**LAB # 13**

**Task # 01: Employee api with CRUD**

**Solution:**

**EmployeesController:**

namespace WebApplication24.Controllers

{

public class EmployeesController : Controller

{

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

public async Task<ActionResult> Index()

{

IEnumerable<Employee> 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<Employee>>();

}

else

{

employees = Enumerable.Empty<Employee>();

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

}

}

return View(employees);

}

public async Task<ActionResult> Details(string id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Employee 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<Employee>();

}

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 = "Name,Address,Gender,Company,Designation")] Employee 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(string id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Employee 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<Employee>();

}

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,Address,Gender,Company,Designation")] Employee 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(string id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Employee 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<Employee>();

}

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(string 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();

} }}

**EmployeeApiController:**

namespace WebApplication24.Controllers

{

public class EmployeesApiController : ApiController

{

private readonly IEmployeeRepository \_iEmployeeRepository = new EmployeeRepository();

[HttpGet]

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

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

{

return await \_iEmployeeRepository.GetEmployees();

}

[HttpPost]

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

public async Task CreateAsync([FromBody]Employee employee)

{

if (ModelState.IsValid)

{

await \_iEmployeeRepository.Add(employee);

}

}

[HttpGet]

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

public async Task<Employee> Details(string id)

{

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

return result;

}

[HttpPut]

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

public async Task EditAsync([FromBody]Employee employee)

{

if (ModelState.IsValid)

{

await \_iEmployeeRepository.Update(employee);

}

}

[HttpDelete]

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

public async Task DeleteConfirmedAsync(string id)

{

await \_iEmployeeRepository.Delete(id);

}} }

**EmployeesRepository:**

namespace WebApplication24.Models

{

public class EmployeeRepository : IEmployeeRepository

{

private readonly SqlDbContext db = new SqlDbContext();

public async Task Add(Employee employee)

{

employee.Id = Guid.NewGuid().ToString();

db.Employees.Add(employee);

try

{

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

public async Task<Employee> GetEmployee(string id)

{

try

{

Employee employee = await db.Employees.FindAsync(id);

if (employee == null)

{

return null;

}

return employee;

}

catch

{

throw;

}

}

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

{

try

{

var employees = await db.Employees.ToListAsync();

return employees.AsQueryable();

}

catch

{

throw;

}

}

public async Task Update(Employee employee)

{

try

{

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

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

public async Task Delete(string id)

{

try

{

Employee employee = await db.Employees.FindAsync(id);

db.Employees.Remove(employee);

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

private bool EmployeeExists(string id)

{

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

}

}}

**IEmployeeRepository:**

namespace WebApplication24.Models

{

interface IEmployeeRepository

{

Task Add(Employee employee);

Task Update(Employee employee);

Task Delete(string id);

Task<Employee> GetEmployee(string id);

Task<IEnumerable<Employee>> GetEmployees();

}

}

**SQLSBContext class:**

namespace WebApplication24.Models

{

public class SqlDbContext:DbContext

{

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

{

}

public DbSet<Employee> Employees { get; set; }

}

}

**Employee class:**

namespace WebApplication24.Models

{

public class Employee

{

public string Id { get; set; }

public string Name { get; set; }

public string Address { get; set; }

public string Gender { get; set; }

public string Company { get; set; }

public string Designation { get; set; }

}

}

**Webconfig file:**

**Connection String tag:**

<connectionStrings>

<add name="SqlConn"

connectionString="Data Source=NCMS-SDL-028;Initial Catalog=SarathlalDB;User ID=sa;Password=admin@spring2020"

providerName="System.Data.SqlClient"

/>

</connectionStrings>

App setting tag:

