ASP.NET MVC Fundamentals

By: Mosh Hamedani

Action Results

Type	Helper Method
ViewResult	**View()**
PartialViewResult	**PartialView()**
ContentResult	**Content()**
RedirectResult	**Redirect()**
RedirectToRouteResult	**RedirectToAction()**
JsonResult	**Json()**
FileResult	**File()**
HttpNotFoundResult	**HttpNotFound()**
EmptyResult	

Action Parameters

Sources

- Embedded in the URL: /movies/edit/1
- In the query string: /movies/edit?id=1
- In the form data

Convention-based Routes

```
routes.MapRoute(
    “MoviesByReleaseDate”,
    “movies/released/{year}/{month}”,
    new {
         controller = “Movies”,
         action = “MoviesReleaseByDate”
    },
    new {
         year = @“\d{4}, month = @“\d{2}”
    }    isFavorite = false;
}
```

Attribute Routes

```
[Route(“movies/released/{year}/{month}”)
public ActionResult MoviesByReleaseDate(int year, int month)
{
}
```

To apply a constraint use a colon:

```
month:regex(\\d{2}):range(1, 12)
```

Passing Data to Views

Avoid using ViewData and ViewBag because they are fragile. Plus, you have to do extra casting, which makes your code ugly. Pass a model (or a view model) directly to a view:

```
return View(movie);
```

Razor Views

```
@if (…)
{
    // C# code or HTML
}
```

```
@foreach (…)
{
}
```

Render a class (or any attributes) conditionally:

```
@{
   var className = Model.Customers.Count > 5 ? “popular” : null;
}
<h2 class=“@className”>…</h2>
```

Partial Views

To render:

```
@Html.Partial(“_NavBar”)
```

Exercise Hints

Creating Links

There are a few different ways to create links in ASP.NET MVC apps. The simplest way is to use raw HTML:

```
<a href="/Movies/Index">View Movies</a>
```

Alternatively, you can use the **ActionLink** method of **HtmlHelper** class:

```
@Html.ActionLink("View Movies", "Index", "Movies")
```

If the target action requires a parameter, you can use an anonymous object to pass parameter values:

```
@Html.ActionLink("View Movies", "Index", "Movies", new { id = 1 })
```

This should generate a link like the following:

/movies/index/1

But for a reason only known to programmers at Microsoft, this method doesn't generate this link, unless you pass another argument to the ActionLink. This argument can be null or an anonymous object to render any additional HTML attributes:

```
@Html.ActionLink("View Movies", "Index", "Movies", new { id = 1 }, null)
```

So, ActionLink or raw HTML, which approach is better? ActionLink queries the routing engine for the URL associated with the given action. If you have a custom URL associated with an action, and change that URL in the future, ActionLink will pick the latest URL, so you don't need to make any changes. But with raw HTML, you need to update your links when URLs are changed.

Having said, changing URLs is something you should avoid because these URLs are the public contract of your app and can be referenced by other apps or bookmarked by users. If you change them, all these bookmarks and external references will be broken.

So, at the end, whether you prefer to use raw HTML or @Html.ActionLink() is your personal choice.

HTML Tables

In the demo I showed you, I used a few CSS classes on my HTML table to give it the look and feel you saw in the video. These classes are part of Bootstrap, a front-end framework for building modern, responsive applications.

```
<table class="table table-bordered table-hover">
```

Type	Helper Method
ViewResult	**View()**

Type	Helper Method
PartialViewResult	**PartialView()**
ContentResult	**Content()**
RedirectResult	**Redirect()**
RedirectToRouteResult	**RedirectToAction()**
JsonResult	**Json()**
FileResult	**File()**
HttpNotFoundResult	**HttpNotFound()**
EmptyResult	

Action Parameters

Sources

- Embedded in the URL: /movies/edit/1
- In the query string: /movies/edit?id=1
- In the form data

