**.NET CORE MVC**

use ASP.NET CORE WEB APP (MODEL-VIEW-CONTROLLER)

MVC ARCHITECHTURE

4. Response

1. Request

3. Get presentation

2. Get data

User

clicks

CONTROLLER

Handles user request and acts as interface between model and view

VIEW

Represent the user interface

MODEL

Represent the shape of the data

Ultimately user can see the complete website.

CRUD

WebApplication2.cs

CREATE

Create a project by selecting C#; ASP.NET CORE WEB APP (MODEL-VIEW-CONTROLLER)

Add; class; to models folder by selecting class; Category.cs

Category.cs class will contain properties that will represent the column of category table

Add in line 1

using System.ComponentModel.DataAnnotations;

Type prop; 2 times enter

public int Id { get; set; }

public string Name { get; set; }

public int DisplayOrder { get; set; }

DATA ANNOTATION

Primary key is required to if Id is not present in the name of primary key. Ex: Id, CategoryId (default primary key is accepted without using [Key])

If Category\_\_Id then add [Key]

[Required] – mandatory column data to be filled

Open ssms; to connect from visual studio go to view open sql serverobject explorer; click on add sql server; select local; see server name and copy it and pate it on ssms (localdb)\MSSQLLocalDB; connect

CONNECTION STRING

appsettings.json

,

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\MSSQLLocalDB; Database=Bulky;Trusted\_Connection=True;TrustServerCertificate=True;"

}

NUGET PACKAGES FOR ENTITY FRAMEWORK CORE

To create database table, go file name and select manage nuget packages; search Microsoft.EntityFrameworkCore; install it; search Microsoft.EntityFrameworkCore.SqlServer; install it; make sure all the package shall be of same release since difference in release will create problem further; search Microsoft.EntityFrameworkCore.Tools; install it; right click on project file name and go to edit project file, we can see all installed packages with the released version.

SETUP ApplicationDbContext

right click on project file name and go to add; new folder name it: DATA;right click on data folder; add; class; name it ApplicationDbContext.cs

configuration

after typing DbContext hit ctrl + . to get the package name;ctor tab 2 times to get constructor;for base… hit tab

using Microsoft.EntityFrameworkCore;

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) {

}

}

Program.cs

Add services to the container

Add name of class from applicationDbContext.cs; give name => add connectionstring name from appsettings.json

builder.Services.AddDbContext<ApplicationDbContext>(options=>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

CREATE DATABASE

Go to tools; nuget package manager; package manager console

Creating database by the command

> update-database

Open ssms; refresh; Databases;Bulky;dbo.\_EFMigrationHistory right click here and go to select top 1000 rows

Below we can see MigrationId and ProductVersion

CREATE CATEGORY TABLE

ApplicationDbContext.cs

using WebApplication2.Models;

This line will create table with category name; we can give name of table i.e Categories here

public DbSet<Category> Categories { get; set; }

Go to tools; nuget package manager; package manager console

add-migration AddCategoryTabletoDb

Open ssms; refresh; Databases;Bulky;dbo.Categories right click here and go to select top 1000 rows

Below we can see Id, name, displayorder all column names of the table

ADD CATEGORY CONTROLLER

Go to controller right click; Add; Controller; Add; name it CategoryController.cs; always make sure you add Controller in the end of the name so that its recognized as a controller

Go to view right click; Add; New folder and name it as same as of model name i.e Category and then right click; Add; View; select empty;Add;name it as Index.cshtml

Index.cshtml

<h1>Category List</h1>

After running add this url to check content of view i.e of Index.cshtml

https://localhost:7286/Category/Index

ADD CATEGORY LINK IN HEADER

\_Layout.cshtml

Add controller name i.e Category; add file name of the views -> Category folder i.e Index; add whatever name we want to display on webpage i.e Category

<li class="nav-item">

<**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="Category" **asp-action**="Index">Category</**a**>

</li>

SEED CATEGORY TABLE

ApplicationDbContext.cs

Adding data to database with asp.net facility

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Category>().HasData(

new Category { Id=1, Name="Action", DisplayOrder=1},

new Category { Id = 2, Name = "SciFi", DisplayOrder = 2 },

new Category { Id = 3, Name = "History", DisplayOrder = 3 }

);

}

Now add migration -> whenever anything has to be updated in database

Go to tools; nuget package manager; package manager console

> add-migration SeedCategoryTable

> update-database

GET ALL CATEGORIES

CategoryController.cs

Type ctor hit enter

public IActionResult Index()

{

List<Category> objCategoryList=\_db.Categories.ToList();

return View();

}

private readonly ApplicationDbContext \_db;

public CategoryController(ApplicationDbContext db)

{

\_db= db;

}

HOT RELOAD

Index.cshtml

<table class=" table table-bordered table-striped">

<tr>

<th>

Category Name

</th>

<th>

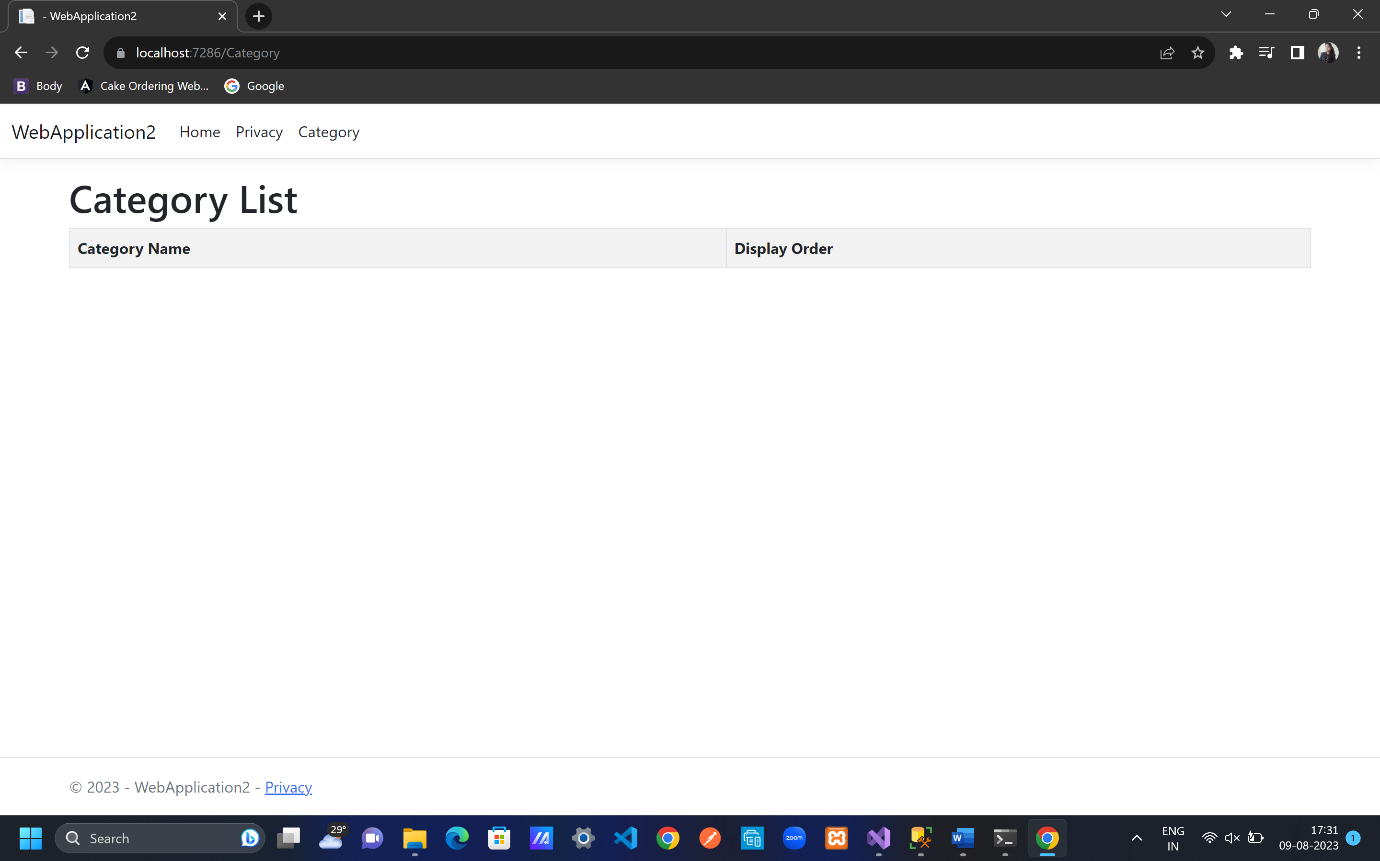
Display Order

</th>

</tr>

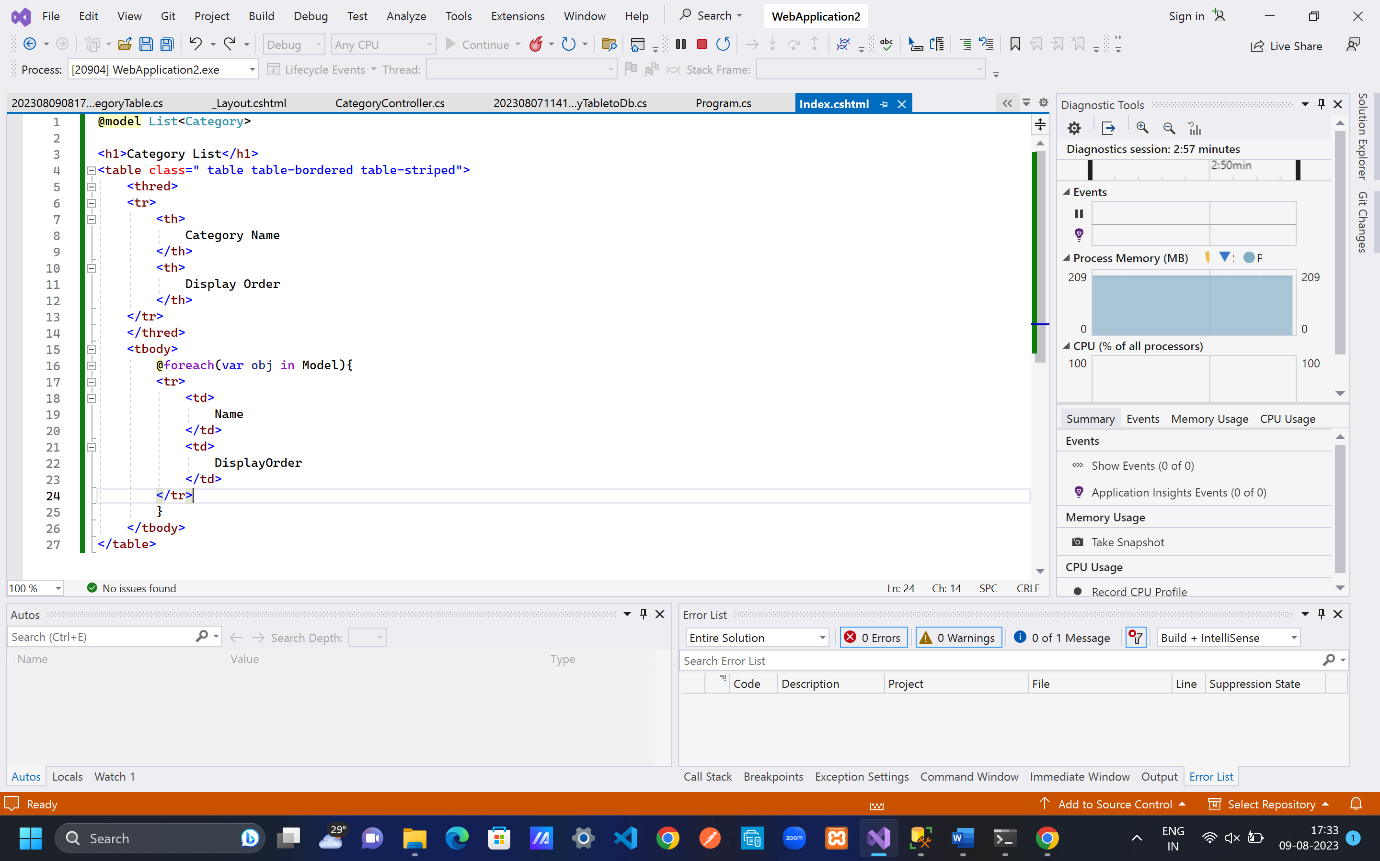
</table>

Hot reload -> whenever we do changes to code without restarting the application/running again simply hit hot reload to changes

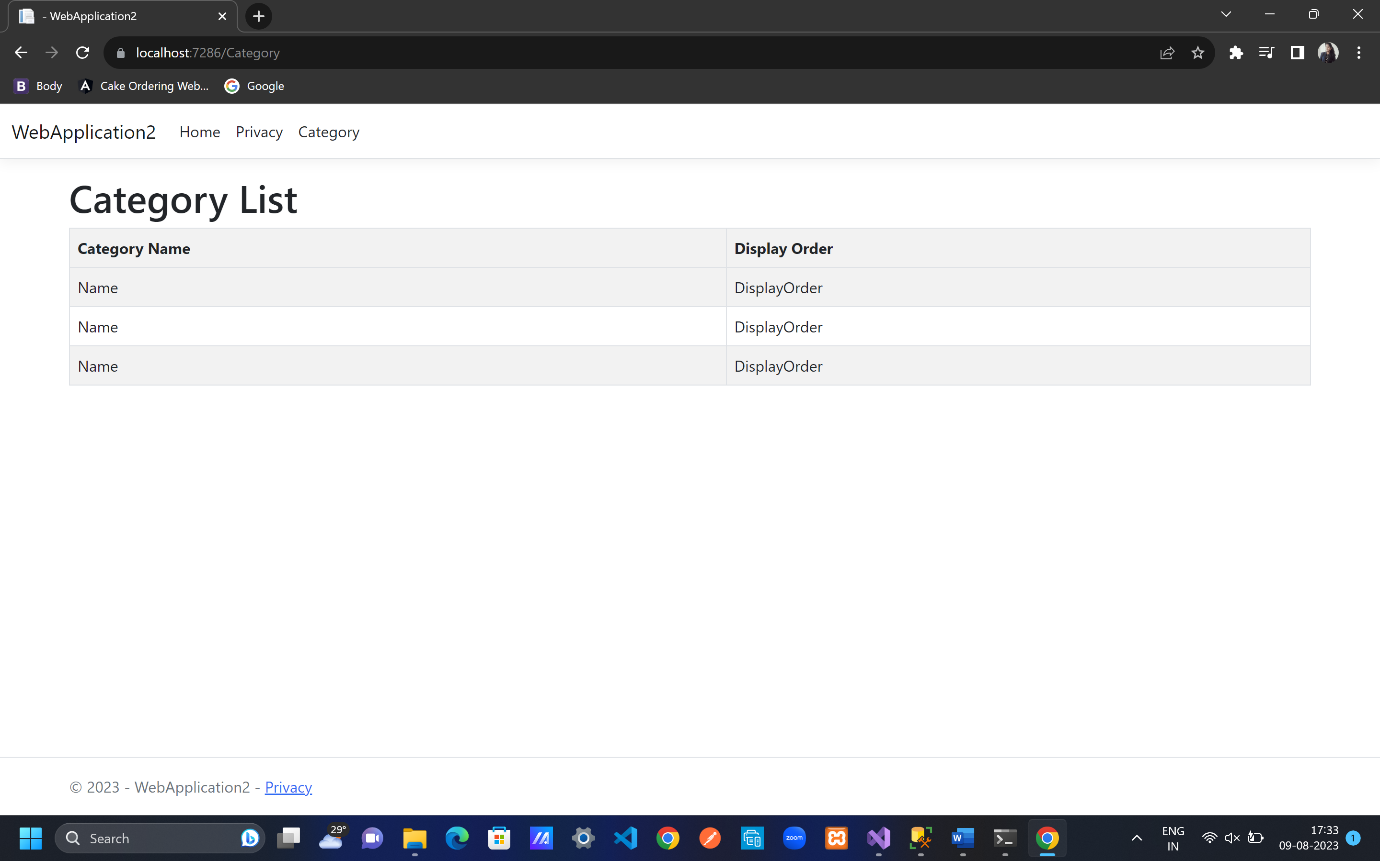


DISPLAY ALL CATEGORIES

If we do this



Then we get



To get all values

CategoryController.cs

public IActionResult Index()

{

List<Category> objCategoryList=\_db.Categories.ToList();

return View(objCategoryList);

}

Index.cshtml

To see data in ascending order by DisplayOrder column ->

OrderBy(u=>u.DisplayOrder)

@model List<Category>

<h1>Category List</h1>

<table class=" table table-bordered table-striped">

<thred>

<tr>

<th>

Category Name

</th>

<th>

Display Order

</th>

</tr>

</thred>

<tbody>

@foreach(var obj in Model.OrderBy(u=>u.DisplayOrder)){

<tr>

<td>

@obj.Name

</td>

<td>

@obj.DisplayOrder

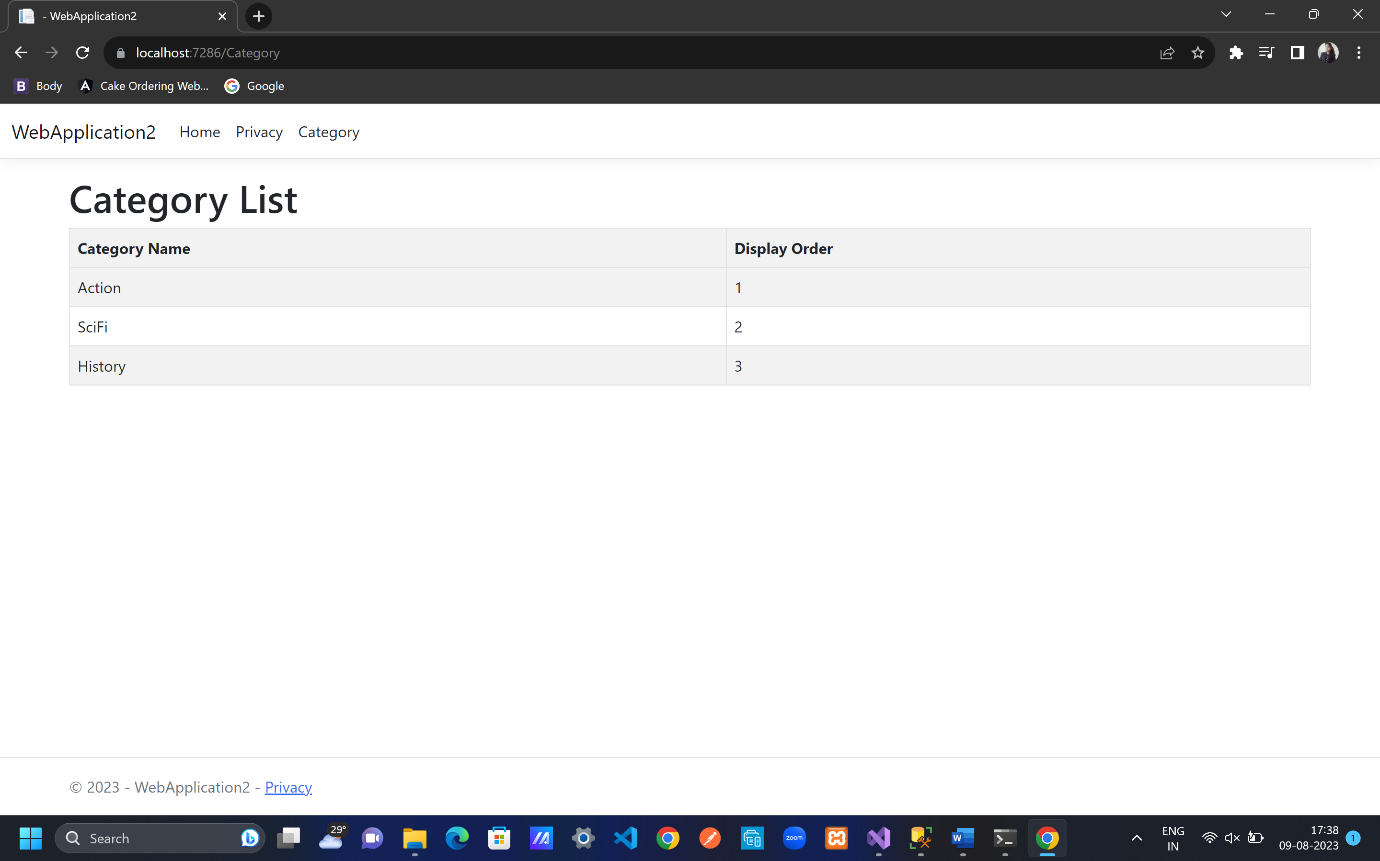
</td>

</tr>

}

</tbody>

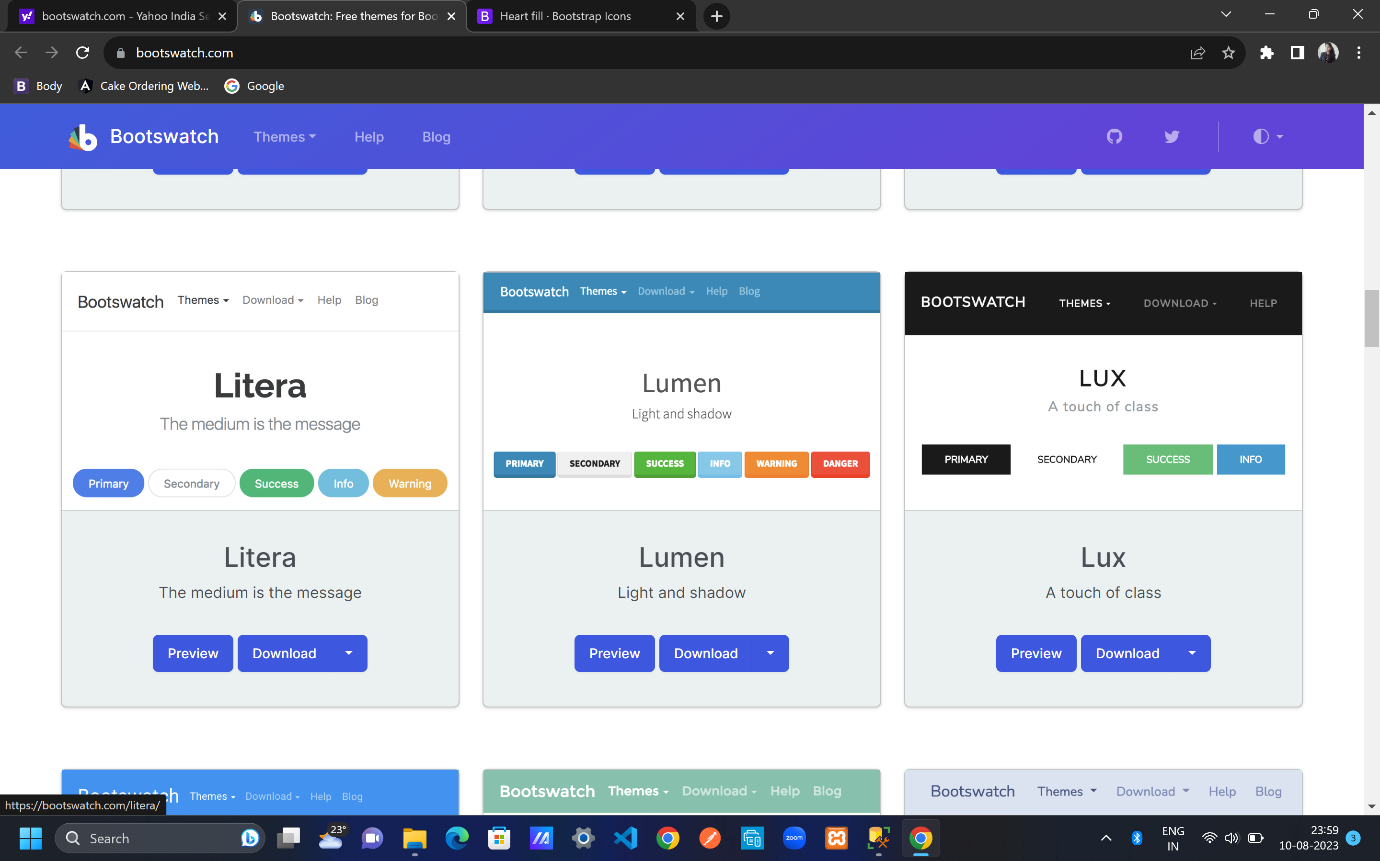
</table>



BOOTSWATCH THEME AND BOOTSTRAP ICONS

bootswatch.com

free themes are bootswatch website



download and copy it

go to wwwroot;lib folder;bootstrap folder;dist folder; css folder;bootstrap.css

paste the copied code in bootstrap.com

go to \_layout.cshtml;

<**link** rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />

<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-dark bg-primary border-bottom box-shadow mb-3">

<li class="nav-item">

<**a** class="nav-link " **asp-area**="" **asp-controller**="Home" **asp-action**="Index">Home</**a**>

</li>

<li class="nav-item">

<**a** class="nav-link " **asp-area**="" **asp-controller**="Home" **asp-action**="Privacy">Privacy</**a**>

</li>

<li class="nav-item">

<**a** class="nav-link " **asp-area**="" **asp-controller**="Category" **asp-action**="Index">Category</**a**>

</li>

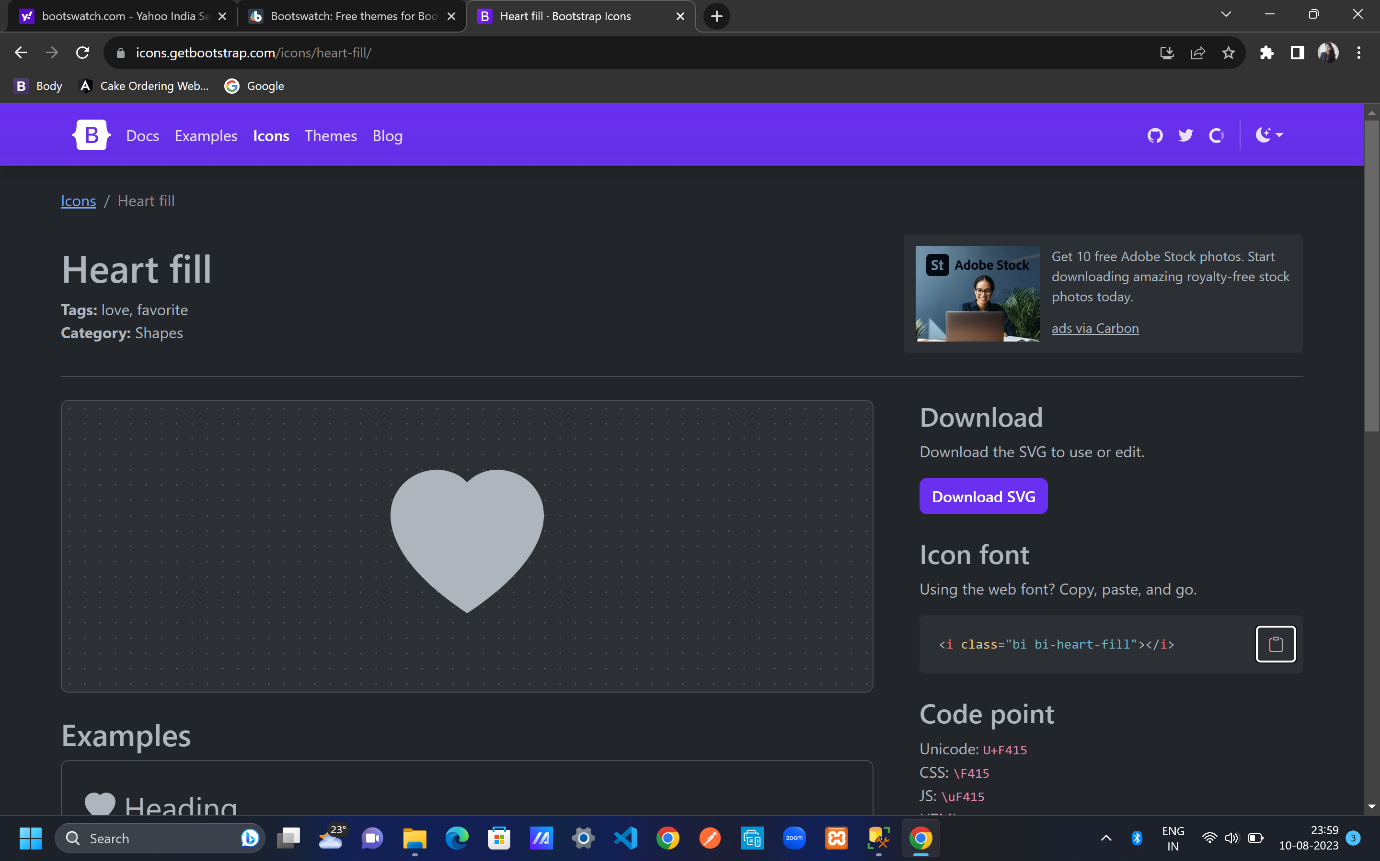
<footer class="border-top footer bg-primary text-muted">

