**SOLUTION**

**Database Accounts:**

CREATE TABLE [dbo].[Accounts](

[Id] int identity(1,1) NOT NULL,

[Name] [nvarchar](50) NULL,

[Balance] [nvarchar](50) NULL,

CONSTRAINT [PK\_Transactions] PRIMARY KEY CLUSTERED

(

[Id] ASC

)

)

GO

**Database Transactions:**

CREATE TABLE [dbo].[Transactions](

[Id] int identity(1,1) NOT NULL,

[ToName] [nvarchar](50) NULL,

[FromName] [nvarchar](50) NULL,

[Amount] [nvarchar] (50) NULL,

CONSTRAINT [PK\_Transactions] PRIMARY KEY CLUSTERED

(

[Id] ASC

)

)

GO

**Accounts.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace OelSecondCloud.Models

{

public class Accounts

{

public int Id { get; set; }

public string Name { get; set; }

public string Balance { get; set; }

}

}

**AccountsRepository:**

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Web;

namespace OelSecondCloud.Models

{

public class AccountsRepository : IAccountsRepository

{

private readonly SqlDbContext db = new SqlDbContext(

);

public async Task Add(Accounts accounts)

{

accounts.Id = Convert.ToInt32(Guid.NewGuid());

db.Accounts.Add(accounts);

try

{

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

public async Task<Accounts> GetEmployee(int id)

{

try

{

Accounts accounts = await db.Accounts.FindAsync(id);

if (accounts == null)

{

return null;

}

return accounts;

}

catch

{

throw;

}

}

public async Task<IEnumerable<Accounts>> GetEmployees()

{

try

{

var accounts = await db.Accounts.ToListAsync();

return accounts.AsQueryable();

}

catch

{

throw;

}

}

public async Task Update(Accounts accounts)

{

try

{

db.Entry(accounts).State = EntityState.Modified;

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

public async Task Delete(int id)

{

try

{

Accounts accounts = await db.Accounts.FindAsync(id);

db.Accounts.Remove(accounts);

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

private bool EmployeeExists(int id)

{

return db.Accounts.Count(e => e.Id == id) > 0;

}

}

}

**IAccountsRepository:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OelSecondCloud.Models

{

public interface IAccountsRepository

{

Task Add(Accounts accounts);

Task Update(Accounts accounts);

Task Delete(int id);

Task<Accounts> GetEmployee(int id);

Task<IEnumerable<Accounts>> GetEmployees();

}

}

**AccountsApiController:**

using OelSecondCloud.Models;

using System.Collections.Generic;

using System.Threading.Tasks;

using System.Web.Http;

namespace OelSecondCloud.Controllers

{

public class AccountsApiController : ApiController

{

private readonly IAccountsRepository \_iEmployeeRepository = new AccountsRepository();

[HttpGet]

[Route("api/Employees/Get")]

public async Task<IEnumerable<Accounts>> Get()

{

return await \_iEmployeeRepository.GetEmployees()

;

}

[HttpPost]

[Route("api/Employees/Create")]

public async Task CreateAsync([FromBody]Accounts accounts)

{

if (ModelState.IsValid)

{

await \_iEmployeeRepository.Add(accounts);

}

}

[HttpGet]

[Route("api/Employees/Details/{id}")]

public async Task<Accounts> Details(int id)

{

var result = await \_iEmployeeRepository.GetEmployee(id);

return result;

}

[HttpPut]

[Route("api/Employees/Edit")]

public async Task EditAsync([FromBody]Accounts accounts)

{

if (ModelState.IsValid)

{

await \_iEmployeeRepository.Update(accounts);

}

}

[HttpDelete]

[Route("api/Employees/Delete/{id}")]

public async Task DeleteConfirmedAsync(int id)

{

await \_iEmployeeRepository.Delete(id);

}

}

}

**AccountsController:**

using OelSecondCloud.Models;

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web.Mvc;

namespace OelSecondCloud.Controllers

{

public class AccountsController : Controller

{

readonly string apiBaseAddress = ConfigurationManager.AppSettings["apiBaseAddress"];

public async Task<ActionResult> Index()

{

IEnumerable<Accounts> employees = null;

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress);

var result = await client.GetAsync("employees/get");

if (result.IsSuccessStatusCode)

{

employees = await result.Content.ReadAsAsync<IList<Accounts>>();

}

else

{

employees = Enumerable.Empty<Accounts>()

;

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

return View(employees);

}

public async Task<ActionResult> Details(int id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Accounts employee = null;

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress)

;

var result = await client.GetAsync($"employees/details/{ id}");

if (result.IsSuccessStatusCode)