Convention-based Routes

```
routes.MapRoute(
    “MoviesByReleaseDate”,
    “movies/released/{year}/{month}”,
    new {
         controller = “Movies”,
         action = “MoviesReleaseByDate”
    },
    new {
         year = @“\d{4}, month = @“\d{2}”
    }    isFavorite = false;
}
```

Attribute Routes

```
[Route(“movies/released/{year}/{month}”)
public ActionResult MoviesByReleaseDate(int year, int month)
{
}
```

To apply a constraint use a colon:

```
month:regex(\\d{2}):range(1, 12)
```

Passing Data to Views

Avoid using ViewData and ViewBag because they are fragile. Plus, you have to do extra casting, which makes your code ugly. Pass a model (or a view model) directly to a view:

```
return View(movie);
```

Razor Views

```
@if (…)
{
    // C# code or HTML
}
```

```
@foreach (…)
{
}
```

Render a class (or any attributes) conditionally:

```
@{
   var className = Model.Customers.Count > 5 ? “popular” : null;
}
<h2 class=“@className”>…</h2>
```

Partial Views

To render:

```
@Html.Partial(“_NavBar”)
```

Entity Framework

By: Mosh Hamedani

Code-first Migrations

```
add-migration <name>
add-migration <name> -force (to overwrite the last migration)
update-database
```

Seeding the Database

Create a new empty migration and use the Sql method:

```
Sql(“INSERT INTO …”)
```

Overriding Conventions

```
[Required]
[StringLength(255)]
public string Name { get; set; }
```

Entity Framework

By: Mosh Hamedani

Querying Objects

```
public class MoviesController
{
    private ApplicationDbContext _context;

    public MoviesController()
    {
        _context = new ApplicationDbContext();
    }

    protected override Dispose()
    {
        _context.Dispose();
    }

    public ActionResult Index()
    {
         var movies = _context.Movies.ToList();

         …
    }
}
```

LINQ Extension Methods

```
_context.Movies.Where(m => m.GenreId == 1)
_context.Movies.Single(m => m.Id == 1);
_context.Movies.SingleOrDefault(m => m.Id == 1);
_context.Movies.ToList();
```

Eager Loading

```
_context.Movies.Include(m => m.Genre);
```

Building Forms

By: Mosh Hamedani

View

```
@using (Html.BeginForm("action", "controller"))
{
    <div class="form-group">
        @Html.LabelFor(m => m.Name)
        @Html.TextBoxFor(m => m.Name, new { @class = "form-
          control")
    </div>
    <button type="submit" class="btn btn-primary">Save</button>
}
```

Markup for Checkbox Fields

```
<div class="checkbox">
    @Html.CheckBoxFor(m => m.IsSubscribed) Subscribed?
</div>
```

Drop-down Lists

```
@Html.DropDownListFor(m => m.TypeId, new SelectList(Model.Types,
"Id", "Name"), "", new { @class = "form-control" }
```

Building Forms

By: Mosh Hamedani

Overriding Labels

```
Display(Name = “Date of Birth”)
public DateTime? Birthdate { get; set; }
```

Saving Data

```
[HttpPost]
public ActionResult Save(Customer customer)
{
     if (customer.Id == 0)
         _context.Customers.Add(customer);
     else
     {
         var customerInDb = _context.Customers.Single(c.Id ==
customer.Id);

         //… update properties
     }

     _context.SaveChanges();

     return RedirectToAction(“Index”, “Customers”)
}
```

Hidden Fields

Building Forms

By: Mosh Hamedani

Required when updating data.

```
@Html.HiddenFor(m => m.Customer.Id)
```

Implementing Validation

By: Mosh Hamedani

Adding Validation

Decorate properties of your model with data annotations. Then, in the controller:

```
if (!ModelState.IsValid)
    return View(…);
```

And in the view:

```
@Html.ValidationMessageFor(m => m.Name)
```

Styling Validation Errors

In site.css:

```
.input-validation-error {
    color: red;
}

.field-validation-error {
    border: 2px solid red;
}
```