For icons go to icons.getbootstrap.com

Go to install;copy the CDN;go to \_layout.cshtml

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.css">

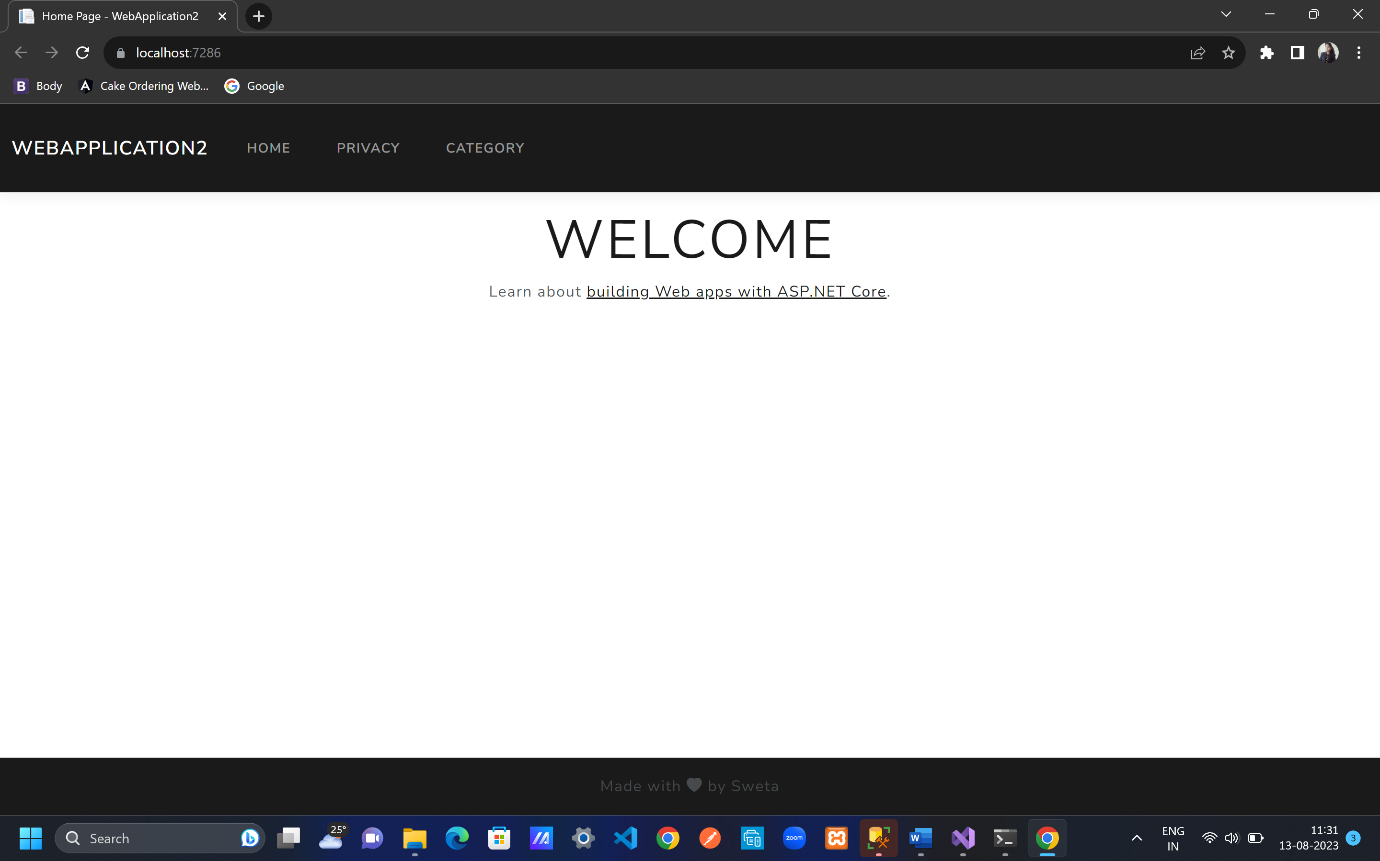
Search for an icon; copy from icon font and paste in \_layout.cshtml



div class="text-center">

Made with <i class="bi bi-heart-fill"></i> by Sweta

</div>



DESIGN CATEGORY LIST PAGE

Go to Views folder; Category folder; Index.cshtml

Go to icons.getbootstrap.com for plus circle icon

<div class="container">

<div class="row pt-4 pb-3">

<div class="col-6">

<h2 class="text-primary">

Category List

</h2>

</div>

<div class="col-6 text-end">

<**a** **asp-controller**="" **asp-action**="" class="btn btn-primary">

<i class="bi bi-plus-circle"></i> Create New Category

</**a**>

</div>

</div>

<table class=" table table-bordered table-striped">

<thred>

<tr>

<th>

Category Name

</th>

<th>

Display Order

</th>

</tr>

</thred>

<tbody>

@foreach(var obj in Model.OrderBy(u=>u.DisplayOrder)){

<tr>

<td>

@obj.Name

</td>

<td>

@obj.DisplayOrder

</td>

</tr>

}

</tbody>

</table>

</div>



CREATE CATEGORY UI

To get the user data from the create new category button

Go to Controllers folder; CategoryController.cs

public IActionResult Create()

{

return View();

}

To create view in Views folder

Select Create() in CategoryController.cs and right click; Add View;Add; Create.cshtml; Add

Name while creating view of Views folder should be same as with the name of function/method in CategoryController.cs i.e Create.cshtml=Create()

Since Index.cshtml is in same folder i.e Category folder;CategoryController.cs so there is no need to add asp-controller=”” but insure that we shall mention controller than action.

Create.cshtml

@model Category // new category object with a default value

<**form** method="post">

<div class="border p-3 mt-4">

<div class="row pb-2">

<h2 class="text-primary">Create Category</h2>

<hr/>

</div>

<div class="mb-3 row p-1"> //row added to make page responsive

<label class="p-0">Category Name</label>

<input type="text" class="form-control"/>

</div>

<div class="mb-3 row p-1"> //row added to make page responsive

<label class="p-0">Display Order</label>

<input type="text" class="form-control" />

</div>

<div class="row">

<div class="col-6 col-md-3">

<button type="submit" class="btn btn-primary form-control" >Create</button>

</div>

<div class="col-6 col-md-3">

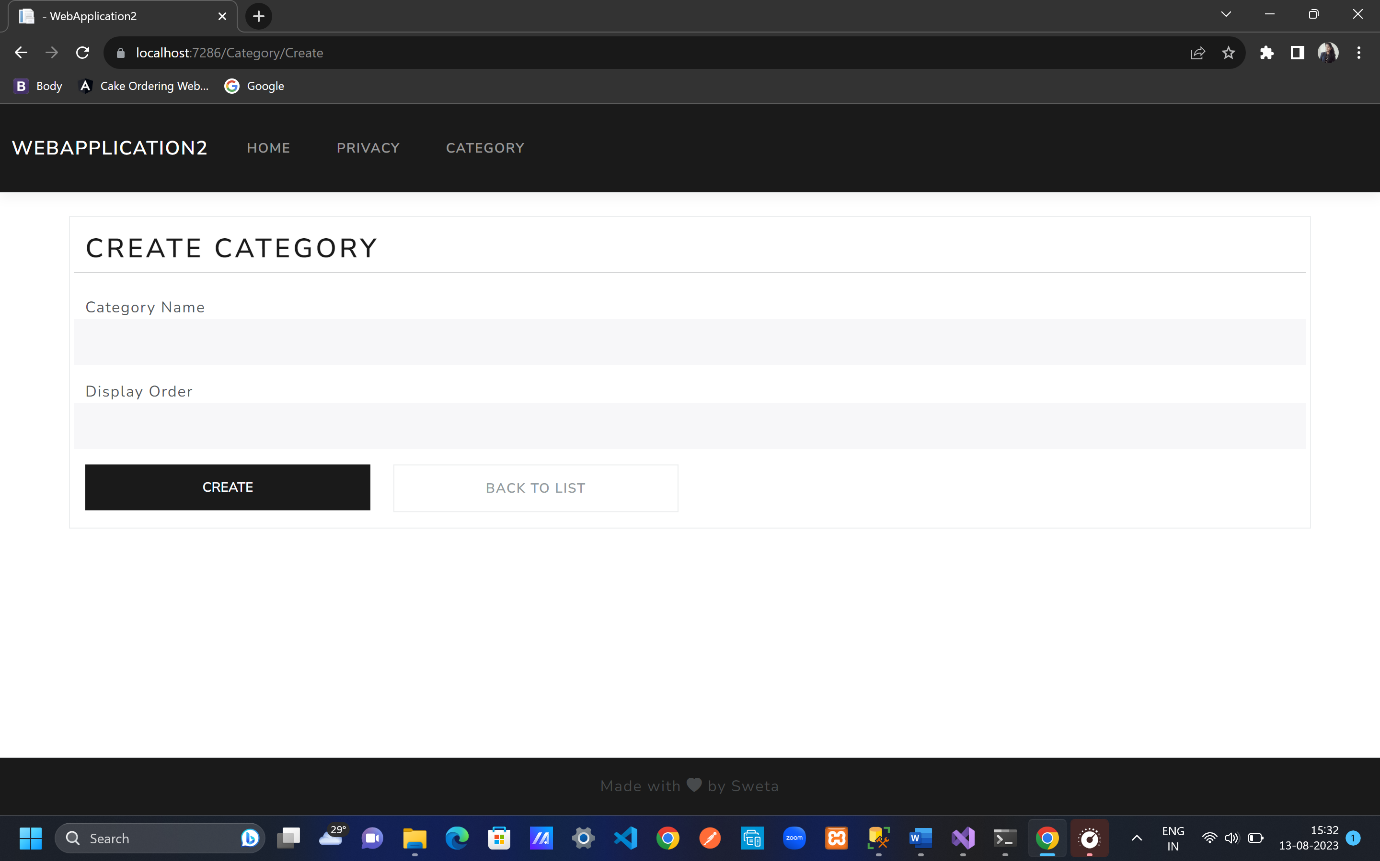
<**a** **asp-controller**="Category" **asp-action**="Index" class="btn btn-outline-secondary border form-control" > Back to List</**a**>

</div>

</div>

</div>

</**form**>



INPUT TAG HELPERS

Create.cshtml

<div class="mb-3 row p-1">

<**label** **asp-for**="Name" class="p-0"></**label**>

<**input** **asp-for**="Name" class="form-control"/>

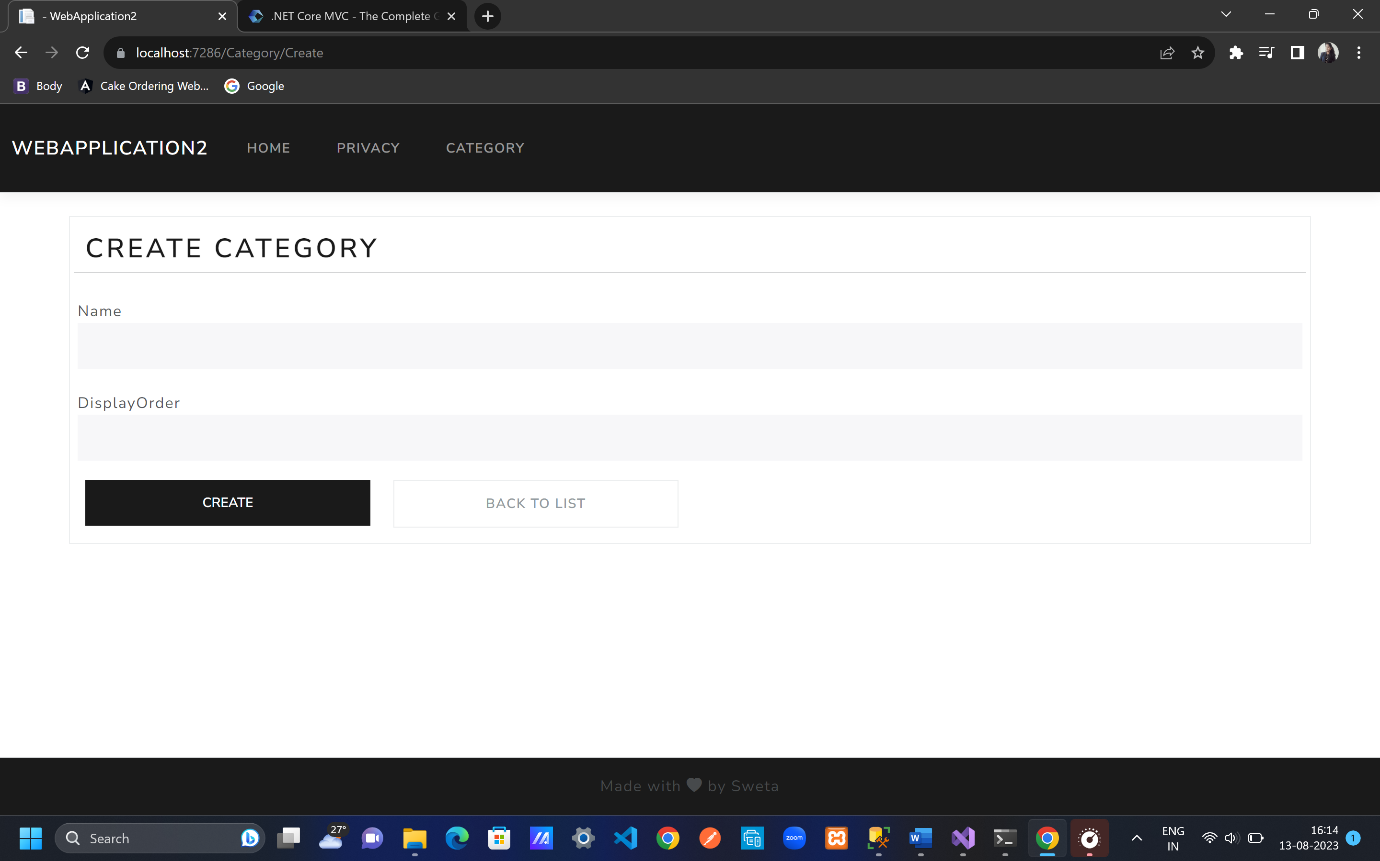
</div>

<div class="mb-3 row p-1">

<**label** **asp-for**="DisplayOrder" class="p-0"></**label**>

<**input** **asp-for**="DisplayOrder" class="form-control" />

</div>



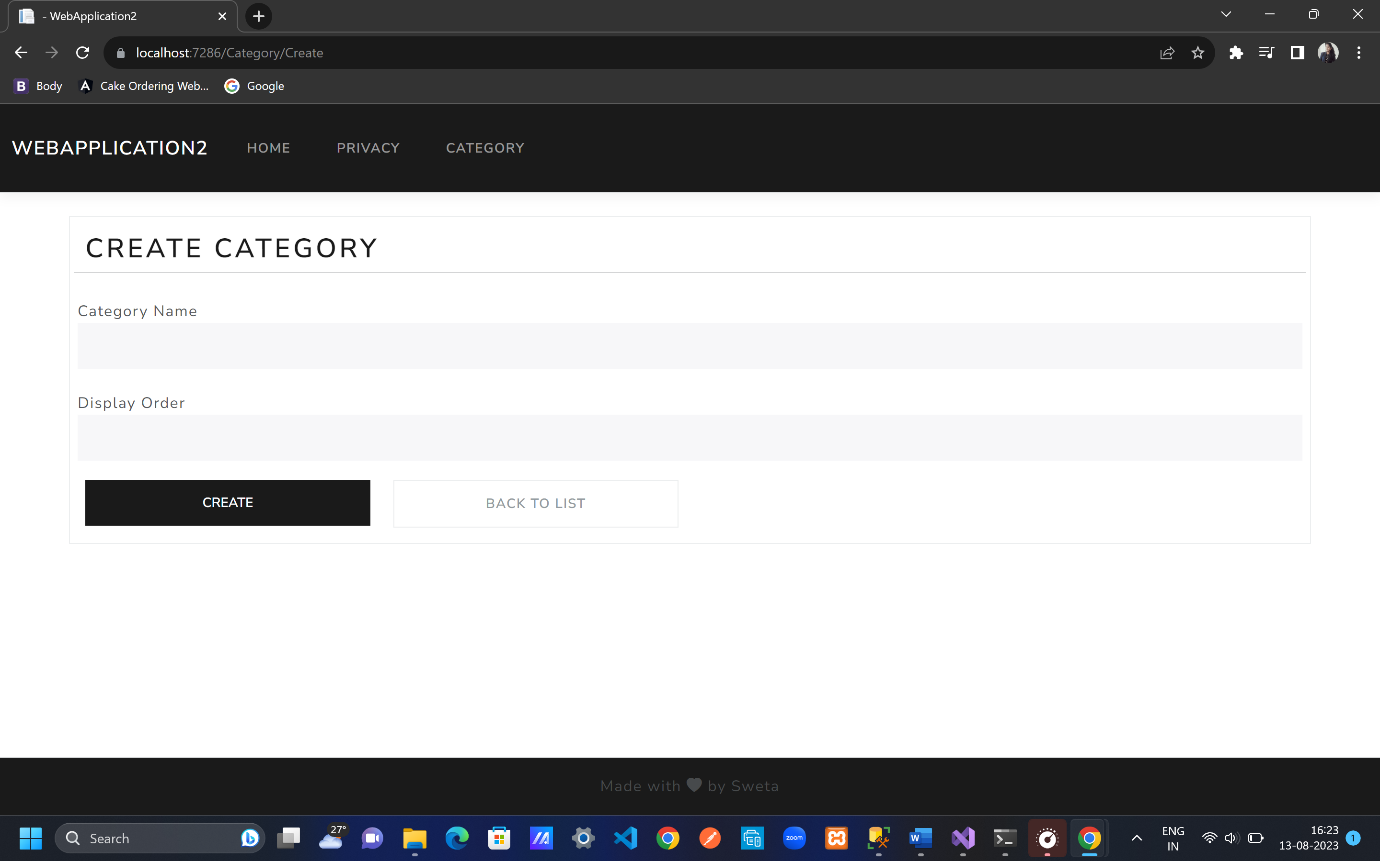
Category.cs

[DisplayName("Category Name")]

public string Name { get; set; }

[DisplayName("Display Order")]

public int DisplayOrder { get; set; }



asp-for property helps to identify the datatype of label name.

CREATE CATEGORY

CategoryController.cs

[HttpPost]

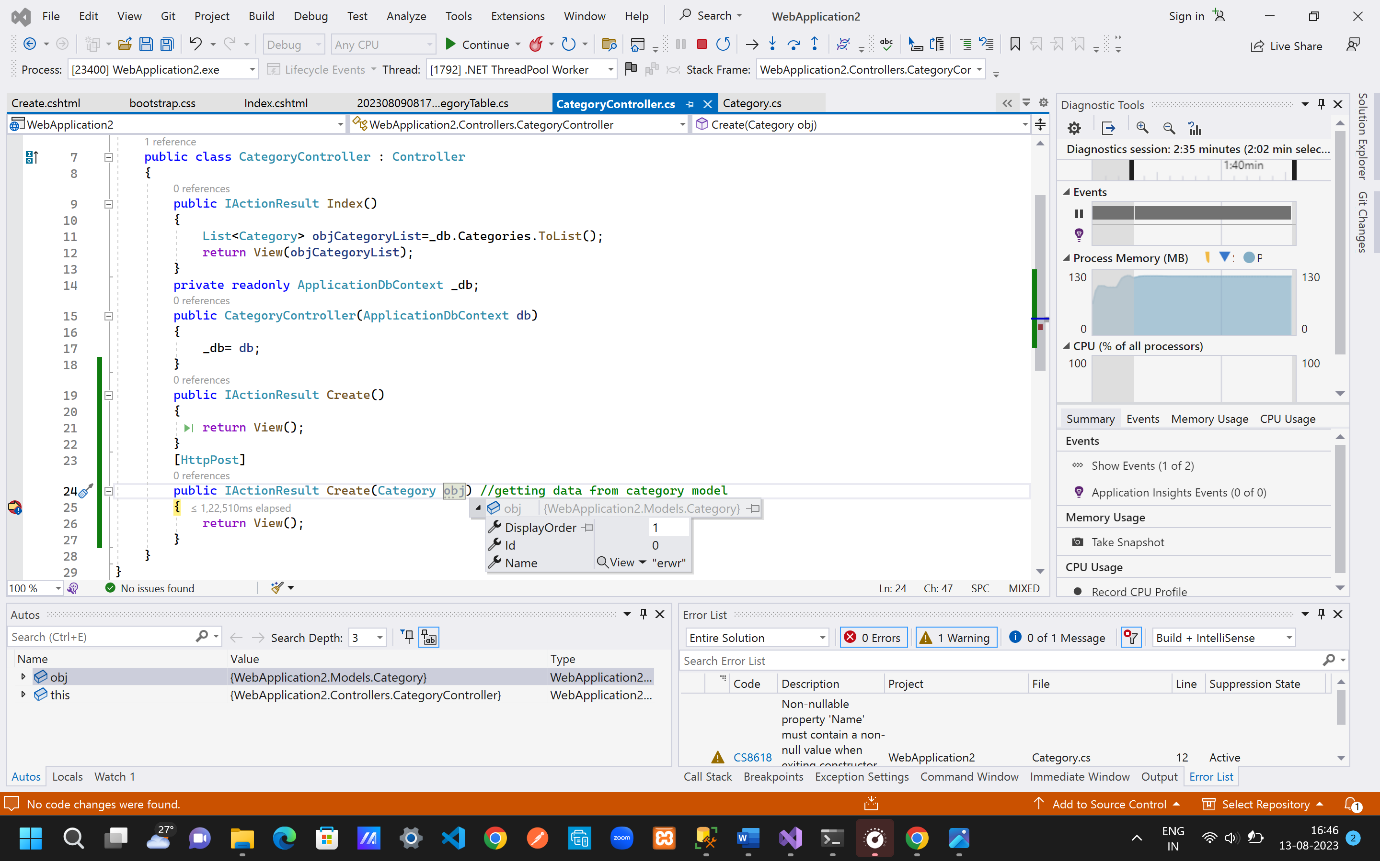
public IActionResult Create(Category obj) //getting data from category model

{

return View();

}

Add break point on line 25; add data on web page then come back check as shown below



CategoryController.cs

[HttpPost]

public IActionResult Create(Category obj) //getting data from category model

{

\_db.Categories.Add(obj); // what is to be add

\_db.SaveChanges(); //go to the database and create category

return RedirectToAction("Index"); //view all categories in index() line 9;

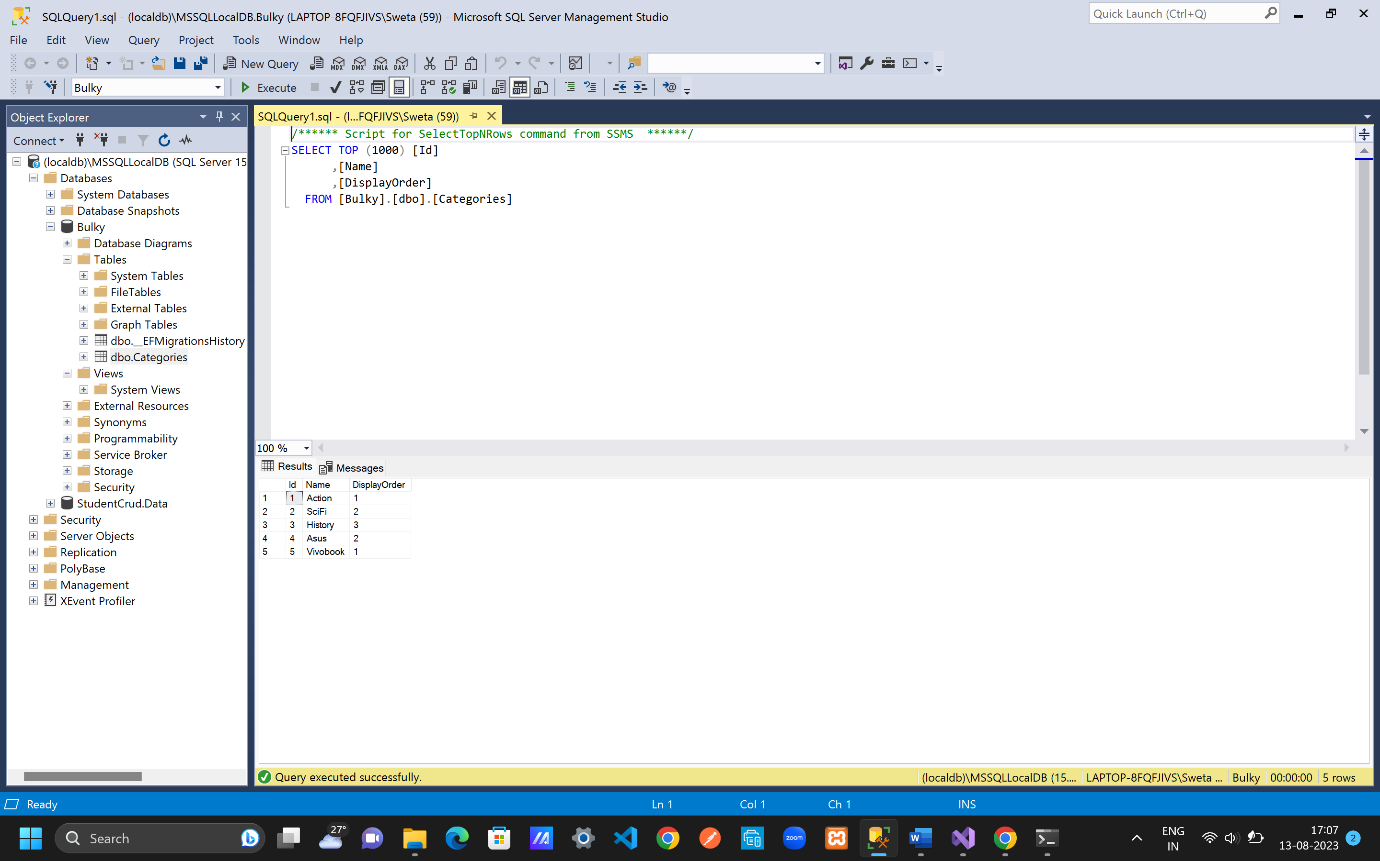
//return RedirectToAction("Index","Category");

