**TÀI LIỆU KỸ THUẬT ỨNG DỤNG WEB MCVN**

**PHẦN I: GIỚI THIỆU CHUNG.**

- Đây là hệ thống hoạt động dưới dạng ứng dụng web. Nên người dùng không cần cài đặt thiết lập, mà chỉ cần sử dụng tài khoản để truy cập trực tuyến từ trình duyệt web trên mọi thiết bị như Laptop, Tablet, Smartphone để nhập liệu và tra cứu thông tin.

- Các tính năng cơ bản gồm: Tạo, chỉnh sửa và tra cứu các hồ sơ dịch vụ, các hoạt động điều trị, thông tin bệnh nhân, người thân và cán bộ PHCN. Ngoài ra, đối với nhân sự MCNV có thể quản lý đào tạo và tạo báo cáo trong lĩnh vực PHCN.

- Được xây dựng dựa trên ngôn ngữ lập trình C#, DevExpress; cấu trúc theo mô hình MVC, thao tác với cơ sở dữ liệu thông qua XPO – ORM Library.

- Hệ thống được chia làm 2 phần chính: WebApp và WebModule.

+ WebApp: tập trung vào các chức năng Hệ Thống, Cấu Hình (có thể tái sử dụng cho các dự án khác).

+ WebModule: tập trung vào các chức năng theo yêu cầu (thiết kế riêng cho từng dự án).

**PHẦN II: CẤU TRÚC HỆ THỐNG.**

**1. Menu**

- Hình ảnh minh họa:

A screenshot of a computer

Description automatically generated

- Được khởi tạo tại: WebModule → App\_Data → menu.xml

WebModule



App\_Data



menu.xml

- Chi tiết:

+ Cấu trúc:

<data>

<Tab>

<Name>[Name]</Name> → [Name]: Tên tab (Dịch vụ, đào tạo,…)

<Group>

<Name>[Name]</Name> → [Name]: Tên group (Hoạt động chính,…)

<Image>[Image]</Image> → [Image]: Ảnh đại diện group

<Item>

<Image>[Image]</Image> → [Image]: Ảnh đại diện module

<Module>[Module]</Module> → [Module]: WebApp/WebModule

<Function>[Function]</Function> → [Function]: Tên Function (xác định module thuộc controller nào)

<Name>[Name]</Name> → [Name]: Tên module (Lượng giá,…)

</Item>

</Group>

</Tab>

</data>

+ Lưu ý: Trong <Tab> có thể có nhiều <Group>, trong <Group> có thể có nhiều <Item>.

**2. View**

- Được khởi tạo tại: WebModule → Views → [Name]

WebModule

Views

Area



- Chi tiết:

+ Các tập tin trong [Name] có phần mở rộng .cshtml, đây là tệp HTML C# được công cụ Razor Markup sử dụng ở phía máy chủ để hiển thị tệp trang web cho trình duyệt của người dùng.

+ Gửi yêu cầu của người dùng lên máy chủ.

+ Tiếp nhận thông tin từ máy chủ gửi về cho người dùng.

**3. Controller**

- Được khởi tạo tại: WebModule → Controllers → [Name]Controller.cs

WebModule

Controllers

AreaController.cs

- Chi tiết:

+ Tiếp nhận thông tin yêu cầu của người dùng và gửi lên máy chủ.

+ Tiếp nhận thông tin từ máy chủ và trả về dữ liệu cho người dùng.

**4. Model**

- Được khởi tạo tại: WebModule → Models → [Name].cs

WebModule

Models

Area.cs

- Chi tiết:

+ Tiếp nhận thông tin yêu cầu từ controller.

+ Xử lý, trả thông tin đã được xử lý về controller.

**5. DataModel**

- Chứa hệ thống cơ sở dữ liệu.

- Được khởi tạo tại: WebApp/WebModule → DataModel.xpo

WebApp/WebModule

DataModel.xpo

- Cách tạo:

+ Mở Toolbox, kéo thả Persistent Object vào khung nhìn.