<appSettings>

<add key="webpages:Version" value="3.0.0.0" />

<add key="webpages:Enabled" value="false" />

<add key="ClientValidationEnabled" value="true" />

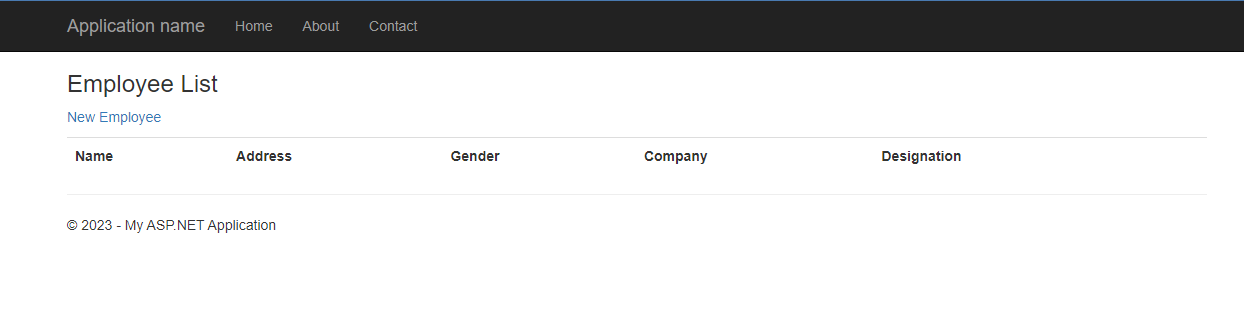
<add key="UnobtrusiveJavaScriptEnabled" value="true" />