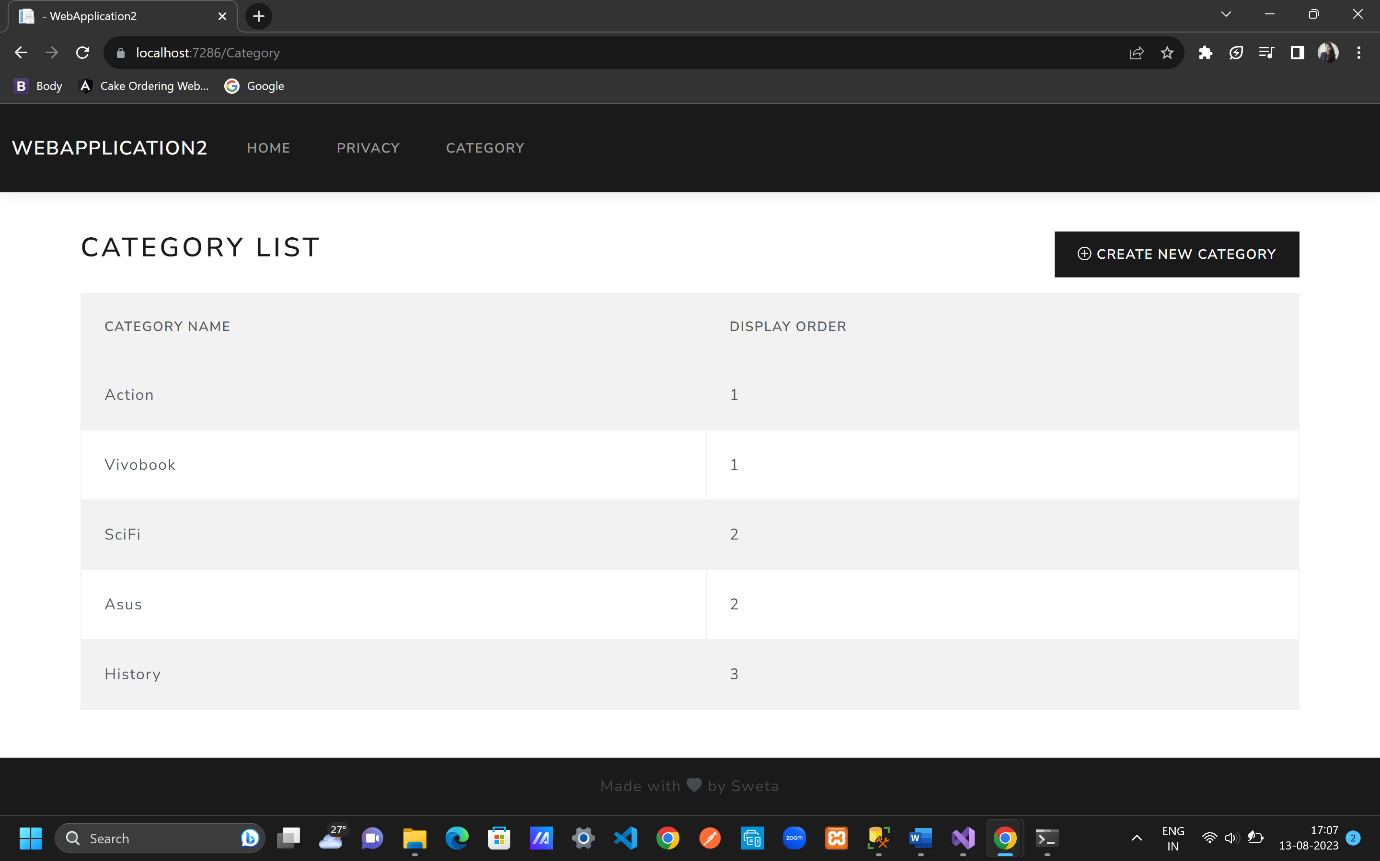
//-> if we have different controller then add like

//this else keep only menthod/function name

}

Data is retrieved from client side to server side





SERVER SIDE VALIDATIONS

Add some server side validation like range, maxlength of the labels

Category.cs

[MaxLength(30)]

[DisplayName("Category Name")]

public string Name { get; set; }

[DisplayName("Display Order")]

[Range(1, 100)]

public int DisplayOrder { get; set; }

CategoryController.cs

if(ModelState.IsValid) {

\_db.Categories.Add(obj); // what is to be add

\_db.SaveChanges(); //go to the database and create category

return RedirectToAction("Index"); //view all categories in index() line 9;

//return RedirectToAction("Index","Category");

//-> if we have different controller then add like

//this else keep only menthod/function name

}

return View();

Create.cshtml

<div class="mb-3 row p-1">

<**label** **asp-for**="Name" class="p-0"></**label**>

<**input** **asp-for**="Name" class="form-control"/>

<**span** **asp-validation-for**="Name" class="text-danger"></**span**>

</div>

<div class="mb-3 row p-1">

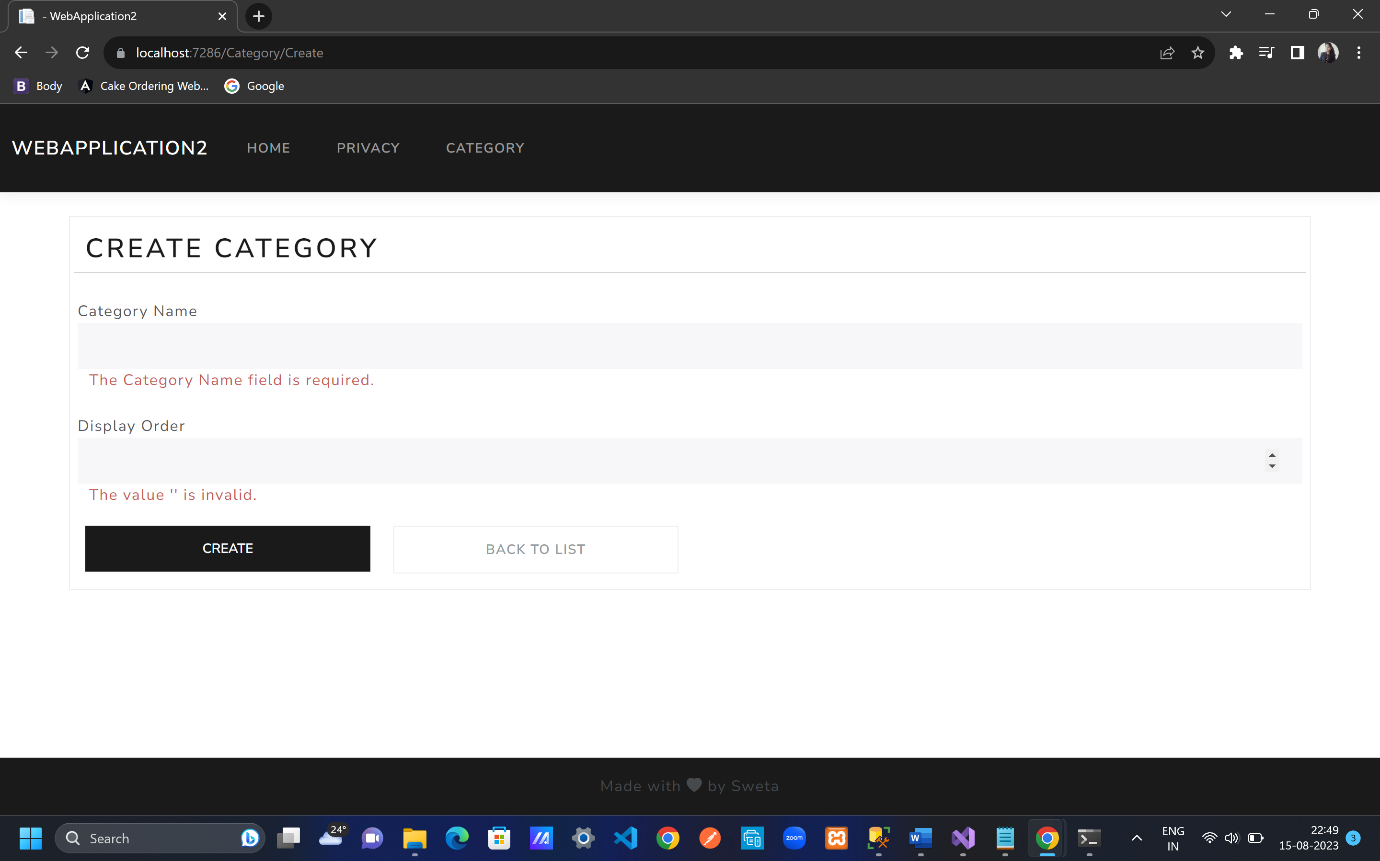
<**label** **asp-for**="DisplayOrder" class="p-0"></**label**>

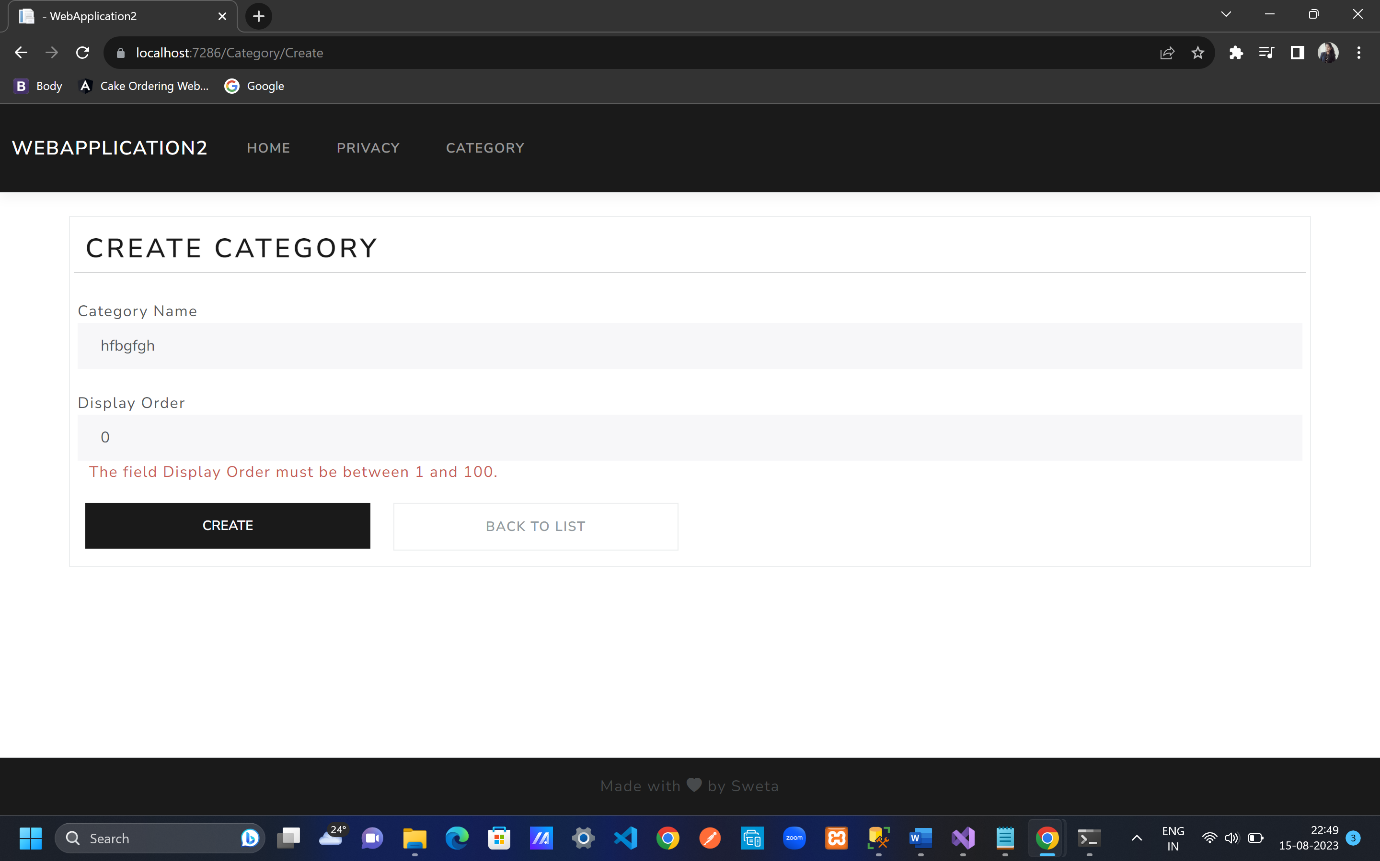
<**input** **asp-for**="DisplayOrder" class="form-control" />

<**span** **asp-validation-for**="DisplayOrder" class="text-danger"></**span**>

</div>

Added errors to the input field of labels with the help of asp.net tags i.e span tag



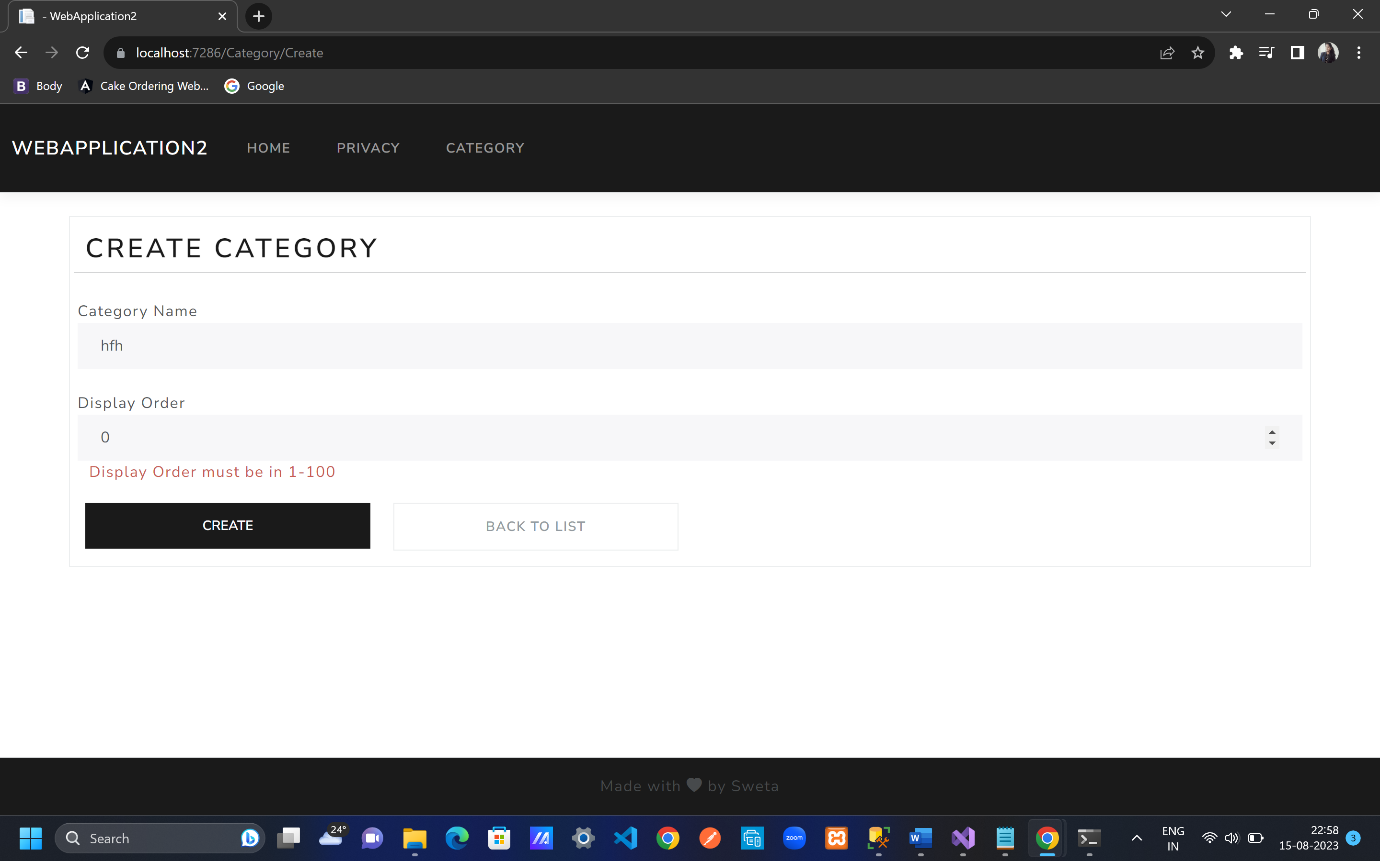


Custom error message

[DisplayName("Display Order")]

[Range(1, 100,ErrorMessage ="Display Order must be in 1-100")]

public int DisplayOrder { get; set; }



CUSTOM VALIDATIONS

In input field whatever asp-for is provided same shall be used in AddModelError() method from Category.cs

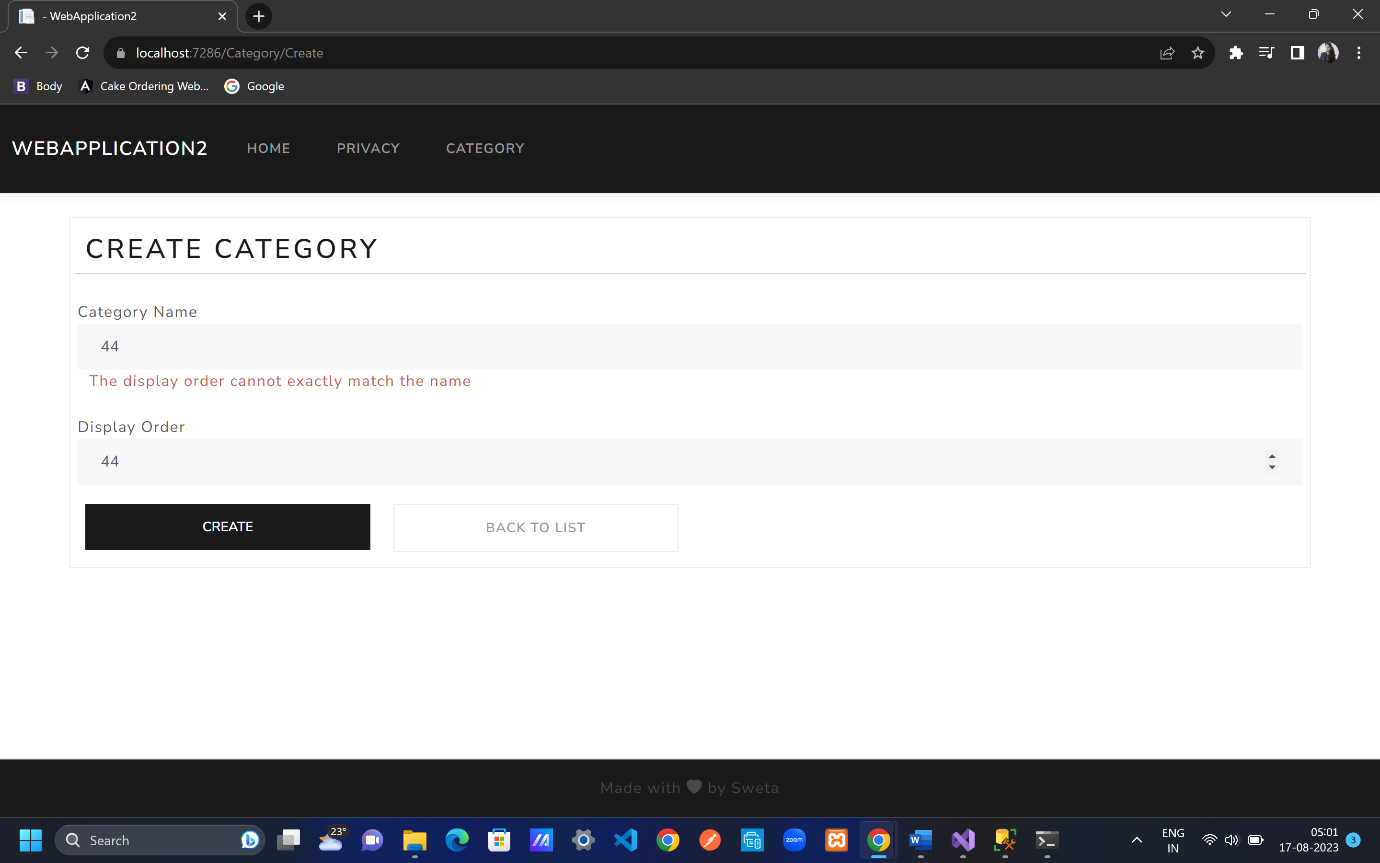
CategoryController.cs

if (obj.Name==obj.DisplayOrder.ToString())

{

ModelState.AddModelError("name", "The display order cannot exactly match the name");

}

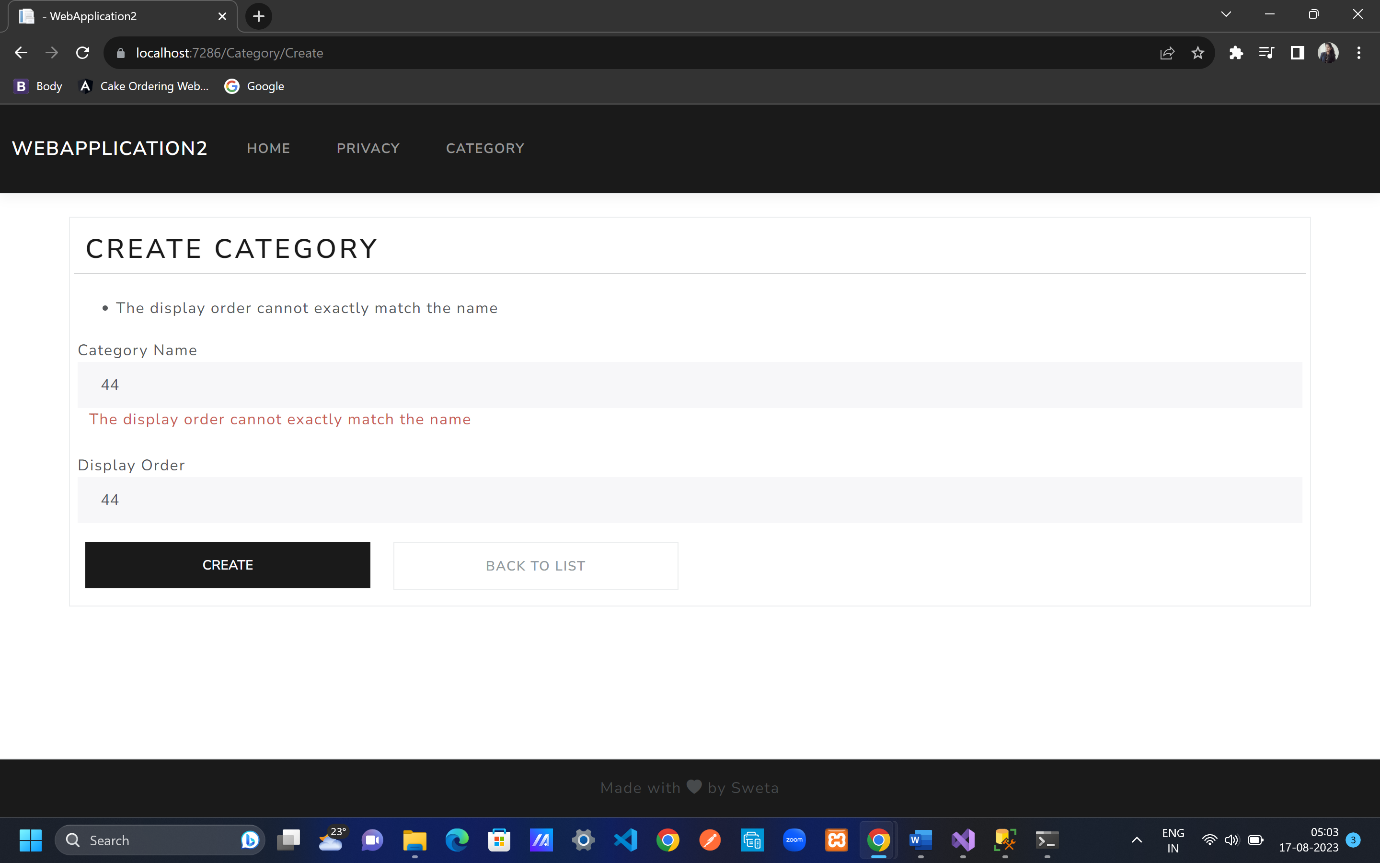


Create.cshtml

<**div** **asp-validation-summary**="All"></**div**>

Overall errors are in form of summary above of the webpage





ASP VALIDATION

CategoryController.cs

if (obj.Name==obj.DisplayOrder.ToString())

{

ModelState.AddModelError("name", "The display order cannot exactly match the name");

}

if (obj.Name!=null && obj.Name.ToLower() == "test")

