSave"
                                type="submit">
                            ارسال
                        </button>
                        <button class="k-button" data-bind="click: resetModel">از نو</button>
```

```
                    </li>
                </ul>
                <span id="doneMsg"></span>
            </form>
        </div>
```

برای اتصال ViewModel تعریف شده به ناحیه‌ی مشخص شده با DIV ایی با Id مساوی coursesSection، می‌توان از متد kendo.bind استفاده کرد.

```
<script type="text/javascript">
        $(function () {
            var model = kendo.data.Model.define({
            // ...
            });

            var viewModel = kendo.observable({
            // ...
            });

            kendo.bind($("#coursesSection"), viewModel);
        });
    </script>
```

به این ترتیب Kendo UI به بر اساس تعریف data-bind یک فیلد، برای مثال تغییرات خواص course.UserName را به text box نام کاربر منتقل می‌کند و همچنین اگر کاربر اطلاعاتی را در این text box وارد کند، بلافاصله این تغییرات در خاصیت course.UserName منعکس خواهند شد.

```
<input type="text" id="UserName" name="UserName" class="k-textbox"
       data-bind="value:course.UserName"
       required />
```

بنابراین تا اینجا به صورت خلاصه، مدلی را توسط متد kendo.data.Model.define، معادل مدل سمت سرور خود ایجاد کردیم. سپس وهله‌ای از این مدل را به صورت یک خاصیت جدید دلخواهی در ViewModel تعریف شده توسط متد kendo.observable در معرض دید View برنامه قرار دادیم. در ادامه اتصال ViewModel و View، با فراخوانی متد kendo.bind انجام شد. اکنون برای دریافت تغییرات کنترل‌های برنامه، تنها کافی است ویژگی‌های data-bind ایی را به آن‌ها اضافه کنیم.
در ناحیه‌ی تعریف شده توسط متد kendo.bind، کلیه خواص ViewModel در دسترس هستند. برای مثال اگر به تعریف ViewModel دقت کنید، یک خاصیت دیگر به نام accepted با مقدار false نیز در آن تعریف شده‌است (این خاصیت چون صرفا کاربرد UI داشت، در model برنامه قرار نگرفت). از آن برای اتصال checkbox تعریف شده، به button ارسال اطلاعات، استفاده کرده‌ایم:

```
<input type="checkbox" name="Accept"
       data-bind="checked:accepted"
       required />

<button class="k-button"
        data-bind="enabled: accepted, click: doSave"
        type="submit">
       ارسال
</button>
```

برای مثال اگر کاربر این checkbox را انتخاب کند، مقدار خاصیت accepted، مساوی true خواهد شد. تغییر مقدار این خاصیت، توسط ViewModel بلافاصله در کل ناحیه coursesSection منتشر می‌شود. به همین جهت ویژگی enabled: accepted که به معنای مقید بودن فعال یا غیرفعال بودن دکمه بر اساس مقدار خاصیت accepted است، دکمه را فعال می‌کند، یا برعکس و برای انجام این عملیات نیازی نیست کدنویسی خاصی را انجام داد. در اینجا بین checkbox و button یک سیم کشی برقرار است.

### ارسال داده‌های تغییر کرده‌ی ViewModel به سرور

تا اینجا 4 جزء اصلی الگوی MVVM که در ابتدای بحث عنوان شد، تکمیل شده‌اند. مدل اطلاعات فرم تعریف گردید. ViewModel ایی

که این خواص را به المان‌های فرم متصل می‌کند نیز در ادامه اضافه شده‌است. توسط ویژگی‌های data-bind کار Declarative data binding انجام می‌شود.

در ادامه نیاز است تغییرات ViewModel را به سرور، جهت ثبت، به روز رسانی و حذف نهایی منتقل کرد.

```
<script type="text/javascript">
        $(function () {
            var model = kendo.data.Model.define({
                //...
            });

            var dataSource = new kendo.data.DataSource({
                type: 'json',
                transport: {
                    read: {
                        url: "api/registrations",
                        dataType: "json",
                        contentType: 'application/json; charset=utf-8',
                        type: 'GET'
                    },
                    create: {
                        url: "api/registrations",
                        contentType: 'application/json; charset=utf-8',
                        type: "POST"
                    },
                    update: {
                        url: function (course) {
                            return "api/registrations/" + course.Id;
                        },
                        contentType: 'application/json; charset=utf-8',
                        type: "PUT"
                    },
                    destroy: {
                        url: function (course) {
                            return "api/registrations/" + course.Id;
                        },
                        contentType: 'application/json; charset=utf-8',
                        type: "DELETE"
                    },
                    parameterMap: function (data, type) {
                        // Convert to a JSON string.  Without this step your content will be form
encoded.
                        return JSON.stringify(data);
                    }
                },
                schema: {
                    model: model
                },
                error: function (e) {
                    alert(e.errorThrown);
                },
                change: function (e) {
                    // فراخوانی در زمان دریافت اطلاعات از سرور و یا تغییرات محلی
                    viewModel.set("coursesDataSourceRows", new
kendo.data.ObservableArray(this.view()));
                }
            });