<add key="apiBaseAddress" value="https://localhost:44361/api/EmployeesApi"/>

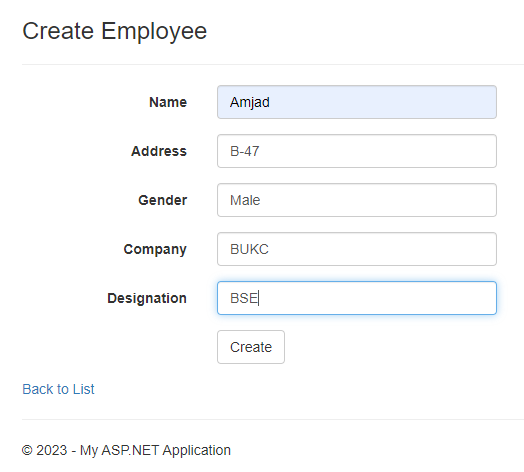
</appSettings>

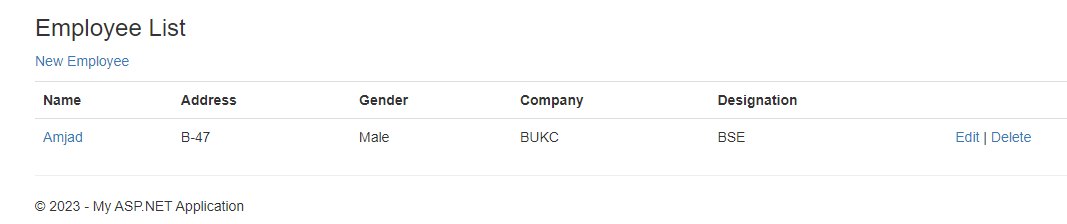
**View Code:**

**Output:**

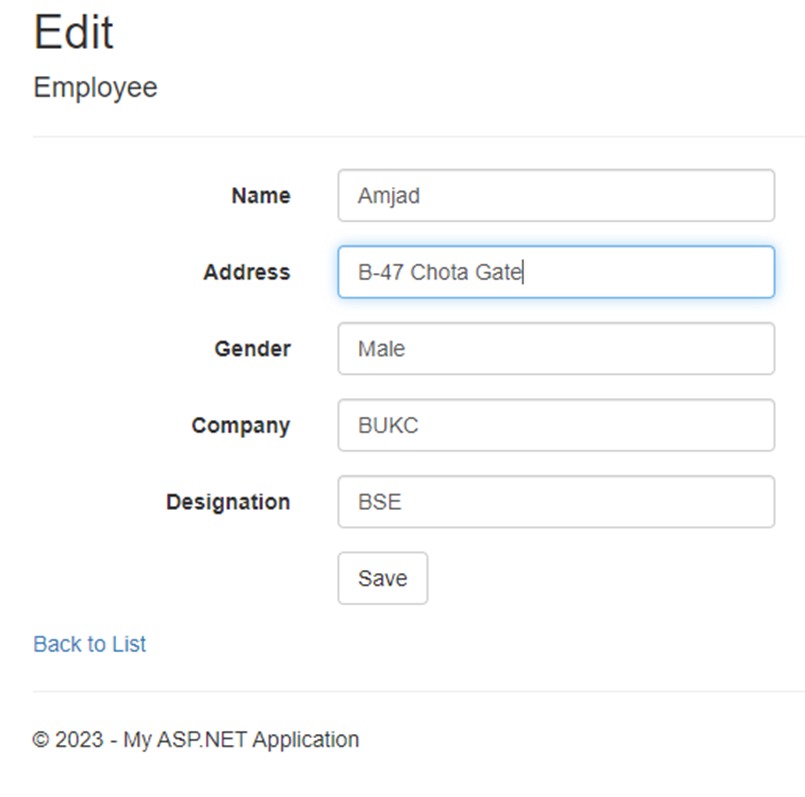
**Index:**

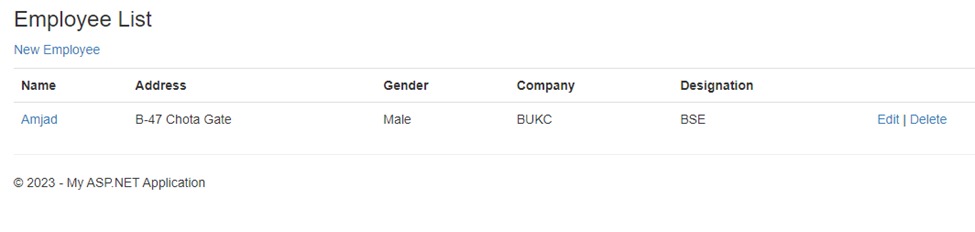
**Create:**

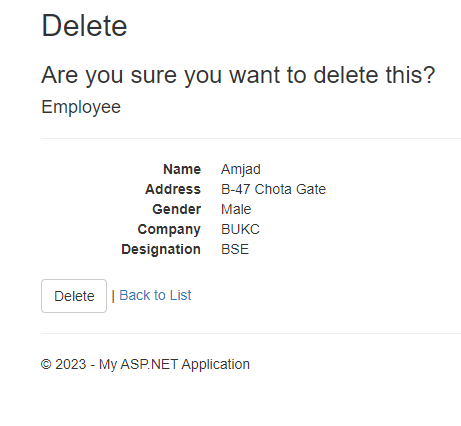


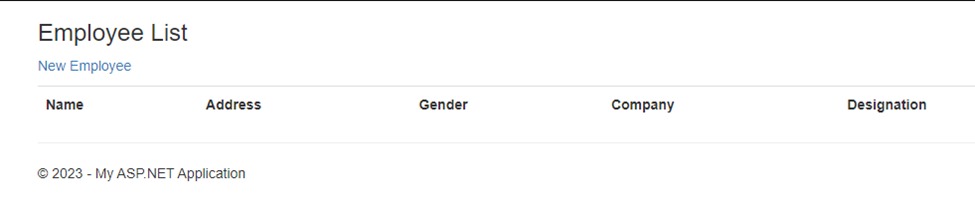


**Edit:**

****

****

**Delete:**

****

**Task # 02: Create a payroll api and perform crud operations on it.**

**Solution:**

**DB Code:**

CREATE DATABASE PayrollDB;

USE PayrollDB;

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Salary DECIMAL(10, 2),

Department VARCHAR(50)

);

**Payroll Model:**

public class Payroll

{

[Key]

public int EmployeeID { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public decimal Salary { get; set; }

public string Department { get; set; }

}

**SqlDbContext Class:**

namespace PayrollApi.Models

{

public class SqlDbContext : DbContext

{

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

{

}

public DbSet<Payroll> payroll { get; set; }

}

}

**SqlConnection String:**

<connectionStrings>

<add name="SqlConn"

connectionString="Data Source=DESKTOP-27TFO05;Initial Catalog=PayrollDB;Integrated Security=True"

providerName="System.Data.SqlClient" />

</connectionStrings>

**IPayroll Repository:**

interface IPayrollRepository

{

Task Add(Payroll payroll);

Task Update(Payroll payroll);

Task Delete(int id);

Task<Payroll> GetPayroll(int id);

Task<IEnumerable<Payroll>> GetPayrolls();