{

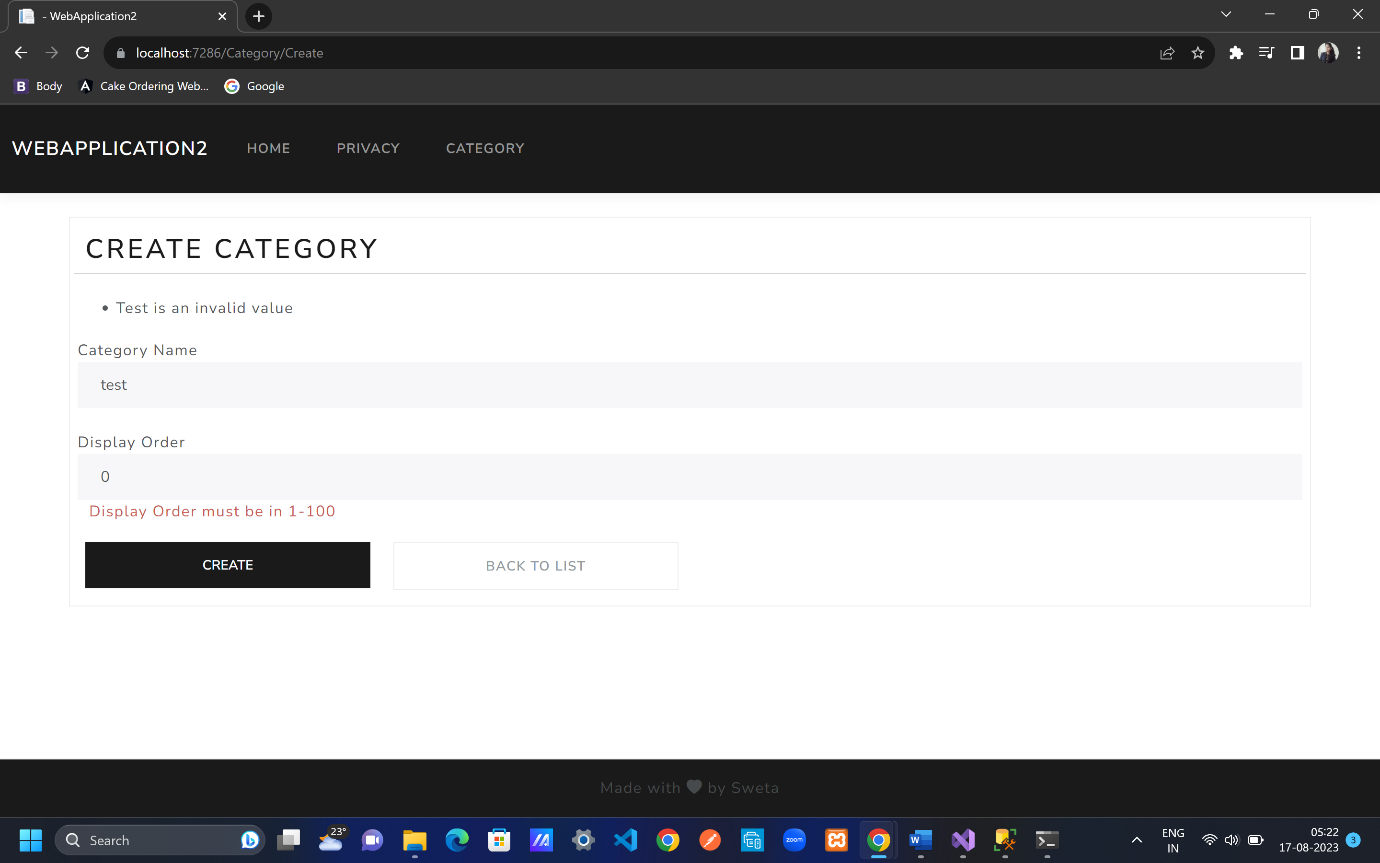
ModelState.AddModelError("", "Test is an invalid value");

}

Create.cshtml

<**div** **asp-validation-summary**="ModelOnly"></**div**>

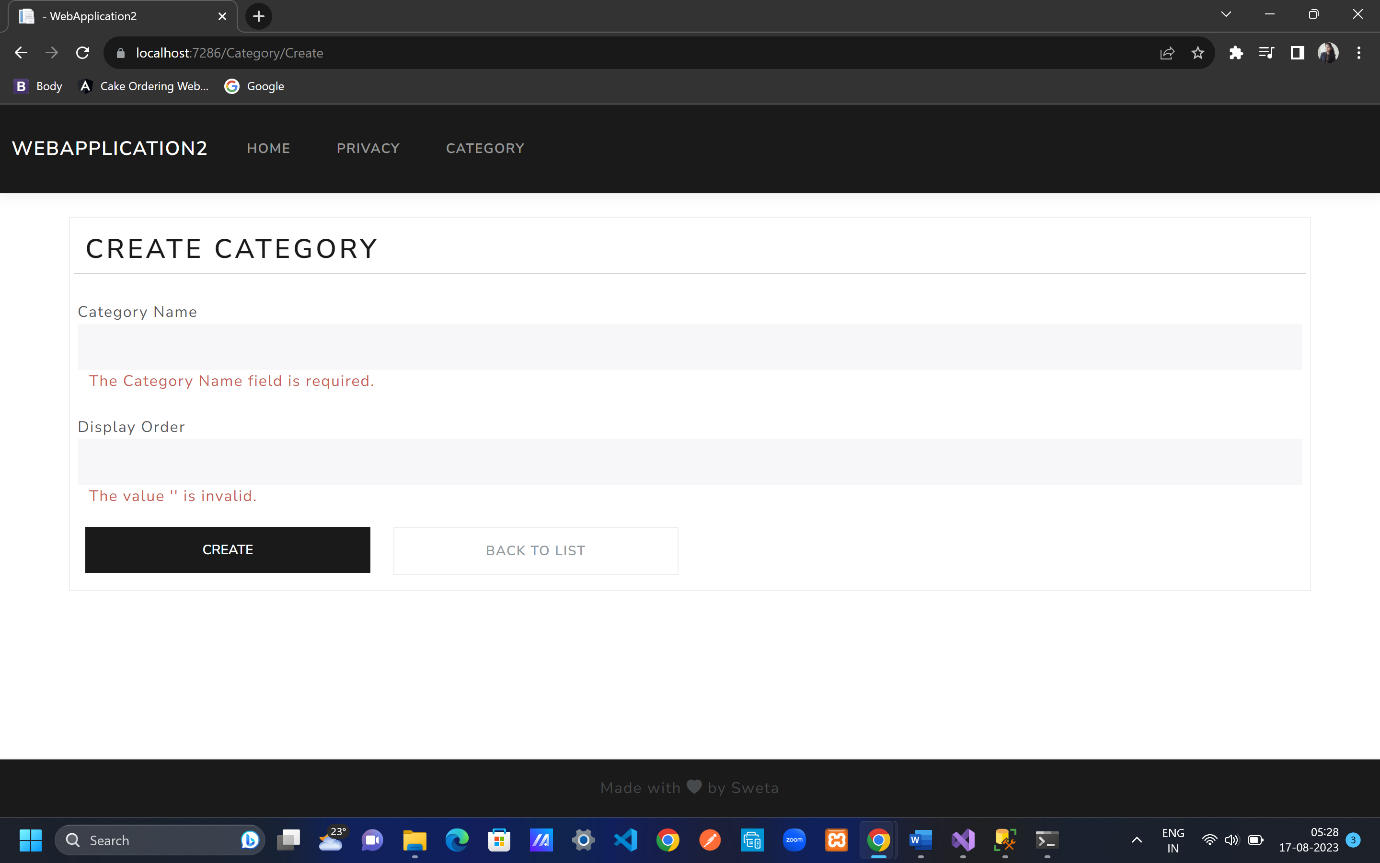
If key is not mentioned then in summary without key error messages are displayed above the webpage rest all are only displayed after the label tag



Create.cshtml

<**div** **asp-validation-summary**="None"></**div**>

No error messages are displayed above in the webpage



If required errors are not displayed with the asp-validation-summary=”All” then use custom validation asp-validation-summary=”ModelOnly”

Commenting some lines since it is not required for now

Create.cshtml

@\*<div asp-validation-summary="None"></div>\*@

CategoryController.cs

//if (obj.Name==obj.DisplayOrder.ToString())

//{

// ModelState.AddModelError("name", "The display order cannot exactly match the name");

//}

//if (obj.Name!=null && obj.Name.ToLower() == "test")

//{

// ModelState.AddModelError("", "Test is an invalid value");

//}

CLIENT SIDE VALIDATION

Create.cshtml

To get client side validation with partial file name provided by .net core in shared folder named with \_ValidationScriptsPartial.cshtml with javascripts is added in create.cshtml for the web page to be applied for all labels.

@section Scripts{

@{

<**partial** **name**="\_ValidationScriptsPartial"/> //add client side validation with all required javascripts

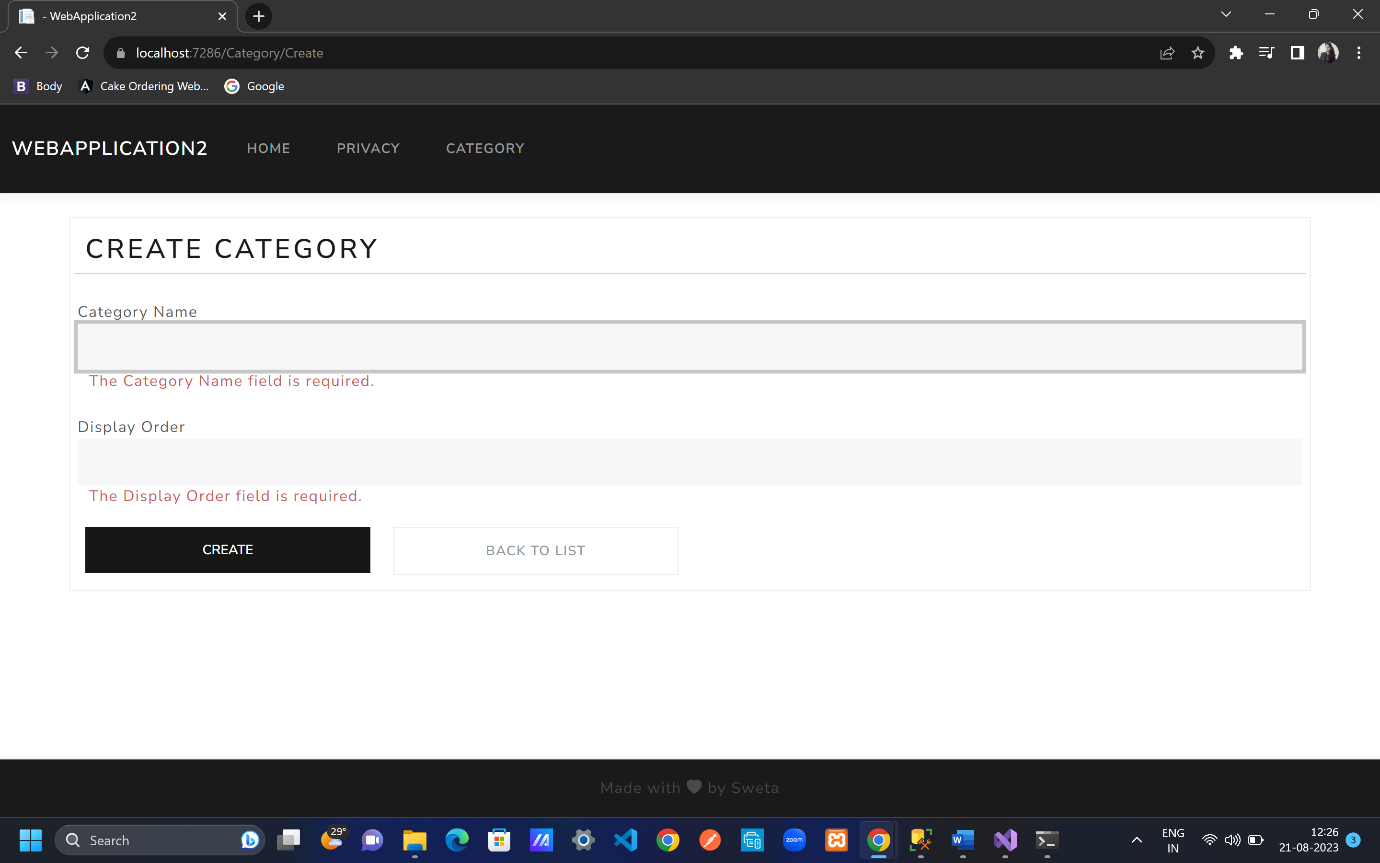
}

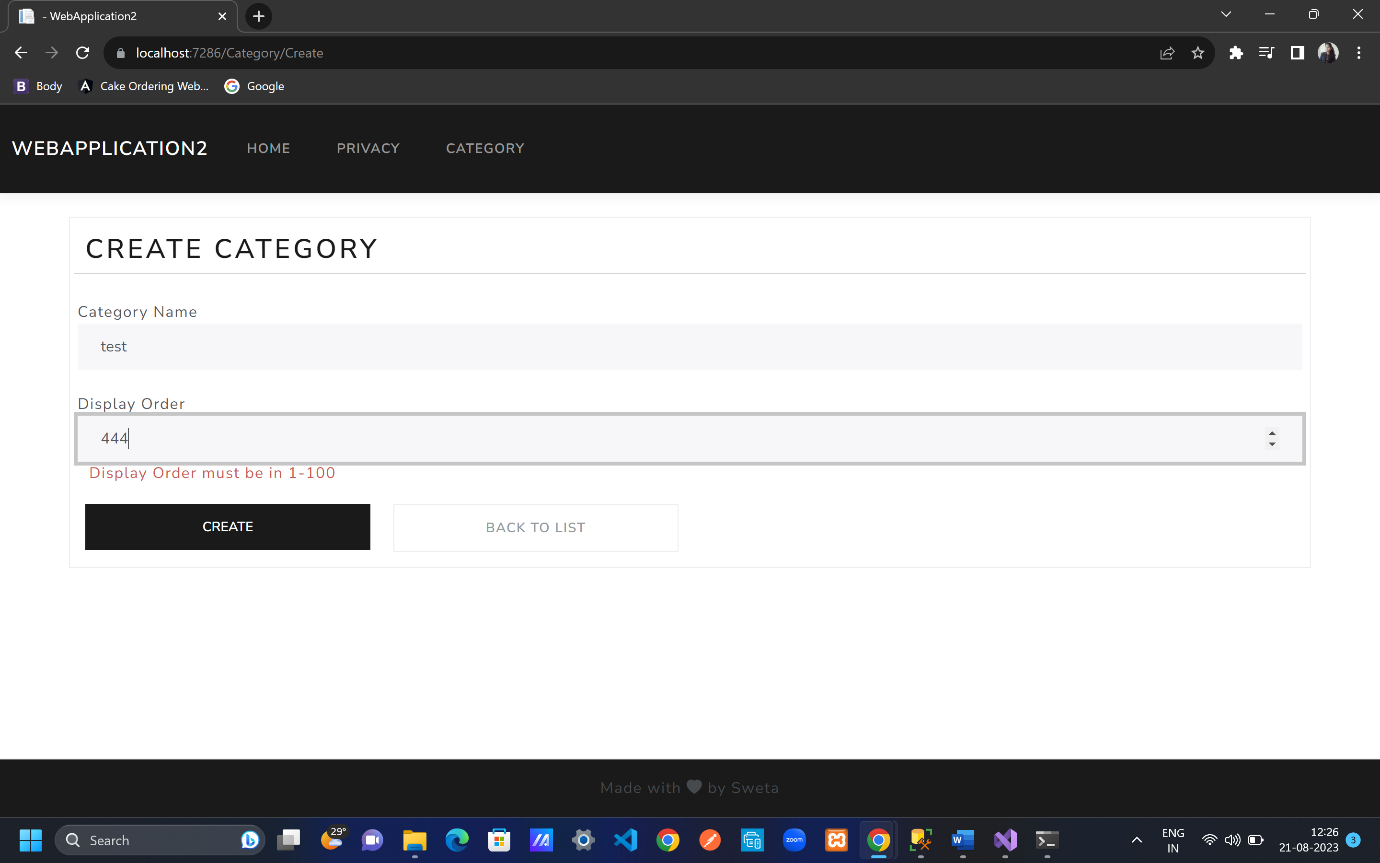
}

We can include <**partial** **name**="\_ValidationScriptsPartial"/> directly in form tag after all tags if want the client side validation in master page.

Firstly client side validation will take place and for custom validation server side validation takes place.

Client side validation





Server side validation

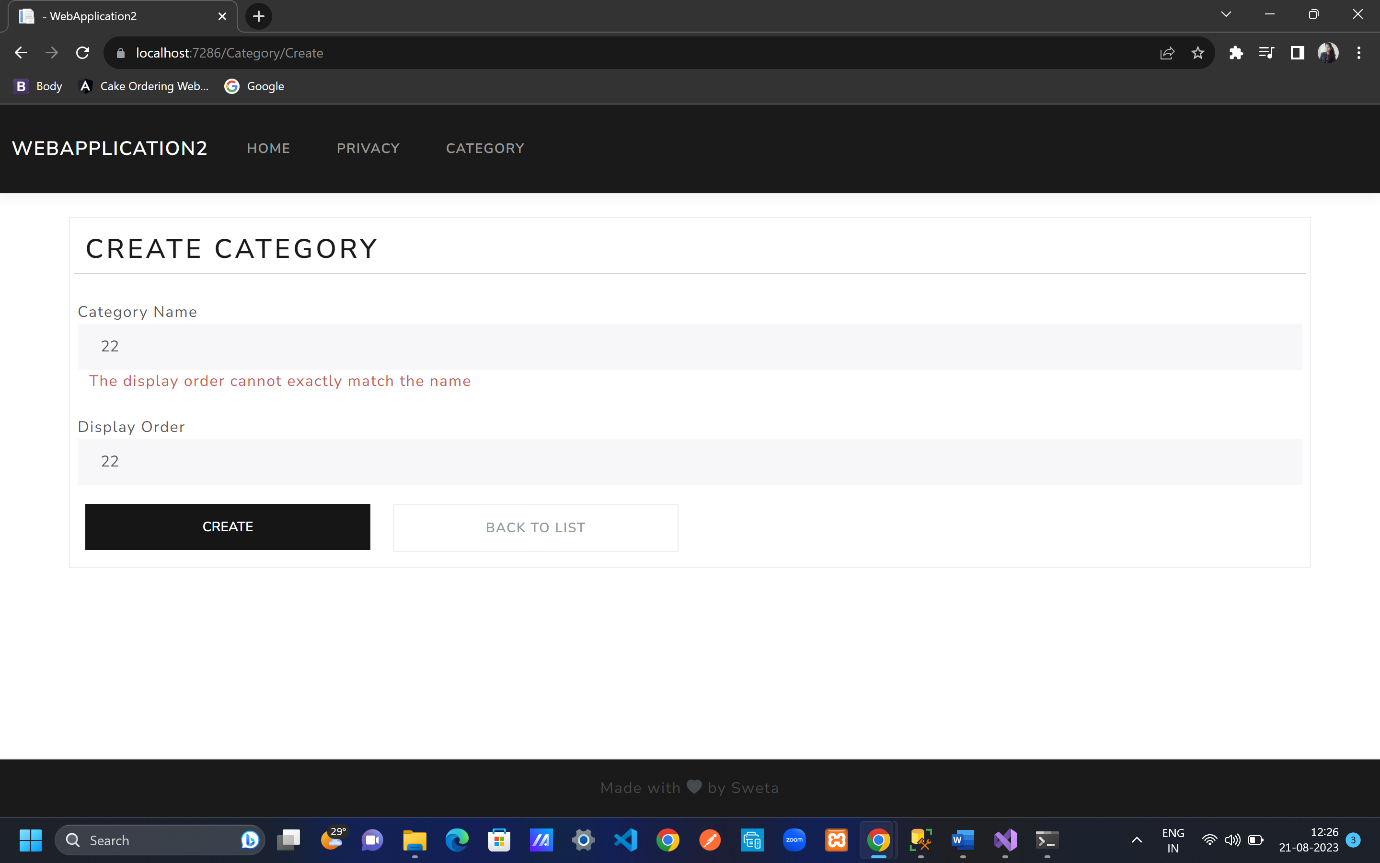
CategoryController.cs

if (obj.Name==obj.DisplayOrder.ToString())

{

ModelState.AddModelError("name", "The display order cannot exactly match the name");

}



EDIT AND DELETE BUTTONS

Go to bootstrap icons; search pencil; copy cdn link; search trash fill; copy cdn link

Views folder; Category folder;

Index.cshtml

<th></th>

<td>

<div class="w-75 btn-group" role="group">

<**a** **asp-controller**="Category" **asp-action**="Edit" class="btn btn-primary mx-2">

<i class="bi bi-pencil-square"></i>Edit

</**a**>

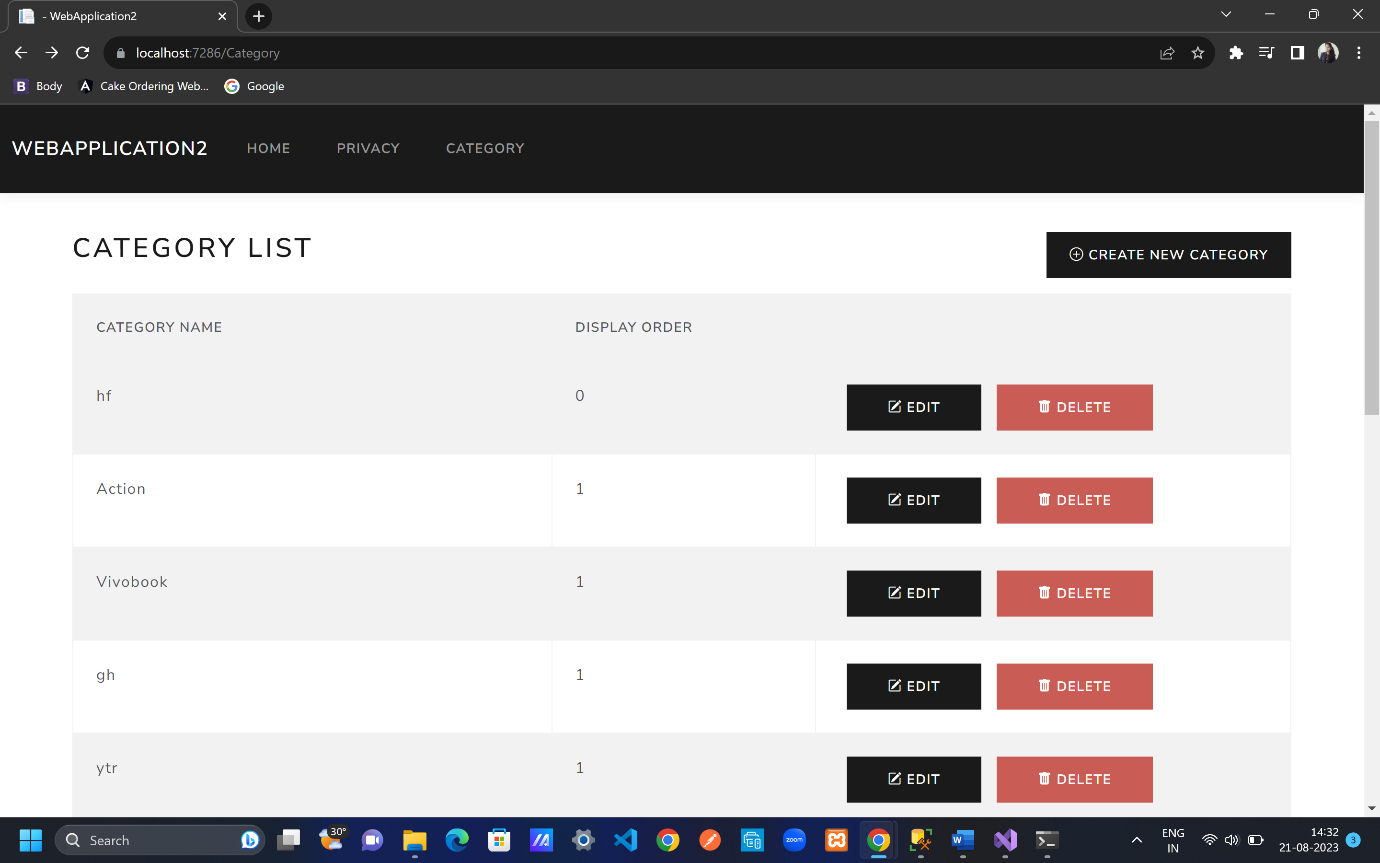
<**a** **asp-controller**="Category" **asp-action**="Delete" class="btn btn-danger mx-2">

<i class="bi bi-trash-fill"></i> Delete

</**a**>

</div>

</td>



GET CATEGORY DETAILS TO EDIT

CategoryController.cs

public IActionResult Edit(int? id)

{

if(id==null || id==0)

{

return NotFound();

}

Category categoryFromDb = \_db.Categories.Find(id);

if (categoryFromDb == null)

{

return NotFound();

}

return View(categoryFromDb);

}

[HttpPost]

public IActionResult Edit(Category obj) //getting data from category model

{

if (obj.Name == obj.DisplayOrder.ToString())

{

ModelState.AddModelError("name", "The display order cannot exactly match the name");

}

//if (obj.Name!=null && obj.Name.ToLower() == "test")

//{

// ModelState.AddModelError("", "Test is an invalid value");

//}

if (ModelState.IsValid)

{

\_db.Categories.Add(obj); // what is to be add

\_db.SaveChanges(); //go to the database and create category

return RedirectToAction("Index"); //view all categories in index() line 9;

//return RedirectToAction("Index","Category");

//-> if we have different controller then add like

//this else keep only menthod/function name

}

return View();

}

Index.cshtml

<div class="w-75 btn-group" role="group">

<**a** **asp-controller**="Category" **asp-action**="Edit" **asp-route-id**="@obj.Id" class="btn btn-primary mx-2">

<i class="bi bi-pencil-square"></i> Edit

</**a**>

**asp-route-id**; here name after asp-routr i.e id shall be same as in named in CategoryController.cs for an object

CategoryController.cs

Other 2 types to check with primary key for editing//updating the item i.e retrieve data from database.