Implementing Validation

By: Mosh Hamedani

Data Annotations

- [Required]
- [StringLength(255)]
- [Range(1, 10)]
- [Compare("OtherProperty")]
- [Phone]
- [EmailAddress]
- [Url]
- [RegularExpression("…")]

Custom Validation

```
public class Min18IfAMember : ValidationAttribute
{
     protected override ValidationResult IsValid(object value,
ValidationContext validationContext)
     {
          …
          if (valid) return ValidationResult.Success;
          else return new ValidationResult("error message");
     }
}
```

Implementing Validation

By: Mosh Hamedani

Validation Summary

```
@Html.ValidationSummary(true, “Please fix the following errors”);
```

Client-side Validation

```
@section scripts {
     @Scripts.Render(“~/bundles/jqueryval”)
}
```

Anti-forgery Tokens

In the view:

```
@Html.AntiForgeryToken()
```

In the controller:

```
[ValidateAntiForgeryToken]
public ActionResult Save() { }
```

Building Web APIs

By: Mosh Hamedani

RESTful Convention

Request	Description
GET /api/customers	Get all customers
GET /api/customers/1	Get customer with ID 1
POST /api/customers	Add a new customer (customer data in the request body)
PUT /api/customers/1	Update customer with ID 1 (customer data in the request body)
DELETE /api/customers/1	Delete customer with ID 1

Building an API

```
public IHttpActionResult GetCustomers() {}


[HttpPost]
public IHttpActionResult CreateCustomer(CustomerDto customer) {}


[HttpPut]
public IHttpActionResult UpdateCustomer(int id, CustomerDto
customer) {}


[HttpDelete]
public IHttpActionResult DeleteCustomer(int id) {}
```

Helper methods

- NotFound()
- Ok()
- Created()
- Unauthorized()

AutoMapper

Create a mapping profile first:

```
public class MappingProfile : Profile
{
     public MappingProfile()
     {
          Mapper.CreateMap<Customer, CustomerDto>();
     }
}
```

Load the mapping profile during application startup (in global.asax.cs):

```
protected void Application_Start()
{
    Mapper.Initialize(c => c.AddProfile<MappingProfile>());
}
```

To map objects:

```
var customerDto = Mapper.Map<Customer, CustomerDto>(customer);
```

Or to map to an existing object:

```
Mapper.Map(customer, customerDto);
```

Enabling camel casing

In WebApiConfig:

```
public static void Register(HttpConfiguration config)
{
    var settings =
config.Formatters.JsonFormatter.SerializerSettings;

    settings.ContractResolver = new
CamelCasePropertyNamesContractResolver();

    settings.Formatting = Formatting.Indented;
}
```

Exercise Hint

By: Mosh Hamedani

As part of this exercise, you'll get this exception when updating a movie:

Property 'Id' is part of object's key information and cannot be modified.

This exception happens at the following line:

```
Mapper.Map(movieDto, movie);
```

The exception is thrown when AutoMapper attempts to set the Id of movie:

```
customer.Id = customerDto.Id;
```

Id is the key property for the Movie class, and a key property should not be changed. That's why we get this exception. To resolve this, you need to tell AutoMapper to ignore Id during mapping of a MovieDto to Movie.

In MappingProfile:

```
CreateMap<Movie, MovieDto>()
    .ForMember(m => m.Id, opt => opt.Ignore());
```

The same configuration should be applied to mapping of customers:

```
CreateMap<Customer, CustomerDto>()
    .ForMember(c => c.Id, opt => opt.Ignore());
```

jQuery Tutorials

By: Mosh Hamedani

Why you need to learn jQuery

While ASP.NET MVC is a server-side (or back-end) web framework, as an ASP.NET MVC developer, there quite a few occasions where you need to work with various client-side (or front-end) technologies. If you only know the server-side, you're considered a *back-end developer*, but if you know both the front-end and the back-end, you're referred to as a *full-stack developer.*

Full-stack development, obviously, requires learning more stuff but that will also boost your salary, compared to a back-end or front-end only developer.

