ASP.NET MVC

MVC Architectural Pattern

1. Model, View, Controller
2. Architectural Pattern for implementing User Interface
3. Model
   1. Represents application data and behaviour in terms of its problem domain and is independent if UI
   2. Consists of classes which contains properties and methods that represents application state and rule
4. View
   1. HTML Markup that we display to the user
5. Controller
   1. Responsible for handling an HTTP Request
   2. Suppose our application is hosted on vidly.com, when a request is sent on [http://vidly.com/movie](http://vidly.com/movies)s, a Controller will be selected to handle this request. This controller will get all the movies from database and put them in a view and return view to client
6. Router
   1. Responsible for selecting right controller for handling a request
   2. Router knows that request <http://vidly.com/movies> should be handled by MoviesController
   3. To be precise, request is handled by one of the method of this class
   4. Methods of controller is defined as Actions
   5. Saying accurately, action in controller is responsible for handling user request

Rule for determining Route

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

First part of URL is considered as Controller

Second part of URL is considered as Action

Third part of URL is optional and passed as parameter to Action

/movies/popular -> MoviesController Popular()

/movies/edit/1 -> MoviesController Edit (int id)

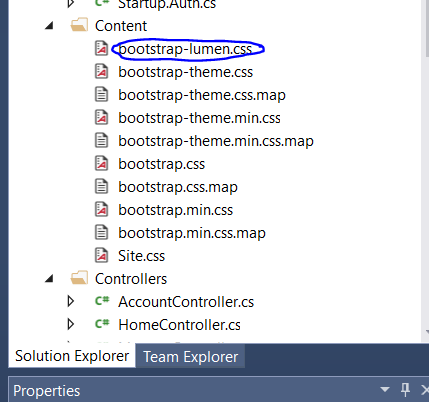
/movies -> MoviesController Index()

Project Structures

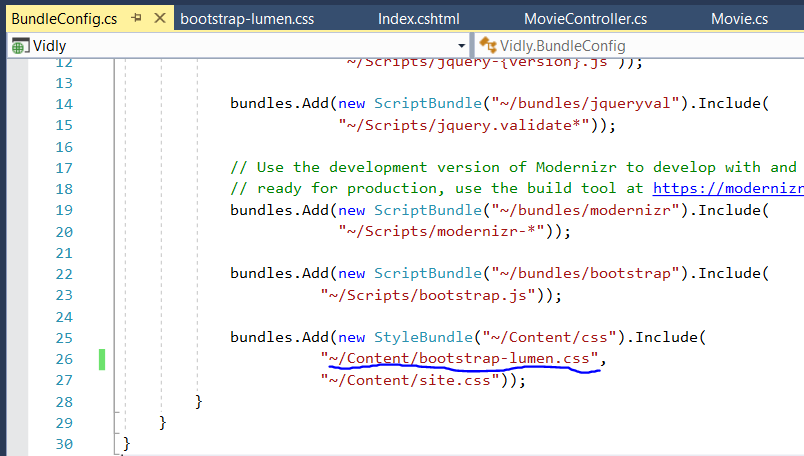
1. App\_Data – Database file will be stored
2. App\_Start – contains classes that are called when application gets started
3. Context – CSS, Images and Client side assets are stored here
4. Controllers
   1. Default project has three controllers AccountController, HomeController and ManageController
   2. AccountController contains actions for sign up, log in, log out
   3. HomeController represents home page
   4. ManageController contains number for actions for handling requests around User Profile like changing Password, enabling Two Password Authentication, Using Social Logins and so on
5. Fonts
6. Models
   1. All domain classes are present here
7. Scripts
   1. Stores JavaScript files
8. Views
   1. Contains folders named after controller in our application
   2. There is one more folder called Shared which includes views that can be used across different controllers
9. Favicon.ico
   1. Icon that is displayed in our application
10. Global.asax
    1. A class that provides hooks for various events in the applications lifecycle.
11. Packages.config
    1. Used by NuGet package manager
12. Startup.cs
    1. For starting the application
13. Web.config
    1. XML that includes configuration for application
    2. Mostly used things are
       1. connectionString which is used for establishing database connection
       2. appSetting where configuration settings for our application is defined

Changing default theme

1. Get required bootstrap.css file
2. Copy it inside the folder named Contents
3. Say this file is renamed to bootstrap-lumen.css

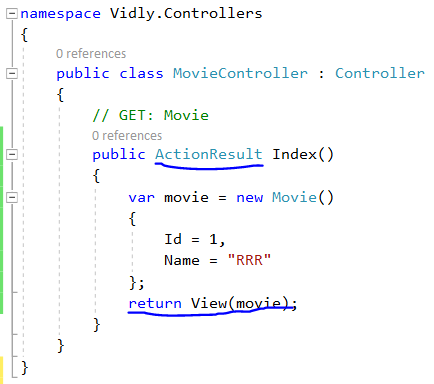


1. In App\_Start -> Bundle\_Config.cs, rename bootstrap.css to bootstrap-lumen.css



Action results

1. Action Results are the output of our actions
2. ActionResult is base class for all actions in ASP.NET MVC
3. Here, ActionResult is type and View() is helper method



1. Action Results

|  |  |
| --- | --- |
| **Type** | **Helper Method** |
| ViewResult | View() |
| PartialViewResult | PartialView() |
| ContentResult | Content() |
| RedirectResult | Redirect() |
| RedirectToRouteResult | RedirectToAction() |
| JsonResult | Json() |
| FileResult | File() |
| HttpNotFoundResult | HttpNotFound() |
| EmptyResult |  |