//Category? categoryFromDb1 = \_db.Categories.FirstOrDefault(c=>c.Id==id);

//Category? categoryFromDb2 = \_db.Categories.Where(c => c.Id == id).FirstOrDefault();

UPDATE CATEGORY

CategoryController.cs

public IActionResult Edit(int? id)

right click on Edit; click on Add View; empty razor, Add; rename it with Edit.cshtml; hit Add

Edit.cshtml

@model Category // new category object with a default value

<**form** method="post">

<div class="border p-3 mt-4">

<div class="row pb-2">

<h2 class="text-primary">Edit Category</h2>

<hr />

</div>

@\*<div asp-validation-summary="None"></div>\*@

<div class="mb-3 row p-1">

<**label** **asp-for**="Name" class="p-0"></**label**>

<**input** **asp-for**="Name" class="form-control" />

<**span** **asp-validation-for**="Name" class="text-danger"></**span**>

</div>

<div class="mb-3 row p-1">

<**label** **asp-for**="DisplayOrder" class="p-0"></**label**>

<**input** **asp-for**="DisplayOrder" class="form-control" />

<**span** **asp-validation-for**="DisplayOrder" class="text-danger"></**span**>

</div>

<div class="row">

<div class="col-6 col-md-3">

<button type="submit" class="btn btn-primary form-control">Update</button>

</div>

<div class="col-6 col-md-3">

<**a** **asp-controller**="Category" **asp-action**="Index" class="btn btn-outline-secondary border form-control"> Back to List</**a**>

</div>

</div>

</div>

</**form**>

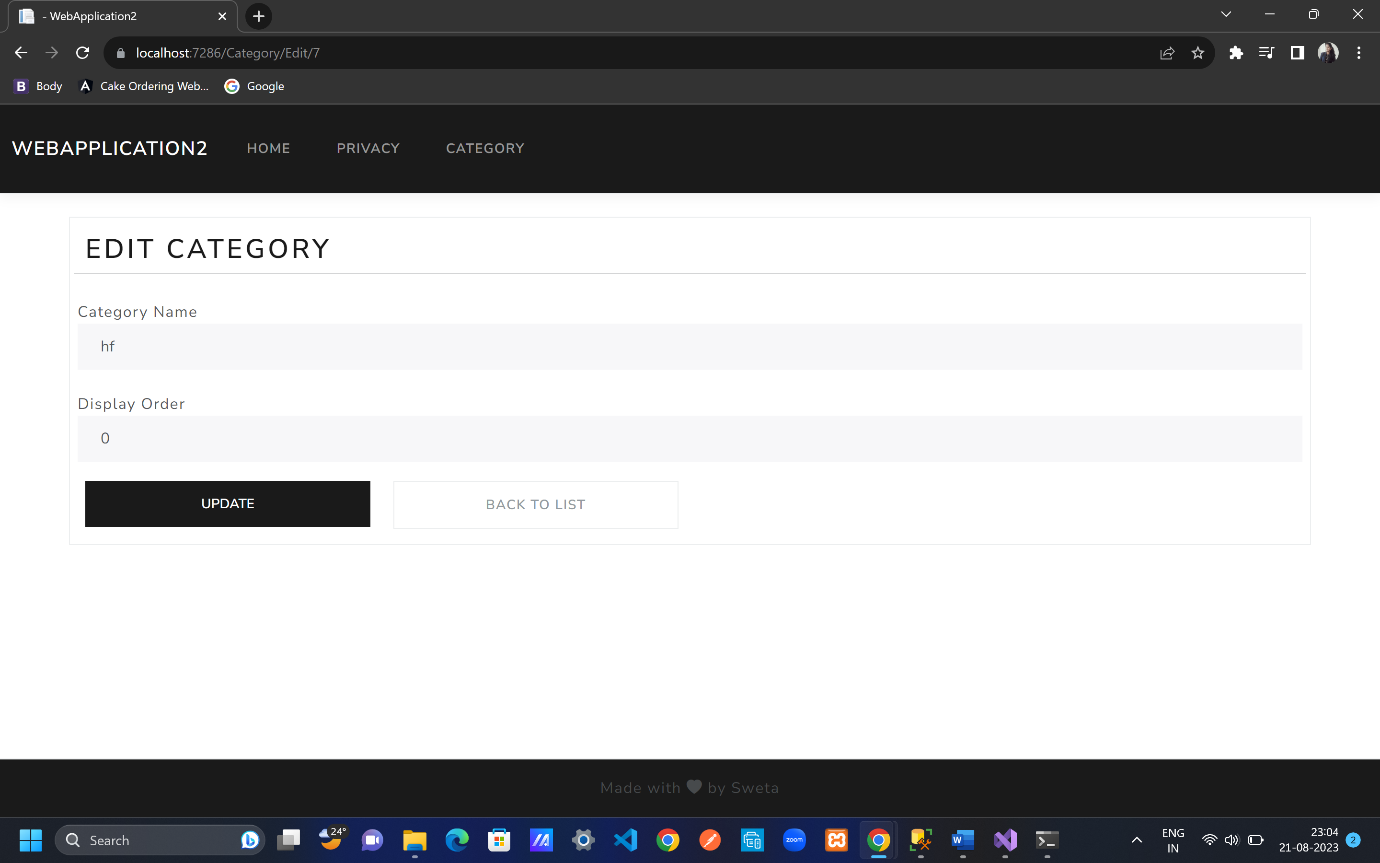
@section Scripts{

@{

<**partial** **name**="\_ValidationScriptsPartial" />

}

}



UPDATE CATEGORY IN ACTION

CategoryController.cs

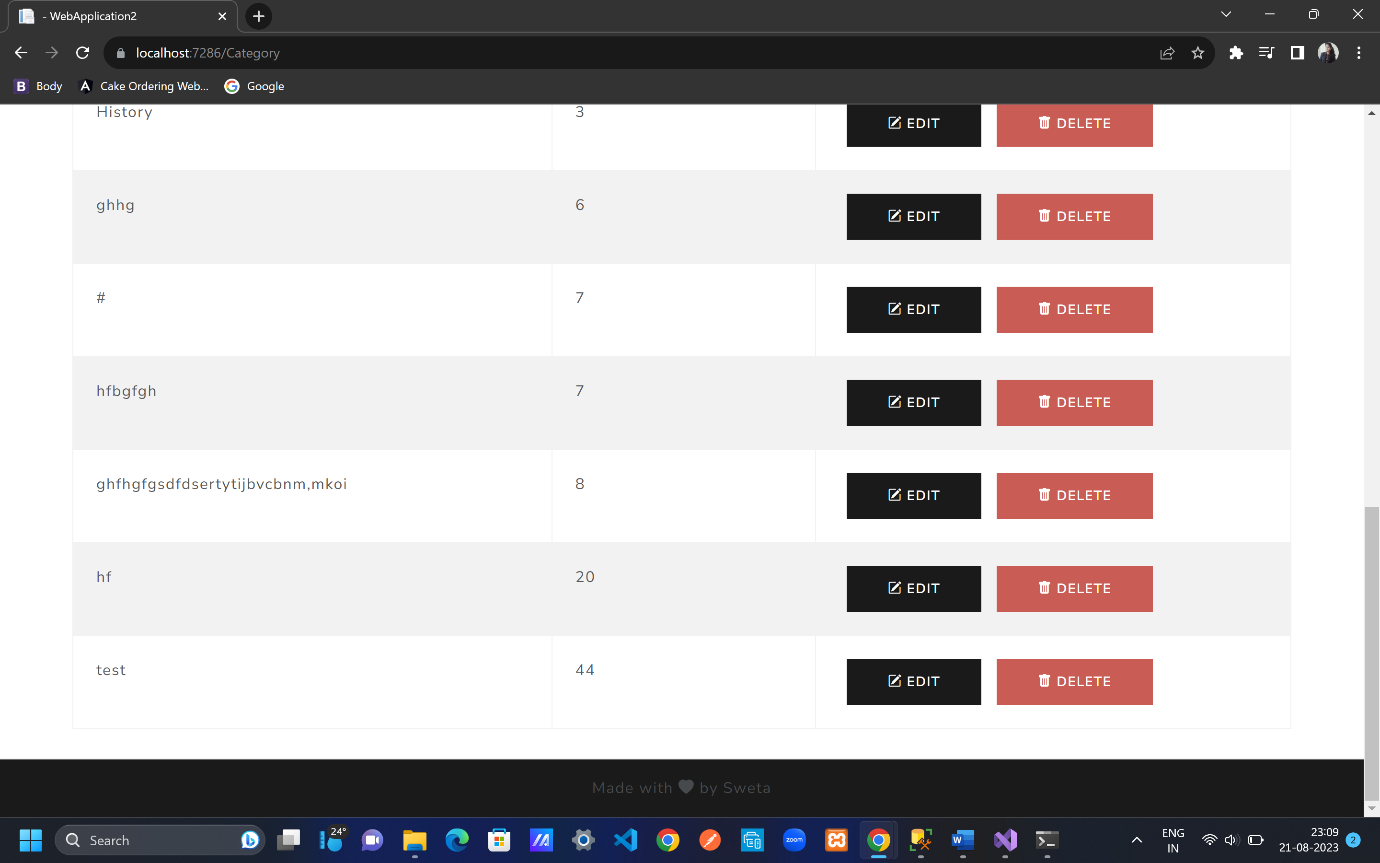
if (ModelState.IsValid)

{

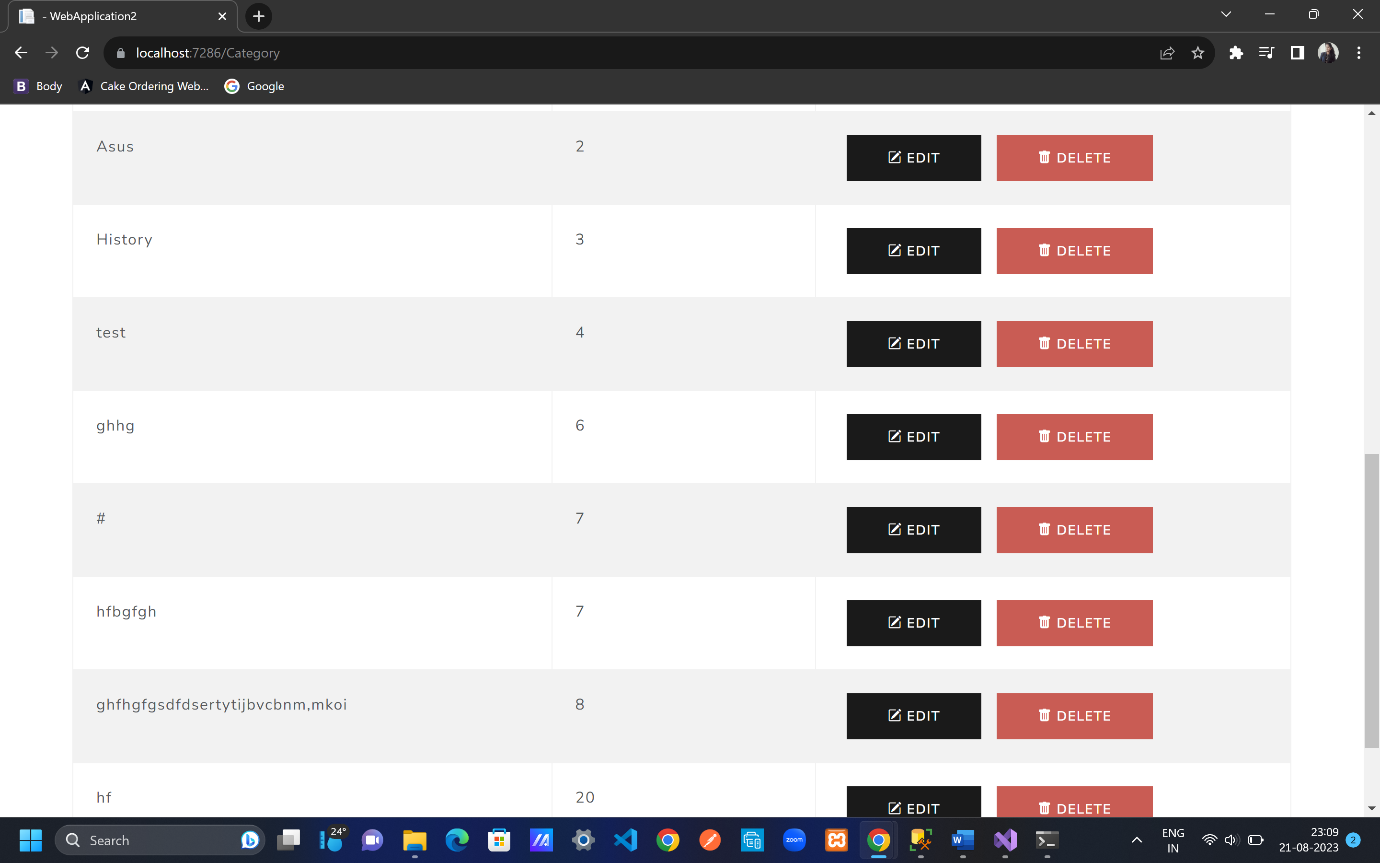
\_db.Categories.Update(obj); // what is to be add

\_db.SaveChanges();

Before editing; test 44



After editing and after changes, hit update button; test 4



GET AND POST ACTION FOR DELETE CATEGORY

Usually function name should be same since the parameter values are same that is why actionname is provided explicitly

CategoryController.cs

public IActionResult Delete(int? id)

{

if (id == null || id == 0)

{

return NotFound();

}

Category? categoryFromDb = \_db.Categories.Find(id);

//Category? categoryFromDb1 = \_db.Categories.FirstOrDefault(c=>c.Id==id);

//Category? categoryFromDb2 = \_db.Categories.Where(c => c.Id == id).FirstOrDefault();

if (categoryFromDb == null)

{

return NotFound();

}

return View(categoryFromDb);

}

[HttpPost, ActionName("Delete")]

public IActionResult DeletePOST(int? id) //getting data from category model

{

Category? obj= \_db.Categories.Find(id);

if (obj == null)

{

return NotFound();

}

\_db.Categories.Remove(obj);

\_db.SaveChanges(); //go to the database and create category

return RedirectToAction("Index"); //view all categories in index() line 9;

//return RedirectToAction("Index","Category");

//-> if we have different controller then add like

//this else keep only menthod/function name

}

DELETE CATEGORY IN ACTION

CategoryController.cs

public IActionResult Delete(int? id)

right click Delete;Add view; empty view – Add; rename with Delete.cshtml; Add

for delete we don’t need any validation

Delete.cshtml

@model Category // new category object with a default value

<**form** method="post">

<**input** **asp-for**="Id" hidden/>

<div class="border p-3 mt-4">

<div class="row pb-2">

<h2 class="text-primary">Delete Category</h2>

<hr />

</div>

@\*<div asp-validation-summary="None"></div>\*@

<div class="mb-3 row p-1">

<**label** **asp-for**="Name" class="p-0"></**label**>

<**input** **asp-for**="Name" class="form-control" />

</div>

<div class="mb-3 row p-1">

<**label** **asp-for**="DisplayOrder" class="p-0"></**label**>

<**input** **asp-for**="DisplayOrder" class="form-control" />

</div>

<div class="row">

<div class="col-6 col-md-3">

<button type="submit" class="btn btn-danger form-control">Delete</button>

</div>

<div class="col-6 col-md-3">

<**a** **asp-controller**="Category" **asp-action**="Index" class="btn btn-outline-secondary border form-control"> Back to List</**a**>

</div>

</div>

</div>

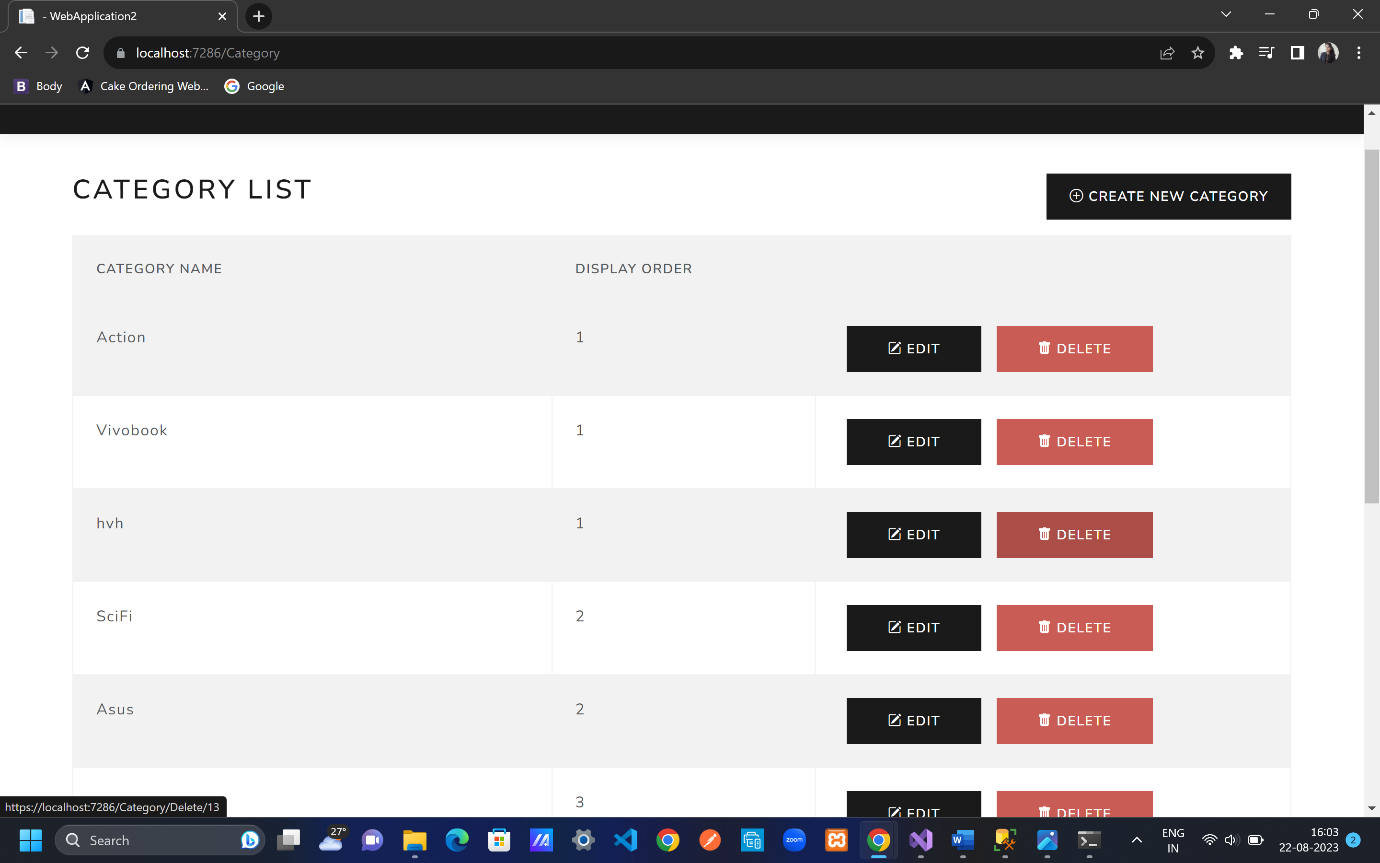
</**form**>

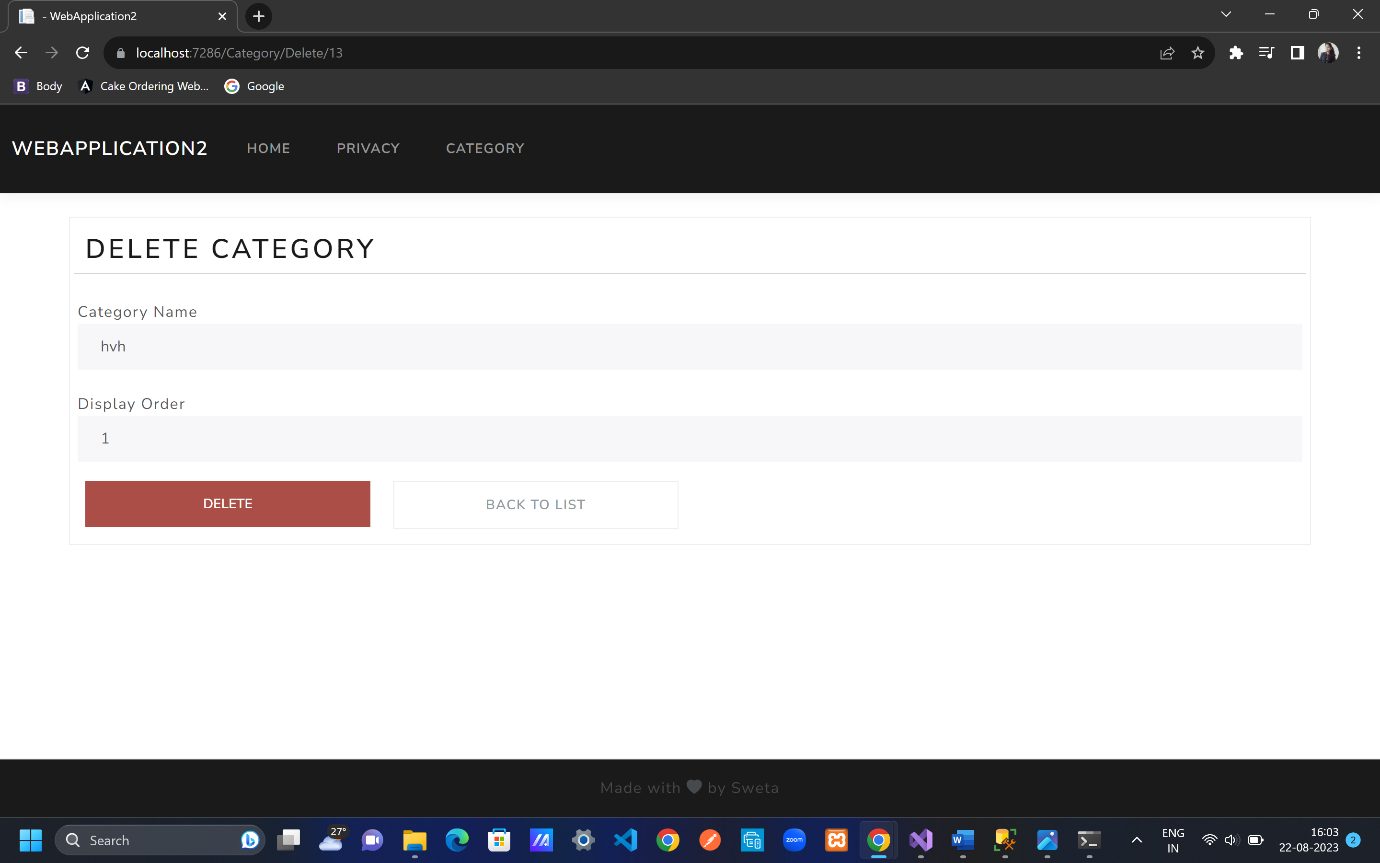
Index.cshtml

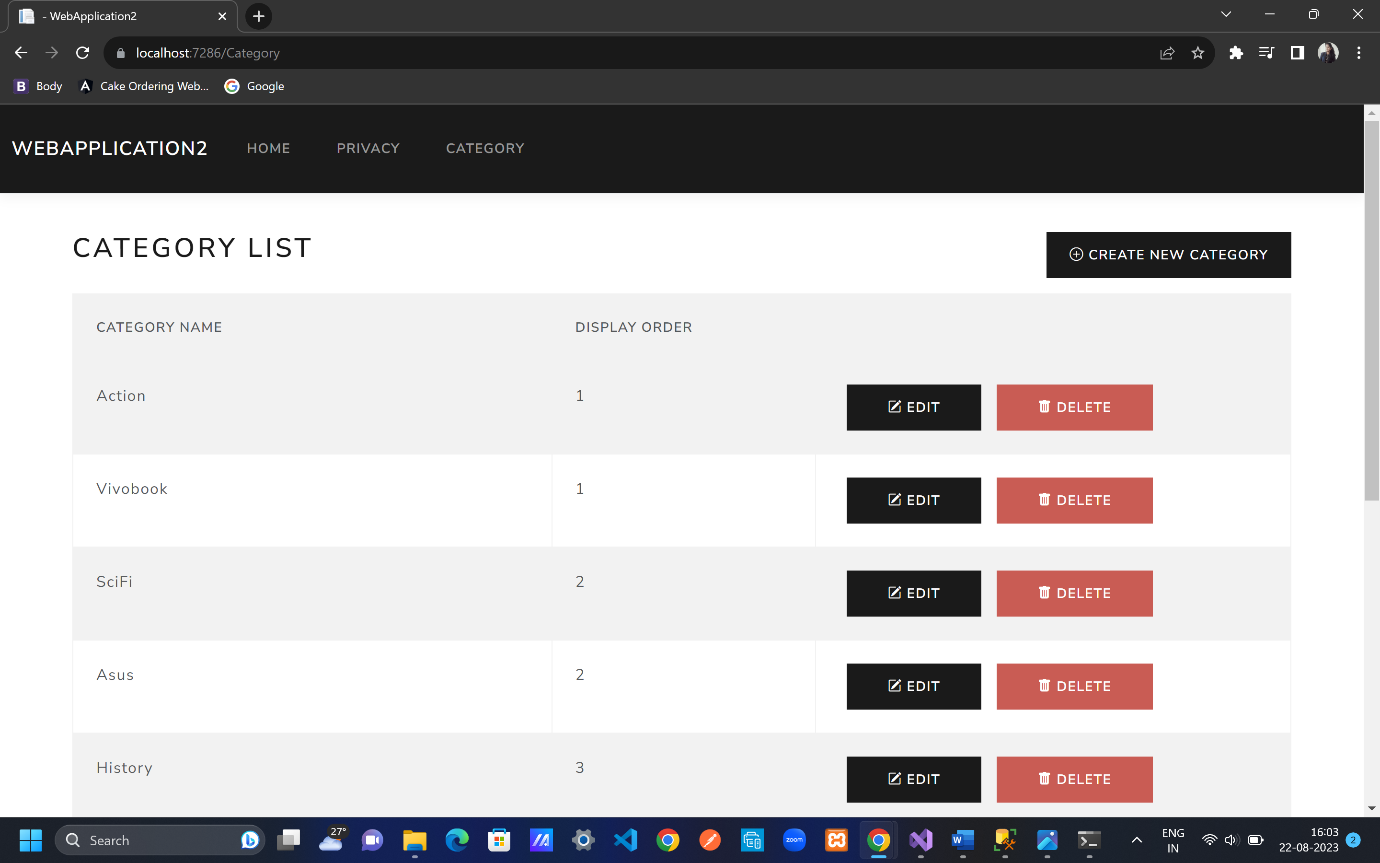
<**a** **asp-controller**="Category" **asp-action**="Delete" **asp-route-id**="@obj.Id" class="btn btn-danger mx-2">

<i class="bi bi-trash-fill"></i> Delete

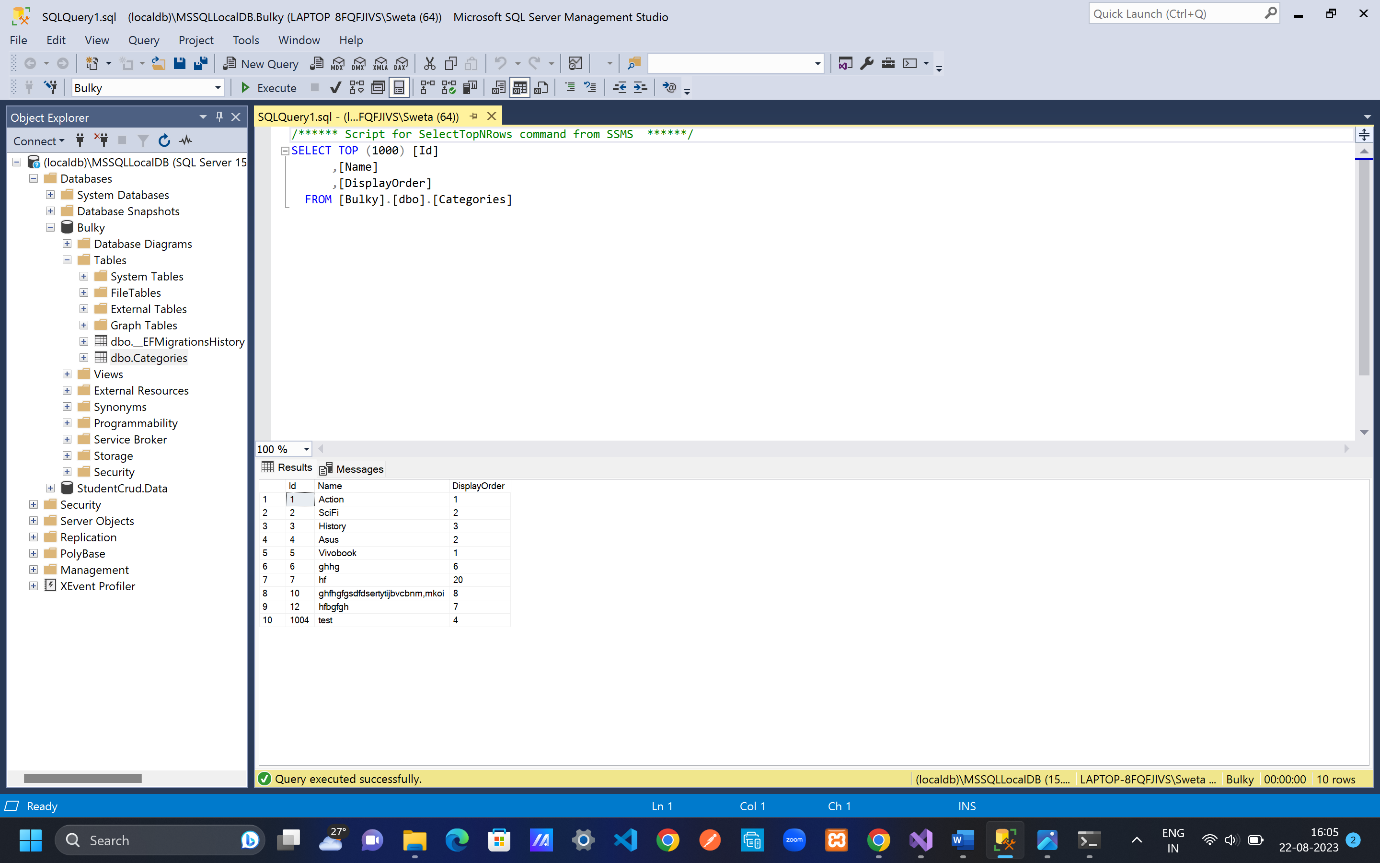
</**a**>







To check in database



TempData

Add tempdata in all crud functions

CategoryController.cs

using Microsoft.AspNetCore.Mvc;

using WebApplication2.Data;

using WebApplication2.Models;

namespace WebApplication2.Controllers

{

public class CategoryController : Controller

{

public IActionResult Index()

