Phần 9. Tìm hiểu hai phương thức Create, Delete và các View liên quan

Ở bài viết trước, bạn đã tìm hiểu về các phương thức Edit ở tập tin **Controller** và hiểu rõ cơ chế hoạt động với View tương ứng. Bài viết này sẽ giúp bạn hiểu thêm về hai phương thức Create và Delete.

# Phương thức Create

Đầu tiên, bạn mở nội dung tập tin Controllers/LinkController.cs và tìm thấy 2 phương thức Create. Phương thức Create() thứ nhất trả về View tương ứng là **Views/Link/Create.cshtml**. Phương thức Create([Bind(Include=”LinkID,LinkName,LinkURL,LinkDescription,CategoryID”)] Link link) thứ hai dùng để lấy dữ liệu từ client do người dùng nhập vào để kiểm tra và lưu vào database.

// GET: /Link/Create

public ActionResult Create()

{

return View();

}

// POST: /Link/Create

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include="LinkID,LinkName,LinkURL,LinkDescription,CategoryID")] Link link)

{

if (ModelState.IsValid)

{

db.Links.Add(link); // thêm đối tượng Link

db.SaveChanges(); // lưu lại --- dammio.com

return RedirectToAction("Index"); // quay trở về trang Index để xem kết quả thay đổi

}

return View(link); // nếu không thể tạo mới thì trả về View như cũ

}

Tiếp theo, bạn mở tập tin **Views/Link/Create.cshtml** để xem nội dung.

# Views/Link/Create.cshtml

@model DammioMVC.Models.Link

@{

ViewBag.Title = "Create";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Create</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Link</h4>

<hr />

@Html.ValidationSummary(true)

<div class="form-group">

@Html.LabelFor(model => model.LinkName, new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.LinkName)

@Html.ValidationMessageFor(model => model.LinkName)

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.LinkURL, new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.LinkURL)

@Html.ValidationMessageFor(model => model.LinkURL)

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.LinkDescription, new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.LinkDescription)

@Html.ValidationMessageFor(model => model.LinkDescription)

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.CategoryID, new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.CategoryID)

@Html.ValidationMessageFor(model => model.CategoryID)

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Create" class="btn btn-default" />

</div>

</div>

</div>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

Trong đoạn mã trên, bạn để ý thấy @Html.AntiForgeryToken() đóng vai trò hiển thị các token bảo mật, còn @Html.ValidationSummary(true) là dùng để hiển thị các thông báo kiểm chứng dữ liệu. Ví dụ, bạn có 1 trường input kiểu số, nhưng bạn lại nhập vào kiểu chuỗi thì hệ thống sẽ báo lỗi.

Ứng với mỗi trường nhập liệu, bạn sẽ thấy một đoạn mã điển hình như sau.

<div class="form-group">

@Html.LabelFor(model => model.LinkName, new { @class = "control-label col-md-2" })

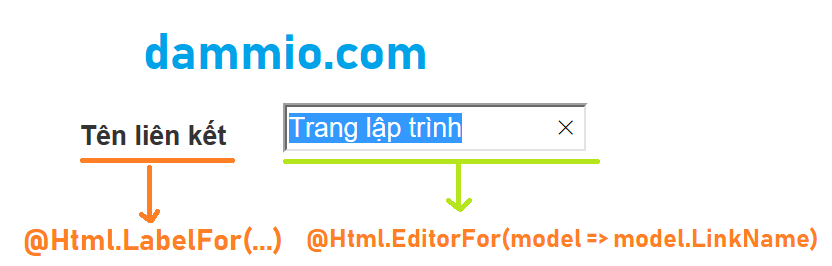
<div class="col-md-10">

@Html.EditorFor(model => model.LinkName)

@Html.ValidationMessageFor(model => model.LinkName)

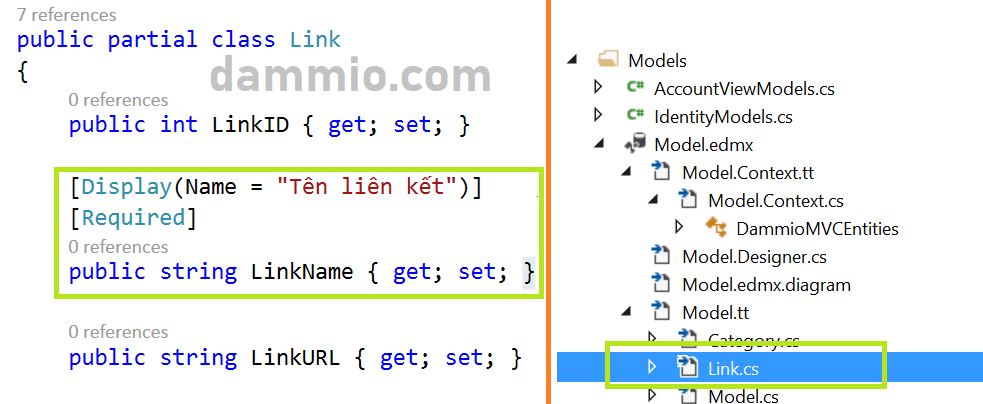
</div>

</div>



Theo hình mô tả trên, @Html.LabelFor dùng để hiển thị nhãn tên, @Html.EditorFor dùng để hiển phần tử input để người dùng nhập dữ liệu, còn @Html.ValidationMessageFor dùng để kiểm chứng dữ liệu nhập vào. Lưu ý nếu bạn không định nghĩa kiểu dữ liệu cho các thuộc tính trong lớp Link.cs thì phần này Visual Studio cho phép nhập dữ liệu trống hoặc null.

Để bắt lỗi cho trường LinkName, bạn mở tập tin Models/Link.cs và thêm mã [Required] vào thuộc tính LinkName. Bạn có thể tham khảo hình sau.



