HOW TO ADD VIEW MODEL AND SERVICE LAYER USING EXSTENSIONS [Answered]

2 replies

Last post Jan 11, 2019 11:13 AM by DA924

HOW TO ADD VIEW MODEL AND SERVICE LAYER USING EXSTENSIONS

Jan 10, 2019 11:04 PM | phmaspnet

hI,

I have a application where I need to display the products using view model and service layer but I dont know how to connect both

```
 public class Product
    {
        public int ID { get; set; }
        public string Prod_SKU { get; set; }
        public string Prod_Name { get; set; }
        public double Price { get; set; }
    }



public class ProductService:IProductService
    {
        private   ProductContext _context;

        public ProductService(ProductContext context)
        {
            _context = context;
        }
```

```
        public  IEnumerable<Product> GetAll()
        {
            return  _context.Products.ToList();

        }
}

product view model

  public class ProductViewModel
    {

        public int ProductId { get; set; }
        public string ProductSku { get; set; }
        public string ProductName { get; set; }
        public double ProductPrice { get; set; }
        public List<Product> Products { get; set; }

    }
```

now in action I have written like this

```
  private IProductService _productService;
        private ProductViewModel _productViewModel;
        public ProductController(IProductService productService,ProductViewModel productViewModel)
        {
            _productService = productService;
            _productViewModel = productViewModel;
        }
        // GET: Product
        public ActionResult Index()
        {
           _productService.GetAll();
            return View(_productService.GetAll());
        }
}
```

now I want to connect all my viewmodel with service class but dono how can u give some help on using extension methods in thsi example

Re: HOW TO ADD VIEW MODEL AND SERVICE LAYER USING EXSTENSIONS

Jan 11, 2019 08:54 AM | Nan Yu

Hi phmaspnet

I'm confused why you have List<Product> Products in view model , if that's by design , you should directly set _productViewModel .Products =_productService.GetAll() .

If you want to map the the product properties to view model's related properties . You can write your own extension class to map the properties of the two entities . Or you can use **AutoMapper (http://automapper.org/)** , an open source and free mapping library which allows to map one type into another, based on conventions .

Best Regards,

Nan Yu

Re: HOW TO ADD VIEW MODEL AND SERVICE LAYER USING EXSTENSIONS

Jan 11, 2019 11:13 AM | DA924

In my option, the service layered should be unaware of the viewmodel. You can consider the WebAPI interface, a folder in the MVC project, to the WebAPi as a service layer that sits between the MVC presentation layered and the WebAPI service, which is using the DTO pattern.

https://www.codeproject.com/Articles/1050468/Data-Transfer-Object-Design-Pattern-in-Csharp (https://www.codeproject.com/Articles/1050468/Data-Transfer-Object-Design-Pattern-in-Csharp)

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/understanding-models-views-and-controllers-cs (https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/understanding-models-views-and-controllers-cs)

<copied>

An MVC model contains all of your application logic that is not contained in a view or a controller. The model should contain all of your application business logic, validation logic, and database access logic. For example, if you are using the Microsoft Entity Framework to access your database, then you would create your Entity

Framework classes (your .edmx file) in the Models folder.

A view should contain only logic related to generating the user interface. A controller should only contain the bare minimum of logic required to return the right view or redirect the user to another action (flow control). Everything else should be contained in the model.

In general, you should strive for fat models and skinny controllers. Your controller methods should contain only a few lines of code. If a controller action gets too fat, then you should consider moving the logic out to a new class in the Models folder

