Mohammad Irfan Uddin

Model:

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
using System;
using System.Globalization;
using Microsoft.EntityFrameworkCore;

namespace GTHRTraining.Models
{
    public class Company
    {
        [Key]
        public Guid ComId { get; set; }
        public string ComName { get; set; }
        public double? Basic { get; set; }
        public double? Hrent { get; set; }
        public double? Medical { get; set; }
        public bool IsInactive { get; set; }
    }

    public class Department
    {
        [Key]
        public Guid DeptId { get; set; }
        [Key]
        public Guid ComId { get; set; }
        [ForeignKey("ComId")]
        public Company Com { get; set; }
        public string DeptName { get; set; }
        [NotMapped]
        public string? ComName { get; set; }
    }

    public class Designation
    {
        [Key]
        public Guid DesigId { get; set; }
        [Key]
        public Guid ComId { get; set; }
        [ForeignKey("ComId")]
        public Company Com { get; set; }
        public string DesigName { get; set; }

        [InverseProperty(nameof(Employee.Designation))]
        public virtual ICollection<Employee> Employees { get; set; }
        [NotMapped]
        public string? ComName { get; set; }
    }

    public class Shift
    {
        [Key]
```

```
    public Guid ShiftId { get; set; }
    [Key]
    public Guid ComId { get; set; }
    [ForeignKey("ComId")]
    public Company Com { get; set; }
    public string ShiftName { get; set; }
    public TimeSpan? ShiftIn { get; set; }
    public TimeSpan? ShiftOut { get; set; }
    public TimeSpan? ShiftLate { get; set; }
    [InverseProperty(nameof(Employee.Shift))]
    public virtual ICollection<Employee> Employees { get; set; }
    [NotMapped]
    public string? ComName { get; set; }
}

public class Employee
{
    public Guid EmpId { get; set; }
    public string EmpCode { get; set; }
    public string EmpName { get; set; }

    public Guid ShiftId { get; set; }

    [ForeignKey(nameof(ShiftId))]
    public virtual Shift Shift { get; set; }

    public Guid ComId { get; set; }
    public Guid DeptId { get; set; }

    [ForeignKey(nameof(DeptId) + "," + nameof(ComId))]
    public virtual Department Department { get; set; }

    public Guid DesigId { get; set; }

    [ForeignKey(nameof(DesigId))]
    public virtual Designation Designation { get; set; }

    public string Gender { get; set; }
    public double? Gross { get; set; }
    public double? Basic { get; set; }
    public double? Hrent { get; set; }
    public double? Medical { get; set; }
    [DisplayFormat(DataFormatString = "{0:0.00}")]
    public double? Others { get; set; }
    [DisplayFormat(DataFormatString = "{0:d}")]
    public DateTime? dtJoin { get; set; }
    [NotMapped]
    public string? ShiftName { get; set; }
    [NotMapped]
    public string? DeptName { get; set; }
    [NotMapped]
    public string? DesigName { get; set; }
}
```

```
public class Attendance
{
    [Key]
    public Guid ComId { get; set; }
    [ForeignKey("ComId")]
    public Company Com { get; set; }
    [Key]
    public Guid EmpId { get; set; }
    [ForeignKey("EmpId")]
    public Employee Emp { get; set; }
    [Key]
    [DisplayFormat(DataFormatString = "{0:d}")]
    public DateTime dtDate { get; set; }
    public string? AttStatus { get; set; }
    public TimeSpan? InTime { get; set; }
    public TimeSpan? OutTime { get; set; }
    [NotMapped]
    public string? ComName { get; set; }
}

public class AttSummary
{
    [Key]
    public Guid ComId { get; set; }
    [ForeignKey("ComId")]
    public Company Com { get; set; }
    [Key]
    public Guid EmpId { get; set; }
    [ForeignKey("EmpId")]
    public Employee Emp { get; set; }
    [Key, StringLength(4)]
    public string dtYear { get; set; }
    [Key, StringLength(4)]
    public string dtMonth { get; set; }
    public int? Present { get; set; }
    public int? Late { get; set; }
    public int? Absent { get; set; }
    public int? WDay { get; set; }
    [NotMapped]
    public string? ComName { get; set; }
}