Content()

namespace Vidly.Controllers

{

public class MovieController : Controller

{

// GET: Movie

public ContentResult Index()

{

var movie = new Movie()

{

Id = 1,

Name = "RRR"

};

return Content(movie.Name);

}

}

}

HttpNotFound()

namespace Vidly.Controllers

{

public class MovieController : Controller

{

// GET: Movie

public HttpNotFoundResult Index()

{

var movie = new Movie()

{

Id = 1,

Name = "RRR"

};

return HttpNotFound();

}

}

}

EmptyResult()

namespace Vidly.Controllers

{

public class MovieController : Controller

{

// GET: Movie

public ActionResult Index()

{

var movie = new Movie()

{

Id = 1,

Name = "RRR"

};

return new EmptyResult();

}

}

}

RedirectToAction()

namespace Vidly.Controllers

{

public class MovieController : Controller

{

// GET: Movie

public RedirectToRouteResult Index()

{

var movie = new Movie()

{

Id = 1,

Name = "RRR"

};

return RedirectToAction("Index", "Home", new { Id=1,Name="test"});

}

}

}

RedirectToAction("Index", "Home", new { Id=1,Name="test"})

1st -> Action

2nd -> Controller

3rd -> Anonymous object that contains arguments to be passed to action. These arguments are sent as part of query string



Action Parameters

1. Action Parameters are inputs to our actions.
2. Source of Action Parameters
   1. In the URL - /movies/edit/1
   2. In query string - /movies/edit/?id=1
   3. In the form data – id=1

public ActionResult Details(int id, string Name)

{

return Content(String.Format("id = {0} Name = {1}",id,Name));

}

1. By default, any string parameter is nullable parameter
2. Id does not accept null value
3. To name id as nullable integer -> int? id

public ActionResult Details(int? id, string Name)

{

return Content(String.Format("id = {0} Name = {1}",id,Name));

}

Convention based Routing

1. Default Route {controller}/{action}/{id}
2. To define custom route

RouteConfig.cs

routes.MapRoute(

"ReleaseDateAndMonth",

"movie/released/{year}/{month}",

new { controller = "Movie", action = "ByReleaseDateAndMonth"},

new { year="\\d{4}",month="\\d{2}"}

);

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

routes.MapRoute(

"ReleaseDateAndMonth", -> Unique name

"movie/released/{year}/{month}", -> URL

new { controller= "Movie", action = "ByReleaseDateAndMonth"}, -> anonymous object defining action

new { year="\\d{4}",month="\\d{2}"} -> Required if some validations are to be performed to parameters

);

1. Then define corresponding action for handling this request

public ActionResult ByReleaseDateAndMonth(int year, int month) {

return Content(String.Format("year = {0} month = {1}", year, month));

}

Attribute Routing

1. Instead of adding route into RouteConfig.cs, it can be added as attribute into corresponding action
2. For this attribute routing should be enabled in RouteConfig.cs by adding following line

routes.MapMvcAttributeRoutes();

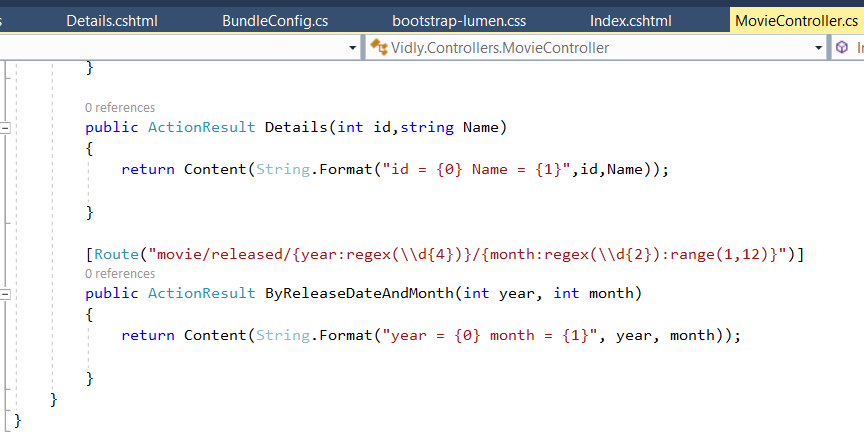


1. Add route as an attribute into corresponding action

[Route("movie/released/{year}/{month}")]

If some validations are to be performed

[Route("movie/released/{year:regex(\\d{4})}/{month:regex(\\d{2}):range(1,12)}")]



Additional constraints that can be used

1. min
2. max
3. minlength
4. maxlength
5. int
6. float
7. guid

View Model

1. Model build specially for a view
2. It includes any data and roles specifically for that view
3. Suppose, we want view where both Movie and Customer data needs to be displayed in a view.
   1. For this, create Customer and Movie class inside Models folder.
   2. Create a folder named ViewModels
   3. Create a class named MovieCustomerViewModel inside this folder
   4. This will contain movie and customer object

public class MovieCustomerViewModel

{

public Movie movie { get; set; }

public Customer customer { get; set; }

}

* 1. In view reference this ViewModel

@model Vidly.ViewModels.MovieCustomerViewModel

@{

ViewBag.Title = "View";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>@Model.customer.Name</h2>

<h2>@Model.movie.Name</h2>

Razor Syntax

1. C# code is prefixed by @
2. For multi-line comments

@\*

\*@

Partial View

1. Create a new .cshtml file inside Shared folder of View
2. Check Partial View option and name it as \_NavBar.cshtml
3. This file will contain html file for navigation bar
4. In \_Layout.cshtml file, add following line of code

@Html.Partial("\_NavBar")

Exercise1

To add hyperlink

@Html.ActionLink(item.Name, "Details",new { Id=item.Id})

item.Name -> text for hyperlink

Details -> Name of Action

New { Id=item.Id } -> Parameters to action

public ActionResult Index()

{

var customer = GetCustomer();

return View(customer);

}

public ActionResult Details(int id)

{

var customer = GetCustomer().SingleOrDefault(c => c.Id == id);

return View(customer);

}

public List<Customer> GetCustomer()

{

return new List<Customer> {

new Customer{Id=1, Name="Customer 1" },

new Customer{Id=2, Name="Customer 2" }

};

}

Entity Framework

Tool used to access database

ORM that maps data in database to objects in our application

It provides class called DbContext which is a gateway to database.