<end>

The service is being called by a model object in the models folder. And the controller is using the object in the models folder for CRUD and other business logic that is working with the VM and passing the viewmodel to the controller.

```
@using ProgMgmntCore2UserIdentity.Models
@model ProjectViewModels.Project

<!DOCTYPE html>

<style type="text/css">
    .editor-field > label {
        float: left;
        width: 150px;
    }
    .txtbox {
        font-family: Arial, Helvetica, sans-serif;
        font-size: 12px;
        background: white;
        color: black;
        cursor: text;
        border-bottom: 1px solid #104A7B;
        border-right: 1px solid #104A7B;
        border-left: 1px solid #104A7B;
        border-top: 1px solid #104A7B;
        padding-top: 10px;
    }
</style>
<html>
<head>
    <title>Edit</title>
</head>
```

```
<body>
    <h1>Project</h1>

    @using (Html.BeginForm())
    {

        @Html.ValidationSummary(false, "", new { @class = "text-danger" })

    <fieldset>

        <legend>Edit</legend>

        @Html.HiddenFor(model => model.ProjectId)

        <div class="editor-field">
            @Html.Label("Project Name:")
            @Html.TextBoxFor(model => model.ProjectName, new { @Cssclass = "txtbox" })
            @Html.ValidationMessageFor(model => model.ProjectName)
        </div>
        <br />
        <div class="editor-field">
            @Html.Label("Client Name:")
            @Html.TextBoxFor(model => model.ClientName, new { @Cssclass = "txtbox" })
            @Html.ValidationMessageFor(model => model.ClientName)
        </div>
        <br />
        <div class="editor-field">
            @Html.Label("Technology:")
            @Html.TextBoxFor(model => model.Technology, new { @Cssclass = "txtbox" })
            @Html.ValidationMessageFor(model => model.Technology)
        </div>
        <br />
        <div class="editor-field">
            @Html.Label("Start Date:")
            @Html.TextBoxFor(model => model.StartDate, new { @Cssclass = "txtbox" })
            @Html.ValidationMessageFor(model => model.StartDate)
        </div>
        <br />
        <div class="editor-field">
            @Html.Label("End Date:")
```

```
            @Html.TextBoxFor(model => model.EndDate, new { @Cssclass = "txtbox" })
            @Html.ValidationMessageFor(model => model.EndDate)
        </div>
        <br />
        <div class="editor-field">
            @Html.Label("Cost:")
            @Html.TextBoxFor(model => model.Cost, new { @Cssclass = "txtbox" })
            @Html.ValidationMessageFor(model => model.Cost)
        </div>
        <br />
        <div class="editor-field">
            @Html.Label("Project Type:")
            @Html.DropDownListFor(model => model.ProjectType, Model.ProjectTypes)
        </div>
        <br />
        <p>
            <input type="submit" name="submit" value="Save" />
            <input type="submit" name="submit" value="Cancel" />
        </p>

    </fieldset>
    }

</body>
</html>
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace ProgMgmntCore2UserIdentity.Models
{
    public class ProjectViewModels
```

```
    {
        public class Project
        {
            public int ProjectId { get; set; }

            [Required(ErrorMessage = "Client Name is required")]
            [StringLength(50)]
            public string ClientName { get; set; }

            [Required(ErrorMessage = "Project Name is required")]
            [StringLength(50)]
            public string ProjectName { get; set; }

            [Required(ErrorMessage = "Technology is required")]
            [StringLength(50)]
            public string Technology { get; set; }

            [Required(ErrorMessage = "Project Type is required")]
            public string ProjectType { get; set; }

            [Required(ErrorMessage = "Start Date is required")]
            [DisplayFormat(ApplyFormatInEditMode = true, DataFormatString = "{0:MM-dd-yyyy}")]
            public DateTime? StartDate { get; set; }

            [Required(ErrorMessage = "End Date is required")]
            [DisplayFormat(ApplyFormatInEditMode = true, DataFormatString = "{0:MM-dd-yyyy}")]
            public DateTime? EndDate { get; set; }

            [Required(ErrorMessage = "Cost is required")]
            public decimal? Cost { get; set; }

            public List<SelectListItem> ProjectTypes { get; set; }
        }

        public List<Project> Projects { get; set; }

    }
}
```

```
namespace ProgMgmntCore2UserIdentity.Models
{
    public interface IProjectModel
    {
        ProjectViewModels GetProjectsByUserId(string userid);
        ProjectViewModels.Project GetProjectById(int id);
        ProjectViewModels.Project Create();