public class Salary
{
    [Key]
    public Guid ComId { get; set; }
    [ForeignKey("ComId")]
    public Company Com { get; set; }
    [Key]
    public Guid EmpId { get; set; }
```

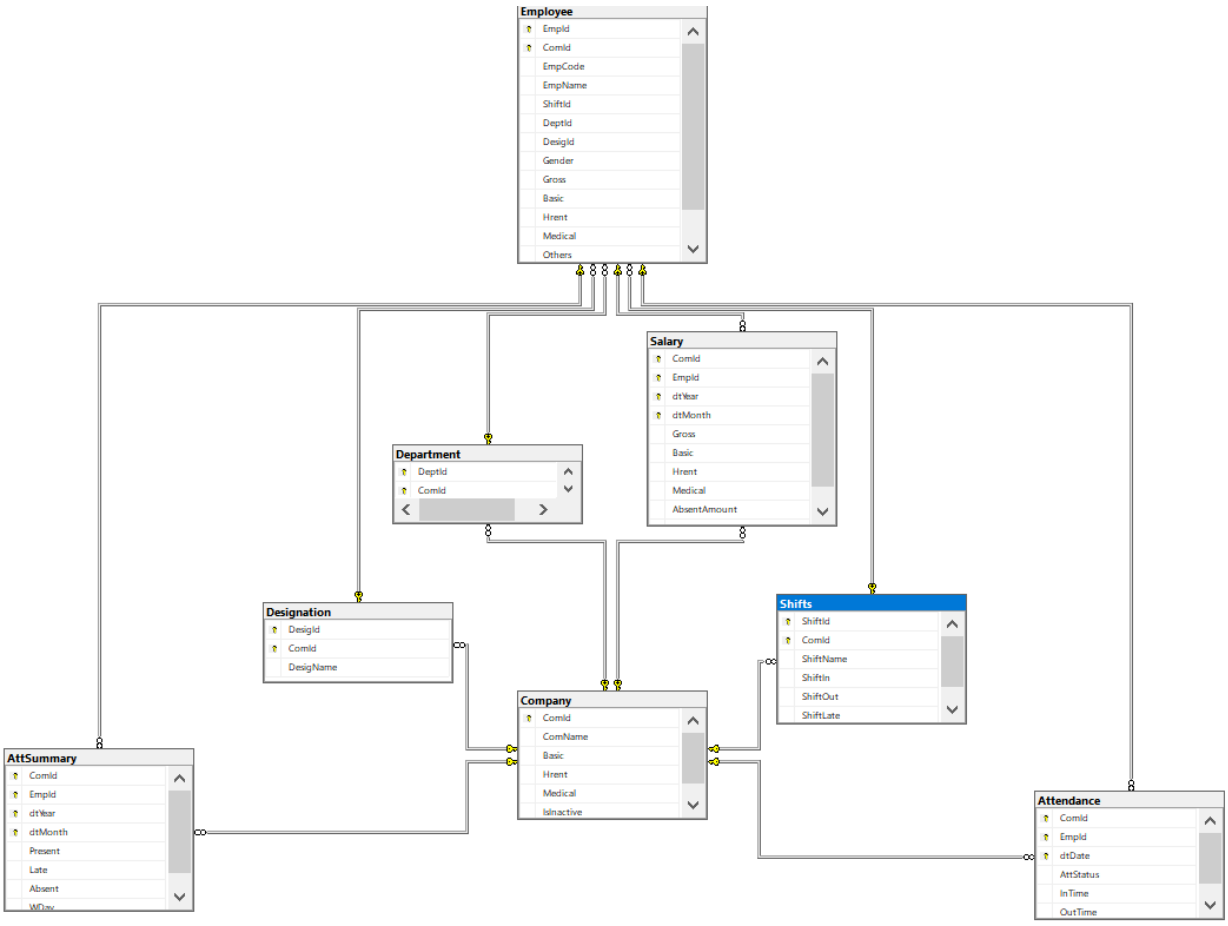
```
        [ForeignKey("EmpId")]
        public Employee Emp { get; set; }
        [Key, StringLength(4)]
        public string dtYear { get; set; }
        [Key, StringLength(4)]
        public string dtMonth { get; set; }
        public double? Gross { get; set; }
        public double? Basic { get; set; }
        public double? Hrent { get; set; }
        public double? Medical { get; set; }
        [DisplayFormat(DataFormatString = "{0:0.00}")]
        public double? AbsentAmount { get; set; }
        [DisplayFormat(DataFormatString = "{0:0.00}")]
        public double? PayableAmount { get; set; }
        [NotMapped]
        public string? ComName { get; set; }
        [NotMapped]
        public string? EmpName { get; set; }
    }
}
```

Context:

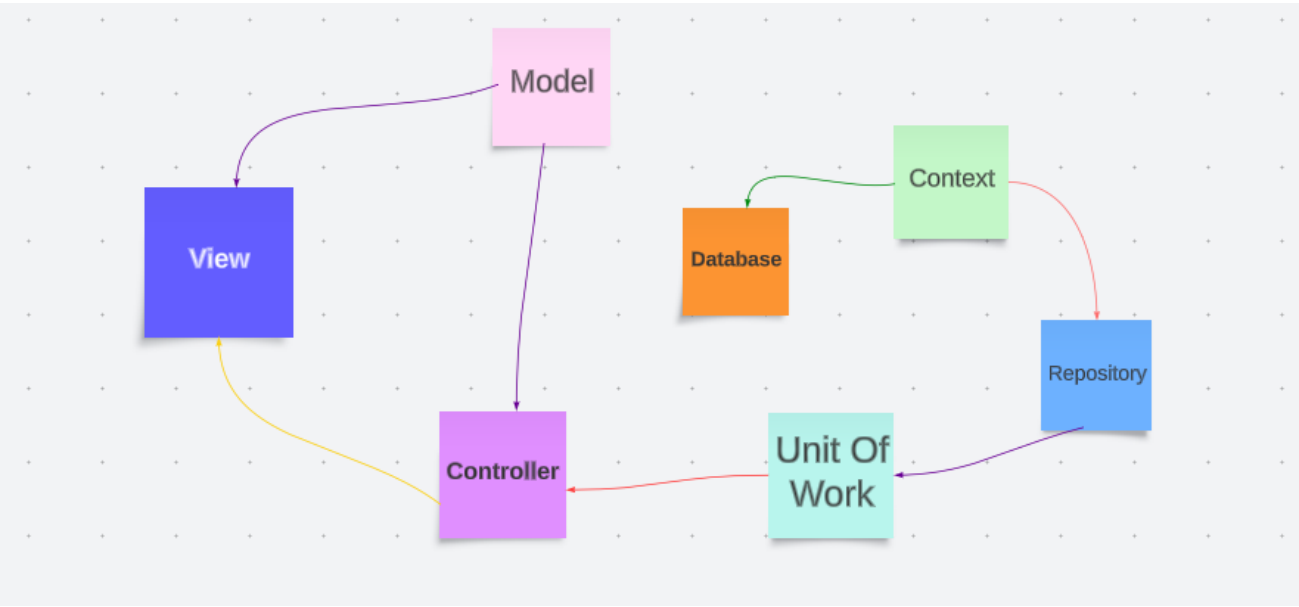
```
using Microsoft.EntityFrameworkCore;

namespace GTHRTraining.Models
{
    public class HRdbContext : DbContext
    {

        public HRdbContext(DbContextOptions<HRdbContext> options)
            : base(options)
        {
        }
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<AttSummary>()
                .HasKey(a => new { a.ComId, a.EmpId, a.dtYear, a.dtMonth });
            modelBuilder.Entity<Attendance>()
              .HasKey(a => new { a.ComId, a.EmpId, a.dtDate});
            modelBuilder.Entity<Salary>()
              .HasKey(a => new { a.ComId, a.EmpId, a.dtYear, a.dtMonth });

            modelBuilder.Entity<Employee>()
        .HasOne(e => e.Department)
        .WithMany()
        .HasForeignKey(e => new { e.DeptId, e.ComId });

            modelBuilder.Entity<Employee>()
        .HasOne(e => e.Designation)
        .WithMany(d => d.Employees)
        .HasForeignKey(e => new { e.DesigId, e.ComId })
        .OnDelete(DeleteBehavior.NoAction);
```

```
            modelBuilder.Entity<Employee>()
        .HasOne(e => e.Shift)
        .WithMany(s => s.Employees)
        .HasForeignKey(e => new { e.ShiftId, e.ComId })
        .OnDelete(DeleteBehavior.NoAction);


            modelBuilder.Entity<Attendance>()
        .HasOne(a => a.Emp)
        .WithMany()
        .HasForeignKey(a => new { a.EmpId, a.ComId })
        .OnDelete(DeleteBehavior.NoAction);

            modelBuilder.Entity<AttSummary>()
        .HasOne(a => a.Emp)
        .WithMany()
        .HasForeignKey(a => new { a.EmpId, a.ComId })
        .OnDelete(DeleteBehavior.NoAction);

            modelBuilder.Entity<Salary>()
        .HasOne(s => s.Emp)
        .WithMany()
        .HasForeignKey(s => new { s.EmpId, s.ComId })
        .OnDelete(DeleteBehavior.NoAction);

            modelBuilder.Entity<Department>()
        .HasKey(d => new { d.DeptId, d.ComId });

            modelBuilder.Entity<Designation>()
        .HasKey(d => new { d.DesigId, d.ComId });

            modelBuilder.Entity<Shift>()
        .HasKey(d => new { d.ShiftId, d.ComId });

            modelBuilder.Entity<Employee>()
        .HasKey(d => new { d.EmpId, d.ComId });



