MVC Books

27 октомври 2016 г. 14:02

```
    public ActionResult Test(string name, int id)
    {
        return Content("name: " + name + " with id " + id);
    }
}
```

http://localhost:54209/Home/Test?name=gogo&id=13

name: gogo with id 13

```
public string Test(string name, int id=1)
{
    return HttpUtility.HtmlEncode("name: " + name + " with id " + id);
}
```

http://localhost:54209/Home/Test/153?name=gogo

name: gogo with id 153

```
routes.MapRoute(
    name: "Test",
    url: "{controller}/{action}/{name}/{id}"
    );
```

http://localhost:54209/Home/Test/Gogo/3252

name: Gogo with id 3252

Adding views!

```
public ActionResult Test(string name, int id=1)
{
    ViewBag.name = name;
    ViewBag.id = id;
    return View();
}
```

++++++++++

```
@{
    ViewBag.Title = "Test";
}

<h2>Hello From The Test View, @ViewBag.name with ID of @ViewBag.id</h2>
```

http://localhost:54209/Home/Test/Gogo/3252
Hello From The Test View, Gogo with ID of 3252
We are going to use a ViewModel in the future!
/a model for the view holding filling data/

Adding Model!

```
public class Book
{
    public int ID { get; set; }
    public string Name{ get; set; }
    public DateTime ReleaseDate { get; set; }
    public string Genre { get; set; }
    public string Author { get; set; }
}

public class MovieDBContext : DbContext
{
    public DbSet<Book> Books { get; set; }
}
```

The **MovieDBContext** class represents the **Entity Framework** movie database context, which handles fetching, storing, and updating Movie class instances in a database. The MovieDBContext derives from the DbContext base class provided by the Entity Framework.
Open the application root **Web**.**config** file
…
<**connectionStrings**>
<add name="**Book**DBContext" connectionString="Data Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\Movies.mdf;Integrated Security=True" providerName="System.Data.SqlClient"/>

By default, the Entity Framework looks for a connection string named the same as the object context class (**Book**DBContext for this project).

You don't actually need to add the **MovieDBContext** connection string. If you don't specify a connection string, Entity Framework will create a LocalDB database in the users directory with the fully qualified name of the DbContext class (in this case

MvcMovie.Models.MovieDBContext). You can name the database anything you like, as long as it has the .MDF suffix. For example, we could name the database MyFilms.mdf

BooksController > with views, using EF > Book Model >

Invalid column name 'ImgURL'.
When we change the **Model**, we need to migrate the DB

View > more windows > Package Manager Console > Enable-Migrations -ContextTypeName MVCBooks.Models.BookDBContext

OR https://www.asp.net/mvc/overview/getting-started/database-first-development/changing-the-database
---
Strongly Typed Models and the @model Keyword

MVC also provides the ability to pass strongly typed objects to a view template. This strongly typed approach enables better compile-time checking of your code and richer IntelliSense in the Visual Studio editor. The scaffolding mechanism in Visual Studio used this approach (that is, passing a strongly typed model) with the MoviesController class and view templates when it created the methods and views.
…
```
    return View(movie);
```

view:
@model MvcMovie.Models.Movie
This @model directive allows you to access the movie that the controller passed to the view by using a Model object that's strongly typed.

@model IEnumerable<MvcMovie.Models.Movie>
accepts a list of movies, then
@foreach (var item in Model){

@Html.DisplayNameFor(model => model.Name)}

Making the date look better

Model >>

```
using System.ComponentModel.DataAnnotations;
[Display(Name = "Release Date")]
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}",
ApplyFormatInEditMode = true)]
        public DateTime ReleaseDate { get; set; }
```

```
[Authorize(Roles = "Admins")]
actionresult Edit…
```

[ValidateAntiForgeryToken]

This ensures that a form being posted to the server was actually generated by the same server. Thus fake forms that do not have the AntiForgeryToken from the correct server, gets rejected

```
public ActionResult
Edit([Bind(Include="ID,Title,ReleaseDate,Genre,Price")] Movie
movie)
```

bind - disabled over possting data, bind to model

+ @Html.AntiForgeryToken() in the View.

@Html.AntiForgeryToken() generates a hidden form anti-forgery token that must match in the **Edit** method of the Movies controller.

Validation

```
    [StringLength(60, MinimumLength = 3)]
    [Required]
    public string Title { get; set; }
```

```
[RegularExpression(@"^[A-Z]+[a-zA-Z''-'\s]*$")]

[Range(1, 100)]
[DataType(DataType.Currency)]
```