A screenshot of a computer

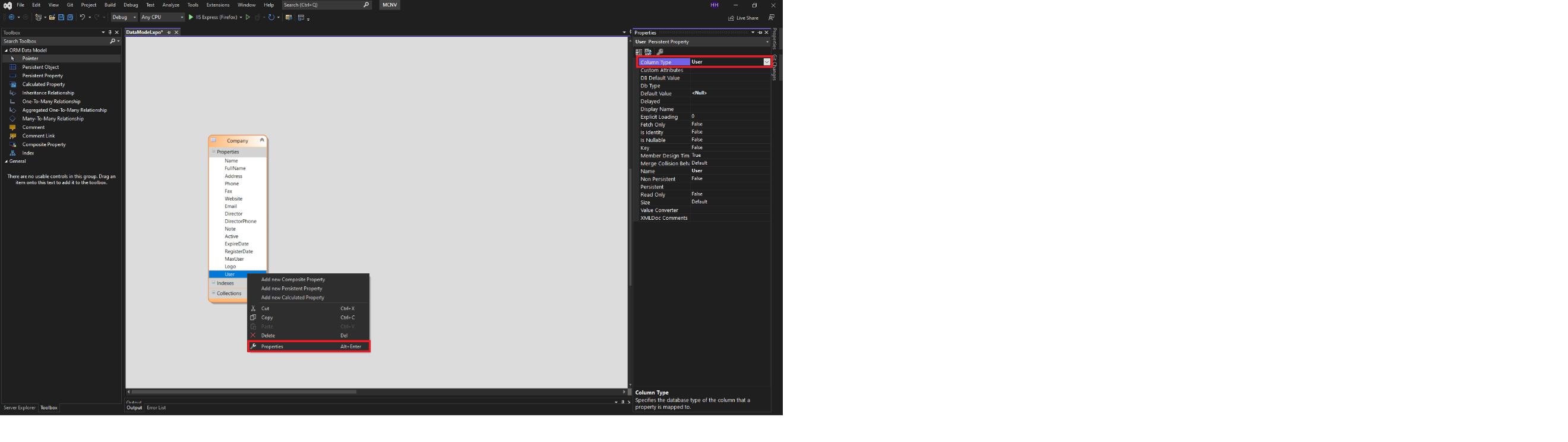
Description automatically generated

+ Bổ sung trường dữ liệu: click chuột phải vào Properties, chọn Add new Persistent Property.

