**Using AutoMapper together with DependencyInjection:**

I altered an existing Microsoft sample project, to use DI, then to easily map the results on resolve, used AutoMapper. Did this in one of the original controllers, and left the remaining controllers untouched for comparison. The project was the ContosoUniversity project. The controller I altered, to work with DI and AutoMapper, was the CourseController. I omitted sending data source to the service via injector, so as to streamline this display of  object mapping for DI.

1. Added an ICourseService interface:

1. using System;
2. using System.Collections.Generic;
3. using ContosoUniversity.Models;
5. namespace ContosoUniversity.Interfaces
6. {
7. public interface ICourseService : IDisposable
8. {
9. List<Department> GetDepartments();
10. List<Course> GetCourses(int? selectedDepartment);
11. Course FindById(int? id);
12. void Create(Course course);
13. void Update(Course course);
14. void DeleteById(int id);
15. int UpdateCourseCredits(int? multiplier);
16. }
17. }

2. Added a CourseService, and moved data access from controller to the service. Did not add a datasource constructor argument (for the Service, but would in a more complete solution.

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using ContosoUniversity.DAL;

using ContosoUniversity.Interfaces;

using ContosoUniversity.Models;

namespace ContosoUniversity.Services

{

    public class CourseService : ICourseService

    {

        private readonly SchoolContext \_db = new SchoolContext();

        public List<Department> GetDepartments()

        {

            return \_db.Departments.OrderBy(q => q.Name).ToList();

        }

        public List<Course> GetCourses(int? selectedDepartment)

        {

            int departmentID = selectedDepartment.GetValueOrDefault();

            return \_db.Courses

                .Where(c => !selectedDepartment.HasValue || c.DepartmentID == departmentID)

                .OrderBy(d => d.CourseID)

                .Include(d => d.Department)

                .ToList();

        }

        public Course FindById(int? id)

        {

            return \_db.Courses.Find(id);

        }

        public void Create(Course course)

        {

            \_db.Courses.Add(course);

            \_db.SaveChanges();

        }

        public void Update(Course course)

        {

            Course dbRecord = FindById(course.CourseID);

            dbRecord.Credits = course.Credits;

            dbRecord.DepartmentID = course.DepartmentID;

            dbRecord.Title = course.Title;

            \_db.SaveChanges();

        }

        public void DeleteById(int id)

        {

            Course course = \_db.Courses.Find(id);

            if (course == null)

            {

                return;

            }

            \_db.Courses.Remove(course);

            \_db.SaveChanges();

        }

        public int UpdateCourseCredits(int? multiplier)

        {

            int recCnt = \_db.Database.ExecuteSqlCommand("UPDATE Course SET Credits = Credits \* {0}", multiplier);

            return recCnt;

        }

        public void Dispose()

        {

            \_db?.Dispose();

        }

    }

}

3. Added a ViewModel to align with the Course type:

using System.Collections.Generic;

using ContosoUniversity.Models;

namespace ContosoUniversity.ViewModels

{

    public class CourseViewModel

    {

        public int CourseID { get; set; }

        public string Title { get; set; }

        public int Credits { get; set; }

        public int DepartmentID { get; set; }

        public virtual Department Department { get; set; }

        public virtual ICollection<Enrollment> Enrollments { get; set; }

        public virtual ICollection<Instructor> Instructors { get; set; }

    }

}

4. Added a ControllerFactory, so that in app startup, container/children resolution would occur per current controller:

using System;

using System.Web.Mvc;

using System.Web.Routing;

using Microsoft.Practices.Unity;

namespace ContosoUniversity.Controllers

{

    public class ControllerFactory : DefaultControllerFactory

    {

        private readonly IUnityContainer \_container;

        public ControllerFactory(IUnityContainer container)

        {

            \_container = container;

        }

        protected override IController GetControllerInstance(RequestContext requestContext, Type controllerType)

        {

            IController controller = \_container.Resolve(controllerType) as IController ??

                                     base.GetControllerInstance(requestContext, controllerType);

            if (controller == null)

            {

                requestContext.HttpContext.Response.StatusCode = 404;

                return null;

            }

            return controller;

        }

    }

}

5. Installed the AutoMapper nuget package

6. Added a class containing a ref to AutoMapper, and an automapper init method where this initial and any ensuing, type mappings, could be added. This method would have many more CreateMap statements if dependency injection were to be setup/leveraged for each controller, and would probably include some projection (custom transform config between source and destination).

using AutoMapper;

using ContosoUniversity.Models;

using ContosoUniversity.ViewModels;

namespace ContosoUniversity

{

    public class AutoMapperConfig

    {

        public static void Initialize()

        {

            Mapper.Initialize((config) =>

            {

                config.CreateMap<Course, CourseViewModel>().ReverseMap();

            });

        }

    }

}

7. Added a call to the AutoMapperConfig.Initialize method, in application start up (global.asax in this case of MVC, would have placed in program.cs when using .NET core, and so on):

protected void Application\_Start()

{

    AreaRegistration.RegisterAllAreas();

    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

    RouteConfig.RegisterRoutes(RouteTable.Routes);

    BundleConfig.RegisterBundles(BundleTable.Bundles);

    AutoMapperConfig.Initialize();

8. Setup container with registration of types in app startup:

AutoMapperConfig.Initialize();

            Container = new UnityContainer();

            Container.RegisterType<ICourseService, CourseService>(new InjectionFactory(c => new CourseService()));

            Container.RegisterType<CourseController>();

            Container.RegisterType<CourseViewModel>();

9. Added call to custom controller factory so resolve would occur for any registered controller and any required object graph items also registered (inside app startup, following type registrations):