}

**Payroll Repository:**

namespace PayrollApi.Models

{

public class PayrollRepository : IPayrollRepository

{

private readonly SqlDbContext db = new SqlDbContext();

public async Task Add(Payroll payroll)

{

payroll.EmployeeID = (int)Guid.NewGuid().GetHashCode();

db.payroll.Add(payroll);

try

{

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

public async Task Delete(int id)

{

try

{

Payroll payroll = await db.payroll.FindAsync(id);

db.payroll.Remove(payroll);

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

public async Task<Payroll> GetPayroll(int id)

{

try

{

Payroll payroll = await db.payroll.FindAsync(id);

if (payroll == null)

{

return null;

}

return payroll;

}

catch

{

throw;

}

}

public async Task<IEnumerable<Payroll>> GetPayrolls()

{

try

{

var employees = await db.payroll.ToListAsync();

return employees.AsQueryable();

}

catch

{

throw;

}

}

public async Task Update(Payroll payroll)

{

try

{

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

await db.SaveChangesAsync();

}

catch

{

throw;

}

}

private bool EmployeeExists(int id)

{

return db.payroll.Count(e => e.EmployeeID == id) > 0;

}

}

}

**Payroll Api Controller:**

namespace PayrollApi.Controllers

{

public class PayrollApiController : ApiController

{

private readonly IPayrollRepository \_iPayrollRepository = new PayrollRepository();

[HttpGet]

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

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

{

return await \_iPayrollRepository.GetPayrolls();

}

[HttpPost]

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

public async Task CreateAsync([FromBody]Payroll employee)

{

if (ModelState.IsValid)

{

await \_iPayrollRepository.Add(employee);

}

}

[HttpGet]

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

public async Task<Payroll> Details(int id)

{

var result = await \_iPayrollRepository.GetPayroll(id);

return result;

}

[HttpPut]

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

public async Task EditAsync([FromBody]Payroll employee)

{

if (ModelState.IsValid)

{

await \_iPayrollRepository.Update(employee);

}

}

[HttpDelete]

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

public async Task DeleteConfirmedAsync(int id)

{

await \_iPayrollRepository.Delete(id);

}

}

}

**AppSetting tag:**

<appSettings>

<add key="webpages:Version" value="3.0.0.0" />

<add key="webpages:Enabled" value="false" />

<add key="ClientValidationEnabled" value="true" />

<add key="UnobtrusiveJavaScriptEnabled" value="true" />

<add key="apiBaseAddress" value="http://localhost:51369/PayrollApi"/>

</appSettings>

**Payroll Controller:**

namespace PayrollApi.Controllers

{

public class PayrollsController : Controller

{

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

private SqlDbContext db = new SqlDbContext();

// GET: Payrolls

public ActionResult Index()

{

return View(db.Payroll.ToList());

}

// GET: Payrolls/Details/5

public ActionResult Details(string id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Payroll payroll = db.Payroll.Find(id);

if (payroll == null)

{

return HttpNotFound();

}

return View(payroll);

}

// GET: Payrolls/Create

public ActionResult Create()

{

return View();

}

// POST: Payrolls/Create

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "Id,FirstName,LastName,Salary,Department")] Payroll payroll)

{

if (ModelState.IsValid)

{

db.Payroll.Add(payroll);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(payroll);

}

// GET: Payrolls/Edit/5

public ActionResult Edit(string id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Payroll payroll = db.Payroll.Find(id);

if (payroll == null)

{

return HttpNotFound();

}

return View(payroll);

}

// POST: Payrolls/Edit/5

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "Id,FirstName,LastName,Salary,Department")] Payroll payroll)

{

if (ModelState.IsValid)

{

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

db.SaveChanges();

return RedirectToAction("Index");

}

return View(payroll);

}

// GET: Payrolls/Delete/5

public ActionResult Delete(string id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Payroll payroll = db.Payroll.Find(id);

if (payroll == null)

{

return HttpNotFound();

}

return View(payroll);

}

// POST: Payrolls/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(string id)