{

List<Category> objCategoryList=\_db.Categories.ToList();

return View(objCategoryList);

}

private readonly ApplicationDbContext \_db;

public CategoryController(ApplicationDbContext db)

{

\_db= db;

}

public IActionResult Create()

{

return View();

}

[HttpPost]

public IActionResult Create(Category obj) //getting data from category model

{

if (obj.Name==obj.DisplayOrder.ToString())

{

ModelState.AddModelError("name", "The display order cannot exactly match the name");

}

//if (obj.Name!=null && obj.Name.ToLower() == "test")

//{

// ModelState.AddModelError("", "Test is an invalid value");

//}

if (ModelState.IsValid)

{

\_db.Categories.Add(obj); // what is to be add

\_db.SaveChanges(); //go to the database and create category

TempData["success"] = "Category created successfully";

return RedirectToAction("Index"); //view all categories in index() line 9;

//return RedirectToAction("Index","Category");

//-> if we have different controller then add like

//this else keep only menthod/function name

}

return View();

}

public IActionResult Edit(int? id)

{

if(id==null || id==0)

{

return NotFound();

}

Category? categoryFromDb = \_db.Categories.Find(id);

//Category? categoryFromDb1 = \_db.Categories.FirstOrDefault(c=>c.Id==id);

//Category? categoryFromDb2 = \_db.Categories.Where(c => c.Id == id).FirstOrDefault();

if (categoryFromDb == null)

{

return NotFound();

}

return View(categoryFromDb);

}

[HttpPost]

public IActionResult Edit(Category obj) //getting data from category model

{

if (obj.Name == obj.DisplayOrder.ToString())

{

ModelState.AddModelError("name", "The display order cannot exactly match the name");

}

//if (obj.Name!=null && obj.Name.ToLower() == "test")

//{

// ModelState.AddModelError("", "Test is an invalid value");

//}

if (ModelState.IsValid)

{

\_db.Categories.Update(obj); // what is to be add

\_db.SaveChanges(); //go to the database and create category

TempData["success"] = "Category updated successfully";

return RedirectToAction("Index"); //view all categories in index() line 9;

//return RedirectToAction("Index","Category");

//-> if we have different controller then add like

//this else keep only menthod/function name

}

return View();

}

public IActionResult Delete(int? id)

{

if (id == null || id == 0)

{

return NotFound();

}

Category? categoryFromDb = \_db.Categories.Find(id);

//Category? categoryFromDb1 = \_db.Categories.FirstOrDefault(c=>c.Id==id);

//Category? categoryFromDb2 = \_db.Categories.Where(c => c.Id == id).FirstOrDefault();

if (categoryFromDb == null)

{

return NotFound();

}

return View(categoryFromDb);

}

[HttpPost, ActionName("Delete")]

public IActionResult DeletePOST(int? id) //getting data from category model

{

Category? obj= \_db.Categories.Find(id);

if (obj == null)

{

return NotFound();

}

\_db.Categories.Remove(obj);

\_db.SaveChanges(); //go to the database and create category

TempData["success"] = "Category deleted successfully";

return RedirectToAction("Index"); //view all categories in index() line 9;

//return RedirectToAction("Index","Category");

//-> if we have different controller then add like

//this else keep only menthod/function name

}

}

Index.cshtml

To get message for other view pages as well add this for example in product, home, contact us page

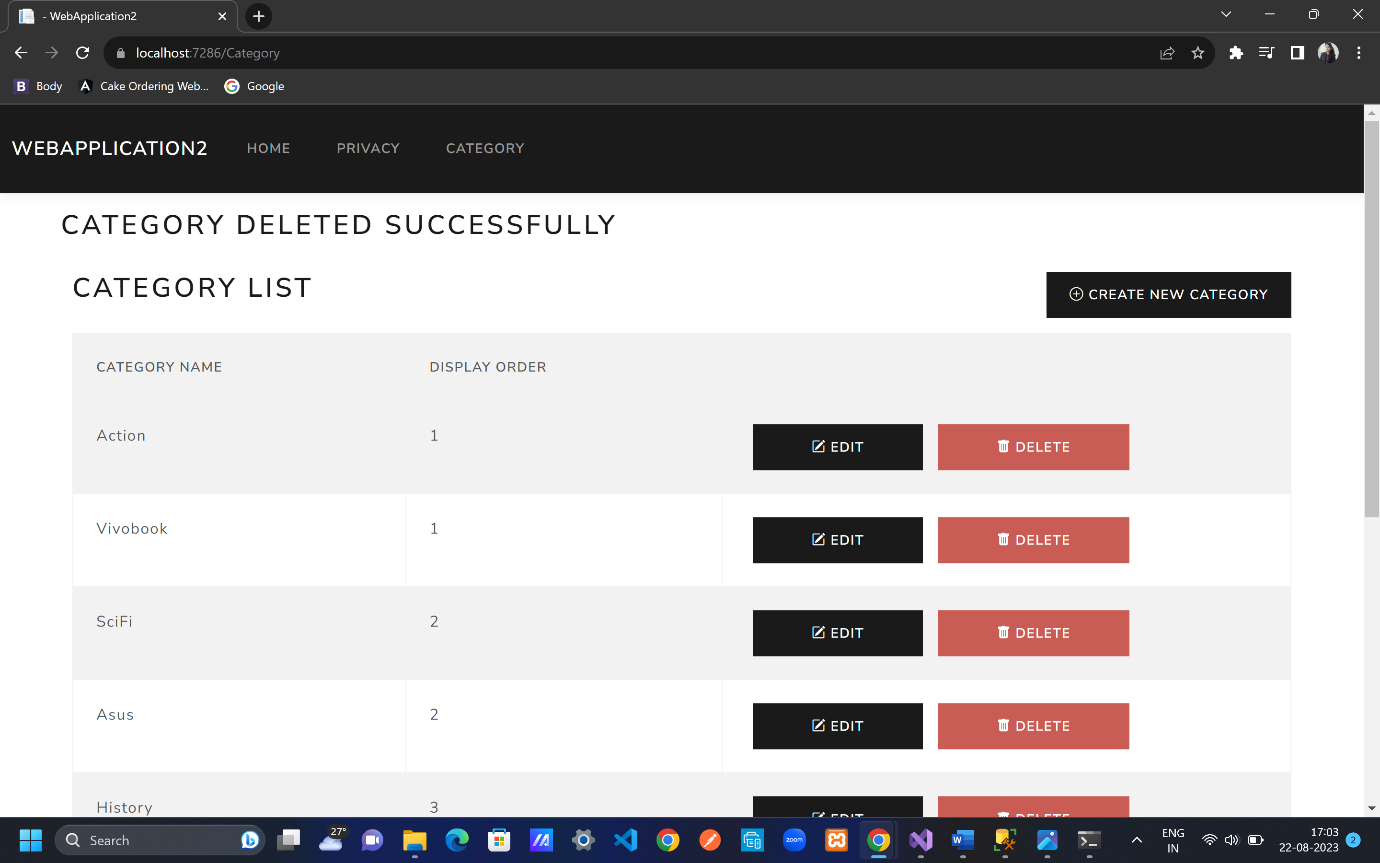
@if (TempData["success"]!=null)

{

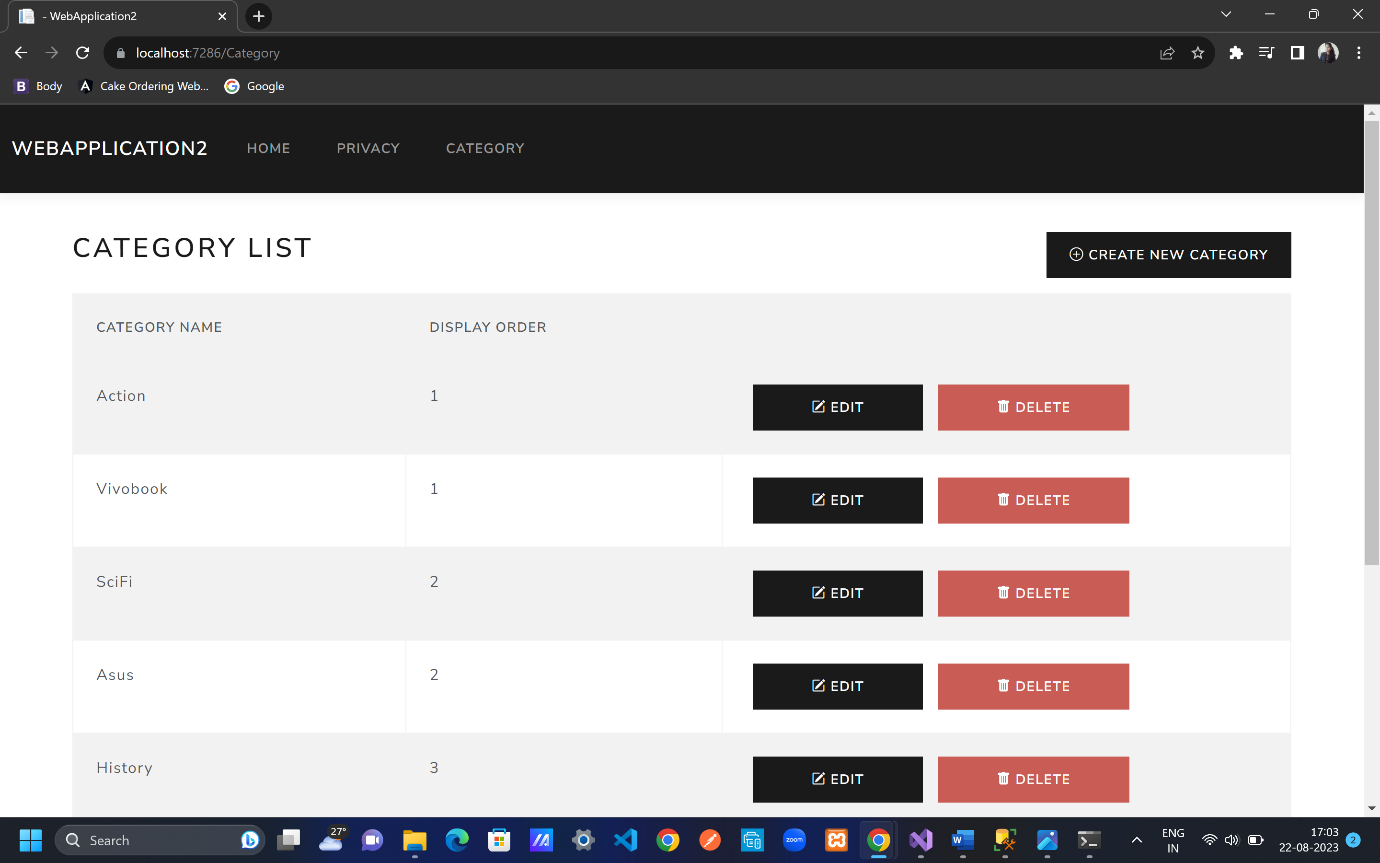
<h2>@TempData["success"]</h2>

}

Since typing everywhere in all seperate web pages we can use partial view as we are using validatonscriptspartial for validations



Tempdata always stays for one request once refreshed or other click is done then it disappears



PARTIAL VIEWS

Right click on Shared folder; Add; View; empty view Add; rename it by \_Notification.cshtml

Remove from Index.cshtml

@if (TempData["success"]!=null)

{

<h2>@TempData["success"]</h2>

}

\_Notification.cshtml

@if (TempData["success"]!=null)

{

<h2>@TempData["success"]</h2>

}

//as of now this is not required

@if (TempData["error"] != null)

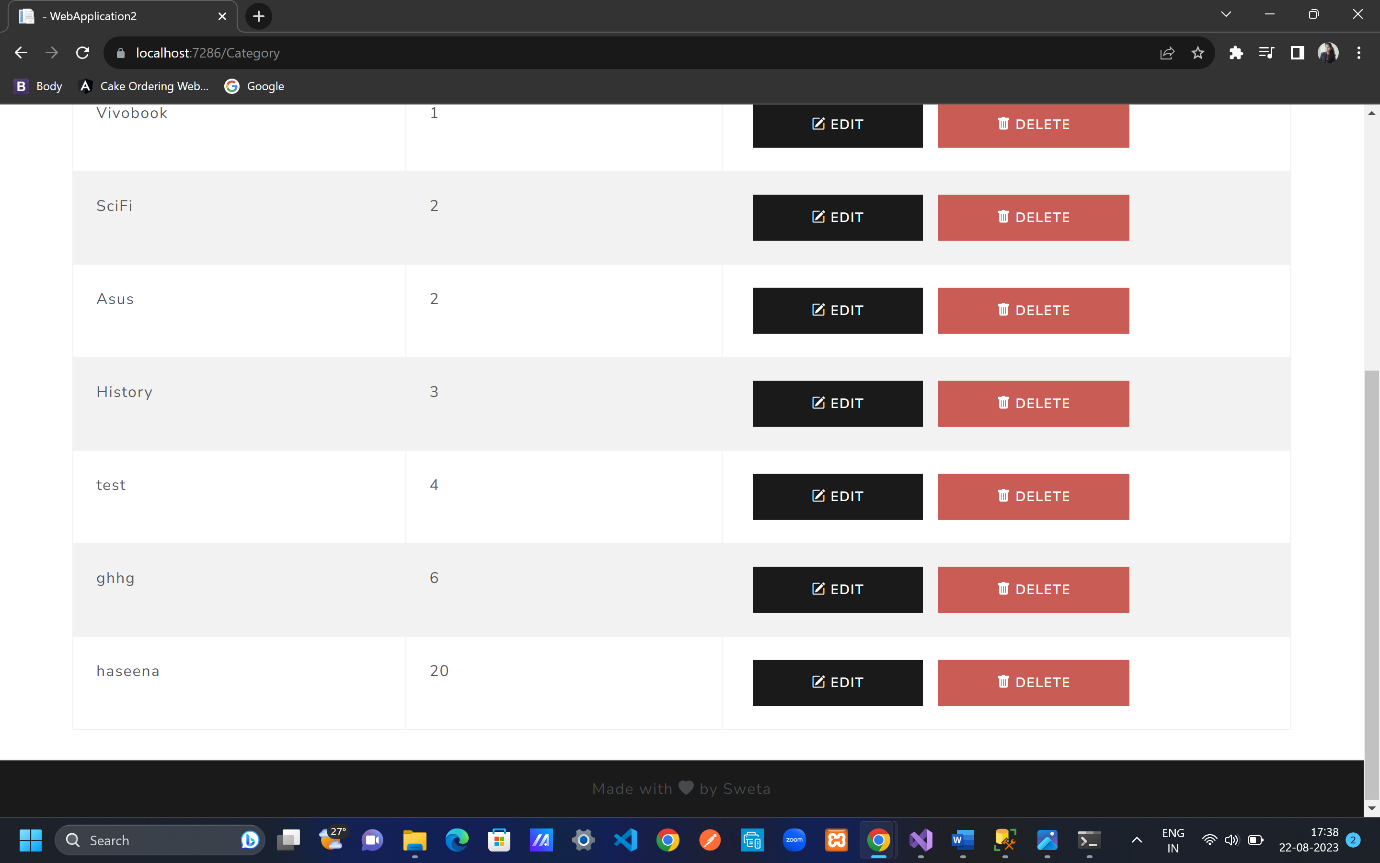
{

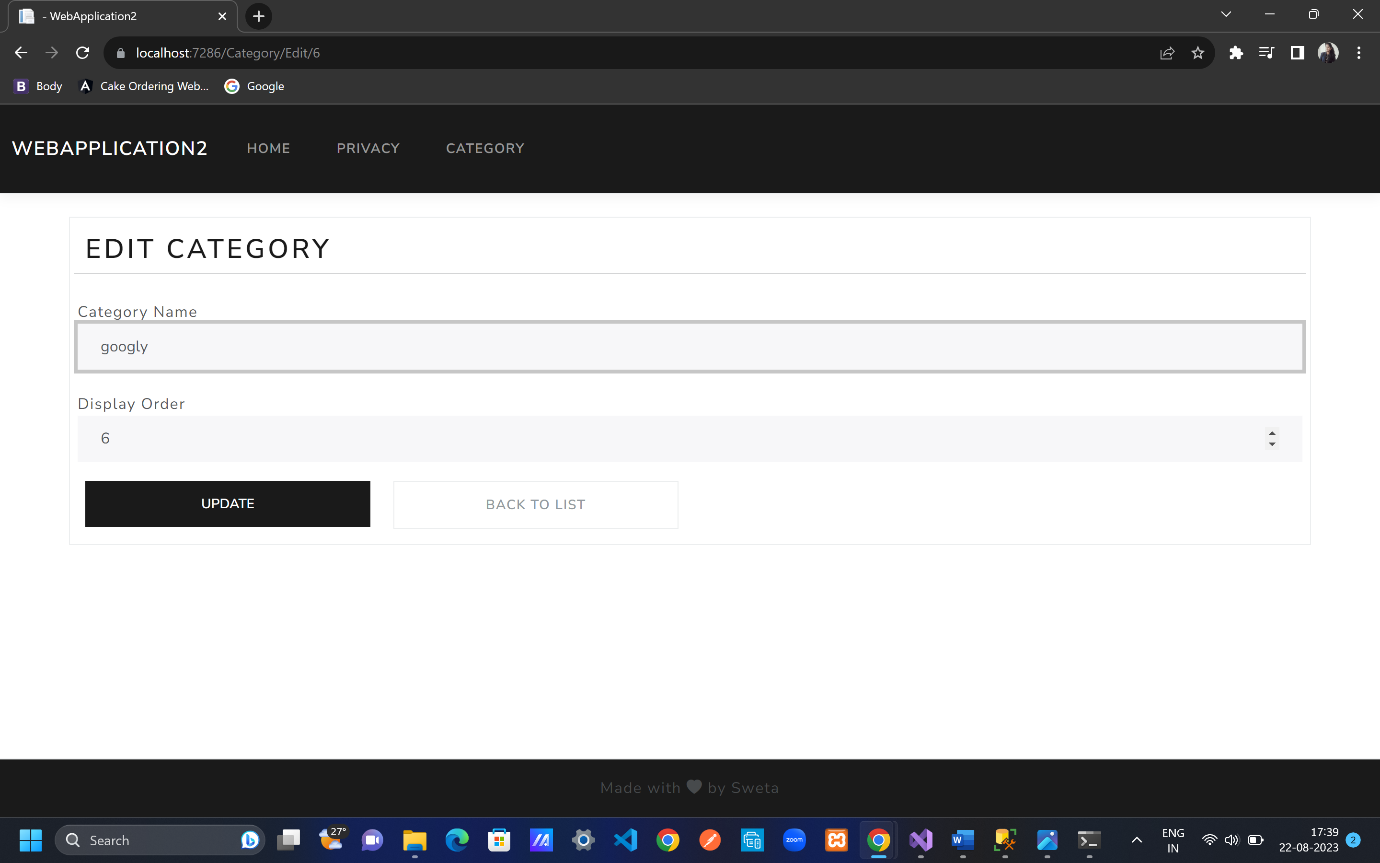
<h2>@TempData["error"]</h2>

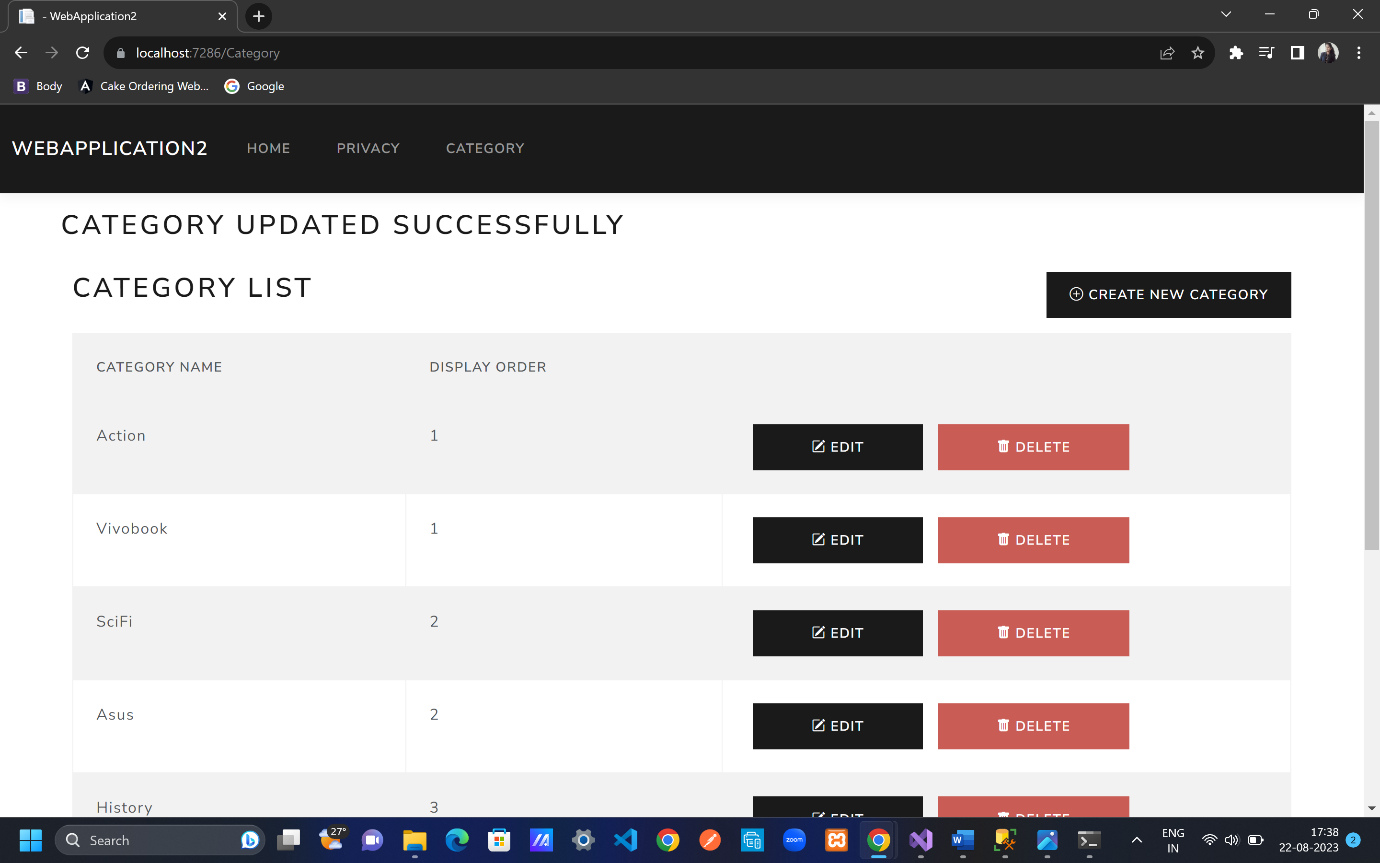
}

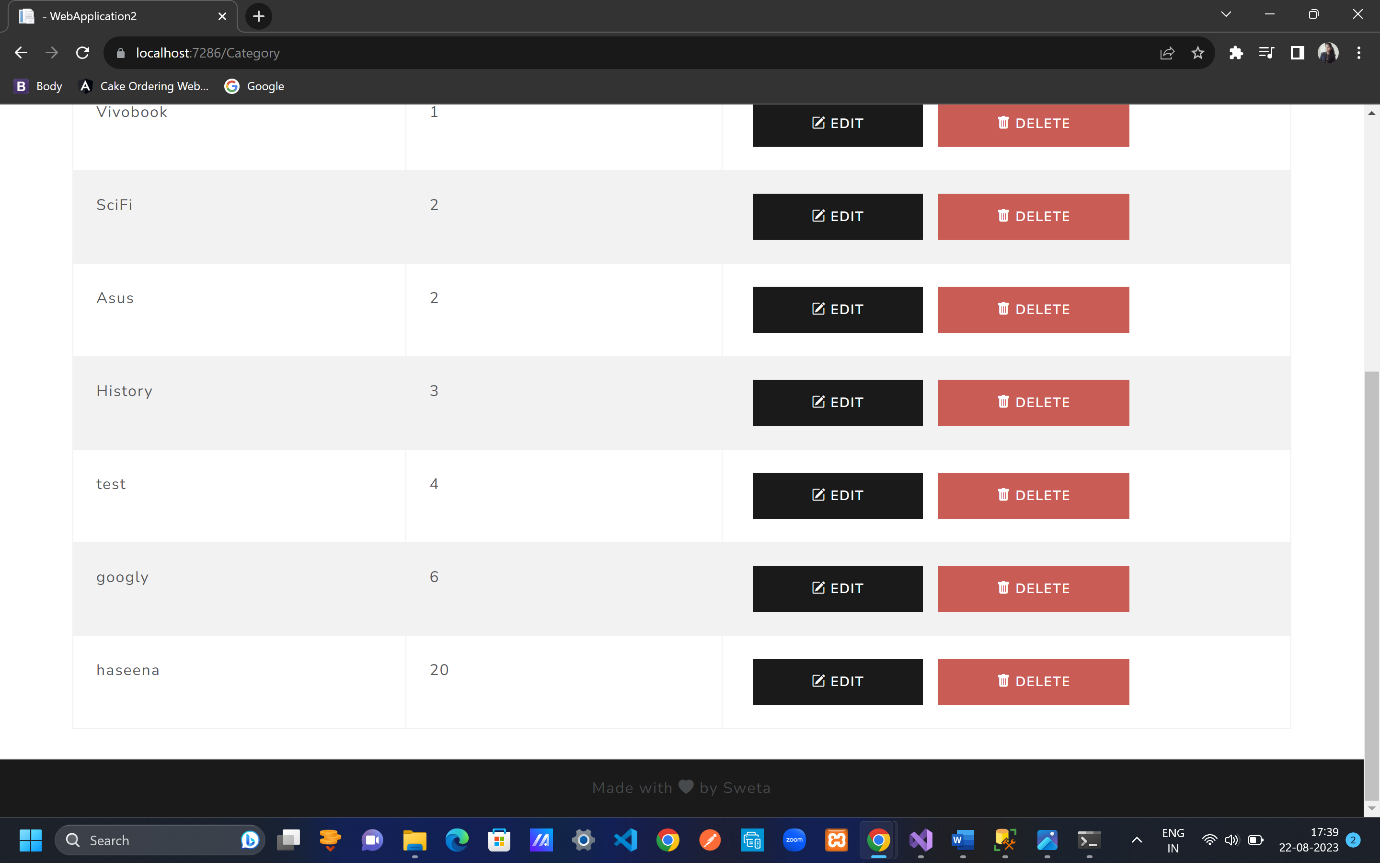
Index.cshtml

<**partial** **name**="\_Notification"/>









For all reusable components use partial views

Toastr NOTIFICATION

Search Toastr by codeseven on chrome; go to demo link; check toastr scripts; go to debug .css link and copy it