ControllerBuilder.Current.SetControllerFactory(new ControllerFactory(Container));

10. Added private member of type ICourseService to CourseController. Added a second contructor to the Controller, accepting an argument of type ICourseService so that the service could be set at runtime, when Resolve occurs.

public class CourseController : Controller

    {

        private ICourseService \_courseService;

        public CourseController(ICourseService courseService)

        {

            \_courseService = courseService;

        }

11. Added ref to AutoMapper in controller. re-pointed all data access code from controller to use the service, and used AutoMapper to align service results with expected ViewModels:

using System.Collections.Generic;

using System.Net;

using System.Web.Mvc;

using ContosoUniversity.Models;

using System.Data.Entity.Infrastructure;

using AutoMapper;

using ContosoUniversity.Interfaces;

using ContosoUniversity.ViewModels;

namespace ContosoUniversity.Controllers

{

    public class CourseController : Controller

    {

        private ICourseService \_courseService;

        public CourseController(ICourseService courseService)

        {

            \_courseService = courseService;

        }

        // GET: Course

        public ActionResult Index(int? selectedDepartment)

        {

            List<Department> departments = \_courseService.GetDepartments();

            ViewBag.SelectedDepartment = new SelectList(departments, "DepartmentID", "Name", selectedDepartment);

            List<Course> courses = \_courseService.GetCourses(selectedDepartment);

            List<CourseViewModel> courseViewModels = Mapper.Map<List<Course>, List<CourseViewModel>>(courses);

            return View(courseViewModels);

        }

        // GET: Course/Details/5

        public ActionResult Details(int? id)

        {

            if (id == null)

            {

                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

            }

            Course course = \_courseService.FindById(id);

            if (course == null)

            {

                return HttpNotFound();

            }

            CourseViewModel courseViewModel = Mapper.Map<Course, CourseViewModel>(course);

            return View(courseViewModel);

        }

        public ActionResult Create()

        {

            PopulateDepartmentsDropDownList();

            return View();

        }

        [HttpPost]

        [ValidateAntiForgeryToken]

        public ActionResult Create([Bind(Include = "CourseID,Title,Credits,DepartmentID")]Course course)

        {

            try

            {

                if (ModelState.IsValid)

                {

                    \_courseService.Create(course);

                    return RedirectToAction("Index");

                }

            }

            catch (RetryLimitExceededException /\* dex \*/)

            {

                //Log the error (uncomment dex variable name and add a line here to write a log.)

                ModelState.AddModelError("", @"Unable to save changes. Try again, and if the problem persists, see your system administrator.");

            }

            PopulateDepartmentsDropDownList(course.DepartmentID);

            CourseViewModel courseViewModel = Mapper.Map<Course, CourseViewModel>(course);

            return View(courseViewModel);

        }

        public ActionResult Edit(int? id)

        {

            if (id == null)

            {

                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

            }

            Course course = \_courseService.FindById(id);

            if (course == null)

            {

                return HttpNotFound();

            }

            PopulateDepartmentsDropDownList(course.DepartmentID);

            CourseViewModel courseViewModel = Mapper.Map<Course, CourseViewModel>(course);

            return View(courseViewModel);

        }

        [HttpPost, ActionName("Edit")]

        [ValidateAntiForgeryToken]

        public ActionResult EditPost(int? id)

        {

            if (id == null)

            {

                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

            }

            Course courseToUpdate = \_courseService.FindById(id);

            if (TryUpdateModel(courseToUpdate, "", new[] { "Title", "Credits", "DepartmentID" }))

            {

                try

                {

                    \_courseService.Update(courseToUpdate);

                    return RedirectToAction("Index");

                }

                catch (RetryLimitExceededException /\* dex \*/)

                {

                    //Log the error (uncomment dex variable name and add a line here to write a log.

                    ModelState.AddModelError("", @"Unable to save changes. Try again, and if the problem persists, see your system administrator.");

                }

            }

            PopulateDepartmentsDropDownList(courseToUpdate.DepartmentID);

            CourseViewModel courseViewModel = Mapper.Map<Course, CourseViewModel>(courseToUpdate);

            return View(courseViewModel);

        }

        private void PopulateDepartmentsDropDownList(object selectedDepartment = null)

        {

            List<Department> departments = \_courseService.GetDepartments();

            ViewBag.DepartmentID = new SelectList(departments, @"DepartmentID", @"Name", selectedDepartment);

        }

        // GET: Course/Delete/5

        public ActionResult Delete(int? id)

        {

            if (id == null)

            {

                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

            }

            Course course = \_courseService.FindById(id);

            if (course == null)

            {

                return HttpNotFound();

            }

            CourseViewModel courseViewModel = Mapper.Map<Course, CourseViewModel>(course);

            return View(courseViewModel);

        }

        // POST: Course/Delete/5

        [HttpPost, ActionName("Delete")]

        [ValidateAntiForgeryToken]

        public ActionResult DeleteConfirmed(int id)

        {

            \_courseService.DeleteById(id);

            return RedirectToAction("Index");

        }

        public ActionResult UpdateCourseCredits()

        {

            return View();

        }

        [HttpPost]

        public ActionResult UpdateCourseCredits(int? multiplier)

        {

            if (multiplier != null)

            {

                ViewBag.RowsAffected = \_courseService.UpdateCourseCredits(multiplier);

            }

            return View();

        }

        protected override void Dispose(bool disposing)

        {

            if (disposing)

            {

                \_courseService.Dispose();

            }

            base.Dispose(disposing);

        }

    }

}

12. Changed Each Course view having return data, to use CourseViewModel rather than the Course type.