Sau khi thêm xong, chạy build dự án (Ctrl + Shift + B), chạy liên kết http://localhost:xxxx/Link/Create, thử không nhập gì vào trường LinkName (Tên liên kết) và nhấn Submit, bạn có thể xem kết quả.



Bạn có thể thấy trong hình trên hiện cảnh báo bằng tiếng Anh, để hiển thị thông báo theo ý muốn bạn muốn, bạn có thể thay đổi đoạn mã ở giao diện Create.cshtml như sau.

<div class="form-group">

@Html.LabelFor(model => model.LinkName, new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.LinkName)

@Html.ValidationMessageFor(model => model.LinkName, "Trường dữ liệu chưa nhập dữ liệu!")

</div>

</div>

Bạn cũng build dự án (Ctrl + Shift + B) và chạy lại http://localhost:xxxx/Link/Create để xem kết quả được thay đổi.



Trong bài này tiếp theo, bạn sẽ học nhiều cách bắt lỗi dữ liệu dùng lớp DataAnnotations. Bài này chỉ hướng dẫn bản sơ về cách thức thực hiện.

# Phương thức Delete

Bây giờ, chúng ta ta chuyển sang tìm hiểu phương thức Delete cũng như View của nó. Đúng như tên gọi, Delete (xóa) là phương thức dễ hiểu nhất, vì thông thường “phá dễ hơn làm”. Nếu bạn đã học về cơ sở dữ liệu và có một sự quan sát tinh tế về [thiết kế Web](https://www.dammio.com/glossary/thiet-ke-web) động thì thao tác xóa ở Web thường xóa theo chỉ số ID (hay chỉ số khóa) chứ không phải theo bất cứ trường nào khác. Lưu ý xóa theo chỉ số ID với [thiết kế Web](https://www.dammio.com/glossary/thiet-ke-web) là nguyên tắc chung áp dụng cho tất cả các ngôn ngữ Web (PHP, Perl, JSP) chứ không riêng gì ASP.NET [MVC](https://www.dammio.com/glossary/mvc).

Quay trở lại và mở tập tin Controllers/LinkController, bạn tìm thấy 2 phương thức Delete như sau.

// GET: /Link/Delete/5

//Khi click vào link delete sẽ gọi Action này

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Link link = db.Links.Find(id);

if (link == null)

{

return HttpNotFound();

}

return View(link);

}

// POST: /Link/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

Link link = db.Links.Find(id);

db.Links.Remove(link);

db.SaveChanges();

return RedirectToAction("Index");

}

Bạn có thể thấy có 2 phương thức khác nhau. Phương thức Delete(int? id) dùng để xóa Link bất kỳ theo chỉ số ID. Trong phương thức này, bạn có thể thấy kiểm tra xem giá trị đầu vào null hay không, nếu không thì dùng phương thức Find để tìm đối tượng Link có trong database theo ID hay không, nếu có thì xóa, không thì trả về phương thức HttpNotFound(), tức là không tìm thấy.

Phương thức Delete thứ hai, DeleteConfirmed(int id) dùng để xác nhận lại với người dùng lần nữa có muốn xóa hay không. Thực chất phương thức này chạy đầu tiên, nếu người dùng xem thông tin xóa, xác nhận thì phương thức Delete(int? id) mới chạy. Đây là cách thông thường khi [thiết kế Web](https://www.dammio.com/glossary/thiet-ke-web), vì vậy thao tác Delete có thể bị người dùng nhấn nhầm nút, vì vậy nên phải chạy ra trang xác nhận xóa để tránh lỗi vô tình như vậy. Một số hệ thống Web không xóa dữ liệu trực tiếp trong database, thay vào đó chỉ xóa tạm bằng cách đặt một trường Active là 0, tức không kích hoạt và dữ liệu vẫn còn trong database.

Tiếp theo, chúng ta mở Views/Link/Delete.cshtml để xem nội dung mã nguồn.

@model DammioMVC.Models.Link

@{

ViewBag.Title = "Delete";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>Link</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.LinkName)

</dt>

<dd>

@Html.DisplayFor(model => model.LinkName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.LinkURL)

</dt>

<dd>

@Html.DisplayFor(model => model.LinkURL)

</dd>

<dt>

@Html.DisplayNameFor(model => model.LinkDescription)

</dt>

<dd>

@Html.DisplayFor(model => model.LinkDescription)

</dd>

<dt>

@Html.DisplayNameFor(model => model.CategoryID)

</dt>

<dd>

@Html.DisplayFor(model => model.CategoryID)

</dd>

</dl>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

<div class="form-actions no-color">

<input type="submit" value="Delete" class="btn btn-default" /> |

@Html.ActionLink("Back to List", "Index")

</div>

}

</div>

Trong đoạn mã, bạn có thể giao diện trước khi xóa để hiển ra để xác nhận xóa. Bạn có thể thay đổi thông điệp “Are you sure you want to delete this?” nếu muốn. Tương tự các phương thức khác, ở View Delete cũng dùng @Html.AntiForgeryToken() để tạo 1 token bảo mật cho thao tác xóa.

1. **Kết luận**

Như vậy, trong bài này bạn đã tìm hiểu được 2 phương thức Create và Delete cũng như các View liên quan. Việc hiểu rõ cấu trúc từng phương thức, từng nội dung tập tin mã nguồn được gieo bởi Visual Studio là hết sức cần thiết giúp bạn hiểu rõ cơ chế hoạt động dự án trước khi xây dựng một ứng dụng ASP.NET [MVC](https://www.dammio.com/glossary/mvc) chuyên nghiệp.

[1]. [/DammioMVC\_Phan9.rar](https://www.dammio.com/wp-content/uploads/2018/07/DammioMVC_Phan9.rar)