            var viewModel = kendo.observable({
                //...
            });

            kendo.bind($("#coursesSection"), viewModel);
            dataSource.read(); // دریافت لیست موجود از سرور در آغاز کار
        });
    </script>
```

در اینجا تعریف DataSource کار با منبع داده راه دور ASP.NET Web API را مشاهده می‌کنید. تعاریف اصلی آن با تعاریف مطرح شده در مطلب « فعال سازی عملیات CRUD در Kendo UI Grid » یکی هستند. هر قسمت آن مانند read، create، update و destory به یکی از متدهای کنترلر ASP.NET Web API اشاره می‌کنند. حالت‌های update و destroy بر اساس Id ردیف انتخابی کار می‌کنند. این Id را باید در قسمت model مربوط به اسکیمای تعریف شده، دقیقا مشخص کرد. عدم تعریف فیلد id، سبب خواهد شد تا عملیات update نیز در حالت create تفسیر شود.

### متصل کردن DataSource به ViewModel

تا اینجا DataSource ایی جهت کار با سرور تعریف شده‌است؛ اما مشخص نیست که اگر رکوردی اضافه شد، چگونه باید اطلاعات خودش را به روز کند. برای این منظور خواهیم داشت:

```
<script type="text/javascript">
        $(function () {
            $("#coursesSection").kendoValidator({
                // ...
            });

            var model = kendo.data.Model.define({
                // ...
            });

            var dataSource = new kendo.data.DataSource({
                // ...
            });

            var viewModel = kendo.observable({
                accepted: false,
                course: new model(),
                doSave: function (e) {
                    e.preventDefault();
                    console.log("this", this.course);
                    var validator = $("#coursesSection").data("kendoValidator");
                    if (validator.validate()) {
                        if (this.course.Id == 0) {
                            dataSource.add(this.course);
                        }
                        dataSource.sync(); // push to the server
                        this.set("course", new model()); // reset controls
                    }
                },
                resetModel: function (e) {
                    e.preventDefault();
                    this.set("course", new model());
                }
             });

            kendo.bind($("#coursesSection"), viewModel);
            dataSource.read(); // دریافت لیست موجود از سرور در آغاز کار
        });
    </script>
```

همانطور که در تعاریف تکمیلی viewModel مشاهده می‌کنید، اینبار دو متد جدید دلخواه doSave و resetModel را اضافه کرده‌ایم. در متد doSave، ابتدا بررسی می‌کنیم آیا اعتبارسنجی فرم با موفقیت انجام شده‌است یا خیر. اگر بله، توسط متد add منبع داده، اطلاعات فرم جاری را توسط شیء course که هم اکنون به تمامی فیلدهای آن متصل است، اضافه می‌کنیم. در اینجا بررسی شده‌است که آیا Id این اطلاعات صفر است یا خیر. از آنجائیکه از همین متد برای به روز رسانی نیز در ادامه استفاده خواهد شد، در حالت به روز رسانی، Id شیء ثبت شده، از طرف سرور دریافت می‌گردد. بنابراین غیر صفر بودن این Id به معنای عملیات به روز رسانی است و در این حالت نیازی نیست کار بیشتری را انجام داد؛ زیرا شیء متناظر با آن پیشتر به منبع داده اضافه شده‌است.

استفاده از متد add صرفا به معنای مطلع کردن منبع داده محلی از وجود رکوردی جدید است. برای ارسال این تغییرات به سرور، از متد sync آن می‌توان استفاده کرد. متد sync بر اساس متد add یک درخواست POST، بر اساس شیءایی که Id غیر صفر دارد، یک درخواست PUT و با فراخوانی متد remove بر روی منبع داده، یک درخواست DELETE را به سمت سرور ارسال می‌کند.

متد دلخواه resetModel سبب مقدار دهی مجدد شیء course با یک وهله‌ی جدید از شیء model می‌شود. همینقدر برای پاک کردن تمامی کنترل‌های صفحه کافی است.

تا اینجا دو متد جدید را در ViewModel برنامه تعریف کرده‌ایم. در مورد نحوه‌ی اتصال آن‌ها به View، به کدهای دو دکمه‌ی موجود در فرم دقت کنید:

```
<button class="k-button"
        data-bind="enabled: accepted, click: doSave"
        type="submit">
```

```
        ارسال
</button>
<button class="k-button" data-bind="click: resetModel">از نو</button>
```

این متدها نیز توسط ویژگی‌های data-bind به هر دکمه نسبت داده شده‌اند. به این ترتیب برای مثال با کلیک کاربر بر روی دکمه‌ی submit، متد doSave موجود در ViewModel فراخوانی می‌شود.

### مدیریت سمت سرور ثبت، ویرایش و حذف اطلاعات

در حالت ثبت، متد Post توسط آدرس مشخص شده در قسمت create منبع داده، فراخوانی می‌گردد. نکته‌ی مهمی که در اینجا باید به آن دقت داشت، نحوه‌ی بازگشت Id رکورد جدید ثبت شده‌است. اگر این تنظیم صورت نگیرد، Id رکورد جدید را در لیست، مساوی صفر مشاهده خواهید کرد و منبع داده این رکورد را همواره به عنوان یک رکورد جدید، مجددا به سرور ارسال می‌کند.

```
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using KendoUI07.Models;

namespace KendoUI07.Controllers
{
    public class RegistrationsController : ApiController
    {
        public HttpResponseMessage Delete(int id)
        {
            var item = RegistrationsDataSource.LatestRegistrations.FirstOrDefault(x => x.Id == id);
            if (item == null)
                return Request.CreateResponse(HttpStatusCode.NotFound);

            RegistrationsDataSource.LatestRegistrations.Remove(item);
            return Request.CreateResponse(HttpStatusCode.OK, item);
        }

        public IEnumerable<Registration> Get()
        {
            return RegistrationsDataSource.LatestRegistrations;
        }

        public HttpResponseMessage Post(Registration registration)
        {
            if (!ModelState.IsValid)
                return Request.CreateResponse(HttpStatusCode.BadRequest);

            var id = 1;
            var lastItem = RegistrationsDataSource.LatestRegistrations.LastOrDefault();
            if (lastItem != null)
            {
                id = lastItem.Id + 1;
            }
            registration.Id = id;
            RegistrationsDataSource.LatestRegistrations.Add(registration);

            // ارسال آی دی مهم است تا از ارسال رکوردهای تکراری جلوگیری شود
            return Request.CreateResponse(HttpStatusCode.Created, registration);
        }

        [HttpPut] // Add it to fix this error: The requested resource does not support http method
'PUT'
        public HttpResponseMessage Update(int id, Registration registration)
        {
            var item = RegistrationsDataSource.LatestRegistrations
                                        .Select(
                                            (prod, index) =>
                                                new
                                                {
                                                    Item = prod,
                                                    Index = index
                                                })
                                        .FirstOrDefault(x => x.Item.Id == id);
            if (item == null)
```

```
                return Request.CreateResponse(HttpStatusCode.NotFound);


            if (!ModelState.IsValid || id != registration.Id)
                return Request.CreateResponse(HttpStatusCode.BadRequest);

            RegistrationsDataSource.LatestRegistrations[item.Index] = registration;
            return Request.CreateResponse(HttpStatusCode.OK);
        }
    }
}
```

در اینجا بیشتر امضای این متدها مهم هستند، تا منطق پیاده سازی شده در آن‌ها. همچنین بازگشت Id رکورد جدید، توسط متد Post نیز بسیار مهم است و سبب می‌شود تا DataSource بداند با فراخوانی متد sync آن، باید عملیات Post یا create انجام شود یا Put و update.

### نمایش آنی اطلاعات ثبت شده در یک لیست

ردیف‌های اضافه شده به منبع داده را می‌توان بلافاصله در همان سمت کلاینت توسط Kendo UI Template که قابلیت کار با ViewModelها را دارد، نمایش داد:

```
<div id="coursesSection" class="k-rtl k-header">
        <div class="box-col">
            <form id="myForm" data-role="validator" novalidate="novalidate">
                        <!--فرم بحث شده در ابتدای مطلب-->
            </form>
        </div>
        <div id="results">
            <table class="metrotable">
                <thead>
                    <tr>
                        <th>Id</th>
                        <th>نام</th>
                        <th>دوره</th>
                        <th>هزینه</th>
                        <th>ایمیل</th>
                        <th>تلفن</th>
                        <th></th>
                        <th></th>
                    </tr>
                </thead>
                <tbody data-template="row-template" data-bind="source: coursesDataSourceRows"></tbody>
                <tfoot data-template="footer-template" data-bind="source: this"></tfoot>
            </table>
            <script id="row-template" type="text/x-kendo-template">
                <tr>
                    <td data-bind="text: Id"></td>
                    <td data-bind="text: UserName"></td>
                    <td dir="ltr" data-bind="text: CourseName"></td>
                    <td>
                        #: kendo.toString(get("Credit"), "c0") #
                    </td>
                    <td data-bind="text: Email"></td>
                    <td data-bind="text: Tel"></td>
                    <td><button class="k-button" data-bind="click: deleteCourse">حذف</button></td>
                    <td><button class="k-button" data-bind="click: editCourse">ویرایش</button></td>
                </tr>
            </script>
            <script id="footer-template" type="text/x-kendo-template">
                <tr>
                    <td colspan="3"></td>
                    <td>
                        جمع کل: #: kendo.toString(totalPrice(), "c0") #
                    </td>
                    <td colspan="2"></td>
                    <td></td>
                    <td></td>
                </tr>
            </script>
        </div>
    </div>
```

در ناحیه‌ی coursesSection که توسط متد kendo.bind به viewModel برنامه متصل شده‌است، یک جدول را برای نمایش ردیف‌های ثبت شده توسط کاربر اضافه کرده‌ایم. thead آن بیانگر سر ستون جدول است. قسمت tbody و tfoot این جدول توسط دو Kendo UI Template مقدار دهی شده‌اند. هر کدام نیز منبع داده‌اشان را از view model دریافت می‌کنند. در row-template معادل خواص شیء course را مشاهده می‌کنید. در footer-template متد totalPrice برای نمایش جمع ستون هزینه اضافه شده‌است. بنابراین مطابق این قسمت از View، به یک خاصیت جدید coursesDataSourceRows و سه متد deleteCourse، editCourse و totalPrice نیاز است:

```
<script type="text/javascript">
        $(function () {
            // ...
            var viewModel = kendo.observable({
                accepted: false,
                course: new model(),
                coursesDataSourceRows: new kendo.data.ObservableArray([]),
                doSave: function (e) {
                       // ...
                },
                resetModel: function (e) {
                      // ...
                },
                totalPrice: function () {
                    var sum = 0;
                    $.each(this.get("coursesDataSourceRows"), function (index, item) {
                        sum += item.Credit;
                    });
                    return sum;
                },
                deleteCourse: function (e) {
                    // the current data item is passed as the "data" field of the event argument
                    var course = e.data;
                    dataSource.remove(course);
                    dataSource.sync(); // push to the server
                },
                editCourse: function(e) {
                    // the current data item is passed as the "data" field of the event argument
                    var course = e.data;
                    this.set("course", course);
                }
            });

            kendo.bind($("#coursesSection"), viewModel);
            dataSource.read(); // دریافت لیست موجود از سرور در آغاز کار
        });
    </script>
```

نحوه‌ی اتصال خاصیت جدید coursesDataSourceRows که به عنوان منبع داده ردیف‌های row-template عمل می‌کند، به این صورت است:

- ابتدا خاصیت دلخواه coursesDataSourceRows به viewModel اضافه می‌شود تا در ناحیه‌ی coursesSection در دسترس قرار گیرد.

- سپس اگر به انتهای تعریف DataSource دقت کنید، داریم:

```
<script type="text/javascript">
        $(function () {
            var dataSource = new kendo.data.DataSource({
                //...
                change: function (e) {
                    // فراخوانی در زمان دریافت اطلاعات از سرور و یا تغییرات محلی
                    viewModel.set("coursesDataSourceRows", new
kendo.data.ObservableArray(this.view()));
                }
            });
        });
    </script>
```

متد change آن، هر زمانیکه اطلاعاتی در منبع داده تغییر کنند یا اطلاعاتی به سمت سرور ارسال یا دریافت گردد، فراخوانی می‌شود. در همینجا فرصت خواهیم داشت تا خاصیت coursesDataSourceRows را جهت نمایش اطلاعات موجود در منبع داده،

مقدار دهی کنیم. همین مقدار دهی ساده سبب اجرای row-template برای تولید ردیف‌های جدول می‌شود. استفاده از new kendo.data.ObservableArray سبب خواهد شد تا اگر اطلاعاتی در فرم برنامه تغییر کند، این اطلاعات بلافاصله در لیست گزارش برنامه نیز منعکس گردد.

**کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید:**

KendoUI07.zip