Phần 14: Kết hợp sắp xếp và tìm kiếm

Trong phần này, bạn sẽ học cách kết hợp và sắp xếp với nhau. Việc kết hợp không khó như nhiều bạn nghĩ, đơn giản là chúng ta chỉ cần thêm 1 số đoạn code để kéo dài câu truy vấn [LINQ](https://www.dammio.com/glossary/linq) theo điều kiện tìm kiếm và xử lý 1 số đoạn mã liên quan.

Bài này sẽ tái sử dụng dự án code của bài trước để các bạn tiện theo dõi nội dung.

# Kết hợp sắp xếp và tìm kiếm

## Sửa phương thức Index, tập tin Controller/LinkController.cs

Bạn chỉ cần thêm 1 số đoạn mã và sửa một ít nội dung mã nguồn như sau. Trước hết, thêm một tham số **string searchString** ở phương thức Index, sau đó ở phần 5.1 bạn thêm điều kiện và câu truy vấn. Ở phần 2.3, sửa liên kết các tên cột kèm thêm tham số searchString cho phép sắp xếp theo điều kiện đã tìm kiếm. Thêm nữa, ở phần 1.1 bạn dùng 1 biến ViewBag searchValue để giữ giá trị tìm kiếm trên View phục vụ cho việc tạo các liên kết sắp xếp theo cột.

public ActionResult Index(string sortProperty, string sortOrder, string searchString)

{

// 1. Tạo biến ViewBag SortOrder để giữ trạng thái sắp tăng hay giảm

//ViewBag.SortOrder = String.IsNullOrEmpty(sortOrder) ? "desc" : sortOrder;

if (sortOrder == "asc") ViewBag.SortOrder = "desc";

if (sortOrder == "desc") ViewBag.SortOrder = "";

if (sortOrder == "") ViewBag.SortOrder = "asc";

// 1.1 Giữ giá trị tìm kiếm bằng biến searchValue

ViewBag.searchValue = searchString;

// 2. Lấy tất cả tên thuộc tính của lớp Link (LinkID, LinkName, LinkURL,...)

var properties = typeof(Link).GetProperties();

// 2.0. Tạo 1 danh List với mỗi phần tử là kiểu Tuple

// Tuple<string, bool, int> với tham số lần lượt là <Name, IsVirtual, Order>

// tức là Tên thuộc tính, Thuộc tính là virtual hay không và Thứ tự thuộc tính

List<Tuple<string, bool, int>> list = new List<Tuple<string, bool, int>>();

foreach (var item in properties)

{

int order = 999;

var isVirtual = item.GetAccessors()[0].IsVirtual;

if (item.Name == "LinkName") order = 1;

if (item.Name == "LinkID") order = 2;

if (item.Name == "LinkDescription") order = 3;

if (item.Name == "LinkURL") order = 4;

if (item.Name == "CategoryID") continue; // Không hiển thị cột này

Tuple<string, bool, int> t = new Tuple<string, bool, int>(item.Name, isVirtual, order);

list.Add(t);

}

// 2.1. Sắp xếp theo thứ tự ở trên

list = list.OrderBy(x => x.Item3).ToList();

foreach (var item in list)

{

// 2.2. Thuộc tính bình thường thì cho phép sắp xếp

if (!item.Item2) // Item2 dùng để kiểm tra thuộc tính ảo hay không?

{

// 2.3. So thuộc tính sortProperty và sortOrder để biết thuộc tính nào cần thay biểu tượng sắp giảm

if (sortOrder == "desc" && sortProperty == item.Item1)

{

ViewBag.Headings += "<th><a href='?sortProperty=" + item.Item1 + "&sortOrder=" +

ViewBag.SortOrder + "&searchString=" + searchString + "'>" + item.Item1 + "<i class='fa fa-fw fa-sort-desc'></i></th></a></th>";

}

else if (sortOrder == "asc" && sortProperty == item.Item1)