            // Other configurations...

            base.OnModelCreating(modelBuilder);
        }

        public DbSet<Company> Company { get; set; }
        public DbSet<Department> Department { get; set; }
        public DbSet<Designation> Designation { get; set; }
        public DbSet<Shift> Shifts { get; set; }
        public DbSet<Employee> Employee { get; set; }
        public DbSet<Attendance> Attendance { get; set; }
        public DbSet<AttSummary> AttSummary { get; set; }
        public DbSet<Salary> Salary { get; set; }

    }
}
```

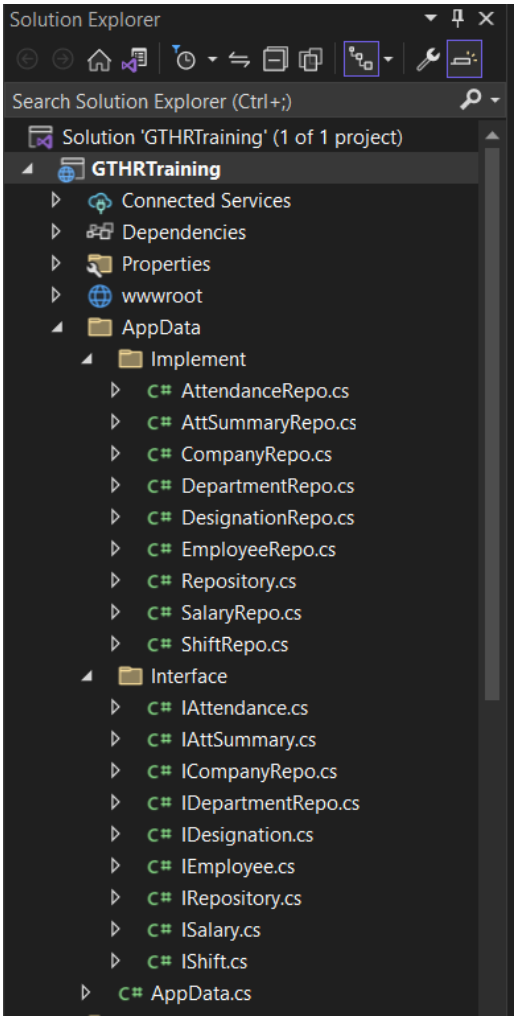
Table Diagram:

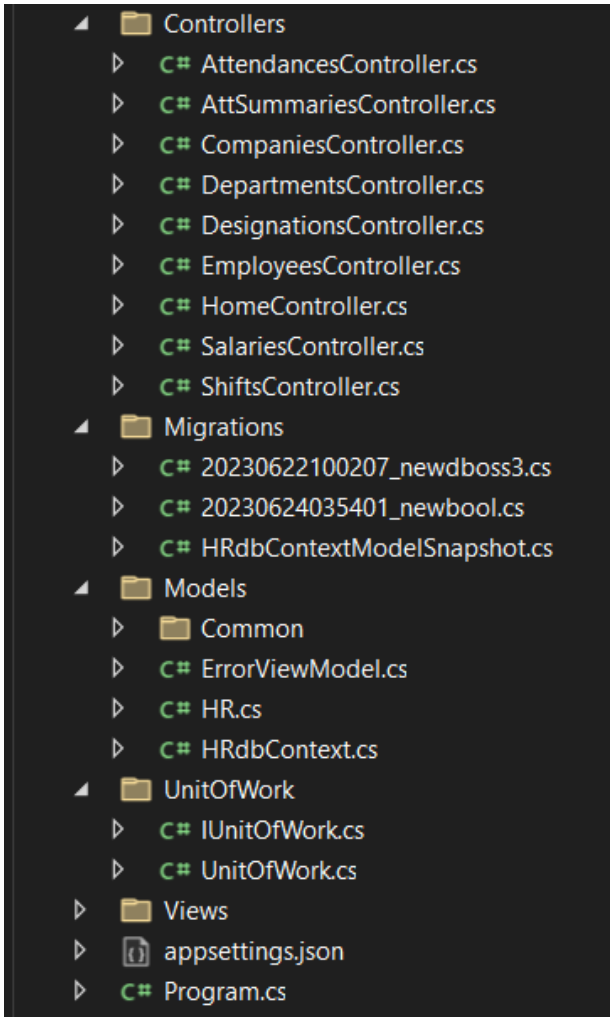


WorkFlow:



Project Files:





Repository:

Main Repository:

```
using GTHRTraining.AppData.Interface;
using GTHRTraining.Models;
using Microsoft.EntityFrameworkCore;
using System.Linq.Expressions;

namespace GTHRTraining.AppData.Implement
{
    public class Repository<T> : IRepository<T> where T : class
    {
        protected readonly HRdbContext _context;
        public Repository(HRdbContext context)
        {
            _context = context;
        }
        public async Task<int> Count(Expression<Func<T, bool>> predicate)
        {
            return await _context.Set<T>().CountAsync(predicate);
        }
```

```
public async Task Create(T model)
{
    await _context.Set<T>().AddAsync(model);
    await SaveChange();
}

public async Task Delete(Guid Id)
{
    var result = await GetById(Id);
    if (result is not null)
    {
        _context.Set<T>().Remove(result);
        await SaveChange();
    }

}
public async Task Delete(Guid Id, Guid ComId)
{
    var result = await GetById(Id,ComId);
    if (result is not null)
    {
        _context.Set<T>().Remove(result);
        await SaveChange();
    }

}
public async Task Delete(Guid ComId, Guid EmpId, DateTime dtDate)
{
    var result = await GetById(ComId, EmpId, dtDate);
    if (result is not null)
    {
        _context.Set<T>().Remove(result);
        await SaveChange();
    }

}
public async Task Delete(Guid ComId, Guid EmpId, string dtMonth, string dtYear)
{
    var result = await GetById(ComId, EmpId, dtMonth, dtYear);
    if (result is not null)
    {
        _context.Set<T>().Remove(result);
        await SaveChange();
    }

}

private Task GetById(object id)
{
    throw new NotImplementedException();
}

public async Task<bool> Exist(Expression<Func<T, bool>> predicate)
{
    return await _context.Set<T>().AnyAsync();
```

```
}

public async Task<IEnumerable<T>> GetAllAsync()
{
    return await _context.Set<T>().ToListAsync();
}

public async Task<T> GetById(Guid id)
{
    return await _context.Set<T>().FindAsync(id);