        void Create(ProjectViewModels.Project project, string userid);
        ProjectViewModels.Project Edit(int id);
        void Edit(ProjectViewModels.Project project, string userid);
        void Delete(int id);
        ProjectViewModels.Project PopulateSelectedList(ProjectViewModels.Project project);
    }
}

=====================================================================

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Entities;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.Extensions.Caching.Memory;
using ProgMgmntCore2UserIdentity.WebApi;

namespace ProgMgmntCore2UserIdentity.Models
{
    public class ProjectModel : IProjectModel
    {
        private readonly IMemoryCache _memoryCache;
        private readonly IWebApi _webApi;

        public ProjectModel(IWebApi webApi, IMemoryCache memoryCache)
        {
            _memoryCache = memoryCache;
            _webApi = webApi;
        }

        public ProjectViewModels GetProjectsByUserId(string userid)
```

```
        {
            var vm = new ProjectViewModels {Projects = new List<ProjectViewModels.Project>()};

            var dtos = _webApi.GetProjsByUserIdApi(userid).ToList();

            vm.Projects.AddRange(dtos.Select(dto => new ProjectViewModels.Project()
            {
                ProjectId = dto.ProjectId,
                ClientName = dto.ClientName,
                ProjectName = dto.ProjectName,
                Technology = dto.Technology,
                ProjectType = dto.ProjectType,
                StartDate = dto.StartDate,
                EndDate = dto.EndDate,
                Cost = dto.Cost
            }).ToList());

            return vm;
        }

        public ProjectViewModels.Project GetProjectById(int id)
        {
            var responseDto = _webApi.GetProjByIdApi(id);

            var project = new ProjectViewModels.Project
            {
                ProjectId = responseDto.ProjectId,
                ClientName = responseDto.ClientName,
                ProjectName = responseDto.ProjectName,
                Technology = responseDto.Technology,
                ProjectType = responseDto.ProjectType,
                StartDate = responseDto.StartDate,
                EndDate = responseDto.EndDate,
                Cost = responseDto.Cost
            };

            return project;
        }

        public ProjectViewModels.Project Create()
        {
```

```
            var project = new ProjectViewModels.Project();
            return PopulateSelectedList(project);
        }

        public void Create(ProjectViewModels.Project project, string userid)
        {
            var dto = new DtoProject
            {
                ProjectId = project.ProjectId,
                ClientName = project.ClientName,
                ProjectName = project.ProjectName,
                ProjectType = project.ProjectType,
                Technology = project.Technology,
                UserId = userid,
                StartDate = (DateTime) project.StartDate,
                EndDate = (DateTime) project.EndDate,
                Cost = (decimal) project.Cost
            };

            _webApi.CreateProjectApi(dto);
        }

        public ProjectViewModels.Project Edit(int id)
        {
            var responseDto = _webApi.GetProjByIdApi(id);

            var project = new ProjectViewModels.Project
            {
                ProjectId = responseDto.ProjectId,
                ClientName = responseDto.ClientName,
                ProjectName = responseDto.ProjectName,
                Technology = responseDto.Technology,
                ProjectType = responseDto.ProjectType,
                StartDate = responseDto.StartDate,
                EndDate = responseDto.EndDate,
                Cost = responseDto.Cost
            };

            project = PopulateSelectedList(project);

            return project;
```

```
        }

        public void Edit(ProjectViewModels.Project project, string userid)
        {
            var dto = new DtoProject
            {
                ProjectId = project.ProjectId,
                ClientName = project.ClientName,
                ProjectName = project.ProjectName,
                ProjectType = project.ProjectType,
                Technology = project.Technology,
                UserId = userid,
                StartDate = (DateTime) project.StartDate,
                EndDate = (DateTime) project.EndDate,
                Cost = (decimal) project.Cost
            };

            _webApi.UpdateProjectApi(dto);
        }

        public void Delete(int id)
        {
            _webApi.DeleteProjectApi(new DtoId{Id = id});
        }

        public ProjectViewModels.Project PopulateSelectedList(ProjectViewModels.Project project)