{

ViewBag.Headings += "<th><a href='?sortProperty=" + item.Item1 + "&sortOrder=" +

ViewBag.SortOrder + "&searchString=" + searchString + "'>" + item.Item1 + "<i class='fa fa-fw fa-sort-asc'></a></th>";

}

else

{

ViewBag.Headings += "<th><a href='?sortProperty=" + item.Item1 + "&sortOrder=" +

ViewBag.SortOrder + "&searchString=" + searchString + "'>" + item.Item1 + "<i class='fa fa-fw fa-sort'></a></th>";

}

}

// 2.4. Thuộc tính virtual (public virtual Category Category...) thì không sắp xếp được

// cho nên không cần tạo liên kết

else ViewBag.Headings += "<th>" + item.Item1 + "</th>";

}

// 3. Truy vấn lấy tất cả đường dẫn

var links = from l in db.Links

select l;

// 4. Tạo thuộc tính sắp xếp mặc định là "LinkID"

if (String.IsNullOrEmpty(sortProperty)) sortProperty = "LinkID";

// 5. Sắp xếp tăng/giảm bằng phương thức OrderBy sử dụng trong thư viện Dynamic LINQ

if (sortOrder == "desc") links = links.OrderBy(sortProperty + " desc");

else if (sortOrder == "asc") links = links.OrderBy(sortProperty);

// 5.1. Thêm phần tìm kiếm

if (!String.IsNullOrEmpty(searchString))

{

links = links.Where(s => s.LinkName.Contains(searchString));

}

// 6. Trả kết quả về Views

return View(links.ToList());

}

## Sửa nội dung tập tin Views/Link/Index.cshtml

Ở tập tin **Views/Link/Index.cshtml**, bạn chỉ cần thêm đoạn cần thêm chứa 1 form HTML với TextBox chứa giá trị ViewBag searchValue.

@model IEnumerable<DammioMVC1.Models.Link>

@{

    ViewBag.Title = "Index";

}

<h2>Index</h2>

<p>

    @Html.ActionLink("Create New", "Create")

</p>

<!-- Đoạn cần thêm -->

@using (Html.BeginForm())

{

    <p>

        LinkName: @Html.TextBox("SearchString", (string) ViewBag.searchValue) <input type="submit" value="Tìm kiếm" />

    </p>

}

<!-- Kết thúc -->

<table class="table">

    <tr>

        @Html.Raw(ViewBag.Headings)

    </tr>

@foreach (var item in Model) {

    <tr>

        <td>

            @Html.DisplayFor(modelItem => item.LinkName)

        </td>

        <td>

            @Html.DisplayFor(modelItem => item.LinkID)

        </td>

        <td>

            @Html.DisplayFor(modelItem => item.LinkDescription)

        </td>

        <td>

            @Html.DisplayFor(modelItem => item.LinkURL)

        </td>

        <td>

            @Html.DisplayFor(modelItem => item.Category.CategoryName)

        </td>

        <td>

            @Html.ActionLink("Edit", "Edit", new { id=item.LinkID }) |

            @Html.ActionLink("Details", "Details", new { id=item.LinkID }) |

            @Html.ActionLink("Delete", "Delete", new { id=item.LinkID })