DbContext has one or more DbSet which represent tables in our database.

LINQ is used to create these DbSet and Entity Framework will translate this LINQ queries to SQL Queries at run time.

Any changes made in DbSet is tracked and corresponding SQL Queries are executed in database.

Two workflows to use Entity Framework

Database First

Code First

DbFirstWorkflow

Design Entity Framework first and have Entity Framework generate corresponding domain classes.

CodeFirst Workflow

Start with Domain Classes and have Entity Framework generate corresponding table for us.

Migrations

1) In the Models folder add a new class and call it whatever you want (I named mine ApplicationContext)

2) Open the class and then you need to add the following:

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Web;

namespace Vidly.Models

{

public class ApplicationContext: DbContext

{

public ApplicationContext()

{

}

public DbSet<Customer> Customers { get; set; }

public DbSet<Movie> Movies { get; set; }

}

}

3) In Packet Manager Console,

PM> Enable-Migrations -ContextTypeName Vidly.Models.ApplicationContext

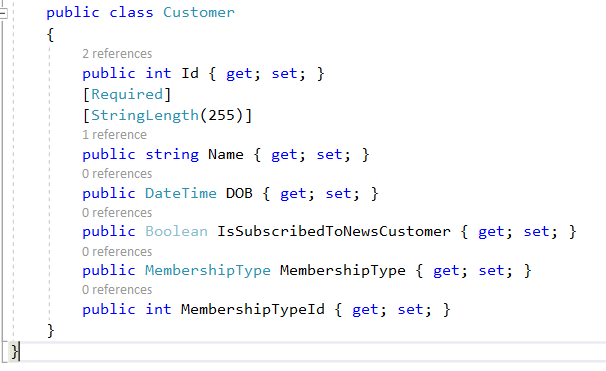
PM> add-migration InitialMigration

PM> add-migration InitialMigration -force 🡪 If this migration already exists

PM> update-database

In other to reference a class to some other class

Suppose, Customer Table should have MembershipType information in it



To revert back to previous migration

Update-Database -TargetMigration 202204061627362\_UpdateNameToMembershipType

Seeding the database

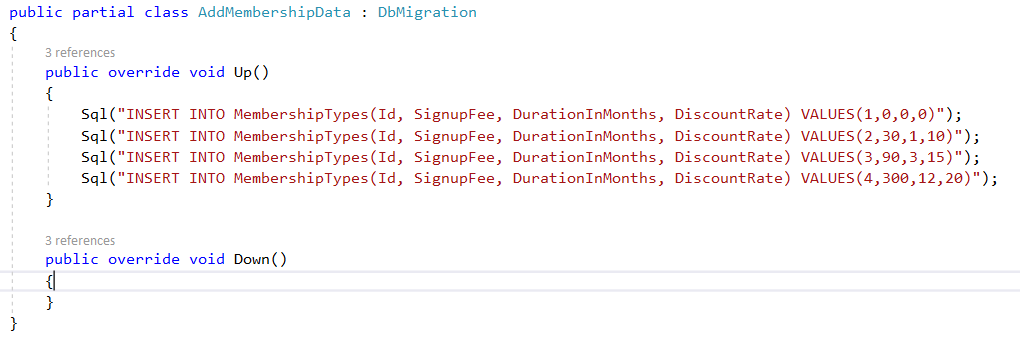
a) Suppose, data needs to be inserted into some table

b) This data should be consistent across all regions. In code first approach we do not insert data into database. Instead it should go through our code.

c) In Project Manager Console,

PM> add-migration AddMembershipData

d) In migration file created for AddMembershipData, add SQL statements for insert statements.



e) In Project Manager Console,

PM> update-migration

For DbContext issue

Run following commands into Packet Manager Console

sqllocaldb.exe stop

sqllocaldb.exe delete

sqllocaldb.exe start

Overriding Conventions

1) By default, string is nullable and can store any number of characters and id is not nullable

2) To override this convention, we can add annotation above respective field in Model class.

public class Customer

{

public int Id { get; set; }

[Required]

[StringLength(255)]

public string Name { get; set; }

public DateTime DOB { get; set; }

public bool IsSubscribedToNewsletter { get; set; }

public MembershipType MembershipType { get; set; }

public int MembershipTypeId { get; set; }

}

3) There annotations are defined in following namespace

using System.ComponentModel.DataAnnotations;

Querying Objects

1) Suppose we need to fetch data from database

2) For this add ApplicationContext in Controller we created.

private ApplicationContext \_context;

public CustomerController()

{

\_context = new ApplicationContext();

}

protected override void Dispose(bool disposing)

{

\_context.Dispose();

}

3) Action can use this context object to fetch data from database.

public ActionResult Index()

{

var customer = \_context.Customers;

return View(customer);

}

public ActionResult Details(int id)

{

var customer = \_context.Customers.SingleOrDefault(c=>c.Id==id);

return View(customer);

}

In view -> Index

@model IEnumerable<Vidly.Models.Customer>

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Index</h2>

<table>

<tr>

<th>

Customer Name

</th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

@Html.ActionLink(item.Name,"Details",new { Id = item.Id })

</td>

</tr>

}

</table>

In View -> Details

@model Vidly.Models.Customer

@{

ViewBag.Title = "Details";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>@Model.Name</h2>

Eager Loading

1) Suppose, we need to load customer details along with membership information.