{

employee = await result.Content.ReadAsAsync<Accounts>();

}

else

{

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

if (employee == null)

{

return HttpNotFound();

}

return View(employee);

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Create([Bind(Include

= "Id,Name,Balance")] Accounts employee)

{

if (ModelState.IsValid)

{

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress);

var response = await client.PostAsJsonAsync("employees/Create", employee);

if (response.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

else

{

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

}

return View(employee);

}

public async Task<ActionResult> Edit(int id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Accounts employee = null;

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress)

;

var result = await client.GetAsync($"employees/details/{ id}");

if (result.IsSuccessStatusCode)

{

employee = await result.Content.ReadAsAsync<Accounts>();

}

else

{

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

if (employee == null)

{

return HttpNotFound();

}

return View(employee);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Edit([Bind(Include =

"Id,Name,Balance")] Accounts employee)

{

if (ModelState.IsValid)

{

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress);

var response = await client.PutAsJsonAsync("employees/edit", employee);

if (response.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

else

{

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

return RedirectToAction("Index");

}

return View(employee);

}

public async Task<ActionResult> Delete(int id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Accounts employee = null;

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress)

;

var result = await client.GetAsync($"employees/details/{ id}");

if (result.IsSuccessStatusCode)

{

employee = await result.Content.ReadAsAsync<Accounts>();

}

else

{

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

if (employee == null)

{

return HttpNotFound();

}

return View(employee);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> DeleteConfirmed(int id)

{

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress)

;

var response = await client.DeleteAsync($"employees/delete/{ id}");

if (response.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

else

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

return View();

}

}

}

**Transactions.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace OelSecondCloud.Models

{

public class Transactions

{

public int Id { get; set; }

public string ToName { get; set; }

public string FromName { get; set; }

public string Amount { get; set; }

}

}

**TransactionsRepository:**

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Web;

namespace OelSecondCloud.Models

{

public class TransactionsRepository : ITransactionsRepository

{

private readonly SqlDbContext db = new SqlDbContext(

);

public async Task Add(Transactions accounts)

{

accounts.Id = Convert.ToInt32(Guid.NewGuid());

db.Transactions.Add(accounts);

try

{

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

public async Task<Transactions> GetEmployee(int id)

{

try

{

Transactions accounts = await db.Transactions.FindAsync(id);

if (accounts == null)

{

return null;

}

return accounts;

}

catch

{

throw;

}

}

public async Task<IEnumerable<Transactions>> GetEmployees()

{

try

{

var accounts = await db.Transactions.ToListAsync();

return accounts.AsQueryable();

}

catch

{

throw;

}

}

public async Task Update(Transactions accounts)

{

try

{

db.Entry(accounts).State = EntityState.Modified;

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

public async Task Delete(int id)

{

try

{

Transactions accounts = await db.Transactions.FindAsync(id);

db.Transactions.Remove(accounts);

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

private bool EmployeeExists(int id)

{

return db.Accounts.Count(e => e.Id == id) > 0;

}

}

}

**ITransactionsRepository:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OelSecondCloud.Models

{

interface ITransactionsRepository

{

Task Add(Transactions accounts);

Task Update(Transactions accounts);

Task Delete(int id);

Task<Transactions> GetEmployee(int id);

Task<IEnumerable<Transactions>> GetEmployees();

}

}

**TransactionsApiController:**

using OelSecondCloud.Models;

using System.Collections.Generic;

using System.Threading.Tasks;

using System.Web.Http;

namespace OelSecondCloud.Controllers

{

public class TransactionsApiController : ApiController

{

private readonly ITransactionsRepository \_iEmployeeRepository = new TransactionsRepository();

[HttpGet]

[Route("api/Transactions/Get")]

public async Task<IEnumerable<Transactions>> Get()

{

return await \_iEmployeeRepository.GetEmployees()

;

}

[HttpPost]

[Route("api/Transactions/Create")]

public async Task CreateAsync([FromBody]Transactions accounts)

{

if (ModelState.IsValid)

{

await \_iEmployeeRepository.Add(accounts);

}

}

[HttpGet]

[Route("api/Transactions/Details/{id}")]

public async Task<Transactions> Details(int id)

{

var result = await \_iEmployeeRepository.GetEmployee(id);

return result;

}

[HttpPut]

[Route("api/Transactions/Edit")]

public async Task EditAsync([FromBody]Transactions accounts)

{

if (ModelState.IsValid)

{

await \_iEmployeeRepository.Update(accounts);

}

}

[HttpDelete]

[Route("api/Transactions/Delete/{id}")]

public async Task DeleteConfirmedAsync(int id)