A screenshot of a computer

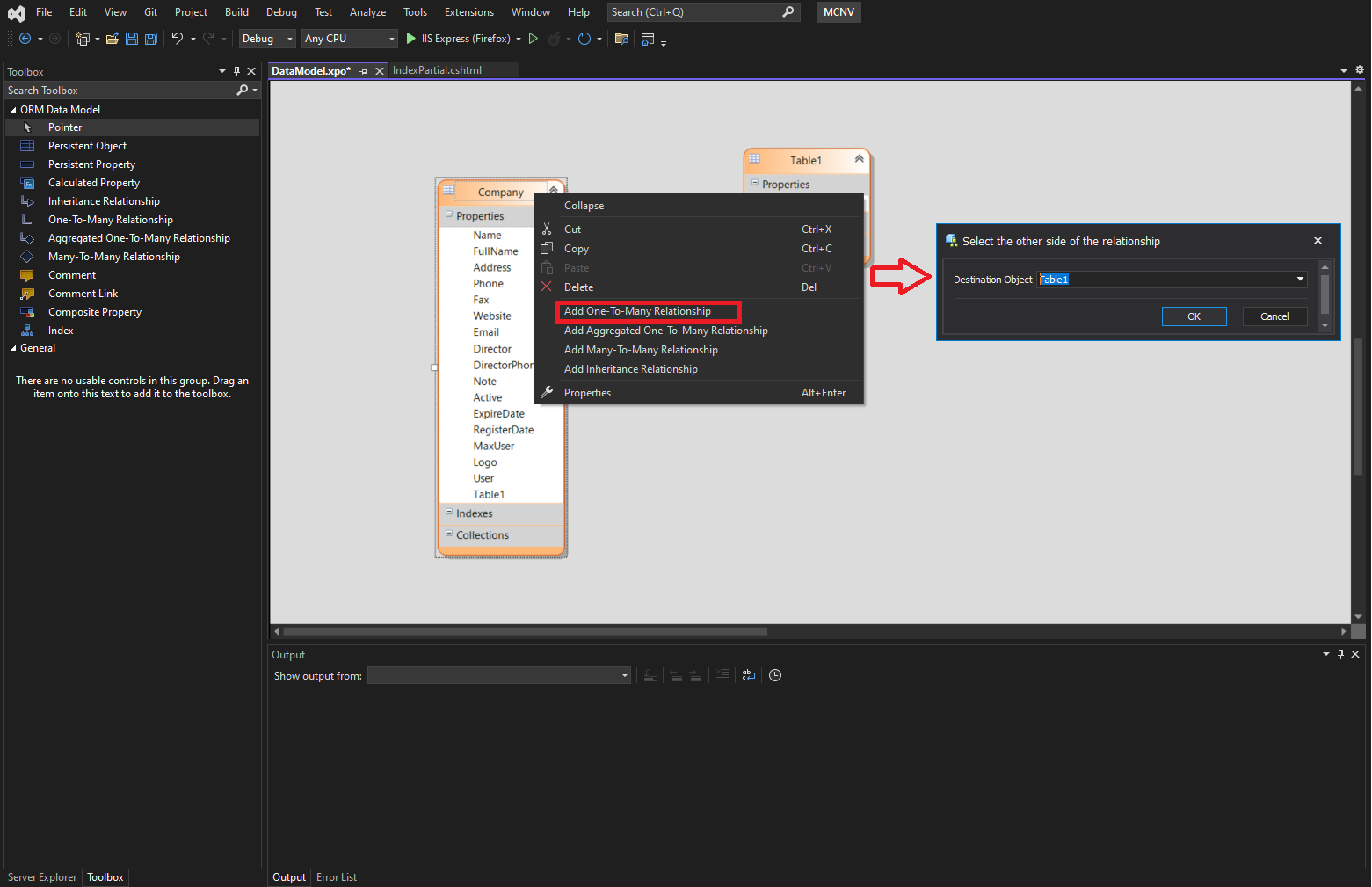
Description automatically generated

+ Thay đổi kiểu dữ liệu: click chuột phải vào trường dữ liệu, chọn properties. Xuất hiện bảng Properties, chọn Column Type tương ứng với trường dữ liệu.



+ Thiết lập mối quan hệ giữa các bảng:

\*Quan hệ One-to-Many: click chuột phải vào table [One], chọn Add One-To-many Relationship, xuất hiện bảng quan hệ chọn table tương ứng. Lưu ý các giữa các table phải có trường dữ liệu liên quan đến nhau.



\*Quan hệ Many-to-Many: click chuột phải vào table, chọn Add Many-To-many Relationship, xuất hiện bảng quan hệ chọn table tương ứng.

A screenshot of a computer

Description automatically generated

- Lưu ý: Khi tạo mối quan hệ giữa các table, sẽ tự sinh ra các table mới với tên được gắn tự động, mở bảng Properties để sửa tên theo ý muốn.

A screenshot of a computer

Description automatically generated

**PHẦN III: CHI TIẾT MODULE.**

**A. Module Công ty**

- Mục đích: Phân tách dữ liệu tương ứng với từng công ty, mỗi công ty sẽ có hệ thống dữ liệu riêng.

- Vị trí trong giao diện: Hệ thống → Công ty

**1. Cơ sở dữ liệu**

- Truy cập tệp: DataModel.xpo → Table Company

WebApp

DataModel.xpo

**2. Vận hành**

WebApp



Models

Company.cs

Controllers

CompanyControllers.cs

Views

Company

- Khi có yêu cầu gọi view từ menu, controller sẽ tiếp nhận yêu cầu, gửi thông tin đến model (nếu có) và trả thông tin đến view.

(Controller)

[Function("Công ty")] → Xác định controller được gọi

public class CompanyController : Core.BaseController

{

public ActionResult Index() → Mặc định chạy Index() khi gọi controller

{

return PartialView(new Models.Company.ListInfo());

}

}

+ “[Function("Công ty")]” dùng để xác định controller nào sẽ được gọi từ menu, khi gọi vào controller sẽ mặc định chạy hàm “Index()”, hàm “Index()” sẽ trả về một thành phần view “Index.cshtml”.

- Tại “Index.cshtml” sẽ xử lý thao tác người dùng, định hướng action, hiển thị thông tin,…

(Index.cshtml)

<div class="page-container">

<div class="page-header grid-middle">

<div class="mr img company"></div>

<div class="title">

@H("Title")

</div>

</div>

<div class="page-content">

<form>

@Html.Partial("IndexPartial") → gọi view “IndexPartial.cshtml”

</form>

</div>

</div>

+ View được chia nhỏ làm nhiều thành phần hỗ trợ việc tái sử dụng và dễ dàng sửa chữa.

+ “IndexPartial.cshtml” đảm nhiệm chức năng hiển thị dữ liệu:

(IndexPartial.cshtml)

@model Company.ListInfo → Model binding

@Html.DevExpress().GridView(st =>{

Html.DefaultGridSheet(st, Model.ShowFilter); → Hàm tìm kiếm

st.Columns.Add(col => → Bổ sung trường dữ liệu hiển thị

{

col.FieldName = "Name";

col.Caption = T("Title");

col.Settings.AutoFilterCondition = AutoFilterCondition.Contains;

});

}).BindToLINQ("", "", (s, e) =>{

e.QueryableSource = Model.GetData(); → Lấy dữ liệu

e.DefaultSorting = "Name";

e.KeyExpression = "Oid";

}).GetHtml()

- Thêm dữ liệu:

+ Khi người dùng thực hiện yêu cầu thêm dữ liệu, cần xác định action được gọi.