* [//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.css](https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.css)

\_Layout.cshtml

<link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.css"/>

Then copy debug javascript link as well

* [//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.js](https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.js)

\_Notification.cshtml

<script src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.js"></script>

\_Notification.cshtml

Also need jquery

<**script** src="~/lib/jquery/dist/jquery.min.js"></**script**>

\_Notification.cshtml

Go to toastr; Minified copy both .js and .css

Use Minified one for .js and .css

@if (TempData["success"] != null)

{

<**script** src="~/lib/jquery/dist/jquery.min.js"></**script**>

<script src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>

<h2>@TempData["success"]</h2>

}

@if (TempData["error"] != null)

{

<**script** src="~/lib/jquery/dist/jquery.min.js"></**script**>

<script src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>

<h2>@TempData["error"]</h2>

}

\_Layout.cshtml

<link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" />

From 3 easy steps choose 3. Toastr.info();

\_Notification.cshtml

@if (TempData["success"] != null)

{

<**script** src="~/lib/jquery/dist/jquery.min.js"></**script**>

<script src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>

<script type="text/javascript">

toastr.success('@TempData["success"]');

</script>

}

@if (TempData["error"] != null)

{

<**script** src="~/lib/jquery/dist/jquery.min.js"></**script**>

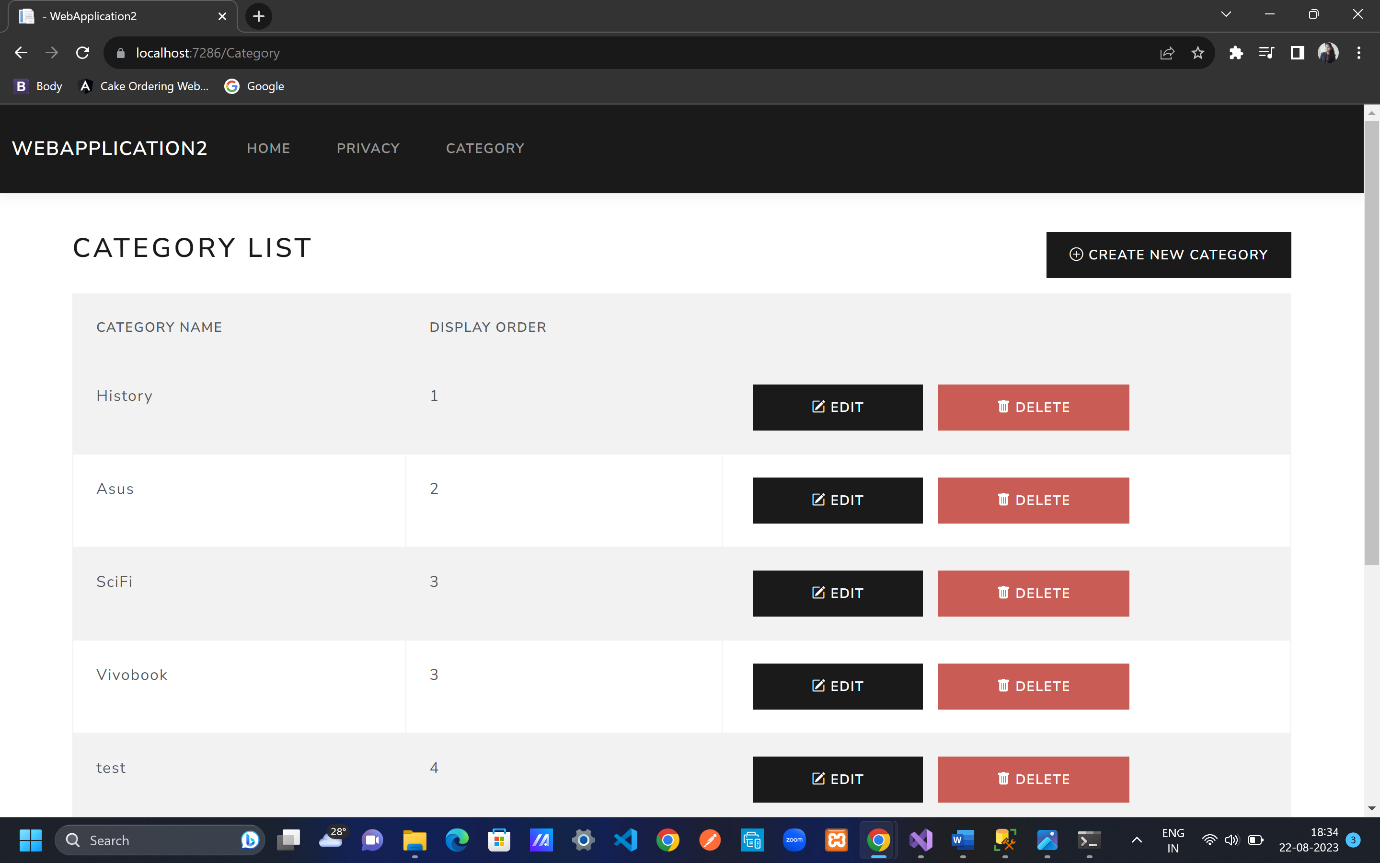
<script src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>

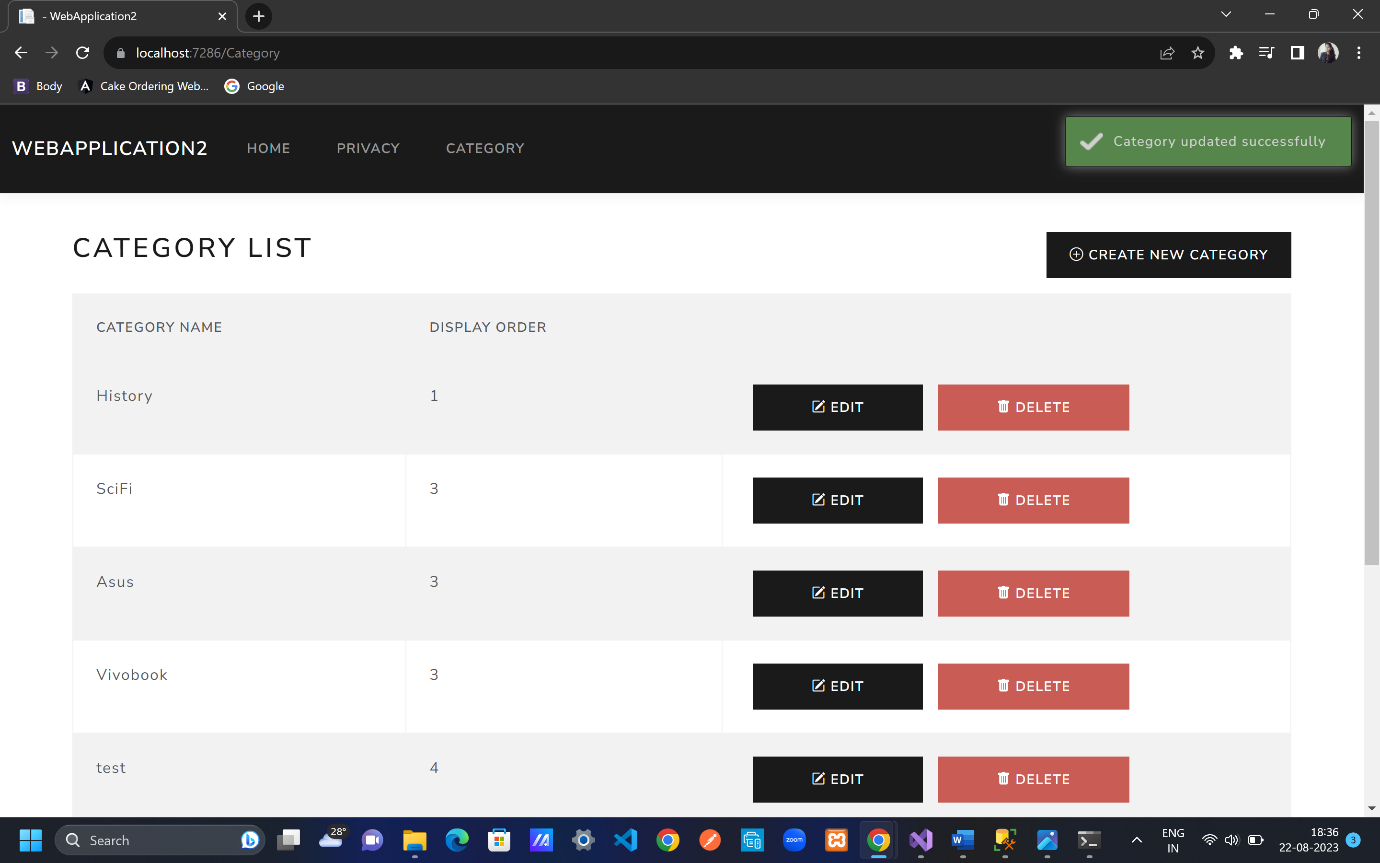
<script type="text/javascript">

toastr.error('@TempData["error"]');

</script>

}





Cut <**partial** **name**="\_Notification"/> from Index.cshtml and paste in

\_Layout.cshtml

</header>

<div class="container">

<**partial** **name**="\_Notification" />

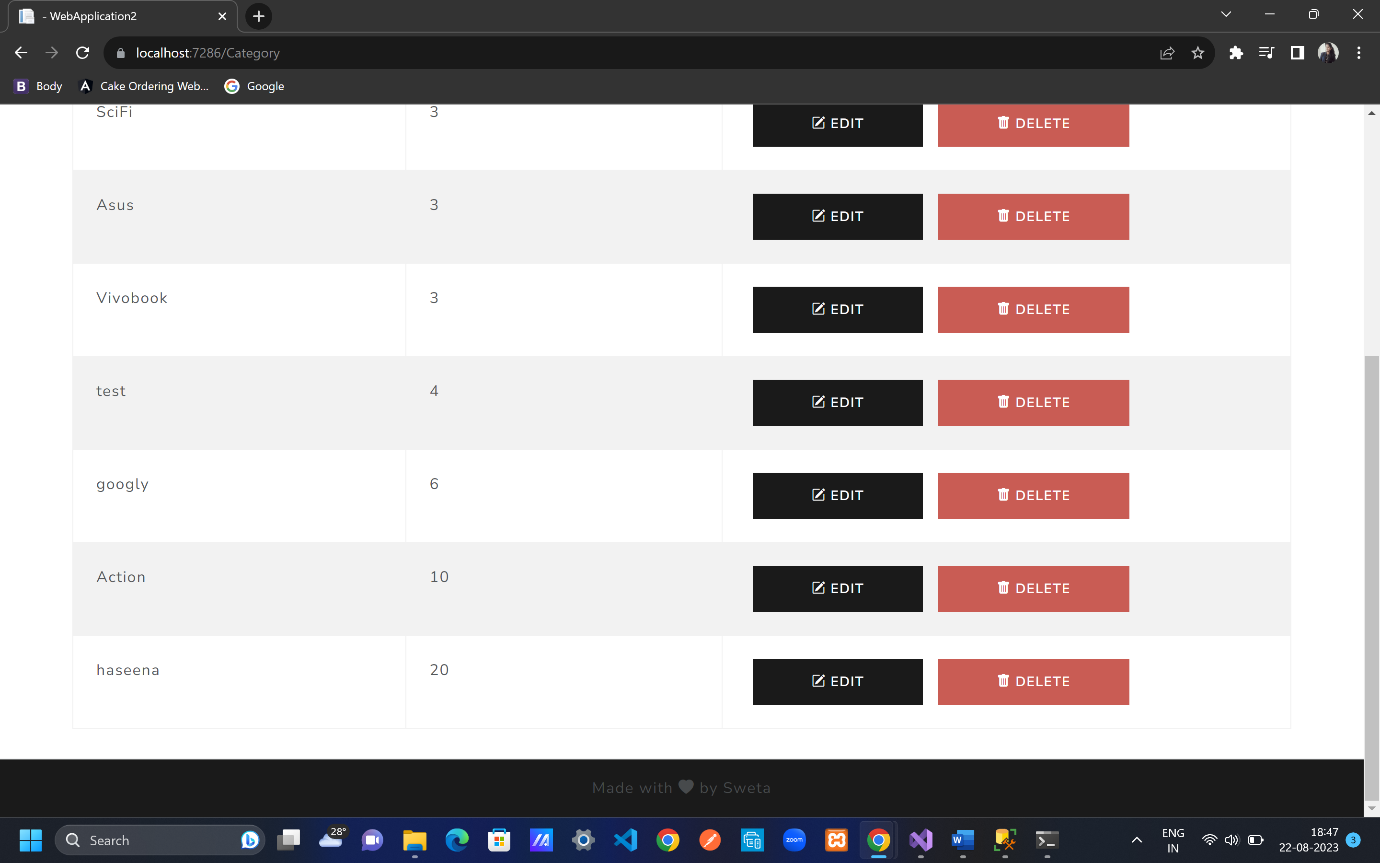
<main role="main" class="pb-3">

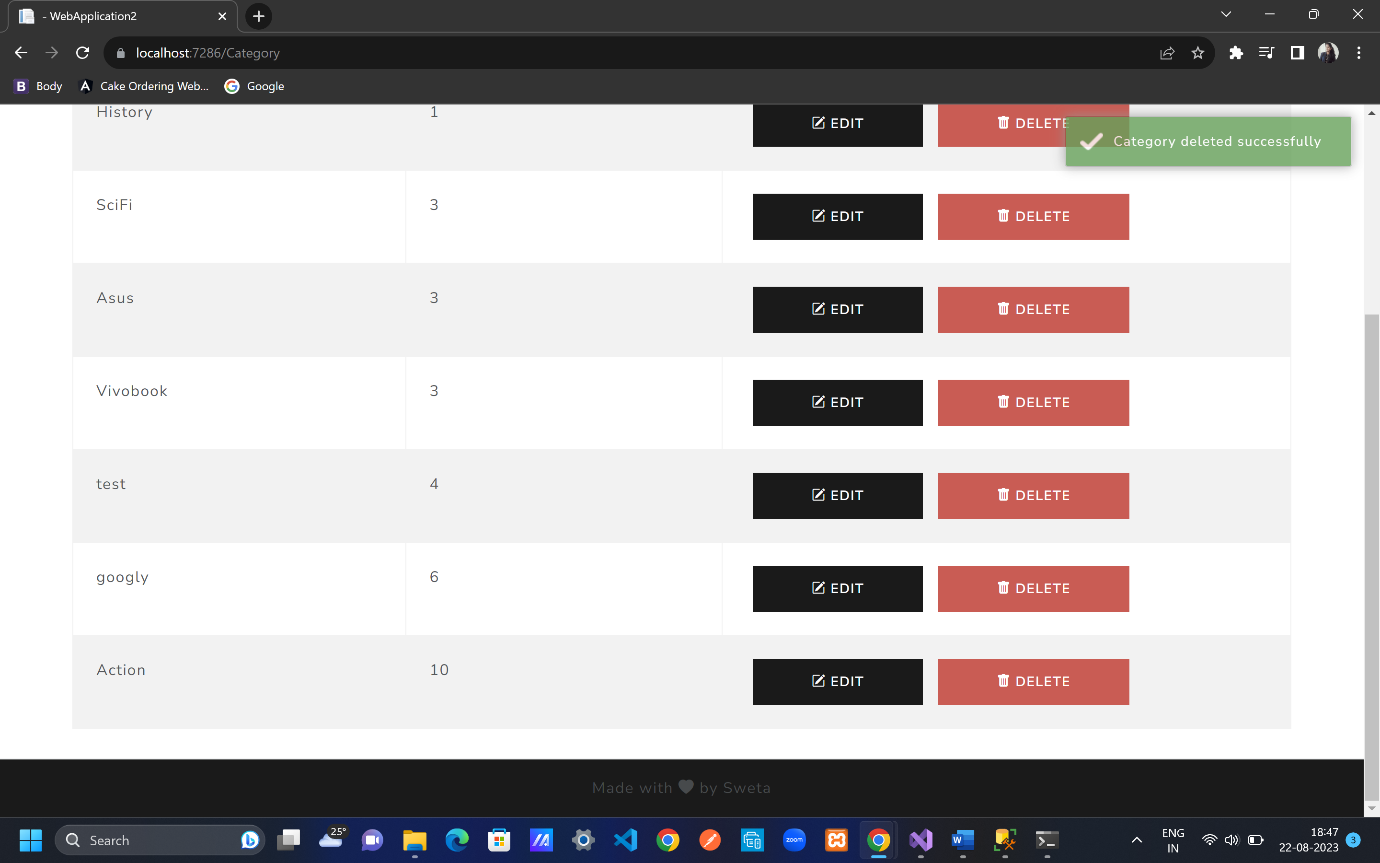
@RenderBody()

</main>

</div>

Since for all pages we need the notification that’s why shift it to \_Layout.cshtml to apply for all web pages





CREATE

Right click on application name; Add; create a new project; search Class Library; Next; name project as Bulky.DataAccess; Next; Create

Right click on application name; Add; create a new project; search Class Library; Next; name project as Bulky.Models; Next; Create

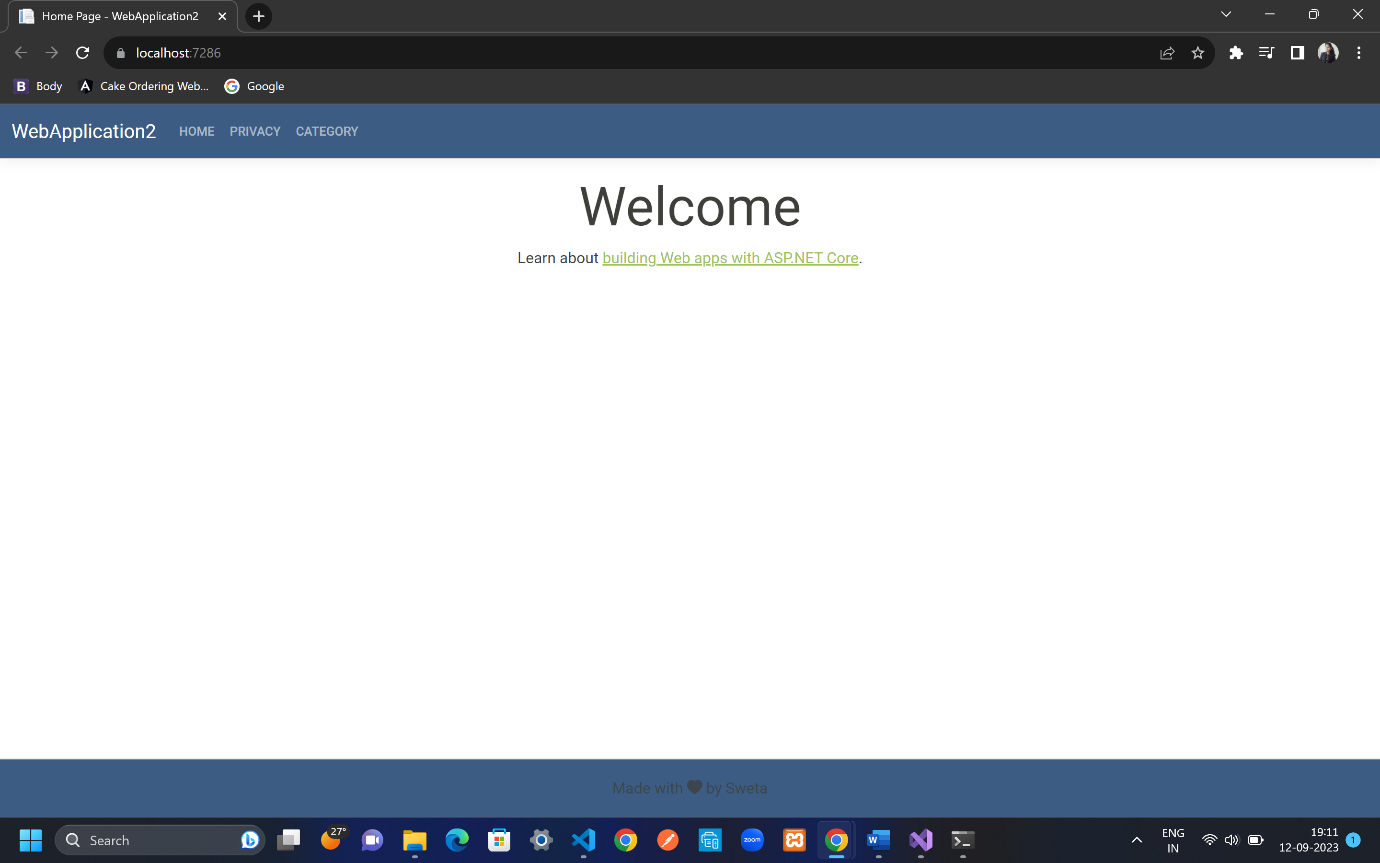
Right click on application name; Add; create a new project; search Class Library; Next; name project as Bulky.Utility; Next; Create

MODIFY STYLING

Delete class files all three projects named as Class1.cs filename.

Browse bootswatch; look for Sandstone theme; download bootstrap.css file ;

Go to webapplication2 folder; wwwroot folder; lib folder; bootstrap folder; dist folder; css folder; copy the downloaded to bootstrap.css



MODIFY UI OF CATEGORY PAGES

Views folder; Category folder; Create.cshtml

@model Category // new category object with a default value

<**form** method="post">

<div class="border p-3 mt-4">

<div class="row pb-2">

<h2 class="text-primary">Create Category</h2>

<hr/>

</div>

@\*<div asp-validation-summary="None"></div>\*@

<div class="form-floating py-2 col-12">

<**input** **asp-for**="Name" class="form-control border-0 shadow" />

<**label** **asp-for**="Name"></**label**>

<**span** **asp-validation-for**="Name" class="text-danger"></**span**>

</div>

<div class="form-floating py-2 col-12">

<**input** **asp-for**="DisplayOrder" class="form-control border-0 shadow" />

<**label** **asp-for**="DisplayOrder"></**label**>

<**span** **asp-validation-for**="DisplayOrder" class="text-danger"></**span**>

</div>

<div class="row">

<div class="col-6 col-md-3">

<button type="submit" class="btn btn-primary form-control" >Create</button>

</div>

<div class="col-6 col-md-3">

<**a** **asp-controller**="Category" **asp-action**="Index" class="btn btn-outline-secondary border form-control" > Back to List</**a**>