{

await \_iEmployeeRepository.Delete(id);

}

}

}

**TransactionsController:**

using OelSecondCloud.Models;

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web.Mvc;

namespace OelSecondCloud.Controllers

{

public class TransactionsController : Controller

{

readonly string apiBaseAddress = ConfigurationManager.AppSettings["apiBaseAddress"];

public async Task<ActionResult> Index()

{

IEnumerable<Transactions> employees = null;

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress);

var result = await client.GetAsync("transactions/get");

if (result.IsSuccessStatusCode)

{

employees = await result.Content.ReadAsAsync<IList<Transactions>>();

}

else

{

employees = Enumerable.Empty<Transactions>()

;

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

return View(employees);

}

public async Task<ActionResult> Details(int id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Accounts employee = null;

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress)

;

var result = await client.GetAsync($"transactions/details/{ id}");

if (result.IsSuccessStatusCode)

{

employee = await result.Content.ReadAsAsync<Accounts>();

}

else

{

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

if (employee == null)

{

return HttpNotFound();

}

return View(employee);

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Create([Bind(Include

= "Id,ToName,FromName,Amount")] Transactions employee)

{

if (ModelState.IsValid)

{

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress);

var response = await client.PostAsJsonAsync("transactions/Create", employee);

if (response.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

else

{

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

}

return View(employee);

}

public async Task<ActionResult> Edit(int id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Accounts employee = null;

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress)

;

var result = await client.GetAsync($"employees/details/{ id}");

if (result.IsSuccessStatusCode)

{

employee = await result.Content.ReadAsAsync<Accounts>();

}

else

{

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

if (employee == null)

{

return HttpNotFound();

}

return View(employee);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Edit([Bind(Include =

"Id,ToName,FromName,Amount")] Transactions employee)

{

if (ModelState.IsValid)

{

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress);

var response = await client.PutAsJsonAsync("transactions/edit", employee);

if (response.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

else

{

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

return RedirectToAction("Index");

}

return View(employee);

}

public async Task<ActionResult> Delete(int id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Accounts employee = null;

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress)

;

var result = await client.GetAsync($"transactions/details/{ id}");

if (result.IsSuccessStatusCode)

{

employee = await result.Content.ReadAsAsync<Accounts>();

}

else

{

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

}

if (employee == null)

{

return HttpNotFound();

}

return View(employee);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> DeleteConfirmed(int id)

{

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(apiBaseAddress)

;

var response = await client.DeleteAsync($"transactions/delete/{ id}");

if (response.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

else

ModelState.AddModelError(string.Empty, "Server error try after some time.");

}

return View();

}

}

}

**SqlDbContext:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.Entity;

namespace OelSecondCloud.Models

{

public class SqlDbContext : DbContext

{

public SqlDbContext() : base("name=SqlConn")

{

}

public DbSet<Accounts> Accounts { get; set; }

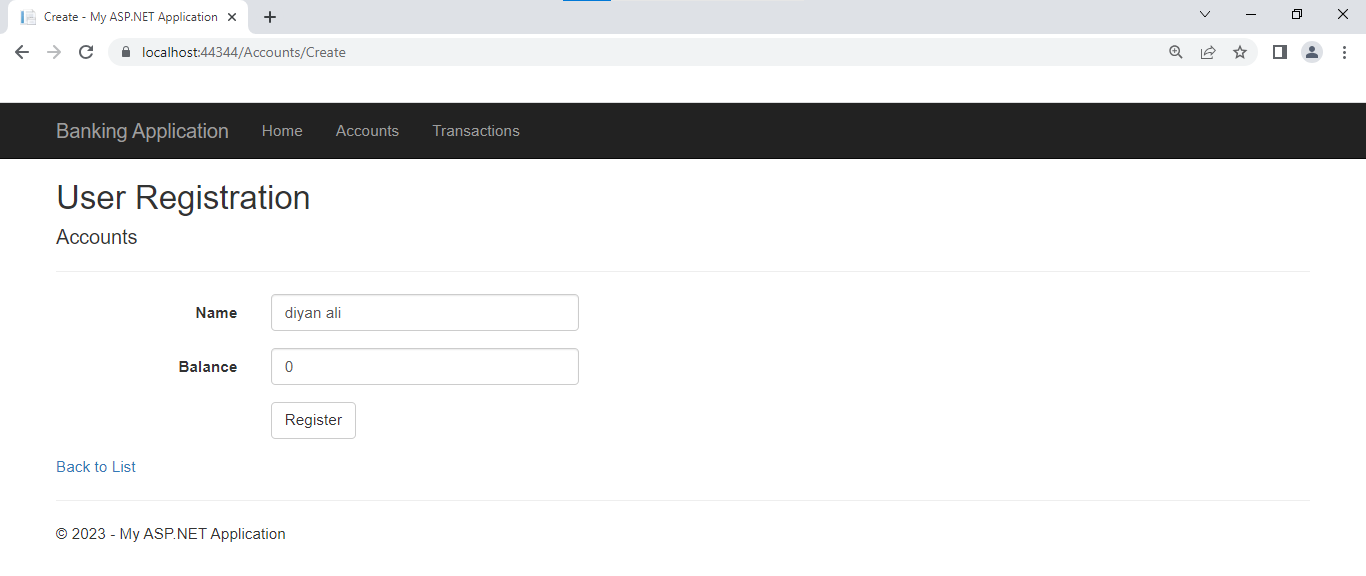
public DbSet<Transactions> Transactions { get; set; }