(Index.cshtml)

add: function () {

WebApp.showDialog({

title: '@H("Thêm")',

url: '@Url.Action("Add")' → Xác định action

}).done(function(v) {

grdSheet.PerformCallback();

});

}

+ Controller tiếp nhận yêu cầu, gọi vào action tương ứng, lấy dữ liệu từ model và trả về một thành phần view.

(Controller)

[Logic("Thêm")]

public ActionResult Add()

{

return PartialView("EditTemplate", Models.Company.Create());

}

+ "EditTemplate.cshtml" đảm nhiệm chức năng Thêm/Sửa:

(EditTemplate.cshtml)

@model Company → Model binding

@Html.HiddenFor(m => m.Id) → Kiểm tra thêm hay sửa dữ liệu

<div class="grid">

<div class="col-6\_xs-12 grid">

<div class="col-12">@Html.EditorFor(m => m.Name)</div> → Gọi từ model

</div>

</div>

+ Model xử lý dữ liệu (hiển thị):

(Model)

// Khai báo biến

public int Id { get; set; }

[Core.Caption("Tên")]

public string Name { get; set; }

// Thực hiện yêu cầu từ controller

public Company() { }

public static Company Create()

{

var vm = new Company()

{

Active = true → Khởi tạo các giá trị mặc định

};

return vm;

}

+ Khi người dùng submit form, controller tiếp nhận yêu cầu, gọi vào action post và gửi dữ liệu đến model.

(Controller)

[Logic("Thêm")]

[HttpPost] → Submit form

public ActionResult Add(Models.Company vm)

{

if (!ModelState.IsValid)

return PartialView("EditTemplate", vm);

vm.Save(); → Truyền dữ liệu đến model

return Nothing();

}

+ Model xử lý dữ liệu (lưu dữ liệu):

(Model)

public void Save(){

DataModel.Company model; → Gọi table tương ứng trong database

model.Name = Name; → Gắn giá trị mới

Core.ApplicationDbContext.Current.Session.CommitChanges(); → Thực thi

}

- Sửa dữ liệu: giống với việc thêm dữ liệu, chỉ khác model sẽ truyền dữ liệu được tìm thấy thông qua yêu cầu của view.

(Model)

// Thực hiện yêu cầu từ controller

public static Company Load(int id)

{

var model = Core.ApplicationDbContext.Current.Session.GetObjectByKey<DataModel.Company>(id); → Lấy dữ liệu tương ứng với ID

if (model == null) return null;

return new Company(model);

}

public Company(DataModel.Company model) {

Id = model.Oid; → Gắn giá trị mới

Name = model.Name;

}

public void Save(){

if (Id != 0)

model = Core.ApplicationDbContext.Current.Session.GetObjectByKey<DataModel.Company>(Id); → Lấy dữ liệu tương ứng với ID

DataModel.Company model; → Gọi table tương ứng trong database

model.Name = Name; → Gắn giá trị mới

Core.ApplicationDbContext.Current.Session.CommitChanges(); → Thực thi

}

- Xóa dữ liệu:

+ Khi người dùng thực hiện yêu cầu xóa dữ liệu, cần xác định action được gọi.

(Index.cshtml)

del: function () {

var idx = grdSheet.GetFocusedRowIndex();

var id = grdSheet.GetRowKey(idx);

if (!id) return;

WebApp.showConfirm(WebApp.message.delete).done(function (r) {

WebApp.ajax('@Url.Action("Delete")', {id: id}).done(function() {

grdSheet.PerformCallback();

});

});

}

+ Controller tiếp nhận yêu cầu, gọi vào action tương ứng và gửi yêu cầu đến model

(Controller)

[Logic("Xóa")]

public ActionResult Delete(int id)

{

Models.Company.Delete(id);

return Nothing();

}

+ Model xử lý dữ liệu

(Model)

public static void Delete(int id)

{

var model = Core.ApplicationDbContext.Current.Session.GetObjectByKey<DataModel.Company>(id); → Lấy dữ liệu tương ứng với ID

if (model == null) return;

model.Delete(); → Gọi lệnh xóa

Core.ApplicationDbContext.Current.Session.CommitChanges(); → Thực thi

}