        {
            bool isExist = _memoryCache.TryGetValue("DtoCache", out DtoCache dtocache);

            if (!isExist)
            {
                dtocache = _webApi.GetCacheApi();

                var cacheEntryOptions = new MemoryCacheEntryOptions()
                    .SetSlidingExpiration(TimeSpan.FromSeconds(30));

                _memoryCache.Set("DtoCache", dtocache, cacheEntryOptions);
            }

            project.ProjectTypes = new List<SelectListItem>();
```

```
            foreach (var pt in dtocache.ProjectTypes)
            {
                var sli = new SelectListItem {Value = pt.Value, Text = pt.Text};
                project.ProjectTypes.Add(sli);
            }

            var selected = (from a in project.ProjectTypes.Where(a => a.Value == project.ProjectType) select a)
                .SingleOrDefault();

            if (selected != null)
                selected.Selected = true;

            return project;
        }
    }
}
```

```
using System.Collections.Generic;
using System.Threading.Tasks;
using Entities;

namespace ProgMgmntCore2UserIdentity.WebApi
{
    public interface IWebApi
    {
        List<DtoProject> GetProjsByUserIdApi(string userid);
        DtoProject GetProjByIdApi(int id);
        void CreateProjectApi(DtoProject dto);
        void UpdateProjectApi(DtoProject dto);
        void DeleteProjectApi(DtoId dto);
     }
}

============================================================
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;
```

```
using Entities;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

namespace ProgMgmntCore2UserIdentity.WebApi
{
    public class WebApi : IWebApi
    {
        #region ProjectApi

        public List<DtoProject> GetProjsByUserIdApi(string userid)
        {
            var dtoprojects = new List<DtoProject>();

            using (var client = new HttpClient())
            {
                var uri = new Uri("http://progmgmntcore2api.com/api/project/GetProjsByUserId?userid=" + userid);

                var response = client.GetAsync(uri).Result;

                if (!response.IsSuccessStatusCode)
                    throw new Exception(response.ToString());

                var responseContent = response.Content;
                var responseString = responseContent.ReadAsStringAsync().Result;

                dynamic projects = JArray.Parse(responseString) as JArray;

                foreach (var obj in projects)
                {
                    DtoProject dto = obj.ToObject<DtoProject>();

                    dtoprojects.Add(dto);
                }
            }

            return dtoprojects;
        }

        public DtoProject GetProjByIdApi(int id)
        {
```

```
            DtoProject dto;

            using (var client = new HttpClient())
            {
                var uri = new Uri("http://progmgmntcore2api.com/api/project/GetProjById?id=" + id);
                HttpResponseMessage getResponseMessage = client.GetAsync(uri).Result;

                if (!getResponseMessage.IsSuccessStatusCode)
                    throw new Exception(getResponseMessage.ToString());

                var responsemessage = getResponseMessage.Content.ReadAsStringAsync().Result;

                dynamic project = JsonConvert.DeserializeObject(responsemessage);

                dto = project.ToObject<DtoProject>();
            }

            return dto;
        }

        public void CreateProjectApi(DtoProject dto)
        {
            using (var client = new HttpClient { BaseAddress = new Uri("http://progmgmntcore2api.com") })
            {
                string serailizeddto = JsonConvert.SerializeObject(dto);

                var inputMessage = new HttpRequestMessage
                {
                    Content = new StringContent(serailizeddto, Encoding.UTF8, "application/json")
                };

                inputMessage.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

                HttpResponseMessage message =
                    client.PostAsync("api/project/CreateProject", inputMessage.Content).Result;

                if (!message.IsSuccessStatusCode)
                    throw new Exception(message.ToString());
            }
        }
```

```
        public void UpdateProjectApi(DtoProject dto)
        {
            using (var client = new HttpClient { BaseAddress = new Uri("http://progmgmntcore2api.com") })
            {
                string serailizeddto = JsonConvert.SerializeObject(dto);

                var inputMessage = new HttpRequestMessage
                {
                    Content = new StringContent(serailizeddto, Encoding.UTF8, "application/json")
                };

                inputMessage.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