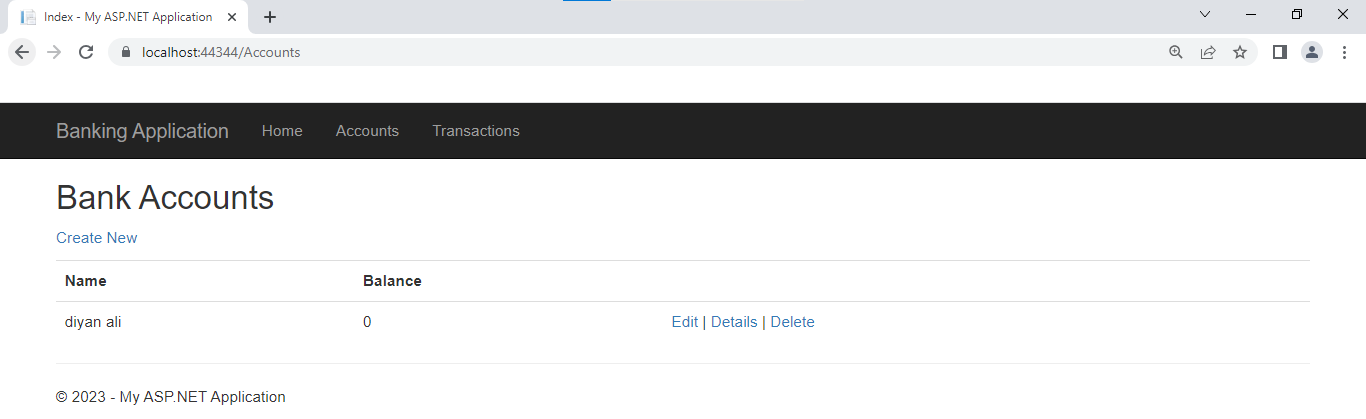
}

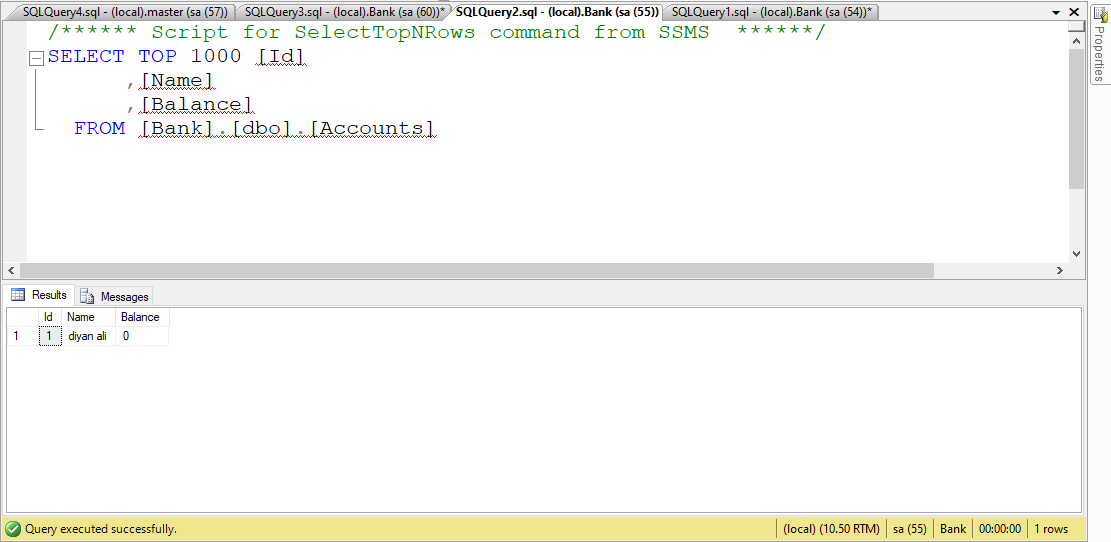
}

**Code Screen Shots:**

