بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

بررسی متدهای Details و Delete

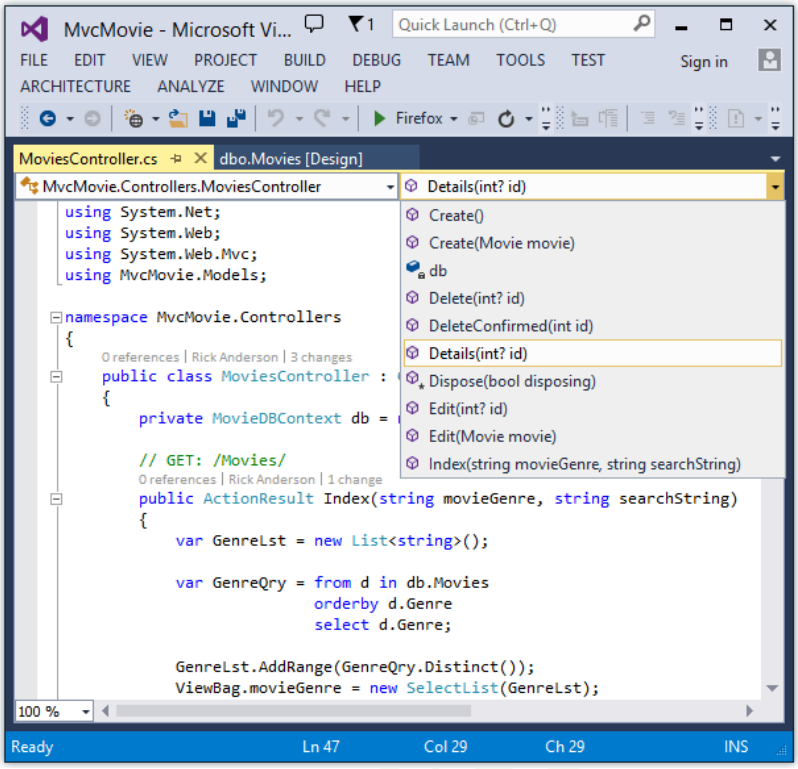
مدرس : مهندس افشین رفوآ

دوره آموزش MVC

بررسی متدهای Details و Delete

در این درس، به بررسی متدهای Details و Delete که به صورت خودکار ایجاد شده اند، خواهم پرداخت.

Movie controller باز کرده و متد Details را بررسی کنید.




```
public ActionResult Details(int? id)
  {
    if (id == null)
    {
      return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
      return HttpNotFound();
    }
    return View(movie);
  }
```

موتور scaffolding چارچوب کاری MVC که این action method را ایجاد کرد، یک توضیح (comment) اضافه می کند که درخواست HTTP فراخواننده ی متد نام برده را مشخص می کند. در این مثال، یک درخواست GET است که دارای URL ای متشکل از سه قسمت می باشد: Movies controller، Details method و یک ID value.

Code First جستجو و یافتن داده ها را به کمک متد Find سهل می سازد. یکی از امکانات امنیتی که به صورت درون ساخته در متد وجود دارد این است که کد اطمینان حاصل کرده و بررسی می کند متد Find پیش

از اینکه کد بتواند عملیاتی را بر روی آیتم (movie) انجام دهد، آیتم مورد نظر (movie) را پیدا کرده باشد.

به عنوان مثال، یک هکر می تواند با تغییر URL ای که توسط لینک ایجاد شده از آدرس

http://localhost:xxxx/Movies/Details/1 به آدرس

http://localhost:xxxx/Movies/Details/12345، error هایی را وارد سایت کند. بنابراین اگر قبلا بررسی نکرده اید که آیا فیلد null وجود دارد یا خیر، این امر ممکن است منجر به رخداد خطا در پایگاه داده شود.