}
public async Task<T> GetById(Guid id, Guid ComId)
{
    return await _context.Set<T>().FindAsync(id,ComId);

}
public async Task<T> GetById(Guid ComId, Guid EmpId, DateTime dtDate)
{
    return await _context.Set<T>().FindAsync(ComId, EmpId, dtDate);

}
public async Task<T> GetById(Guid ComId, Guid EmpId, string dtMonth, string dtYear)
{
    return await _context.Set<T>().FindAsync(ComId, EmpId, dtMonth, dtYear);

}

public async Task<T> Single(Expression<Func<T, bool>> predicate)
{
    return await _context.Set<T>().SingleOrDefaultAsync(predicate);

}

public async Task Update(T model, Guid Id)
{
    if (model == null)
    {
        return;
    }
    T oldData = await _context.Set<T>().FindAsync(Id);
    if (oldData != null)
    {
        _context.Entry(oldData).CurrentValues.SetValues(model);
        await SaveChange();
    }

}
public async Task Update(T model, Guid Id, Guid ComId)
{
    if (model == null)
    {
        return;
    }
    T oldData = await _context.Set<T>().FindAsync(Id, ComId);
    if (oldData != null)
```

```
        {
            _context.Entry(oldData).CurrentValues.SetValues(model);
            await SaveChange();
        }

    }
    public async Task Update(T model, Guid ComId, Guid EmpId, DateTime dtDate)
    {
        if (model == null)
        {
            return;
        }
        T oldData = await _context.Set<T>().FindAsync(ComId, EmpId, dtDate);
        if (oldData != null)
        {
            _context.Entry(oldData).CurrentValues.SetValues(model);
            await SaveChange();
        }

    }
    public async Task Update(T model, Guid ComId, Guid EmpId, string dtMonth, string dtYear)
    {
        if (model == null)
        {
            return;
        }
        T oldData = await _context.Set<T>().FindAsync(ComId, EmpId, dtMonth, dtYear);
        if (oldData != null)
        {
            _context.Entry(oldData).CurrentValues.SetValues(model);
            await SaveChange();
        }

    }
    public Task<IEnumerable<T>> Where(Expression<Func<T, bool>> predicate)
    {
        throw new NotImplementedException();
    }
    public async Task SaveChange()
    {
        await _context.SaveChangesAsync();

    }
  }
}
```

Attendance repositoy

```
using GTHRTraining.AppData.Interface;
using GTHRTraining.Models;

namespace GTHRTraining.AppData.Implement
{
    public class AttendanceRepo : Repository<Attendance>, IAttendance
    {
        public AttendanceRepo(HRdbContext context) : base(context)
        {
        }
    }
}
```

Company repository

```
using GTHRTraining.AppData.Interface;
using GTHRTraining.Models;

namespace GTHRTraining.AppData.Implement
{
    public class CompanyRepo : Repository<Company>, ICompanyRepo
    {
    public CompanyRepo(HRdbContext context) : base(context)
    {
    }
}
}
```

Department repository

```
using GTHRTraining.AppData.Interface;
using GTHRTraining.Models;
using System.Linq.Expressions;

namespace GTHRTraining.AppData.Implement
{
    public class DepartmentRepo : Repository<Department>, IDepartmentRepo
    {
        public DepartmentRepo(HRdbContext context) : base(context)
        {
        }
    }
}
```

Designation Repository

```
using GTHRTraining.AppData.Interface;
using GTHRTraining.Models;

namespace GTHRTraining.AppData.Implement
{
    public class DesignationRepo : Repository<Designation>, IDesignation
    {
        public DesignationRepo(HRdbContext context) : base(context)
        {
        }
    }
}
```

Employee repository

```
using GTHRTraining.AppData.Interface;
using GTHRTraining.Models;

namespace GTHRTraining.AppData.Implement
{
    public class EmployeeRepo : Repository<Employee>, IEmployee
    {
        public EmployeeRepo(HRdbContext context) : base(context)
        {
        }
    }
}
```

Salary Repository

```
using GTHRTraining.AppData.Interface;
using GTHRTraining.Models;

namespace GTHRTraining.AppData.Implement
{
    public class SalaryRepo : Repository<Salary>, ISalary
    {
        public SalaryRepo(HRdbContext context) : base(context)
        {
        }
    }
}
```

Shift Repository

```
using GTHRTraining.AppData.Interface;
using GTHRTraining.Models;

namespace GTHRTraining.AppData.Implement
{
    public class ShiftRepo : Repository<Shift>, IShift
    {
        public ShiftRepo(HRdbContext context) : base(context)
        {
        }
    }
}
```

Repository interface

Main repository interface

```
using System.Linq.Expressions;

namespace GTHRTraining.AppData.Interface
{
    public interface IRepository<T> where T : class
    {

        Task<IEnumerable<T>> GetAllAsync();
        Task<IEnumerable<T>> Where(Expression<Func<T, bool>> predicate);
        Task<T> Single(Expression<Func<T, bool>> predicate);
```

```
        Task<T> GetById(Guid id);
        Task<T> GetById(Guid id, Guid ComId);
        Task<T> GetById(Guid ComId, Guid EmpId, DateTime dtDate);
        Task<T> GetById(T model, Guid ComId, Guid EmpId, string dtMonth, string dtYear);
        Task Create(T model);
        Task Update(T model, Guid Id);
        Task Update(T model, Guid Id, Guid ComId);
        Task Update(T model, Guid ComId, Guid EmpId, DateTime dtDate);
        Task Update(T model, Guid ComId, Guid EmpId, string dtMonth, string dtYear);
        Task Delete(Guid Id);
        Task Delete(Guid Id, Guid ComId);
        Task Delete(T model, Guid ComId, Guid EmpId, DateTime dtDate);
        Task Delete(T model, Guid ComId, Guid EmpId, string dtMonth, string dtYear);

        Task<int> Count(Expression<Func<T, bool>> predicate);
        Task<bool> Exist(Expression<Func<T, bool>> predicate);
    }
}
```

Common figure for all repository interface

```
using GTHRTraining.Models;

namespace GTHRTraining.AppData.Interface
{
    public interface IAttendance : IRepository<Attendance>
    {
    }
}
```

Unit of Work:

```
using GTHRTraining.AppData.Implement;
using GTHRTraining.AppData.Interface;
using GTHRTraining.Models;

namespace GTHRTraining.UnitOfWork
{
    public class UnitOfWork : IUnitOfWork
    {
        private readonly HRdbContext _dbContext;
        private readonly Dictionary<Type, object> _repositories;

        public UnitOfWork(HRdbContext dbContext)
        {
            _dbContext = dbContext;
            _repositories = new Dictionary<Type, object>();
        }

        public IRepository<T> GetRepository<T>() where T : class
        {
            if (_repositories.ContainsKey(typeof(T)))
            {
                return (IRepository<T>)_repositories[typeof(T)];
            }
```