2) By default, Entity Frame only loads Customer object (in our case), not their related objects. So, Membership Details will remain null.

3) For this, customers and their membership types should be loaded together.

4) This is called Eager Loading.

public ActionResult Index()

{

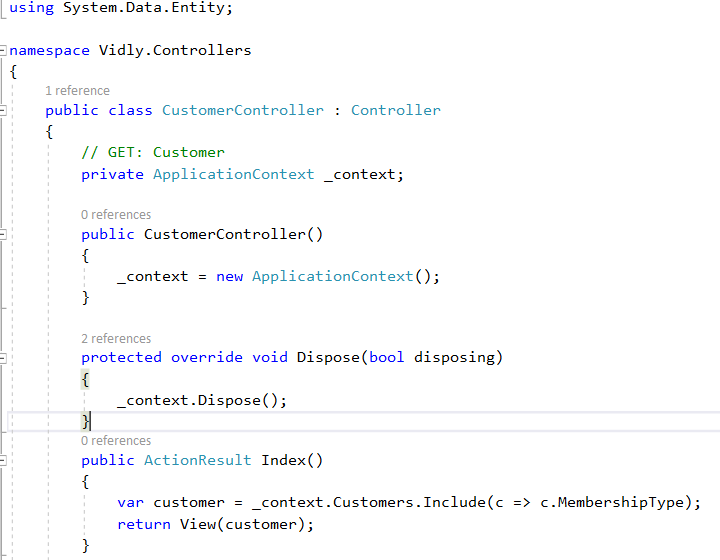
var customer = \_context.Customers.Include(c => c.MembershipType);

return View(customer);

}

5) Include is available in following namespace.

using System.Data.Entity;



Eager loading multiple table data

var ticket = \_context.Tickets.Include(c=>c.Status).Include(c=>c.Priority).Include(c => c.TicketType).SingleOrDefault(c => c.Id == id);

Add Shortcuts for Package Manager Console

1) Tools -> Options -> Environment -> Keyboard

2) Show command containing -> packagemanagerconsole

3) In press shortcut keys section -> Alt+/+.

4) OK

Form

1) Create New Action and View for that action

2) In View, to create form

@using (Html.BeginForm("Create", "Customer"))

{

<div class="form-group">

@Html.LabelFor(m => m.Name)

@Html.TextBoxFor(m => m.Name, new { @class = "form-control" })

</div>

<div class="form-group">

@Html.LabelFor(m => m.DOB)

@Html.TextBoxFor(m => m.DOB, new { @class="form-control"})

</div>

<div class="form-group">

<label>

@Html.CheckBoxFor(m => m.IsSubscribedToNewsletter) Is Subscribed To Newsletter?

</label>

</div>

}

@using (Html.BeginForm("Create", "Customer")) 🡪 When a form is submitted, action to be performed and name of controller

{

<div class="form-group">

@Html.LabelFor(m => m.Name)

@Html.TextBoxFor(m => m.Name, new { @class = "form-control" })

</div>

<div class="form-group">

@Html.LabelFor(m => m.DOB)

@Html.TextBoxFor(m => m.DOB, new { @class="form-control"})

</div>

<div class="form-group">

<label>

@Html.CheckBoxFor(m => m.IsSubscribedToNewsletter) Is Subscribed To Newsletter?

</label>

</div>

}

Changing label for form object

Method 1

1) Add annotation to corresponding field on model class

[Display(Name="Date of Birth")]

public DateTime? DOB { get; set; }

In View

<div class="form-group">

@Html.LabelFor(m => m.DOB)

@Html.TextBoxFor(m => m.DOB, new { @class = "form-control" })

</div>

Note: On doing this, On clicking label, text box will get focused

2) Add label in corresponding View

<div class="form-group">

<label>Date of Birth</label>

@Html.TextBoxFor(m => m.DOB, new { @class = "form-control" })

</div>

Note: On doing this, On clicking label, text box will not get focused

3) Add label for in corresponding view

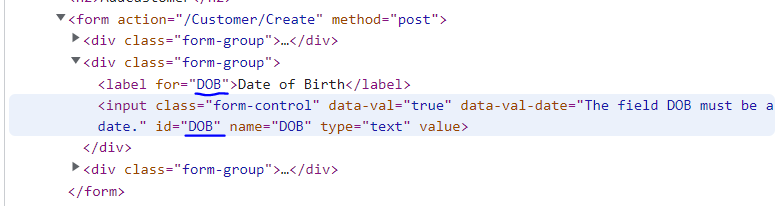
<div class="form-group">

<label for="DOB">Date of Birth</label>

@Html.TextBoxFor(m => m.DOB, new { @class = "form-control" })

</div>

Note: On doing this, On clicking label, text box will get focused because passing id on for section on label



For DropDownList

<div class="form-group">

@Html.LabelFor(m => m.Customer.MembershipTypeId)

@Html.DropDownListFor(m => m.Customer.MembershipTypeId, new SelectList(Model.MembershipType, "Id", "Name"), "Select Membership Type", new { @class = "form-control" })