                HttpResponseMessage message =
                    client.PostAsync("api/project/UpdateProject", inputMessage.Content).Result;

                if (!message.IsSuccessStatusCode)
                    throw new Exception(message.ToString());
            }
        }

        public void DeleteProjectApi(DtoId dto)
        {
            using (var client = new HttpClient { BaseAddress = new Uri("http://progmgmntcore2api.com") })
            {
                string serailizeddto = JsonConvert.SerializeObject(dto);

                var inputMessage = new HttpRequestMessage
                {
                    Content = new StringContent(serailizeddto, Encoding.UTF8, "application/json")
                };

                inputMessage.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

                HttpResponseMessage message =
                    client.PostAsync("api/project/DeleteProject", inputMessage.Content).Result;

                if (!message.IsSuccessStatusCode)
                    throw new Exception(message.ToString());
            }
        }
```

```
        #endregion

 }
```

```
using System;
using System.Linq;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using ProgMgmntCore2UserIdentity.Models;

namespace ProgMgmntCore2UserIdentity.Controllers
{
    public class ProjectController : Controller
    {
        private readonly IProjectModel _projectModel;
        private readonly IModelHelper _modelHelper;

        public ProjectController(IProjectModel projectModel,   IModelHelper modelHelper)
        {
            _projectModel = projectModel;
            _modelHelper = modelHelper;
        }

        // GET: Project
        [Authorize]
        public ActionResult Index()
        {
            return View(_projectModel.GetProjectsByUserId(User.Identity.Name));
        }

        [Authorize]
        public ActionResult Details(int id = 0)
        {
            return id == 0 ? null : View(_projectModel.Edit(id));
        }

        [Authorize]
```

```
        public ActionResult Create()
        {
            return View(_projectModel.Create());
        }

        [Authorize]
        [HttpPost]
        public ActionResult Create(ProjectViewModels.Project project, string submit)
        {
            if (submit == "Cancel") return RedirectToAction("Index");

            ValidateddlProjectTypes();

            project.ProjectType = (Request.Form["ddlProjectTypes"]);

            if (ModelState.IsValid && _modelHelper.IsEndDateLessThanStartDate(project, "Project"))
                ModelState.AddModelError(string.Empty, "End Date cannot be less than Start Date.");

            if (!ModelState.IsValid) return View(_projectModel.PopulateSelectedList(project));

            _projectModel.Create(project, User.Identity.Name);
            return RedirectToAction("Index");
        }

        [Authorize]
        public ActionResult Edit(int id = 0)
        {
            return id == 0 ? null : View(_projectModel.Edit(id));
        }

        [Authorize]
        [HttpPost]
        public ActionResult Edit(ProjectViewModels.Project project, string submit)
        {
            if (submit == "Cancel") return RedirectToAction("Index");

            if (ModelState.IsValid && _modelHelper.IsEndDateLessThanStartDate(project, "Project"))
                ModelState.AddModelError(String.Empty, "End Date cannot be less than Start Date.");

            if (!ModelState.IsValid) return View(_projectModel.PopulateSelectedList(project));
```

```
            var theproject = new ProjectViewModels.Project();

            theproject = project;

            theproject.ProjectType = Request.Form["ProjectType"];

            _projectModel.Edit(theproject, User.Identity.Name);
            return RedirectToAction("Index");
        }

        public ActionResult Delete(int id = 0)
        {
            if (id > 0) _projectModel.Delete(id);

            return RedirectToAction("Index");
        }

        public ActionResult Cancel()
        {
            return RedirectToAction("Index", "Home");
        }

        public ActionResult UploadFile(int id)
        {
            return RedirectToAction("Index", "Upload", new { id = id, type = "PM" });
        }

        private void ValidateddlProjectTypes()
        {
            if (Request.Form["ddlProjectTypes"] == string.Empty)
              return;

            foreach (var key in ModelState.Keys.ToList().Where(key => ModelState.ContainsKey(key)))
            {
                if (key != "ProjectType") continue;
                ModelState[key].Errors.Clear();
                ModelState[key].ValidationState = ModelValidationState.Valid;
            }
        }
    }
}
```