```
            var repository = new Repository<T>(_dbContext);
            _repositories.Add(typeof(T), repository);
            return repository;
        }

        public async Task SaveAsync()
        {
            await _dbContext.SaveChangesAsync();
        }

        public void Dispose()
        {
            _dbContext.Dispose();
        }
    }
}
```

Interface of Unit of Work:

```
using GTHRTraining.AppData.Interface;

namespace GTHRTraining.UnitOfWork
{
    public interface IUnitOfWork : IDisposable
    {
        IRepository<T> GetRepository<T>() where T : class;
        Task SaveAsync();
    }
}
```

Appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "Server=DESKTOP-17QPMKC\\SQLEXPRESS;Database=MVCTask; User Id =TeamMVCTask; Password
=TeamMVCTask; TrustServerCertificate=True"
  },
  "AllowedHosts": "*"
}
```

Program.cs

```
using GTHRTraining.AppData.Implement;
using GTHRTraining.AppData.Interface;
using GTHRTraining.Models;
using Microsoft.EntityFrameworkCore;
```

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddDbContext<HRdbContext>(
    options => options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();
var app = builder.Build();

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();
```

Controller

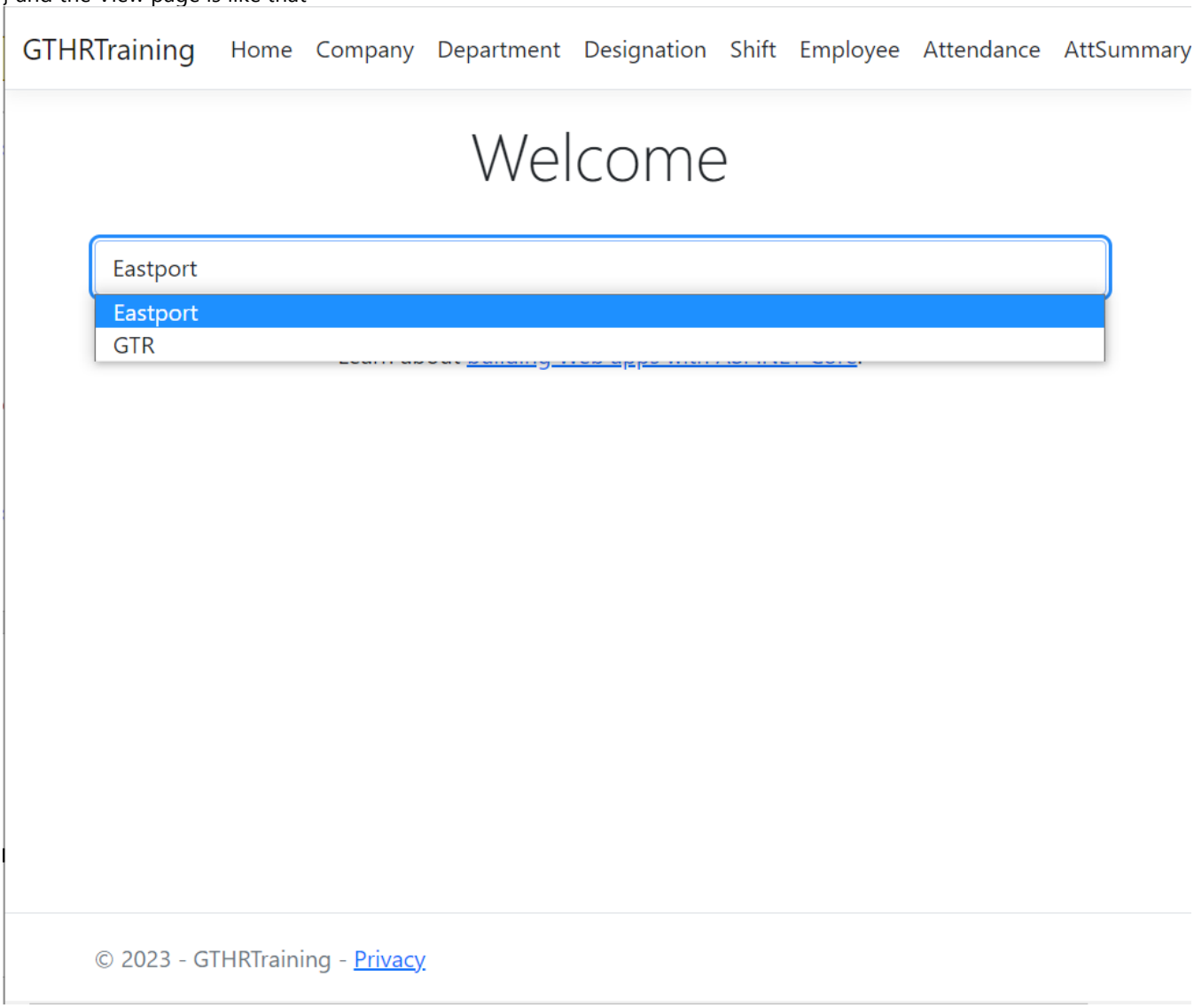
Home Controller

Here to set Company Id in cookie I add a simple code in Index.Cshtml. Here it is-

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <div class="form-group">
        <label  class="control-label"></label>
        <select id="companySelect"  class="form-control" asp-items="ViewBag.ComId"></select>

        <button onclick="setCompanyCookie()">Set Company</button>
    </div>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
</div>

@section Scripts
{
```

```
<script>
  function setCompanyCookie() {
  var dropdown = document.getElementById("companySelect");
  var selectedOption = dropdown.options[dropdown.selectedIndex];
  var comId = selectedOption.value;
  var comName = selectedOption.text;

  // Set cookie with company ID
  document.cookie = "comId=" + comId + "; expires=Fri, 31 Dec 9999 23:59:59 GMT";

  // Set cookie with company name
  document.cookie = "comName=" + encodeURIComponent(comName) + "; expires=Fri, 31 Dec 9999 23:59:59 GMT";

  // Optionally, you can display a confirmation message
  alert("Company set successfully!");
  }
</script>
```