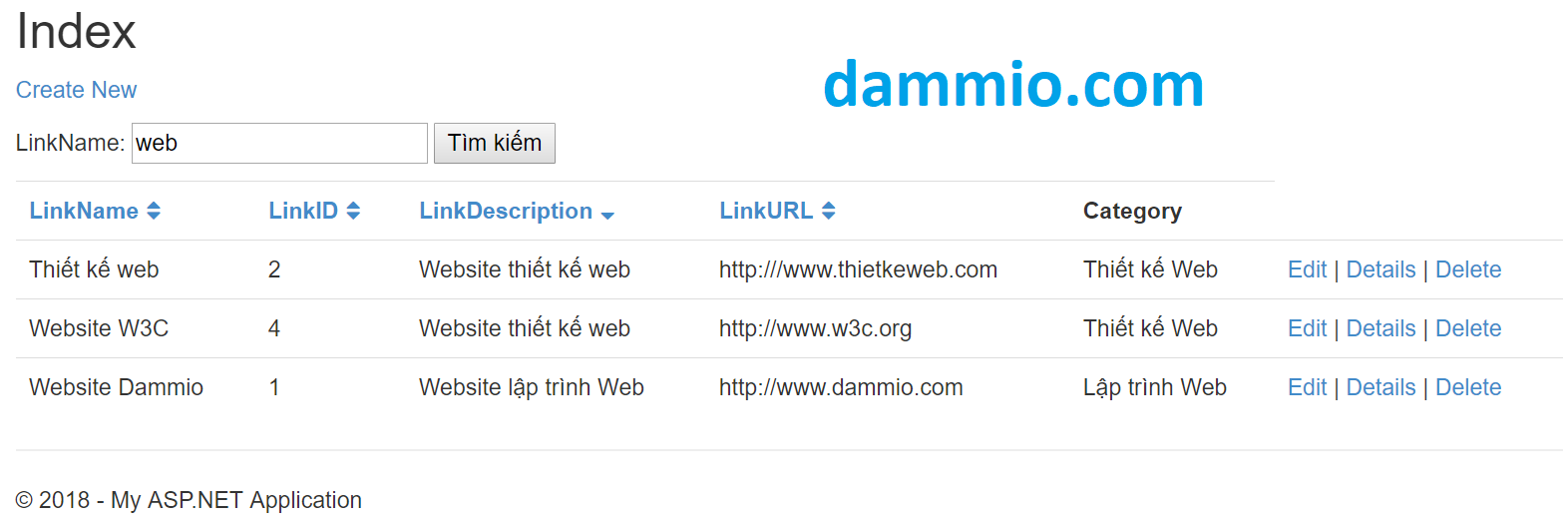
        </td>

    </tr>

}

</table>

Sau đó, bạn nhấn Build dự án, chạy đường dẫn **http://localhost:xxxx/Link/**, thử gõ tìm kiếm rồi nhấn nút Tìm kiếm, tiếp theo thử sắp xếp theo trường nào đó, chẳng hạn như LinkDescription để thử nghiệm và xem kết quả.



# Thêm nút “Reset” ở giao diện View

Nút “Reset” giúp bạn thiết lập lại kết quả tìm kiếm và đặt trang Web ở lại trạng thái ban đầu trước khi tìm kiếm hay là lúc trang Web load lần đầu tiên.

## Sửa giao diện Views/Link/Index.cshtml

Ở giao diện View, bạn chỉ thêm đoạn mã **input type=”submit” name=”Reset” value=”Mặc định”** để đặt tả nút nhấn với thuộc tính **name=”Reset”**.

<!-- Đoạn cần thêm -->

@using (Html.BeginForm())

{

   <p>

      LinkName: @Html.TextBox("SearchString", (string) ViewBag.searchValue)

<input type="submit" value="Tìm kiếm" />

<input type="submit" name="Reset" value="Mặc định"  />

   </p>

}

<!-- Kết thúc -->

## Sửa tập tin Controller/LinkController.cs

Để có thể dò tìm 2 nút nhấn submit trên View và nút nào tương ứng với phương thức nào bạn buộc phải thêm **phương thức HttpParamActionAttribute** để dò tìm. Phương thức này sử dụng thư viện **using System.Reflection;** vì vậy bạn nhúng thêm thư viện này vào dự án.

... nội dung code ...

using System.Reflection; // nhúng vào dự án

namespace DammioMVC1.Controllers

{

public class LinkController : Controller

{

private DammioEntities db = new DammioEntities();

public class HttpParamActionAttribute : ActionNameSelectorAttribute

{

public override bool IsValidName(ControllerContext controllerContext, string actionName, MethodInfo methodInfo)

{

if (actionName.Equals(methodInfo.Name, StringComparison.InvariantCultureIgnoreCase))

return true;

var request = controllerContext.RequestContext.HttpContext.Request;

return request[methodInfo.Name] != null;

}

}

... nội dung code ...