What is jQuery anyway?

jQuery is a Javascript library that allows us to work with objects in an HTML document. The HTML markup we use to build a web page, is eventually loaded into a tree of objects called **Document Object Model (or DOM)**. In front-end development, we need to look for certain elements in the DOM and show/hide them based on some condition. This is just one simple example, but there are many more complex tasks that requires traversing and modifying DOM elements.

We can use pure Javascript to work with these DOM, but this requires extra effort to deal with inconsistencies of various browsers. Each browser, exposes a client-side API to work with these DOM elements and sometimes these APIs are not consistent in their contract and implementation. This applies specifically to Internet Explorer, which has always been like a browser made in another planet!

So, jQuery was released to address this issue. jQuery is a Javascript library that provides a simple and unified API to work with with the DOM, no matter what browser the client-side app is running in.

So, jQuery is just a Javascript library and is not a programming language.

jQuery Tutorials

By: Mosh Hamedani

How to learn jQuery?

Ideally, you may want to take a course or read a book. But you can also get the basics from this tutorial:

http://www.impressivewebs.com/jquery-tutorial-for-beginners/

Now, before you get started, I just let you know that in this course, we'll have only a few lectures using jQuery. If you get stuck following the next few videos, don't get disappointed. I can guarantee that once you go through them one more time and think of the materials in these tutorials, you'll have a reasonable understanding of how everything works. So, don't get stuck, just code along with me and complete these features.

Ok, what after jQuery?

While jQuery has had a great impact on the development of client-side apps, it has been criticised for resulting in spaghetti and untestable Javascript code. So, different libraries have been developed that allows us to work with DOM while giving us a framework to write modular and testable Javascript code. Angular and React are two popular examples of such frameworks.

But before studying any of such frameworks, you need to master jQuery because this will give you better understanding of how DOM works. If you fast forward to Angular, you may be able to get a few things done, but you won't have an understanding of how things are happening behind the scene.

Client-side Development

By: Mosh Hamedani

Calling an API Using jQuery

```
$.ajax({
    url: "…",
    method: "…", // DELETE, POST, PUT, optional for GET
    success: function(result){
        …
    }
});
```

Bootbox

```
bootbox.confirm("Are you sure?", function(result){
    if (result) {
    }
});
```

DataTables - Zero Configuration

```
$("#customers").DataTable();
```

DataTables - Ajax Source

```
$("#customers").DataTable({
    ajax: {
        url: "…",
        dataSrc: ""
    },
    columns: [
         { data: "name" },
         {
           data: "id",
           render: function(data, type, row){
               return "…";
           }
         }
    ]
});
```

DataTables - Removing Records

```
var table = $("…").DataTable(…);


var $tr = $("…");
table.rows(tr).remove().draw();
```

Client-side Development

By: Mosh Hamedani

Exercise

1- Add DataTable plug-in to the list of movies so we get searching, sorting and pagination out of the box.

2- DataTable should be fed using the API you built in the last section (/api/movies).

3- Remember to remove all the code for rendering the list of movies on the server. (That includes both the controller and view code).

4- This exercise requires modifying the movies API to return genre. So you need to create a separate DTO for that and configure AutoMapper accordingly.

5- Add a delete column with a link to delete a movie. Use Bootbox to display a confirmation box and use jQuery to call the movies API.

Approach

As always, one step at a time.

1- Add the DataTable plug-in first without using the movies API as the data source. Ensure the plumbing code works even when you generate the movies on the server. If that works, then modify the DataTable configuration and supply an AJAX source.

2- Modify the DataTable to use movies API as its data source. Do not worry about deleting movies. Do not worry about eager loading genres. Just ensure that you can get a simple table rendering with what you currently get from movies API.

3- Once that works, return genre from movies API and add it in the table.

4- Add a delete button in the third column of the table. Use native JS confirm method and make sure when you call the delete link, confirmation box is shown. Do not worry about calling the API.

5- Once that works, call the API to delete the movie.

6- Once that works, replace the native JS confirmation box with Bootbox.

7- Finally, remove the corresponding record from DataTable.