} and the View page is like that



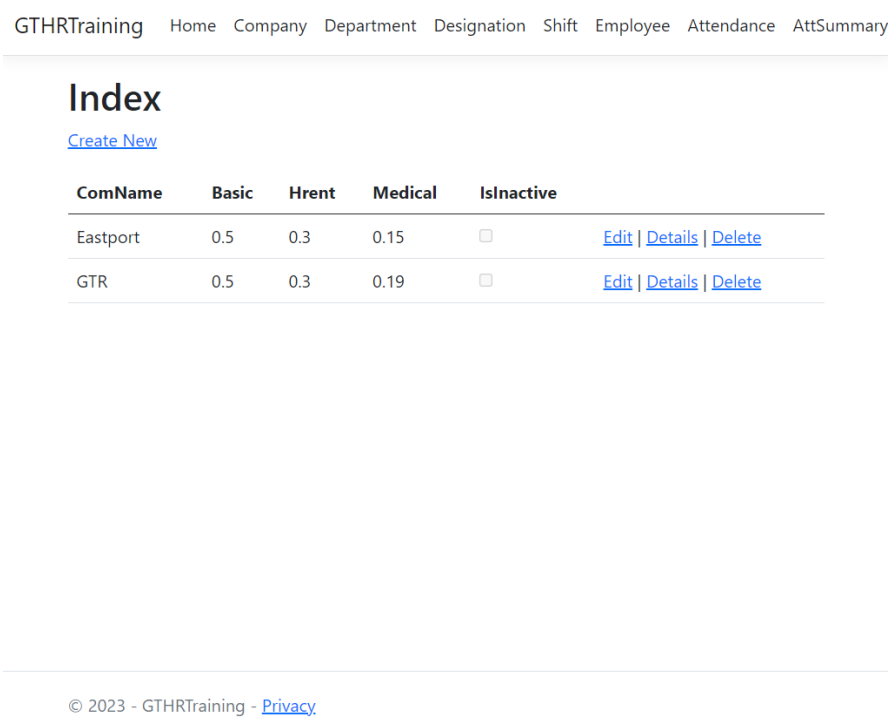
Company Controller

```
public class CompaniesController : Controller
  {
    private readonly IUnitOfWork _unitOfWork;

    public CompaniesController(IUnitOfWork unitOfWork)
    {
      _unitOfWork = unitOfWork;
    }

    // GET: Companies
    public async Task<IActionResult> Index()
    {
      var companyRepo = _unitOfWork.GetRepository<Company>();
      var companies = await companyRepo.GetAllAsync();
      return View(companies);
    }

    public async Task<IActionResult> Details(Guid id)
    {
      var companyRepo = _unitOfWork.GetRepository<Company>();
      var company = await companyRepo.GetById(id);
      if (company is null)
      {
        return NotFound();
      }

      return View(company);
    }

    public IActionResult Create()
    {
      return View();
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create([Bind("ComId,ComName,Basic,Hrent,Medical,IsInactive")] Company company)
    {
      if (ModelState.IsValid)
      {
        var companyRepo = _unitOfWork.GetRepository<Company>();
        await companyRepo.Create(company);
        await _unitOfWork.SaveAsync();
        return RedirectToAction(nameof(Index));
      }
      return View(company);
    }

    public async Task<IActionResult> Edit(Guid id)
    {
      var companyRepo = _unitOfWork.GetRepository<Company>();
      var company = await companyRepo.GetById(id);
      if (company is null)
      {
```

```
            return NotFound();
        }
        return View(company);
    }


    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(Guid id, [Bind("ComId,ComName,Basic,Hrent,Medical,IsInactive")] Company
company)
    {

        if (id != company.ComId)
        {
            return NotFound();
        }
        var companyRepo = _unitOfWork.GetRepository<Company>();
        var olcompany = await companyRepo.GetById(id);
        if (olcompany is null)
        {
            return NotFound();
        }
        if (ModelState.IsValid)
        {
            try
            {
                await companyRepo.Update(company, id);
                await _unitOfWork.SaveAsync();
            }
            catch (DbUpdateConcurrencyException e)
            {
                return BadRequest(e.Message);
            }
            return RedirectToAction(nameof(Index));
        }
        return View(company);
    }

    public async Task<IActionResult> Delete(Guid id)
    {
        var companyRepo = _unitOfWork.GetRepository<Company>();
        var company = await companyRepo.GetById(id);
        if (company is null)
        {
            return NotFound();
        }

        return View(company);
    }

    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(Guid id)
    {
        var companyRepo = _unitOfWork.GetRepository<Company>();
        var company = await companyRepo.GetById(id);
```

```
            if (company is null)
            {
                return NotFound();
            }
            await companyRepo.Delete(id);
            await _unitOfWork.SaveAsync();
            return RedirectToAction(nameof(Index));
        }
    }
```

To Create A company click Company from the dashboard and Click Create New.