{

Payroll payroll = db.Payroll.Find(id);

db.Payroll.Remove(payroll);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

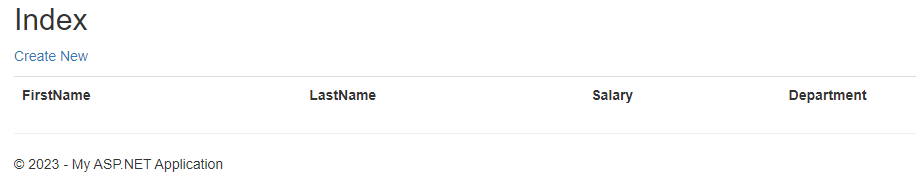
base.Dispose(disposing);

}

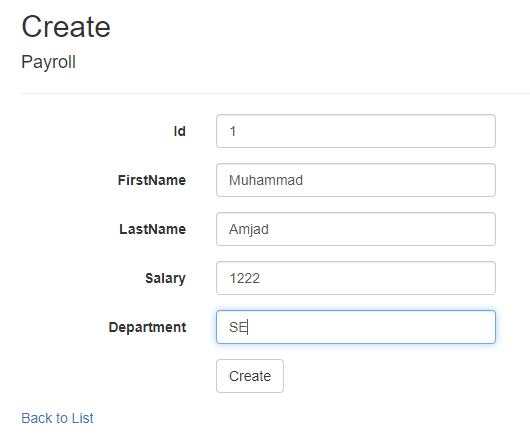
}

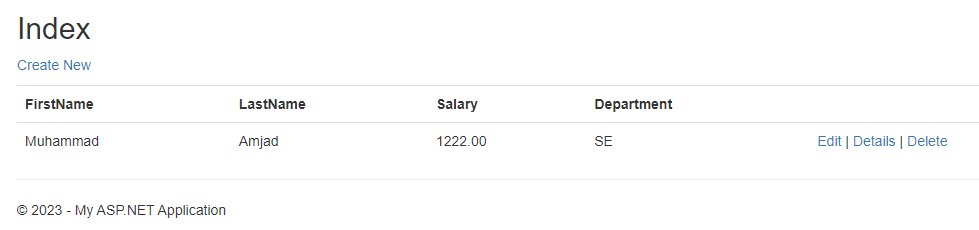
}

**Output:**

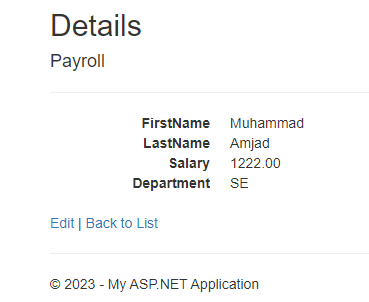


**Create:**

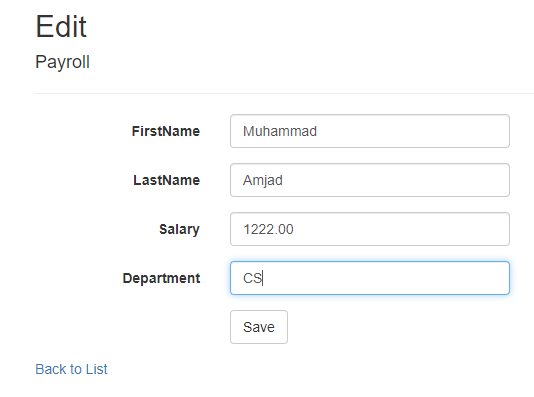


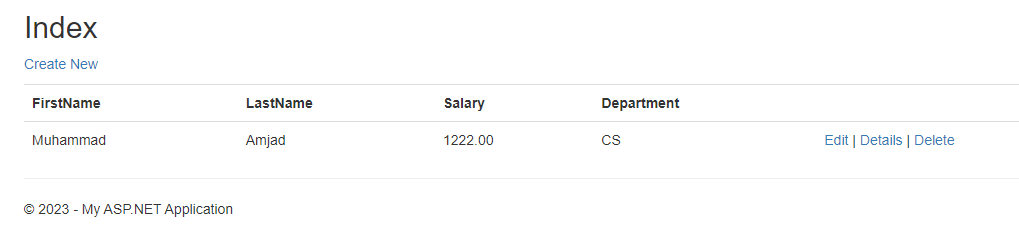


**Details:**



**Edit:**





**Delete:**