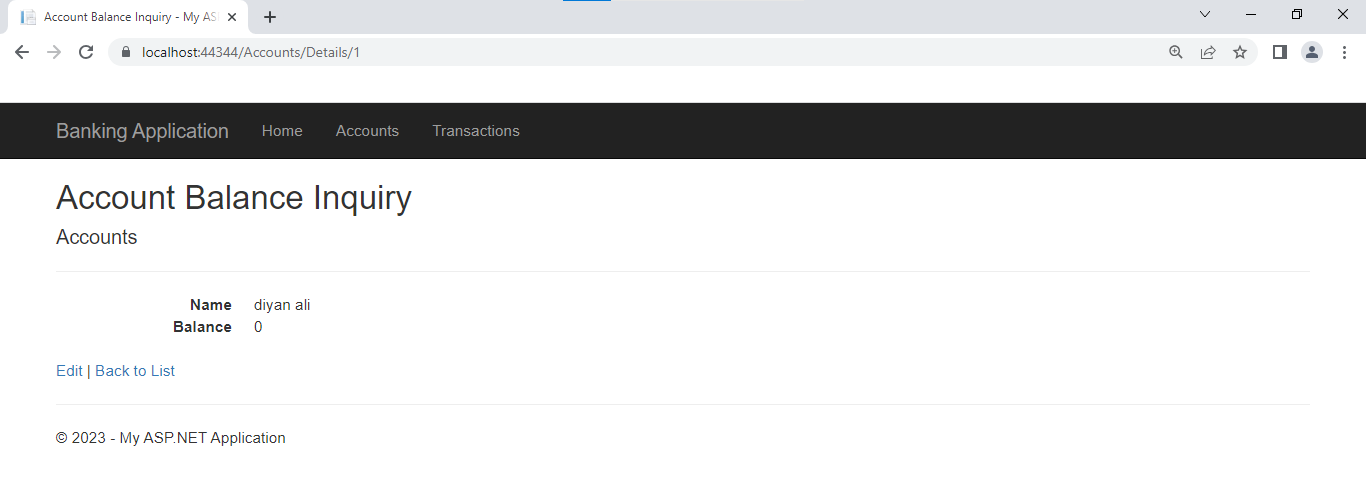
**Task 1:**

****

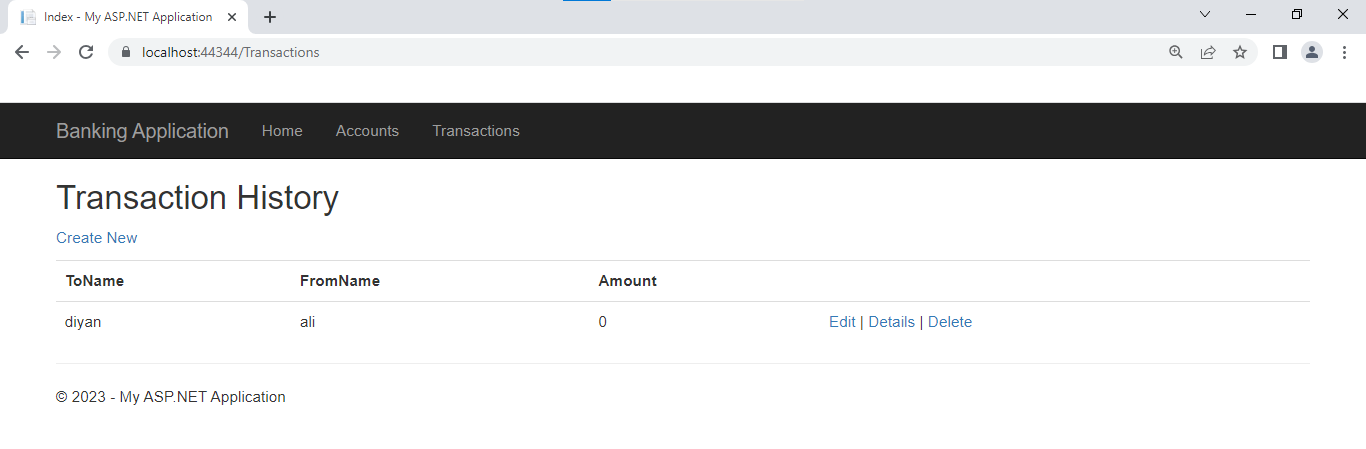
****

****

**Task 2:**

****

**Task 6:**

****