Department Controller

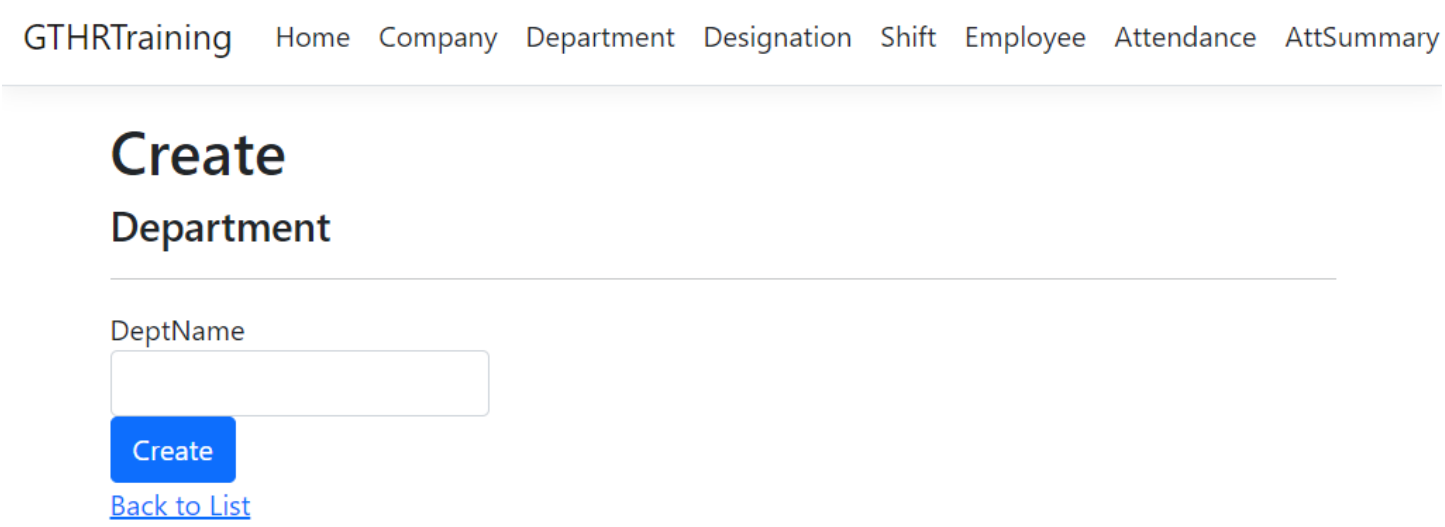
```
public class DepartmentsController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;

        public DepartmentsController(IUnitOfWork unitOfWork)
        {
            _unitOfWork = unitOfWork;
        }

        // GET: Departments
        public async Task<IActionResult> Index()
        {
            var departmentRepo = _unitOfWork.GetRepository<Department>();
            var companyRepo = _unitOfWork.GetRepository<Company>();

            var data = await departmentRepo.GetAllAsync();
            var comid = Guid.Parse(Request.Cookies["comId"]);
```

```
    data = data.Where(x => x.ComId == comid);

    foreach (var flag in data)
    {
        Expression<Func<Company, bool>> predicate = x => x.ComId == flag.ComId;
        flag.ComName = (await companyRepo.Where(predicate)).Select(y => y.ComName).FirstOrDefault();
    }

    return View(data);
}


public IActionResult Create()
{
    return View();
}


[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("DeptId,ComId,DeptName")] Department department)
{
    department.ComId = Guid.Parse(Request.Cookies["comId"]);
    department.DeptId = Guid.NewGuid();

    var departmentRepo = _unitOfWork.GetRepository<Department>();
    await departmentRepo.Create(department);

    await _unitOfWork.SaveAsync();

    return RedirectToAction(nameof(Index));
}

// GET: Departments/Edit/5
public async Task<IActionResult> Edit(Guid DeptId, Guid ComId)
{
    var departmentRepo = _unitOfWork.GetRepository<Department>();
    var companyRepo = _unitOfWork.GetRepository<Company>();

    var depart = await departmentRepo.GetById(DeptId, ComId);
    if (depart is null)
    {
        return NotFound();
    }

    if (depart != null)
    {
        Expression<Func<Company, bool>> predicate = x => x.ComId == depart.ComId;
        depart.ComName = (await companyRepo.Where(predicate)).Select(y => y.ComName).FirstOrDefault();
    }

    return View(depart);
}

[HttpPost]
[ValidateAntiForgeryToken]
```

```
public async Task<IActionResult> Edit([Bind("DeptId,ComId,DeptName")] Department department)
{
    var departmentRepo = _unitOfWork.GetRepository<Department>();

    var olcompany = await departmentRepo.GetById(department.DeptId, department.ComId);
    if (olcompany is null)
    {
        return NotFound();
    }


        try
        {
            await departmentRepo.Update(department, department.DeptId, department.ComId);
            await _unitOfWork.SaveAsync();
        }
        catch (DbUpdateConcurrencyException e)
        {
            return BadRequest(e.Message);
        }

        return RedirectToAction(nameof(Index));


}

// GET: Departments/Delete/5
public async Task<IActionResult> Delete(Guid DeptId, Guid ComId)
{
    var departmentRepo = _unitOfWork.GetRepository<Department>();

    var depart = await departmentRepo.GetById(DeptId, ComId);
    if (depart is null)
    {
        return NotFound();
    }

    return View(depart);
}

// POST: Departments/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed([Bind("DeptId,ComId,DeptName")] Department departments)
{
    var departmentRepo = _unitOfWork.GetRepository<Department>();

    var company = await departmentRepo.GetById(departments.DeptId, departments.ComId);
    if (company is null)
    {
        return NotFound();
    }

    await departmentRepo.Delete(departments.DeptId, departments.ComId);
    await _unitOfWork.SaveAsync();
```

```
            return RedirectToAction(nameof(Index));
        }
    }
```

To Create A Department just Department name Is required. It will be created under the company that I had selected.



Designation Controller

```
public class DesignationsController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly HRdbContext _context;

        public DesignationsController(IUnitOfWork unitOfWork, HRdbContext context)
        {
            _unitOfWork = unitOfWork;
            _context = context;
        }

        public async Task<IActionResult> Index()
        {
            var designationRepo = _unitOfWork.GetRepository<Designation>();
            var data = await designationRepo.GetAllAsync();

            var comid = Guid.Parse(Request.Cookies["comId"]);
            data = data.Where(x => x.ComId == comid);

            foreach (var flag in data)
            {
                Expression<Func<Company, bool>> predicate = x => x.ComId == flag.ComId;
                flag.ComName = (await _unitOfWork.GetRepository<Company>().Where(predicate)).Select(y =>
y.ComName).FirstOrDefault();
            }
```

```
    return View(data);
}


public async Task<IActionResult> Create()
{
    ViewBag.ComId = new SelectList(await _unitOfWork.GetRepository<Company>().GetAllAsync(), "ComId", "ComName");
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("DesigId,ComId,DesigName")] Designation designation)
{
    designation.ComId = Guid.Parse(Request.Cookies["comId"]);
    designation.DesigId = Guid.NewGuid();

    var designationRepo = _unitOfWork.GetRepository<Designation>();
    await designationRepo.Create(designation);
    await _unitOfWork.SaveAsync();
    return RedirectToAction(nameof(Index));
}

public async Task<IActionResult> Edit(Guid DesigId, Guid ComId)
{
    if (ComId == null || _context.Designation == null)
    {
        return NotFound();
    }

    var designationRepo = _unitOfWork.GetRepository<Designation>();
    var designation = await designationRepo.GetById(DesigId, ComId);

    if (designation == null)
    {
        return NotFound();
    }

    ViewData["ComId"] = new SelectList(_context.Company, "ComId", "ComId", designation.ComId);
    return View(designation);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(Guid DesigId, [Bind("DesigId,ComId,DesigName")] Designation designation)
{
    var designationRepo = _unitOfWork.GetRepository<Designation>();

    try
    {
        await designationRepo.Update(designation, designation.DesigId, designation.ComId);
        await _unitOfWork.SaveAsync();
    }
    catch (DbUpdateConcurrencyException e)
    {
        return BadRequest(e.Message);
```

```
        }

        return RedirectToAction(nameof(Index));
    }

    public async Task<IActionResult> Delete(Guid DesigId, Guid ComId)
    {
        if (DesigId == null || _context.Designation == null)
        {
            return NotFound();
        }

        var designationRepo = _unitOfWork.GetRepository<Designation>();
        var designation = await designationRepo.GetById(DesigId, ComId);

        if (designation == null)
        {
            return NotFound();
        }

        return View(designation);
    }

    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(Guid DesigId, [Bind("DesigId,ComId,DesigName")] Designation
designations)
    {
        var designationRepo = _unitOfWork.GetRepository<Designation>();
        var designation = await designationRepo.GetById(designations.DesigId, designations.ComId);

        if (designation is null)
        {
            return NotFound();
        }

        await designationRepo.Delete(designations.DesigId, designations.ComId);
        await _unitOfWork.SaveAsync();

        return RedirectToAction(nameof(Index));
    }

    private bool DesignationExists(Guid id)
    {
        return (_context.Designation?.Any(e => e.DesigId == id)).GetValueOrDefault();
    }
}
```

Here the mechanism is like Department where the designation will be created under the selected company.