</div>

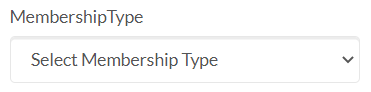
SelectList() is used to initialize dropdown list

1st argument is list of items

2nd argument property that holds id of list

3rd argument property that hold text for that list

"Select Membership Type" -> Name that appears at top on drop down list



Model Binding

a) binds model to request data

To add data from form to database, it should be added to context

SaveChanges will generate SQL statement at run time

Saving Data

a) In our code, we have specified in a form that Create action will be called when the form is submitted.

b) So create a Create Action inside controller

[HttpPost]

public ActionResult Create(CustomerViewModel customerVM)

{

var cust = new Customer()

{

Name = customerVM.Customer.Name,

DOB = customerVM.Customer.DOB,

IsSubscribedToNewsCustomer = customerVM.Customer.IsSubscribedToNewsCustomer,

MembershipTypeId = customerVM.Customer.MembershipTypeId

};

\_context.Customer.Add(cust);

\_context.SaveChanges();

return RedirectToAction("Index", "Customer");

}

Editing and Updating Data

Edit Action

public ActionResult Edit(int id)

{

var customer = \_context.Customer.SingleOrDefault(c => c.Id == id);

if (customer == null)

return HttpNotFound();

else

{

var vModel = new CustomerViewModel()

{

Customer = customer,

MembershipType = \_context.MembershipType.ToList()

};

return View("AddCustomer", vModel);

}

Create Action

[HttpPost]

public ActionResult Create(CustomerViewModel customerVM)

{

if (customerVM.Customer.Id == 0)

{

var cust = new Customer()

{

Name = customerVM.Customer.Name,

DOB = customerVM.Customer.DOB,

IsSubscribedToNewsCustomer = customerVM.Customer.IsSubscribedToNewsCustomer,

MembershipTypeId = customerVM.Customer.MembershipTypeId

};

\_context.Customer.Add(cust);

}

else

{

var custInDb = \_context.Customer.SingleOrDefault(c => c.Id == customerVM.Customer.Id);

custInDb.Name = customerVM.Customer.Name;

custInDb.DOB = customerVM.Customer.DOB;

custInDb.IsSubscribedToNewsCustomer = customerVM.Customer.IsSubscribedToNewsCustomer;

custInDb.MembershipTypeId = customerVM.Customer.MembershipTypeId;

// To update all fields

// TryUpdateModel(custInDb);

}

\_context.SaveChanges();

return RedirectToAction("Index", "Customer");

}

Note: to include id as hidden field in form

@Html.HiddenFor(m=>m.Customer.Id)

Delete Action

public ActionResult Delete(int id)

{

var ticketInDb = \_context.Tickets.SingleOrDefault(c=>c.Id==id);

if (ticketInDb == null)

{

return RedirectToAction("Index");

}

else

{

\_context.Tickets.Remove(ticketInDb);

\_context.SaveChanges();

return RedirectToAction("Index");

}

}

Adding Validation

a) ASP.NET also uses Annotation to validate action parameter

b) To add data validation

1) add data annotation to entities

[Required]

[StringLength(255)]

public string Name { get; set; }

2) Use ModelState.IsValid to in action to change the flow of the program. If the model state is not valid, use same view

[HttpPost]

public ActionResult Create(CustomerViewModel customerVM)

{

if (!ModelState.IsValid)

{

var Customer = customerVM.Customer;

var MembershipType = \_context.MembershipType.ToList();

var model\_ = new CustomerViewModel()

{

Customer = Customer,

MembershipType = MembershipType

};

return View("AddCustomer", model\_);

}

else

{

if (customerVM.Customer.Id == 0)

{

var cust = new Customer()

{

Name = customerVM.Customer.Name,

DOB = customerVM.Customer.DOB,

IsSubscribedToNewsCustomer = customerVM.Customer.IsSubscribedToNewsCustomer,

MembershipTypeId = customerVM.Customer.MembershipTypeId

};

\_context.Customer.Add(cust);

}

else

{

var custInDb = \_context.Customer.SingleOrDefault(c => c.Id == customerVM.Customer.Id);

custInDb.Name = customerVM.Customer.Name;

custInDb.DOB = customerVM.Customer.DOB;

custInDb.IsSubscribedToNewsCustomer = customerVM.Customer.IsSubscribedToNewsCustomer;

custInDb.MembershipTypeId = customerVM.Customer.MembershipTypeId;

// To update all fields

// TryUpdateModel(custInDb);

}

\_context.SaveChanges();

return RedirectToAction("Index", "Customer");

}

}

3) Add placeholder in form for Validation messages

<div class="form-group">

@Html.LabelFor(m => m.Customer.Name)

@Html.TextBoxFor(m => m.Customer.Name, new { @class = "form-control" })

@Html.ValidationMessageFor(m=>m.Customer.Name)

</div>

Styling Validation Errors

a) Inside Contents -> Site.css

b) Add following lines at the bottom of .css file

.field-validation-error {

color: red;

}

.input-validation-error {

border: 2px solid red;

}

c) Here, we can get class name by inspecting the page

Data Annotations

|  |  |
| --- | --- |
| [Required] |  |
| [StringLength(255)] |  |
| [Range(1,10)] |  |
| [Compare(“OtherProperty”)] | To compare two properties like password to password |
| [Phone] |  |
| [EmailAddress] |  |
| [Url] |  |
| [RegularExpression(“..”)] |  |

1) All these data annotations have their default data validation message

2) If we need to override validation message,

[Required(ErrorMessage = “Please enter customer’s name.”)]

