Bài 13: Áp dụng Dependency Injection vào dự án sử dụng autofac.

* Bước 1 vào nuget cài 3 thư viện autofac,autofacmvc,autofacwebapi2
* Tạo file startup (Owin startup class)
* Debug PostCategoryService
* Cài đặt NuGet Microsoft.Owin.Host.SystemWeb
* Sửa lại postcategorycontroller GetAll method

Add dependence inject by autofac to web.

Bai 14: Sử dụng AutoMapper để map đối tượng

* Cài AutoMapper 4.2..1
* Tạo tầng ViewModel
* Tạo thư mục Mappings

+ Tạo class AutoMapperConfiguration.cs

+ Khai báo AutoMapperConfiguration vào Global.asax

* Tạo Class EntityExtension ( tìm hiểu extension method )

\*\* Giúp đẩy giá trị trừ ViewModel vào Model chính

* Khởi tạo Mapper vào API PostCategoryController ( get, add,update

**Bài 15: tích hợp asp.net identity để chứng thực người dung**

\*\*Identity là một cơ chế xác thực và quản lý người dung mới nhất của Microsoft dành cho ứng dụng ASP.net, tiền thân của ASP.Net Identity là Membership

Đặc điểm : là cơ chế dung chung cho tất cả các ứng dụng Web bao gồm ASP.Net MVC, Web API, WebForm và SignalR, dễ dàng them mới các trường dữ liệu khác vào user, dễ dàng unit test, quản lý quyền, hỗ trợ login với các Social dễ dàng, độc lập với web vì sử dụng cơ chế OWin, cài đặt từ Nuget

Các bước thực hiện cài đặt DI Autofac

1. Cài đặt 3 thư viện EntityFramework, Core, Owin
2. Tạo mới class User kết thừa từ IdentityUser
3. Tạo mới Role kết thừa từ IdentityRole
4. Kết thừa lớp DbContext từ dentityDbContext<User>
5. Thực hiện Migration vào database
6. Tạo class quản lý authen

Bước 1 > tạo một project mới để lấy mẫu Identity của ASP.Net > cài đặt thư viện > trong project Model tạo model ApplicationUser : IdentityUser, them mới một số properties > sửa trong project Data DBContext kết thừa từ IdentityDbContext, them phương thức tạo mới chính nó > trong project Web tạo class IdentityConfig trong thư mục App\_Start, cần 2 phương thức application SignIn và User > trong folder app\_start tạo mới một class OwinStartup, Startup.Auth > Migration add-migration Intergrate-aspnetIdentity > them vào DBContext builder.Entity > update-database lại > trong file startup.cs them configureAuth > Tron thư mục Api tạo AccountController > triển khai dependence cho các class mới trong Startup.cs > viết thử một service để test class mới AccountController > Seed sample data trong project Data > update database > có bug sửa lại phương thức trong dbContext > add-migration Changekey> điều chỉnh accountController thanh có [RoutePrefix] Login > có bug không thể test bằng postman, them mới và kiểm tra và cấu hình lại ApplicationUserStore trong IdentityConfig của thư mục App\_Start > Commit lên git add asp identity.

**Bài 16:** tổng quan về AngularJS và cách cài đặt sử dụng Bower > tạo file BowerConfig trở đường dẫn về Script> plugin > cài đặt angular 1.5.5 > tạo AdminController > tạo một view và gán angular vào

Bai 21: dựng cấu trúc cho phần quản trị với Single Page Application

Bài 22: Triển khai routing cho sử dụng Angular UI Router

Router là một cơ chế rất quan trọng trong ứng dụng SPA

Nó độc lập với cơ chế điều hướng của ASP.NET MVC

Nó tự động sử dụng view tương ứng với URL

Trong AngularJS:

* ngRoute: là một module core của AngularJS giúp điều hướng trong các kịch bản đơn giản.
* Ui-router: được phát triển bởi cộng đồng, khắc phục và them các tính năng mạnh mẽ cho ng-Route