Sau đó, bạn tạo thêm 1 Action tên là **Reset**, lưu ý phải trùng tên với thuộc tính **name=”Reset”** ở View. Bạn chạy hàm Index với tham số rỗng giống như lần đầu trang tải về.

Một View khi chạy Controller **chỉ cho phép 1 form có 1 GET hoặc 1 POST duy nhất**, vì vậy để chạy được bạn phải thêm HttpPost, để mô tả tham số Reset khi nhấn nút Reset thì bạn phải dùng HttpParamAction.

[HttpPost, HttpParamAction]

public ActionResult Reset()

{

    ViewBag.searchValue = "";

    Index("","","");

    return View();

}

Sau đây là code đầy đủ tập tin **Controller/LinkController.cs** dành cho bạn nào không biết thêm chính xác vị trí nào.

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using DammioMVC1.Models;

using System.ComponentModel;

using System.Linq.Dynamic;

using System.Linq.Expressions;

using System.Reflection;

namespace DammioMVC1.Controllers

{

public class LinkController : Controller

{

private DammioEntities db = new DammioEntities();

public class HttpParamActionAttribute : ActionNameSelectorAttribute

{

public override bool IsValidName(ControllerContext controllerContext, string actionName, MethodInfo methodInfo)

{

if (actionName.Equals(methodInfo.Name, StringComparison.InvariantCultureIgnoreCase))

return true;

var request = controllerContext.RequestContext.HttpContext.Request;

return request[methodInfo.Name] != null;

}

}

// GET: /Link/

public ActionResult Index(string sortProperty, string sortOrder, string searchString)

{

// 1. Tạo biến ViewBag SortOrder để giữ trạng thái sắp tăng hay giảm

//ViewBag.SortOrder = String.IsNullOrEmpty(sortOrder) ? "desc" : sortOrder;

if (sortOrder == "asc") ViewBag.SortOrder = "desc";

if (sortOrder == "desc") ViewBag.SortOrder = "";

if (sortOrder == "") ViewBag.SortOrder = "asc";

// 1.1 Giữ giá trị tìm kiếm bằng biến searchValue

ViewBag.searchValue = searchString;

// 2. Lấy tất cả tên thuộc tính của lớp Link (LinkID, LinkName, LinkURL,...)

var properties = typeof(Link).GetProperties();

// 2.0. Tạo 1 danh List với mỗi phần tử là kiểu Tuple

// Tuple<string, bool, int> với tham số lần lượt là <Name, IsVirtual, Order>

// tức là Tên thuộc tính, Thuộc tính là virtual hay không và Thứ tự thuộc tính

List<Tuple<string, bool, int>> list = new List<Tuple<string, bool, int>>();

foreach (var item in properties)

{

int order = 999;

var isVirtual = item.GetAccessors()[0].IsVirtual;

if (item.Name == "LinkName") order = 1;

if (item.Name == "LinkID") order = 2;

if (item.Name == "LinkDescription") order = 3;

if (item.Name == "LinkURL") order = 4;

if (item.Name == "CategoryID") continue; // Không hiển thị cột này

Tuple<string, bool, int> t = new Tuple<string, bool, int>(item.Name, isVirtual, order);

list.Add(t);

}

// 2.1. Sắp xếp theo thứ tự ở trên

list = list.OrderBy(x => x.Item3).ToList();

foreach (var item in list)

{

// 2.2. Thuộc tính bình thường thì cho phép sắp xếp

if (!item.Item2) // Item2 dùng để kiểm tra thuộc tính ảo hay không?