متدهای Delete و DeleteConfirmed را در کد زیر بررسی کنید.

```
// GET: /Movies/Delete/5
  public ActionResult Delete(int? id)
  {
    if (id == null)
    {
      return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
      return HttpNotFound();
    }
    return View(movie);
  }

  // POST: /Movies/Delete/5
  [HttpPost, ActionName("Delete")]
  [ValidateAntiForgeryToken]
  public ActionResult DeleteConfirmed(int id)
  {
    Movie movie = db.Movies.Find(id);
    db.Movies.Remove(movie);
    db.SaveChanges();
    return RedirectToAction("Index");
  }
```

همان طور که در کد مشاهده می کنید، متد HTTP Get Delete، movie مشخص شده را حذف نمی کند، بلکه یک view از movie برمی گرداند که می توانید در آن deletion (HttpPost) را وارد (submit) کنید. اجرای عملیات حذف و یا هر عملیات دیگری که اطلاعات را تغییر می دهد همچون ایجاد، ویرایش و غیره .. در پاسخ به درخواست GET، باعث به وجود آمدن یک حفره ی امنیتی می شود.

متد HttpPost ای که اطلاعات را حذف می کند، DeleteConfirmed نام گذاری شده تا متد HTTP POST دارای یک ورودی (signature) یا اسم منحصر بفرد باشد. ورودی (signature) دو متد مورد نظر در زیر نمایش داده شده:

```
    // GET: /Movies/Delete/5
public ActionResult Delete(int? id)

//
// POST: /Movies/Delete/5
[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id)
```

زبان مشترک زمان اجرا (CLR) ایجاب می کند که متدهای overload شده دارای یک پارامتر ورودی یا parameter signature منحصر بفرد باشند (متدی یکسان که دارای مجموعه ی متفاوتی از پارامترهاست). اما، در اینجا به دو متد Delete نیاز دارید – یکی ویژه ی GET و دیگری برای POST – که هر دو دارای پارامتر ورودی (parameter signature) یکسان باشند. (هر دو متد بایستی یک عدد صحیح را به عنوان پارامتر ورودی بپذیرد.)

برای این منظور، شما دو کار می توانید انجام دهید. اولی این است که به متدها، اسم های متفاوت تخصیص دهید. این دقیقا همان کاری است که مکانیزم scaffolding در مثال قبلی پیاده کرد. اما استفاده از این روش یک مشکل کوچک را بر سر راه شما قرار می دهد: ASP.NET بخش هایی از یک URL را بر اساس اسم به action method ها نگاشت می کند و چنانچه اسم آن متد را تغییر دهید، مسیریابی (routing) از یافتن متد مزبور ناتوان خواهد ماند. یک راه حل وجود دارد و آن در مثال زیر نمایش داده شده.

در این مثال، خصیصه ی ActionName("Delete") به متد DeleteConfirmed اضافه می شود. این کار در اصل نگاشت (mapping) را برای سیستم مسیریابی (routing system) انجام می دهد تا URL ای که (برای درخواست POST) شامل /Delete/ for می باشد، بتواند متد DeleteConfirmed را پیدا کند.

یک روش رایج دیگر برای جلوگیری از رخداد مشکل در استفاده از متدهایی که دارای اسم ها یا ورودی های (signatures) یکسان می باشند، این است که ورودی (signatures) متد POST را برای اضافه کردن پارامتر جدید (استفاده نشده) به صورت ساختگی تغییر دهید. به عنوان نمونه می توان به موردی اشاره کرد

که برنامه نویس در آن یک نوع پارامتر ورودی FormCollection اضافه می کند که به متد POST ارسال می شود، ولی از آن استفاده نمی کند:

```
    public ActionResult Delete(FormCollection fcNotUsed, int id = 0)
{
  Movie movie = db.Movies.Find(id);
  if (movie == null)
  {
    return HttpNotFound();
  }
  db.Movies.Remove(movie);
  db.SaveChanges();
  return RedirectToAction("Index");
}
```

خلاصه

اکنون شما صاحب یک برنامه ی تحت وب ASP.NET MVC کامل هستید که داده های خود را در یک پایگاه داده ی محلی ذخیره کرده است. می توانید در آن اطلاعات پایگاه داده را بخوانید، فیلم مورد نظر را جستجو کنید بروز رسانی و اصلاح بر روی داده ها انجام دهید و در صورت لزوم آن ها را حذف نمایید.