Ui-router: cho phép view lồng nhau. Điều này là rất hưu ích với ứng dụng lớn, nơi bạn có thể có các trang mà kế thừa từ các phần khác, cho phép

Thực hành :

Bước 1 : Cài đặt Angular-ui-router : 0.2.18 > View trang quản trị > download template AdminLTE > Chép file dist của template vào Assets Admin > chép html template vào view index.html, thay đổi đường dẫn các file dist > sử dụng bower để them các thư viện mà template cần (jquery 2.2.3), bootstrap 3.3.6, slimScroll , fastclick > test nếu lỗi , xóa code trong bundleConfig > nhúng angularJS và UI-Router > đổi tên file js products.router.js thành products.module.js trong components > them homeController.js vào thư mục components > tạo thự mục modules trong shared > tạo common.js > khai báo app.js khai báo để tiến hành routing > tạo module common.js > khai báo product.module.js > khai báo homeController.js > khai báo angular cho html và them các đường dẫn của controller.js cho html.

Bước 2 : triển khai ui-router

Add thư mục home vào components > tìm phần html thân tách riêng > chỉ định khu vực để ren giao diện ra sử dụng ui-view của ui-router > sử dụng ui-sref để chuyển trang cơ chế ui-router

Bài 23 : binding dữ liệu từ WebAPI ra bảng trong HTML bằng AngularJS

Thực hành: tạo them service ProductCategoryService > tạo API ProductCategoryController > tạo them ViewModel > test API, seed them dữ liệu > Mapping cho view model mới > tạo apiService cho SPA > nhúng API vào view chính > tạo mới productCategoryListView và Controller, tạo mới module cho ProductCategory > include vào app chính > nhúng vào view chính > thay đổi HomeView > tạo productCategoryListView > khai báo ProductCategoryListController, khai báo một phương thức để lấy Data từ server, inject APIService > sử dụng ng-repeat triển khai trong productCategoryListView

Bài 24: Sử filter trong AngularJS

Thực hành > update table ProductCategory set CreatedDate > sử dụng filter để định dạng lại ngày > tạo thưc mục filter, tạo một statusFilter angularJS custom > nhúng filter vào view > sử dụng ng-class để gán điều kiện > thêm cột thao tác cho table front end > tìm icon phù hợp trên fontawesone

Bài 25: Sử dụng directive để tạo phân trang cho list

Các bước chính :

-Backend : tạo class generic chứa thông tin paging trả về, áp dụng vào phần trang cho webAPI sử dụng LINQ

-Frontend: viết mới một pager directive để phân trang, áp dụng directive vào để phân trang

Thực hành: tạo class PaginationSet vào thư mục Core > tạo phân trang vào class API ProductcategoryController vào phương thức getAll > test API bằng postman > tạo thư mục directives và pagerDirective angular > tạo một pagerdirective.html > nhung pân trang vào productCategoryListController >khai báo và nhúng vào view > commit paging for list

Bài 26: Tìm kiếm dữ liệu trong bảng bằng AngularJS

-Backend : Thêm một tham số filter keyword vào phương thức của API, test bằng post man

-Frontend: Chỉnh sửa controller để thêm tham số, Call lên hàm sử dụng binding sự kiện trong angular

Thực hành: chỉnh sửa API productcategoryController > Thêm phương thức vào service GetAll(string keyword), dung phương thức GetMulti của Repository > tiếp tục sửa Clien productCategoryListController thêm scope keyword > thêm row tìm kiếm cho VIEW thêm ng-model để binding keyword > khai báo sự kiện để nhận binding model trong productCategoryListController > đặt sự kiện ng-click cho button search() > thêm tfooter hiển thị tổng số bản ghi > commit search for list

Bài 27: Triển khai service thông báo cho người dung

Các bước : sử dụng bower để cài đặt toastr, tạo custom service notificationService gắn vào module, nhúng service vào index, inject vào controller cần dung, gọi notification thông báo mỗi khi cần.