{

// 2.3. So thuộc tính sortProperty và sortOrder để biết thuộc tính nào cần thay biểu tượng sắp giảm

if (sortOrder == "desc" && sortProperty == item.Item1)

{

ViewBag.Headings += "<th><a href='?sortProperty=" + item.Item1 + "&sortOrder=" +

ViewBag.SortOrder + "&searchString=" + searchString + "'>" + item.Item1 + "<i class='fa fa-fw fa-sort-desc'></i></th></a></th>";

}

else if (sortOrder == "asc" && sortProperty == item.Item1)

{

ViewBag.Headings += "<th><a href='?sortProperty=" + item.Item1 + "&sortOrder=" +

ViewBag.SortOrder + "&searchString=" + searchString + "'>" + item.Item1 + "<i class='fa fa-fw fa-sort-asc'></a></th>";

}

else

{

ViewBag.Headings += "<th><a href='?sortProperty=" + item.Item1 + "&sortOrder=" +

ViewBag.SortOrder + "&searchString=" + searchString + "'>" + item.Item1 + "<i class='fa fa-fw fa-sort'></a></th>";

}

}

// 2.4. Thuộc tính virtual (public virtual Category Category...) thì không sắp xếp được

// cho nên không cần tạo liên kết

else ViewBag.Headings += "<th>" + item.Item1 + "</th>";

}

// 3. Truy vấn lấy tất cả đường dẫn

var links = from l in db.Links

select l;

// 4. Tạo thuộc tính sắp xếp mặc định là "LinkID"

if (String.IsNullOrEmpty(sortProperty)) sortProperty = "LinkID";

// 5. Sắp xếp tăng/giảm bằng phương thức OrderBy sử dụng trong thư viện Dynamic LINQ

if (sortOrder == "desc") links = links.OrderBy(sortProperty + " desc");

else if (sortOrder == "asc") links = links.OrderBy(sortProperty);

// 5.1. Thêm phần tìm kiếm

if (!String.IsNullOrEmpty(searchString))

{

links = links.Where(s => s.LinkName.Contains(searchString));

}

// 6. Trả kết quả về Views

return View(links.ToList());

}

[HttpPost, HttpParamAction]

public ActionResult Reset()

{

ViewBag.searchValue = "";

Index("","","");

return View();

}

// GET: /Link/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

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

if (link == null)

{

return HttpNotFound();

}

return View(link);

}

// GET: /Link/Create

public ActionResult Create()

{

ViewBag.CategoryID = new SelectList(db.Categories, "CategoryID", "CategoryName");

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);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.CategoryID = new SelectList(db.Categories, "CategoryID", "CategoryName", link.CategoryID);

return View(link);

}

// GET: /Link/Edit/5

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

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

if (link == null)

{

return HttpNotFound();

}

ViewBag.CategoryID = new SelectList(db.Categories, "CategoryID", "CategoryName", link.CategoryID);

return View(link);

}

// POST: /Link/Edit/5

// 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 Edit([Bind(Include="LinkID,LinkName,LinkURL,LinkDescription,CategoryID")] Link link)

{

if (ModelState.IsValid)

{

db.Entry(link).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.CategoryID = new SelectList(db.Categories, "CategoryID", "CategoryName", link.CategoryID);

return View(link);

}

// GET: /Link/Delete/5

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");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Cuối cùng, bạn Build dự án, chạy đường dẫn **http://localhost:45033/Link/** ở trình duyệt Web để test kết quả.



# Kết luận

Bài viết đã chỉ cho bạn cách kết hợp sắp xếp và tìm kiếm trên View giúp người dùng có thể tương tác tốt hơn trên giao diện. Ngoài ra, bạn có thể kết hợp nhiều chức năng khác như phân trang, đánh số trang, thay đổi dữ liệu các cột, phong cách hiển thị,… Trên thực tế, các lập trình viên đều viết các chức năng này trong thư viện DLL hoặc các lớp code riêng và nhúng vào Controller và Views khi cần, do đó code sẽ gọn nhẹ và mang tính toàn cục hơn. Mời bạn tiếp tục theo dõi bài tiếp theo.

# References

[1]. [DammioMVC1\_phan14.rar](https://www.dammio.com/wp-content/uploads/2018/10/DammioMVC1_phan14.rar)