Custom Validation

1) Add new class inside Models say Min18YearsIfAMember

2) Derive this class from ValidationAttribute which is defined in namespace using System.ComponentModel.DataAnnotations;

3) override IsValid method

var customer= (Customer)validationContext.ObjectInstance;

Success case -> return ValidationResult.Success

Failure case -> return new ValidationResult(“Validation Message”)

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace Vidly.Models

{

public class \_18YearCheck:ValidationAttribute

{

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

{

var customer=(Customer) validationContext.ObjectInstance;

if (DateTime.Today.Year - customer.DOB.Year < 18)

{

return new ValidationResult("Customer is less than 18 years old");

}

else

{

return ValidationResult.Success;

}

}

}

}

4) Add this validation as annotation inside model

[\_18YearCheck]

Refactoring Magic Number

Inside respective Model class, add following properties

Model Class -> Customer.cs

public static readonly int AgeLimit = 10;

In validation or wherever applicable, use it as

Eg. MembershipType.Unknown or MembershipType.PayAsYouGo

Model Class -> 18YearCheck

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace Vidly.Models

{

public class \_18YearCheck:ValidationAttribute

{

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

{

var customer=(Customer) validationContext.ObjectInstance;

if (DateTime.Today.Year-customer.DOB.Year < Customer.AgeLimit)

{

String error\_msg = String.Format("Customer is < {0} years old", Customer.AgeLimit);

return new ValidationResult(error\_msg);

}

else

{

return ValidationResult.Success;

}

}

}

}

Implementing Validation Summary

To display summary of validation,

@Html.ValidationSummary()

@model Vidly.ViewModels.CustomerViewModel

@{

ViewBag.Title = "AddCustomer";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2> Add Customer </h2>

@using (@Html.BeginForm("Create", "Customer"))

{

@Html.ValidationSummary()

<div class="form-group">

@Html.LabelFor(m => m.Customer.Name)

@Html.TextBoxFor(m => m.Customer.Name, new { @class = "form-control" })

@Html.ValidationMessageFor(m=>m.Customer.Name)

</div>

<div class="form-group">

@Html.LabelFor(m => m.Customer.DOB)

@Html.TextBoxFor(m => m.Customer.DOB, new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.Customer.DOB)

</div>

<div class="form-group">

<label>

@Html.CheckBoxFor(m => m.Customer.IsSubscribedToNewsCustomer) Is Subscribed To Newsletter?

</label>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Customer.MembershipTypeId)

@Html.DropDownListFor(m => m.Customer.MembershipTypeId, new SelectList(Model.MembershipType, "Id", "Name"), "Select Membership Type", new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.Customer.MembershipTypeId)

</div>

@Html.HiddenFor(m=>m.Customer.Id)

<button type="submit" class="btn btn-primary">

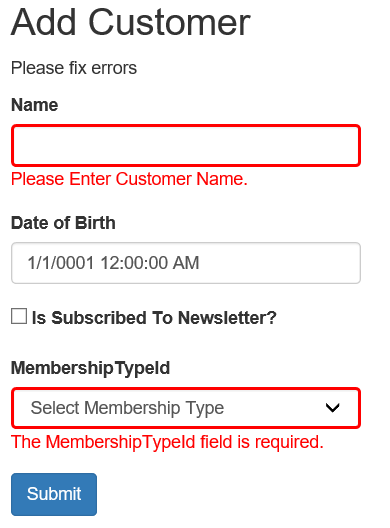
Submit

</button>

}

@Html.ValidationSummary(false) -> Ignore individual error is false

@Html.ValidationSummary(true,"Please fix errors") -> ignore induvial error



Client side Validation

At the bottom of form in View created add following lines

@section scripts{

@Scripts.Render("~/bundles/jqueryval")

}

Anti-forgery Token

Two steps to do so

a) In form add AntiForgeryToken

@Html.AntiForgeryToken()

b) In post action add following Annotation

[ValidateAntiForgeryToken]

@model Vidly.ViewModels.CustomerViewModel

@{

ViewBag.Title = "AddCustomer";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2> Add Customer </h2>

@using (@Html.BeginForm("Create", "Customer"))

{

@Html.ValidationSummary(true,"Fix below errors to proceed !!!")

<div class="form-group">

@Html.LabelFor(m => m.Customer.Name)

@Html.TextBoxFor(m => m.Customer.Name, new { @class = "form-control" })

@Html.ValidationMessageFor(m=>m.Customer.Name)

</div>

<div class="form-group">

@Html.LabelFor(m => m.Customer.DOB)

@Html.TextBoxFor(m => m.Customer.DOB, new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.Customer.DOB)

</div>

<div class="form-group">

<label>

@Html.CheckBoxFor(m => m.Customer.IsSubscribedToNewsCustomer) Is Subscribed To Newsletter?