Thực hành : cài đặt toastr 2.1.2 bằng bower > nhúng vào index css và js(thư viện phải nhúng vào trước cái file tự viết) > vào thư mục service tạo file notificationService > gắn toastr vào module > nhúng service vào > thêm vào productCategoryListController inject notificationService > thay đổi view thêm số trang và căn phải

Bài 28: Thêm mới dữ liệu trong form sản phẩm

Các bước : tạo phương thức API thêm mới dữ liệu > tạo form thêm dữ liệu sử dụng bootstrap > valid dữ liệu đàu vào bằng AngularJS > tạo mới controller thêm mới > nhúng controller vào index > Binding sự kiện

Thực hành: Tạo mới phương thức Create trong controller ProductCategoryController > Tạo thêm EntityExtensions cho UpdateProductCategory > Tạo productCategoryAddView và ProductCategoryAddController > thêm state add cho productCategory.module > khai báo cho productcategoryAddController > thêm phương thức post cho apiService > tạo form cho view horizontal view > trên view list sử dụng ui-sref để chuyển sang trang thêm mới > Thêm button hủy, và submit cho addview > nhúng add controller vào view chính > binding dữ liệu cho nút select bằng cách khai báo hàm LoadParent trong productCategoryAddController > tạo phương thức getAllParent trong API ProductCategoryController > đặt mặt định cho các model Status trên view > tạo sự kiện submit AddProductCategory > viết thêm phương thức post cho apiService + thêm notificationService khi fail, thêm điều kiện kiểm tra báo lỗi 401

Bài 29: Cập nhật dữ liệu trong AngularJS

Các bước: Backend: thêm API cập nhật dữ liệu, Frontend: tạo form cập nhật dữ liệu và loading dữ liệu sẵn, tạo thêm phương thức cho apiService, tạo controller edit, validate dữ liệu đầu vào, binding sự kiện cập nhật, thông báo cho người dùng

Thực hành: tạo một api update sử dụng [httpput] vào ProductCategoryController, sử dụng region để gộp nhóm > tạo thêm một phương thức GetbyID trong API > thêm phương thứ.c Put trong apiService > tạo form ProductCategoryEditView và ProductCategoryEditController > thêm state cho Controller mới > khởi tạo controller tương tự add controller > thêm function loadproductCategoryDetail > lấy ID trên parameter sử dụng $stateParams > liên kết listview sang editvew, sử dụng ui-sref gọi url có parameter > nhúng controller vào view chính > thêm createdate = date time > tạo một commonService getSeoTitle > tiêm phương thức commonService vào Controller > nhúng commonService vào view chính > binding phương thức getSeoTitle vào view > add vào branch mới commit update form and auto makeSeoTitle.

Bài 30: Validate form trong AngularJS

* Lý thuyết: trong pristine trạng thái sạch, chưa từng sử dụng, dirty đã được sử dụng, touched đã được chạm vào một lần, valid đúng được gán mặc định, invalid sai nếu có required thì là mặc định.
* Để truy cập vào form: <formname>.<angular property>
* Để truy cập vào input: <formname>.<inputname>.<angularproperty>
* Một số điểm cần nhớ : thuộc tính novalidate cho form sẽ disable cơ chế validate mặc định của html, chúng ta sẽ dung của AngularJS, áp dụng directive ng-model cho các input để liên kết với model, sử dụng ng-minlength và ng-maxlength để giới hạn độ dài ngắn cho input, thuộc tính bắt buộc là required, nếu cần valid email chỉ cần cho type=”email” và nhiều thuộc tính tại web angular directive/input

Thực hành: thực hiện chỉnh sửa productCategoryAddView, xóa giá trị mặc định của field danh mục > chèn thêm thẻ span sử dụng ng-show=”frmAddProdctCategory.name.$invalid” > tạo custom.css nhúng vào view > sử dụng ng-disable cho button > sử dụng $error.maxlength để validate > áp dụng vào productEditView > ngoài ra để tốt hơn nên sử dụng validate trên model điều chỉnh model ProductCategoryViewModel