</div>

</div>

</div>

</**form**>

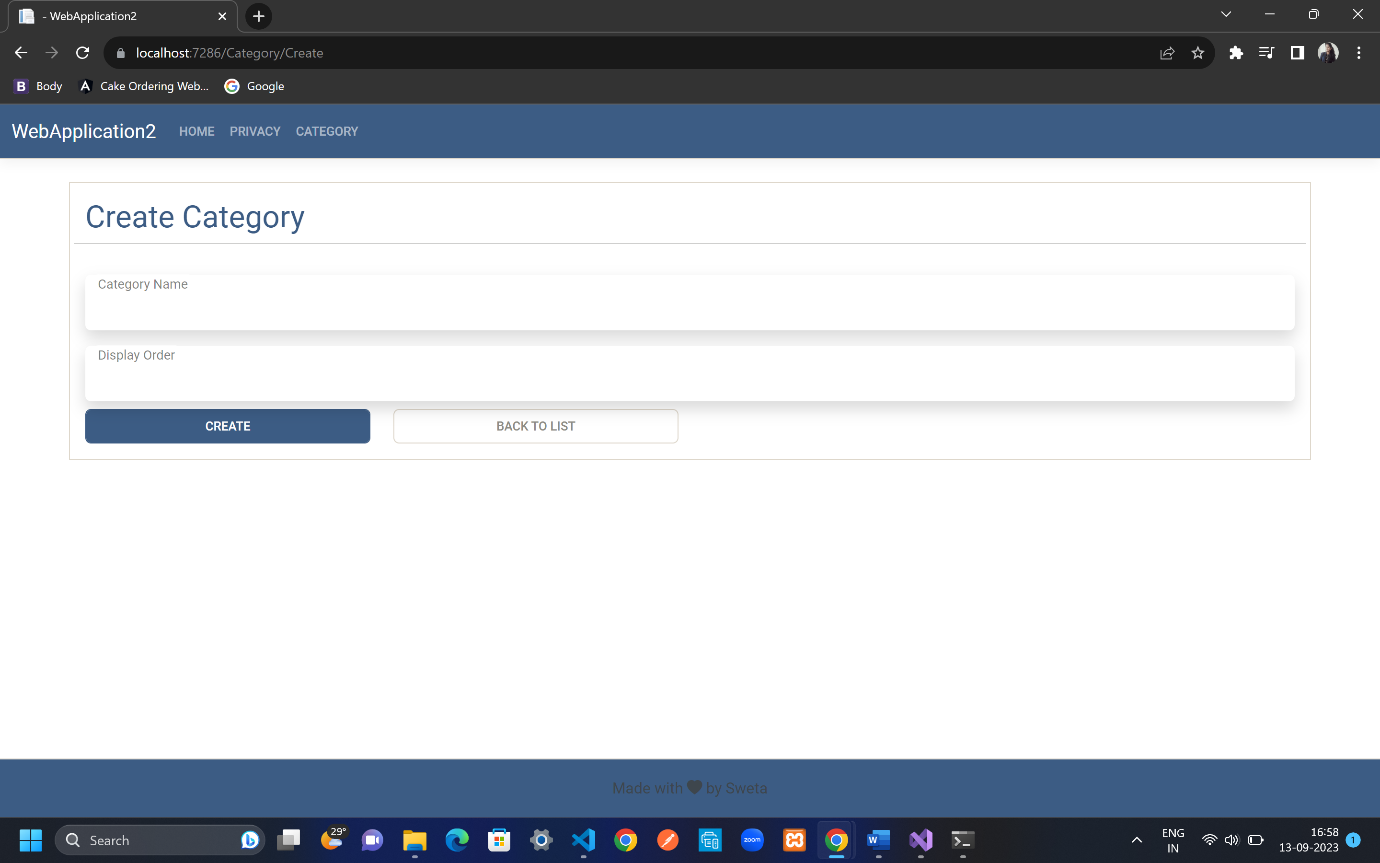
@section Scripts{

@{

<**partial** **name**="\_ValidationScriptsPartial"/>

}

}



Views folder; Category folder; Delete.cshtml

@model Category // new category object with a default value

<**form** method="post">

<**input** **asp-for**="Id" hidden/>

<div class="border p-3 mt-4">

<div class="row pb-2">

<h2 class="text-primary">Delete Category</h2>

<hr />

</div>

@\*<div asp-validation-summary="None"></div>\*@

<div class="form-floating py-2 col-12">

<**input** **asp-for**="Name" class="form-control border-0 shadow" />

<**label** **asp-for**="Name"></**label**>

<**span** **asp-validation-for**="Name" class="text-danger"></**span**>

</div>

<div class="form-floating py-2 col-12">

<**input** **asp-for**="DisplayOrder" class="form-control border-0 shadow" />

<**label** **asp-for**="DisplayOrder"></**label**>

<**span** **asp-validation-for**="DisplayOrder" class="text-danger"></**span**>

</div>

<div class="row">

<div class="col-6 col-md-3">

<button type="submit" class="btn btn-danger form-control">Delete</button>

</div>

<div class="col-6 col-md-3">

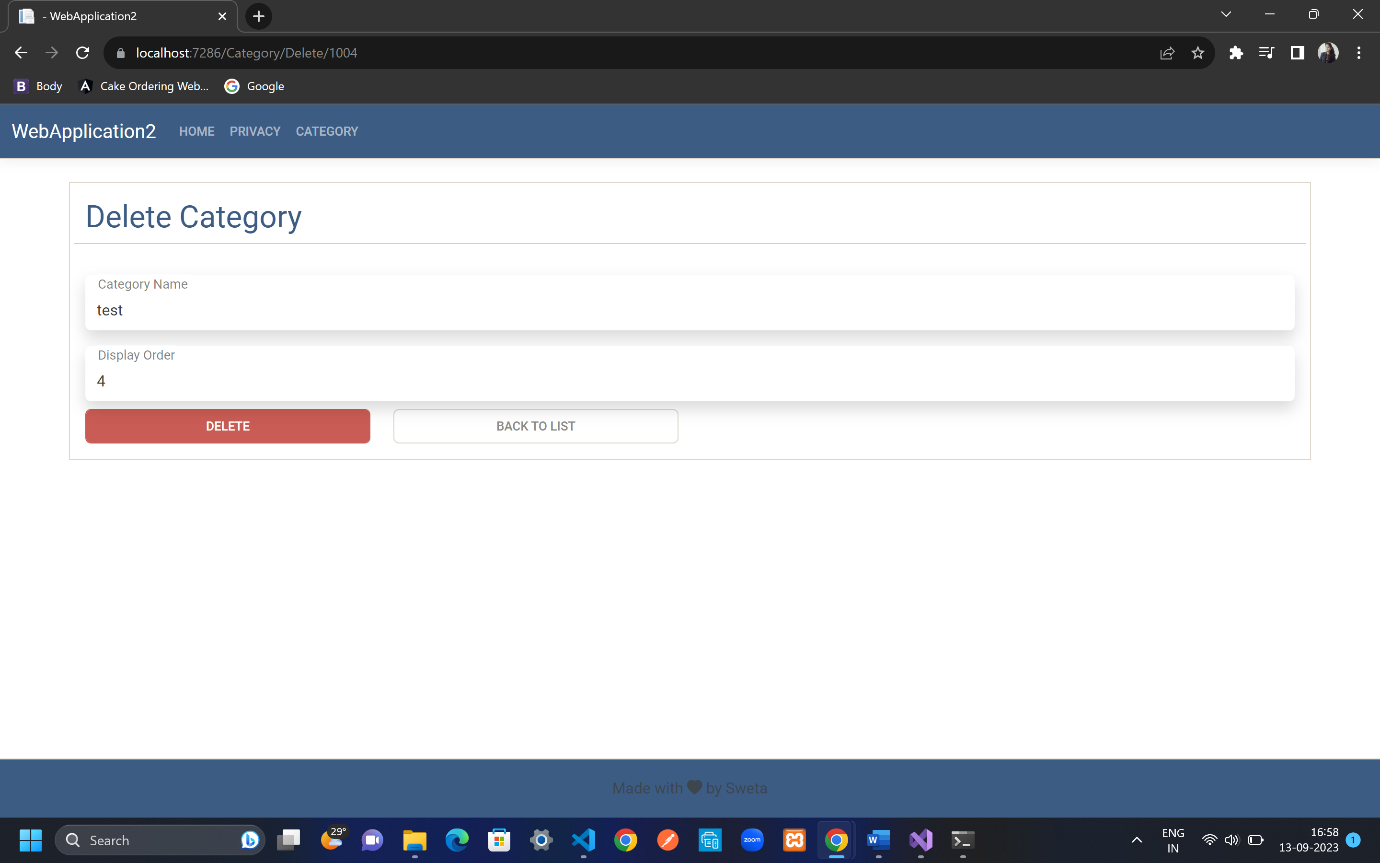
<**a** **asp-controller**="Category" **asp-action**="Index" class="btn btn-outline-secondary border form-control"> Back to List</**a**>

</div>

</div>

</div>

</**form**>



Views folder; Category folder; Edit.cshtml

@model Category // new category object with a default value

<**form** method="post">

<div class="border p-3 mt-4">

<div class="row pb-2">

<h2 class="text-primary">Edit Category</h2>

<hr />

</div>

@\*<div asp-validation-summary="None"></div>\*@

<div class="form-floating py-2 col-12">

<**input** **asp-for**="Name" class="form-control border-0 shadow" />

<**label** **asp-for**="Name"></**label**>

<**span** **asp-validation-for**="Name" class="text-danger"></**span**>

</div>

<div class="form-floating py-2 col-12">

<**input** **asp-for**="DisplayOrder" class="form-control border-0 shadow" />

<**label** **asp-for**="DisplayOrder"></**label**>

<**span** **asp-validation-for**="DisplayOrder" class="text-danger"></**span**>

</div>

<div class="row">

<div class="col-6 col-md-3">

<button type="submit" class="btn btn-primary form-control">Update</button>

</div>

<div class="col-6 col-md-3">

<**a** **asp-controller**="Category" **asp-action**="Index" class="btn btn-outline-secondary border form-control"> Back to List</**a**>

</div>

</div>

</div>

</**form**>

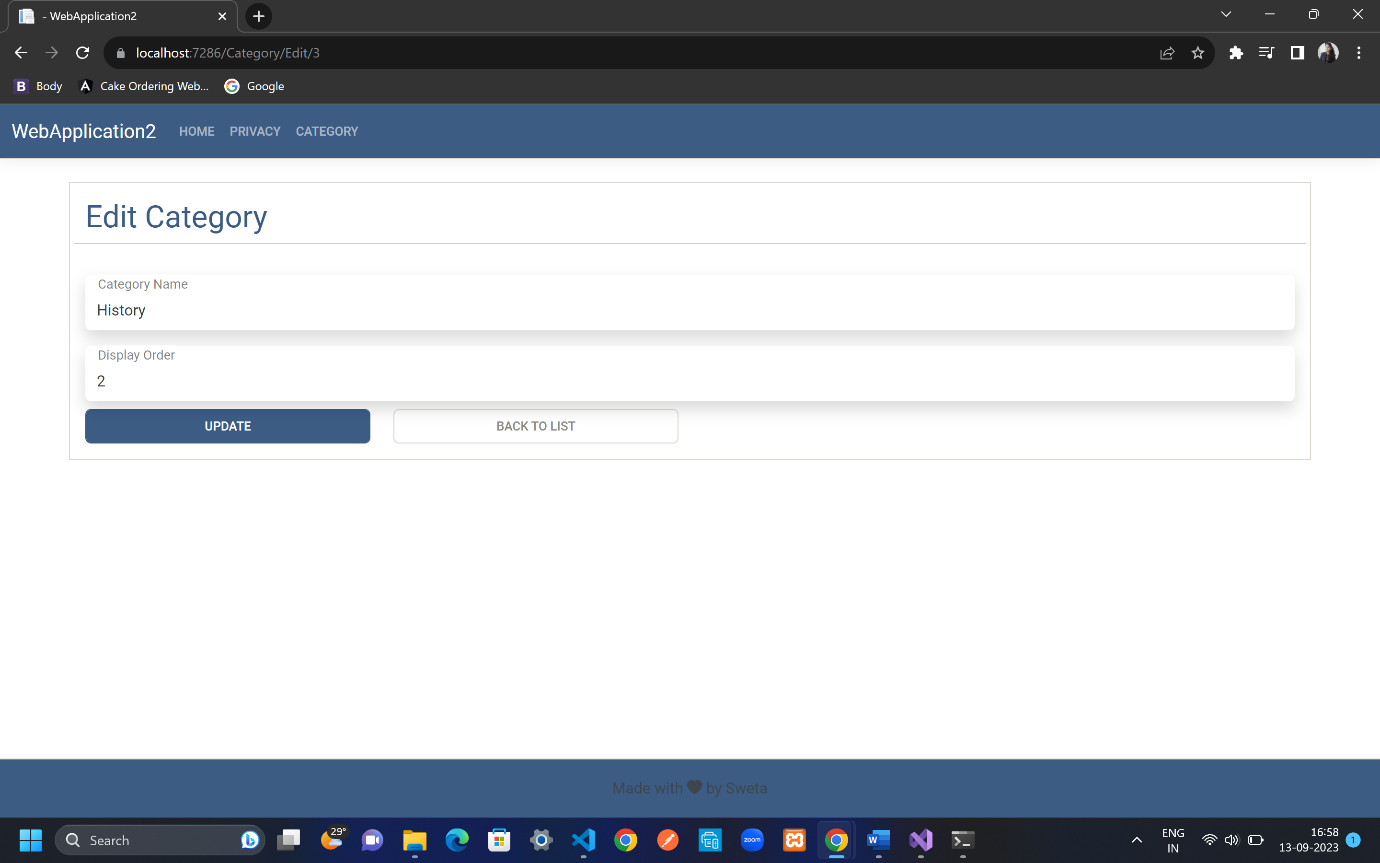
@section Scripts{

@{

<**partial** **name**="\_ValidationScriptsPartial" />

}

}



Views folder; Category folder; Index.cshtml

@model List<Category>

<div class="card shadow border-0 mt-4">

<div class="card-harder bg-secondary bg-gradient ml-0 py-3">

<div class="row">

<div class="col-12 text-center">

<h2 class="text-white py-2">Category List</h2>

</div>

</div>

</div>

<div class="card-body pt-4 ">

<div class="row pb-3">

<div class="col-13 text-end">

<**a** **asp-controller**="Category" **asp-action**="Create" class="btn btn-primary">

<i class="bi bi-plus-circle"></i> Create New Category

</**a**>

</div>

</div>

<table class=" table table-bordered table-striped">

<thred>

<tr>

<th>

Category Name

</th>

<th>

Display Order

</th>

<th></th>

</tr>

</thred>

<tbody>

@foreach(var obj in Model.OrderBy(u=>u.DisplayOrder)){

<tr>

<td>

@obj.Name

</td>

<td>

@obj.DisplayOrder

</td>

<td>

<div class="w-75 btn-group" role="group">

<**a** **asp-controller**="Category" **asp-action**="Edit" **asp-route-id**="@obj.Id" class="btn btn-primary mx-2">

<i class="bi bi-pencil-square"></i> Edit

</**a**>

<**a** **asp-controller**="Category" **asp-action**="Delete" **asp-route-id**="@obj.Id" class="btn btn-danger mx-2">

<i class="bi bi-trash-fill"></i> Delete

</**a**>

</div>

</td>

</tr>

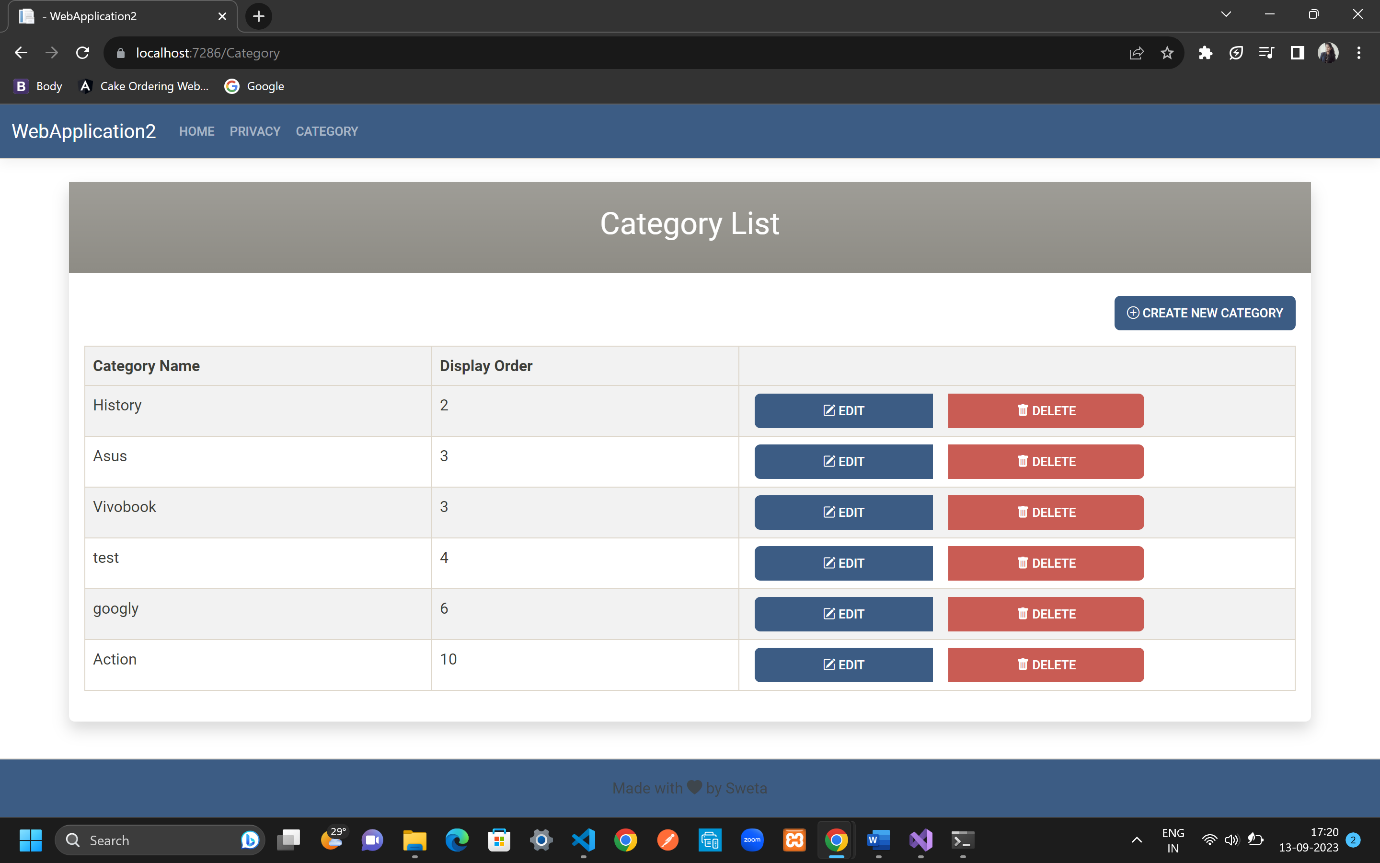
}

</tbody>

</table>

</div>

</div>



N-TIER ARCHITECTURE

Move Data folder form WebApplication2 to Bulky.DataAccess and delete Data folder from WebApplication2

Move Models folder form WebApplication2 to Bulky.DataAccess and delete Models folder from WebApplication2

Right click on Bulky.Utility folder; Add; class; rename it as SD.cs; SD(static details)

Bulky.Utility folder; SD.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bulky.Utility

{

public static class SD

{

}

}

Move Migrations folder form WebApplication2 to Bulky.DataAccess and delete Migrations folder from WebApplication2

Right click on project name; Manage NuGet Packages; install Microsoft.EntityFrameworkCore and select Bulky.DataAccess in right then install; install Microsoft.EntityFrameworkCore.Tools and select Bulky.DataAccess in right then install; install Microsoft.EntityFrameworkCore.SqlServer and select Bulky.DataAccess in right then install; install Microsoft.VisualStudio.Web.CodeGeneration.Design and select Bulky.DataAccess in right then install;

Add all namespace properly and remove all the errors; by right click on application project name and click on build project.

Bulky.DataAccess; Data folder; ApplicationDbContext.cs

using Bulky.Models.Models;

using Microsoft.EntityFrameworkCore;

namespace Bulky.DataAccess.Data

{

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) {

}

public DbSet<Category> Categories { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Category>().HasData(

new Category { Id=1, Name="Action", DisplayOrder=1},

new Category { Id = 2, Name = "SciFi", DisplayOrder = 2 },

new Category { Id = 3, Name = "History", DisplayOrder = 3 }

);

}

}

}

Bulky.DataAccess; Migrations folder; 20230807114146\_AddCategoryTabletoDb.cs

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace Bulky.DataAccess.Migrations

{

/// <inheritdoc />

public partial class AddCategoryTabletoDb : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "Categories",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

Name = table.Column<string>(type: "nvarchar(max)", nullable: false),

DisplayOrder = table.Column<int>(type: "int", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Categories", x => x.Id);

});

}

/// <inheritdoc />

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "Categories");

}

}

}

Bulky.DataAccess; Migrations folder; 20230807114146\_AddCategoryTabletoDb.cs; 20230807114146\_AddCategoryTabletoDb.Designer.cs

// <auto-generated />

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.EntityFrameworkCore.Metadata;

using Microsoft.EntityFrameworkCore.Migrations;

using Bulky.DataAccess.Data;

#nullable disable

namespace Bulky.DataAccess.Migrations

{

[DbContext(typeof(ApplicationDbContext))]

[Migration("20230807114146\_AddCategoryTabletoDb")]

partial class AddCategoryTabletoDb

{

/// <inheritdoc />

protected override void BuildTargetModel(ModelBuilder modelBuilder)

{

#pragma warning disable 612, 618

modelBuilder

.HasAnnotation("ProductVersion", "7.0.9")

.HasAnnotation("Relational:MaxIdentifierLength", 128);

SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder);

modelBuilder.Entity("WebApplication2.Models.Category", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<int>("DisplayOrder")

.HasColumnType("int");

b.Property<string>("Name")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.ToTable("Categories");

});

#pragma warning restore 612, 618

}

}

}

Bulky.DataAccess; Migrations folder; 20230809081757\_SeedCategoryTable.cs

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

#pragma warning disable CA1814 // Prefer jagged arrays over multidimensional

namespace Bulky.DataAccess.Migrations

{

/// <inheritdoc />

public partial class SeedCategoryTable : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.InsertData(

table: "Categories",

columns: new[] { "Id", "DisplayOrder", "Name" },

values: new object[,]

{

{ 1, 1, "Action" },

{ 2, 2, "SciFi" },

{ 3, 3, "History" }

});

}

/// <inheritdoc />

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DeleteData(

table: "Categories",

keyColumn: "Id",

keyValue: 1);

migrationBuilder.DeleteData(

table: "Categories",

keyColumn: "Id",

keyValue: 2);

migrationBuilder.DeleteData(

table: "Categories",

keyColumn: "Id",

keyValue: 3);

}

}

}

Bulky.DataAccess; Migrations folder; 20230809081757\_SeedCategoryTable.cs; 20230809081757\_SeedCategoryTable.Designer.cs

// <auto-generated />

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.EntityFrameworkCore.Metadata;

using Microsoft.EntityFrameworkCore.Migrations;

using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

using Bulky.DataAccess.Data;

#nullable disable

namespace Bulky.DataAccess.Migrations

{

[DbContext(typeof(ApplicationDbContext))]

[Migration("20230809081757\_SeedCategoryTable")]

partial class SeedCategoryTable

{

/// <inheritdoc />

protected override void BuildTargetModel(ModelBuilder modelBuilder)

{

#pragma warning disable 612, 618

modelBuilder

.HasAnnotation("ProductVersion", "7.0.9")

.HasAnnotation("Relational:MaxIdentifierLength", 128);

SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder);

modelBuilder.Entity("WebApplication2.Models.Category", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<int>("DisplayOrder")

.HasColumnType("int");

b.Property<string>("Name")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.ToTable("Categories");

b.HasData(

new

{

Id = 1,

DisplayOrder = 1,

Name = "Action"

},

new

{

Id = 2,

DisplayOrder = 2,

Name = "SciFi"

},

new

{

Id = 3,

DisplayOrder = 3,

Name = "History"

});

});

#pragma warning restore 612, 618

}

}

}

Bulky.DataAccess; Migrations folder; ApplicationDbContextModelSnapshot.cs

// <auto-generated />

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.EntityFrameworkCore.Metadata;

using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

using Bulky.DataAccess.Data;

#nullable disable

namespace Bulky.DataAccess.Migrations

{

[DbContext(typeof(ApplicationDbContext))]

partial class ApplicationDbContextModelSnapshot : ModelSnapshot

{

protected override void BuildModel(ModelBuilder modelBuilder)

{

#pragma warning disable 612, 618

modelBuilder

.HasAnnotation("ProductVersion", "7.0.9")

.HasAnnotation("Relational:MaxIdentifierLength", 128);

SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder);

modelBuilder.Entity("WebApplication2.Models.Category", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<int>("DisplayOrder")

.HasColumnType("int");

b.Property<string>("Name")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.ToTable("Categories");

b.HasData(

new

{

Id = 1,

DisplayOrder = 1,

Name = "Action"

},

new

{

Id = 2,

DisplayOrder = 2,

Name = "SciFi"

},

new

{

Id = 3,

DisplayOrder = 3,

Name = "History"

});

});

#pragma warning restore 612, 618

}

}

}

Bulky.DataAccess; Migrations folder; ApplicationDbContextModelSnapshot.cs; ApplicationDbContextModelSnapshot

// <auto-generated />

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.EntityFrameworkCore.Metadata;

using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

using Bulky.DataAccess.Data;

#nullable disable

namespace Bulky.DataAccess.Migrations

{

[DbContext(typeof(ApplicationDbContext))]

partial class ApplicationDbContextModelSnapshot : ModelSnapshot

{

protected override void BuildModel(ModelBuilder modelBuilder)

{

#pragma warning disable 612, 618

modelBuilder

.HasAnnotation("ProductVersion", "7.0.9")

.HasAnnotation("Relational:MaxIdentifierLength", 128);

SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder);

modelBuilder.Entity("WebApplication2.Models.Category", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<int>("DisplayOrder")

.HasColumnType("int");

b.Property<string>("Name")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.ToTable("Categories");

b.HasData(

new

{

Id = 1,

DisplayOrder = 1,

Name = "Action"

},

new

{

Id = 2,

DisplayOrder = 2,

Name = "SciFi"

},

new

{

Id = 3,

DisplayOrder = 3,

Name = "History"

});

});

#pragma warning restore 612, 618

}

}

}

Bulky.Models; Models; Category.cs;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

namespace Bulky.Models.Models

{

public class Category

{

[Key]

public int Id { get; set; }

[Required]

[MaxLength(30)]

[DisplayName("Category Name")]

public string Name { get; set; }

[DisplayName("Display Order")]

[Range(1, 100,ErrorMessage ="Display Order must be in 1-100")]

public int DisplayOrder { get; set; }

}

}

Bulky.Models; Models; ErrorViewModel.cs

namespace Bulky.Models.Models

{

public class ErrorViewModel

{

public string? RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

}

}

Bulky.Utility; SD.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bulky.Utility

{

public static class SD

{

}

}

WebApplication2; Controllers; CategoryController.cs

using Microsoft.AspNetCore.Mvc;

using Bulky.DataAccess.Data;

using Bulky.Models.Models;

namespace Bulky.DataAccess.Controllers

{

public class CategoryController : Controller

{

public IActionResult Index()

{

List<Category> objCategoryList = \_db.Categories.ToList();

return View(objCategoryList);

}

private readonly ApplicationDbContext \_db;

public CategoryController(ApplicationDbContext db)

{

\_db= db;

}

public IActionResult Create()

{

return View();

}

[HttpPost]

public IActionResult Create(Category obj) //getting data from category model

{

if (obj.Name==obj.DisplayOrder.ToString())

{

ModelState.AddModelError("name", "The display order cannot exactly match the name");

}

//if (obj.Name!=null && obj.Name.ToLower() == "test")

//{

// ModelState.AddModelError("", "Test is an invalid value");

//}

if (ModelState.IsValid)

{

\_db.Categories.Add(obj); // what is to be add

\_db.SaveChanges(); //go to the database and create category

TempData["success"] = "Category created successfully";

return RedirectToAction("Index"); //view all categories in index() line 9;

//return RedirectToAction("Index","Category");

//-> if we have different controller then add like

//this else keep only menthod/function name

}

return View();

}

public IActionResult Edit(int? id)

{

if(id==null || id==0)

{

return NotFound();

}

Category? categoryFromDb = \_db.Categories.Find(id);

//Category? categoryFromDb1 = \_db.Categories.FirstOrDefault(c=>c.Id==id);

//Category? categoryFromDb2 = \_db.Categories.Where(c => c.Id == id).FirstOrDefault();

if (categoryFromDb == null)

{

return NotFound();

}

return View(categoryFromDb);

}

[HttpPost]

public IActionResult Edit(Category obj) //getting data from category model

{

if (obj.Name == obj.DisplayOrder.ToString())

{

ModelState.AddModelError("name", "The display order cannot exactly match the name");

}

//if (obj.Name!=null && obj.Name.ToLower() == "test")

//{

// ModelState.AddModelError("", "Test is an invalid value");

//}

if (ModelState.IsValid)

{

\_db.Categories.Update(obj); // what is to be add

\_db.SaveChanges(); //go to the database and create category

TempData["success"] = "Category updated successfully";

return RedirectToAction("Index"); //view all categories in index() line 9;

//return RedirectToAction("Index","Category");

//-> if we have different controller then add like

//this else keep only menthod/function name

}

return View();

}

public IActionResult Delete(int? id)

{

if (id == null || id == 0)

{

return NotFound();

}

Category? categoryFromDb = \_db.Categories.Find(id);

//Category? categoryFromDb1 = \_db.Categories.FirstOrDefault(c=>c.Id==id);

//Category? categoryFromDb2 = \_db.Categories.Where(c => c.Id == id).FirstOrDefault();

if (categoryFromDb == null)

{

return NotFound();

}

return View(categoryFromDb);

}

[HttpPost, ActionName("Delete")]

public IActionResult DeletePOST(int? id) //getting data from category model

{

Category? obj= \_db.Categories.Find(id);

if (obj == null)

{

return NotFound();

}

\_db.Categories.Remove(obj);

\_db.SaveChanges(); //go to the database and create category

TempData["success"] = "Category deleted successfully";

return RedirectToAction("Index"); //view all categories in index() line 9;

//return RedirectToAction("Index","Category");

//-> if we have different controller then add like

//this else keep only menthod/function name

}

}

}

WebApplication2; Controllers; HomeController.cs

using Bulky.Models.Models;

using Microsoft.AspNetCore.Mvc;

using System.Diagnostics;

namespace WebApplication2.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

public IActionResult Index()

{

return View();

}

public IActionResult Privacy()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

WebApplication2; Views; Shared; ViewImports.cshtml

@using WebApplication2

@using Bulky.Models.Models

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

WebApplication2; Program.cs

using Bulky.DataAccess.Data;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

builder.Services.AddDbContext<ApplicationDbContext>(options=>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

HOW TO RESET DATABASE(If migrations are corrupted then go ahead with following steps)

Open ssms and connect server.

Delete database from ssms; delete migration from the application project; then go to nuget package manager and add migration and then update database.

Go to Databases; delete Bulky

Go to visual studio; Bulky.DataAccess; delete Migrations

Go to tools; NuGet package manager; Package manager console

> add-migration AddCategoryToDbAndSeedTable

Now we will get error so change Default project: Bulky.DataAccess

> add-migration AddCategoryToDbAndSeedTable

This will add migration

>update-database

Now refresh the ssms; Databases; Bulky; Tables; right click dbo.Categories; Select top 1000 rows;

Earlier database Category content gets deleted and only seeded data is available.