</label>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Customer.MembershipTypeId)

@Html.DropDownListFor(m => m.Customer.MembershipTypeId, new SelectList(Model.MembershipType, "Id", "Name"), "Select Membership Type", new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.Customer.MembershipTypeId)

</div>

@Html.HiddenFor(m=>m.Customer.Id)

@Html.AntiForgeryToken()

<button type="submit" class="btn btn-primary">

Submit

</button>

}

@section scripts{

@Scripts.Render("~/bundles/jqueryval")

}

[HttpPost]

ValidateAntiForgeryToken]

public ActionResult Create(CustomerViewModel customerVM)

{

if (!ModelState.IsValid)

{

var Customer = customerVM.Customer;

var MembershipType = \_context.MembershipType.ToList();

var model\_ = new CustomerViewModel()

{

Customer = Customer,

MembershipType = MembershipType

};

return View("AddCustomer", model\_);

}

else

{

if (customerVM.Customer.Id == 0)

{

var cust = new Customer()

{

Name = customerVM.Customer.Name,

DOB = customerVM.Customer.DOB,

IsSubscribedToNewsCustomer = customerVM.Customer.IsSubscribedToNewsCustomer,

MembershipTypeId = customerVM.Customer.MembershipTypeId

};

\_context.Customer.Add(cust);

}

else

{

var custInDb = \_context.Customer.SingleOrDefault(c => c.Id == customerVM.Customer.Id);

custInDb.Name = customerVM.Customer.Name;

custInDb.DOB = customerVM.Customer.DOB;

custInDb.IsSubscribedToNewsCustomer = customerVM.Customer.IsSubscribedToNewsCustomer;

custInDb.MembershipTypeId = customerVM.Customer.MembershipTypeId;

// To update all fields

// TryUpdateModel(custInDb);

}

\_context.SaveChanges();

return RedirectToAction("Index", "Customer");

}

}

Change Heading based on condition

Example-> Edit or Add Customer

@if (Model.Customer!=null){

<h2> Edit Customer </h2>}

else{

<h2>Add Customer</h2>

}

Hidden Value can also be set as

@Html.Hidden("Customer.Id", (Model.Customer != null) ? Model.Customer.Id : 0)

Building RESTful Services with ASP.NET WEB API

API

/api/customers

|  |  |  |
| --- | --- | --- |
| GET | READ | /api/customers |
| GET | READ | /api/customers/1 |
| POST | CREATE | /api/customers |
| PUT | UPDATE | /api/customers/1 |
| DELETE | DELETE | /api/customers/1 |

Create Web API Controller

1) Create folder named API inside Controller

2) Create a controller say Empty controller with name CustomersController

Notice plural naming convention being used

3) Add statement into Global.asax.cs as per instruction provided in Readme.

Code for CustomersController Web API

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using Vidly.Models;

namespace Vidly.Controllers.API

{

public class CustomersController : ApiController

{

private AppContext \_context;

public CustomersController()

{

\_context = new AppContext();

}

// GET /api/customers

public IEnumerable<Customer> GetCustomers()

{

return \_context.Customer.ToList();

}

// GET /api/customers/1

public Customer GetCustomers(int id)

{

var customer= \_context.Customer.SingleOrDefault(c=>c.Id==id);

if (customer == null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

else

{

return customer;

}

}

// POST /api/customers

[HttpPost]

public Customer CreateCustomer(Customer customer)

{

if (!ModelState.IsValid)

{

throw new HttpResponseException(HttpStatusCode.BadRequest);

}

else

{

\_context.Customer.Add(customer);

\_context.SaveChanges();

return customer;

}

}

// PUT /api/customers/1

[HttpPut]

public void UpdateCustomer(int id, Customer customer)

{

if (!ModelState.IsValid)

{

throw new HttpResponseException(HttpStatusCode.BadRequest);

}

else

{

var customerInDb = \_context.Customer.SingleOrDefault(c => c.Id == id);

if (customerInDb == null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

else

{

customerInDb.Name = customer.Name;

customerInDb.DOB = customer.DOB;

customerInDb.IsSubscribedToNewsCustomer = customer.IsSubscribedToNewsCustomer;

customerInDb.MembershipTypeId = customer.MembershipTypeId;

\_context.SaveChanges();

}

}

}

[HttpDelete]

public void DeleteCustomer(int id)

{

var customerInDb = \_context.Customer.SingleOrDefault(c => c.Id == id);

if(customerInDb is null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

else

{

\_context.Customer.Remove(customerInDb);

\_context.SaveChanges();

}

}

}

}

Whenever we need to pass resource, specify Content-Type: application/json

DTO (Data Transfer Objects)

Right now, our web API exposes the database entities to the client. The client receives data that maps directly to your database tables. However, that's not always a good idea. Sometimes you want to change the shape of the data that you send to client. For example, you might want to:

* Remove circular references (see previous section).
* Hide particular properties that clients are not supposed to view.
* Omit some properties in order to reduce payload size.
* Flatten object graphs that contain nested objects, to make them more convenient for clients.
* Avoid "over-posting" vulnerabilities. (See [Model Validation](https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/model-validation-in-aspnet-web-api) for a discussion of over-posting.)
* Decouple your service layer from your database layer.

To accomplish this, you can define a data transfer object (DTO). A DTO is an object that defines how the data will be sent over the network.

To create DTO

1) Under Project folder, create folder named Dtos.

2) Create a class named CustomerDto

3) This will contain properties same/less/more like Customer Model.

4) Any domain class should be excluded as it creates dependency of DTO on our domain model. To solve this problem either use primitive type (int, string, etc) or create hierarchical structure by creating another DTO.

5) Now back in the controller, we need to remap everything mapped to Customer to CustomerDto. For this we can use AutoMapper

Customer class inside Model

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace Vidly.Models

{

public class Customer

{

public int Id { get; set; }

[Required(ErrorMessage ="Please Enter Customer Name.")]