Bài 31: xóa dữ liệu với Confirm trong bootBox

Các bước thực hiện :

* Backend thêm phương thức xóa cho API
* Frontend cài đặt bootbox và ngBootbox trên bower, sử dụng confirm của bootbox, gọi phương thức xóa, thông báo thành công

Thực hành : cài đặt bootbox và ngBootBox bằng Bower > nhúng vào index view > inject ngBootbox vào productCategoryListController và common> viết thêm một phương thức delete vào productCategoryListController > thêm hàm del vào apiService > binding vào view > tạo webAPI HttpDelete

Bài 32: Xóa nhiều bản ghi trong AngularJS

Các bước thực hiện:

+ Backend: Thêm phương thức xóa nhiều bản ghi cho API

+ Frontend: Tạo checkbox cho mỗi dòng, sử dụng $watch để check sự thay đổi của các checkbox, submit các dòng dẫ check lên server để xóa

Thực hành: Tạo phương thức xóa nhiều API ProductCategoryController > thêm button deletemulti vào productCategoryListView , thêm cột cdm checkbox chọn tất cả > thêm service selectAll và deleteMultiple vào productCategoryListController sử dụng service Watch của angular.

Bài 33: nhúng ckeditor cho phần nội dung

Các bước thực hiện :

+ Tạo view productAddView.html và controller , nhúng vào index

+ Tạo route cho add productView trong product.module.js

+ Tạo nội dung cho form html

+ Dùng bower để nhúng CK Editor > nhúng CK editor

Thực hành : Tạo productAddView và productController copy từ productcategoryAdd > thêm view và controller mới vào products.module.js > nhúng vào view chính > tạo API productController,copy từ productCategoryController > tạo productService > thêm EntityExtensions.cs > thêm ckeditor và ng-ckeditor > nhúng vào index chính > include vào common > thêm vào view > thêm vào controller

Bài 34: nhúng CKFinder cho quản lý ảnh

Các bước thực hiện:

+ Download và nhúng CK Finder

+ Nhúng CK Finder

+ Cấu hình CK Finder trỏ đến thư mục muốn quản lý

+ Trỏ CK Editor vào CK Finder

Ngoài ra còn có thể sử dụng Roxyfileman

Thực hành: download ckfinder bản 2.6 chép vào thư mục assets lib, xóa một số simple cột > nhúng ck finder.js vào view chính > thay đổi file config của ckfinder vào thư mục điều chỉnh UploadedFiles > tạo một thư mục mới Libs bên ngoài sau đó add file dll Release CKFinder.dll và add inorsource và add references cho file này> điều chỉnh tạm chời check authentication là true, sau này kiểm tra đăng nhập sẽ chỉnh sửa lại > vào Addview them một nút chọn ảnh > viết sự kiện vào AddController ChooseImage > tích hợp vào ckeditor them vào config của ckeditor > chỉnh ignore cho các file trong thư mục UploadedFiles

Bài 35: Quản lý tag cho từng sản phẩm

Thực hành: Vào view productAddViewthêm trường Tags > thêm trường Tags vào Product.cs Model > add migration AddTagField và update database để cập nhật db > Cập nhật infrastructure, entityExtensions thêm trường Tags > Vào API và thêm phương thức vào Services gốc Add > Tạo Class StringHelper trong Common để chuyển đổi > tiếp tục triển khai phước thức vào ProductService, tiêm dependence injection > thêm lớp CommonConstants vào Common > frontEnd thêm phương thức vào productAddController > lấy phương thức GetTitleSeo từ productCategoryEdit > lưu ý them thuộc tính của nút chọn ảnh bằng Button > them vào productModule product\_edit > tạo mới productEditController và View > kiểm tra đã nhúng vào view chính controller mới hay chưa > lưu ý tiêm stateparam một thành phần của ui-router > kiểm tra và debug fix lỗi