[StringLength(255)]

public string Name { get; set; }

[Display(Name="Date of Birth")]

[\_18YearCheck]

public DateTime DOB { get; set; }

public Boolean IsSubscribedToNewsCustomer { get; set; }

public MembershipType MembershipType { get; set; }

public int MembershipTypeId { get; set; }

public static readonly int AgeLimit = 10;

}

}

CustomerDto class inside Dtos

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace Vidly.Dtos

{

public class CustomerDto

{

public int Id { get; set; }

[Required]

[StringLength(255)]

public string Name { get; set; }

public DateTime DOB { get; set; }

public Boolean IsSubscribedToNewsCustomer { get; set; }

public int MembershipTypeId { get; set; }

}

}

AutoMapper

1) It is convention based mapping tool as it uses properties name as convention to map between objects.

2) To install AutoMapper, On Package Manager Console

Install-Package automapper -version:4.1

3) In App\_Start, add a new class named MappingProfile.cs

This class needs to be derived from class Profile which is defined in namespace AutoMapper.

Add a constructor and use Mapper.CreateMap() method to create mapping configuration between two types.

using AutoMapper;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using Vidly.Dtos;

using Vidly.Models;

namespace Vidly.App\_Start

{

public class MappingProfile:Profile

{

public MappingProfile()

{

Mapper.CreateMap<Customer, CustomerDto>();

Mapper.CreateMap<CustomerDto, Customer>();

}

}

}

4) MappingProfile needs to be loaded when the application started.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Http;

using System.Web.Mvc;

using System.Web.Optimization;

using System.Web.Routing;

using AutoMapper;

using Vidly.App\_Start;

namespace Vidly

{

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

Mapper.Initialize(c => c.AddProfile<MappingProfile>());

GlobalConfiguration.Configure(WebApiConfig.Register);

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

}

5) Now controller class should be modified as

using AutoMapper;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using Vidly.Dtos;

using Vidly.Models;

namespace Vidly.Controllers.API

{

public class CustomersController : ApiController

{

private AppContext \_context;

public CustomersController()

{

\_context = new AppContext();

}

// GET /api/customers

public IEnumerable<CustomerDto> GetCustomers()

{

return \_context.Customer.ToList().Select(Mapper.Map<Customer,CustomerDto>);

}

// GET /api/customers/1

public CustomerDto GetCustomers(int id)

{

var customer= \_context.Customer.SingleOrDefault(c=>c.Id==id);

if (customer == null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

else

{

return Mapper.Map<Customer, CustomerDto>(customer);

}

}

// POST /api/customers

[HttpPost]

public CustomerDto CreateCustomer(CustomerDto customerdto)

{

if (!ModelState.IsValid)

{

throw new HttpResponseException(HttpStatusCode.BadRequest);

}

else

{

var customer=Mapper.Map<CustomerDto,Customer>(customerdto);

\_context.Customer.Add(customer);

\_context.SaveChanges();

customerdto.Id = customer.Id;

return customerdto;

}

}

// PUT /api/customers/1

[HttpPut]

public void UpdateCustomer(int id, CustomerDto customerdto)

{

customerdto.Id = id;

if (!ModelState.IsValid)

{

throw new HttpResponseException(HttpStatusCode.BadRequest);

}

else

{

var customerInDb = \_context.Customer.SingleOrDefault(c => c.Id == id);

if (customerInDb == null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

else

{

Mapper.Map<CustomerDto, Customer>(customerdto, customerInDb);

\_context.SaveChanges();

}

}

}

[HttpDelete]

public void DeleteCustomer(int id)

{

var customerInDb = \_context.Customer.SingleOrDefault(c => c.Id == id);

if(customerInDb is null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

else

{

\_context.Customer.Remove(customerInDb);

\_context.SaveChanges();

}

}

}

}

Configure Web API to return JSON object using CAMEL Notation

a) To make sure json object return will be in camel case format and in proper indentation

b) Inside WebApiConfig.cs file, add following lines

var settings = config.Formatters.JsonFormatter.SerializerSettings;

settings.ContractResolver = new CamelCasePropertyNamesContractResolver();

settings.Formatting = Formatting.Indented;

with namespace

using Newtonsoft.Json;

using Newtonsoft.Json.Serialization;

WebApiConfig.cs

using Newtonsoft.Json;

using Newtonsoft.Json.Serialization;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web.Http;

namespace Vidly

{

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

var settings = config.Formatters.JsonFormatter.SerializerSettings;

settings.ContractResolver = new CamelCasePropertyNamesContractResolver();

settings.Formatting = Formatting.Indented;

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

}

IHttpActionResult

1) Currently, when a object is created, return type is 200 OK

2) But return type should actually be 201 CREATED.

3) For this, we use IHttpActionResult.

|  |  |  |  |
| --- | --- | --- | --- |
|  | STATUS CODE / MSG |  | RETURNS |
| CREATE | 201 CREATED | CREATED(URI OF NEWLY CREATED RESOURCE,OBJECT) | api/customers/101  101 is id of new object created |
| GET | 200 OK | OK(OBJECT) |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[HttpPost]

public IHttpActionResult CreateCustomer(CustomerDto customerdto)

{

if (!ModelState.IsValid)

{

return BadRequest();

}

else

{

var customer=Mapper.Map<CustomerDto,Customer>(customerdto);

\_context.Customer.Add(customer);

\_context.SaveChanges();

customerdto.Id = customer.Id;

return Created(new Uri(Request.RequestUri+ "/" +customer.Id),customerdto);

}

}

public IHttpActionResult GetCustomers(int id)

{

var customer= \_context.Customer.SingleOrDefault(c=>c.Id==id);

if (customer == null)

{

return NotFound();

}

else

{

return Ok(Mapper.Map<Customer, CustomerDto>(customer)) ;

}

}