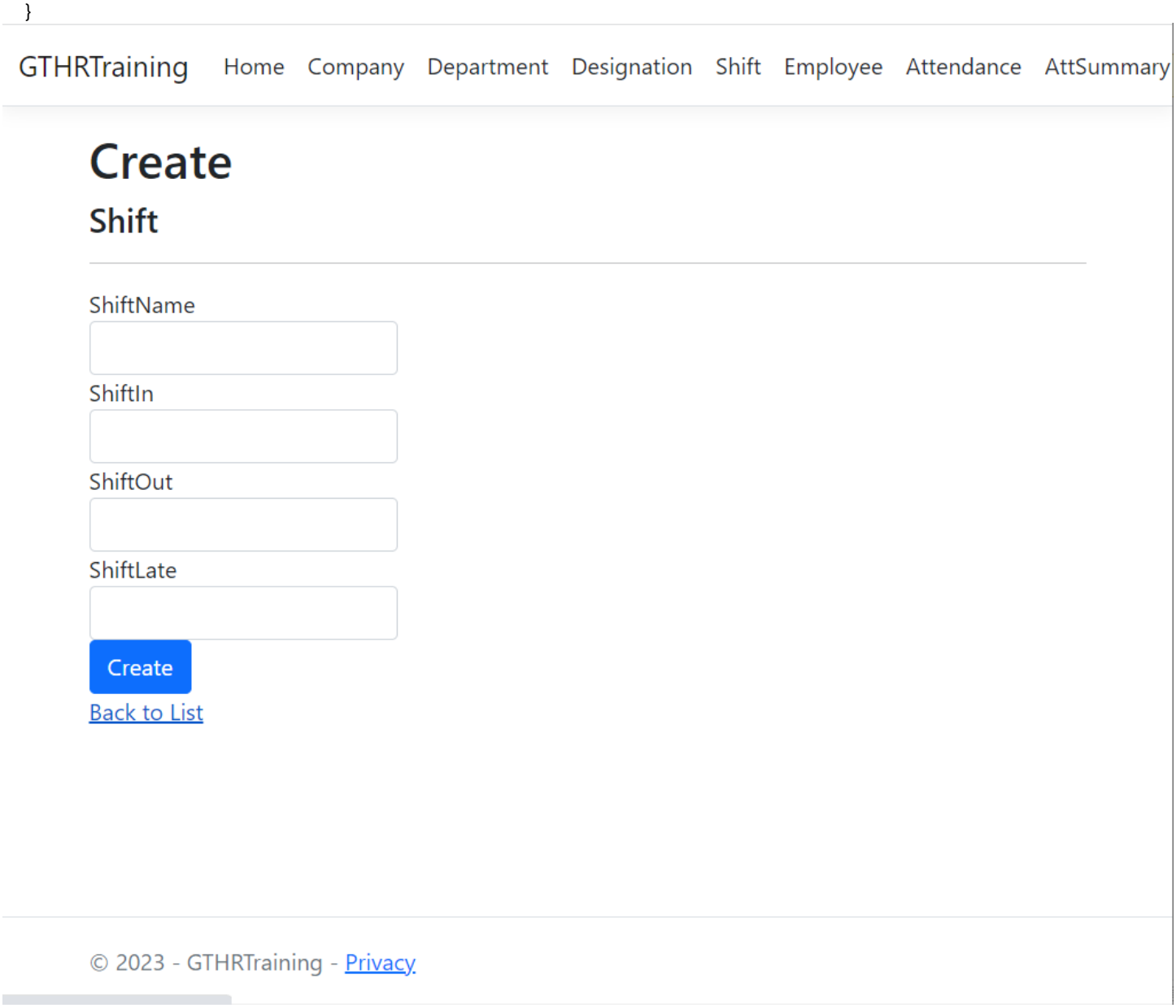
Shift Controller

```
public class ShiftsController : Controller
{
```

```
private readonly IUnitOfWork _unitOfWork;
private readonly HRdbContext _context;

public ShiftsController(IUnitOfWork unitOfWork, HRdbContext context)
{
    _unitOfWork = unitOfWork;
    _context = context;
}

// GET: Shifts
public async Task<IActionResult> Index()
{
    var shiftRepo = _unitOfWork.GetRepository<Shift>();
    var data = await shiftRepo.GetAllAsync();
    var comid = Guid.Parse(Request.Cookies["comId"]);
    data = data.Where(x => x.ComId == comid);
    foreach (var flag in data)
    {
        Expression<Func<Company, bool>> predicate = x => x.ComId == flag.ComId;
        var companyRepo = _unitOfWork.GetRepository<Company>();
        flag.ComName = (await companyRepo.Where(predicate)).Select(y => y.ComName).FirstOrDefault();
    }
    return View(data);
}


public IActionResult Create()
{
    ViewData["ComId"] = new SelectList(_context.Company, "ComId", "ComName");
    return View();
}

// POST: Shifts/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("ShiftId,ComId,ShiftName,ShiftIn,ShiftOut,ShiftLate")] Shift shift)
{
    shift.ComId = Guid.Parse(Request.Cookies["comId"]);
    shift.ShiftId = Guid.NewGuid();

    var shiftRepo = _unitOfWork.GetRepository<Shift>();
    await shiftRepo.Create(shift);
    await _unitOfWork.SaveAsync();
    return RedirectToAction(nameof(Index));
}

// GET: Shifts/Edit/5
public async Task<IActionResult> Edit(Guid ShiftId, Guid ComId)
{
    if (ComId == null || _context.Shifts == null)
    {
        return NotFound();
    }
```

```
    var shiftRepo = _unitOfWork.GetRepository<Shift>();
    var shift = await shiftRepo.GetById(ShiftId, ComId);
    if (shift == null)
    {
        return NotFound();
    }
    ViewData["ComId"] = new SelectList(_context.Company, "ComId", "ComName", shift.ComId);
    return View(shift);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(Guid ShiftId, [Bind("ShiftId,ComId,ShiftName,ShiftIn,ShiftOut,ShiftLate")] Shift shift)
{
    var shiftRepo = _unitOfWork.GetRepository<Shift>();
    try
    {
        await shiftRepo.Update(shift, shift.ShiftId, shift.ComId);
        await _unitOfWork.SaveAsync();
    }
    catch (DbUpdateConcurrencyException e)
    {
        return BadRequest(e.Message);
    }
    return RedirectToAction(nameof(Index));
}

// GET: Shifts/Delete/5
public async Task<IActionResult> Delete(Guid ShiftId, Guid ComId)
{
    if (ComId == null || _context.Shifts == null)
    {
        return NotFound();
    }

    var shiftRepo = _unitOfWork.GetRepository<Shift>();
    var shift = await shiftRepo.GetById(ShiftId, ComId);
    if (shift == null)
    {
        return NotFound();
    }

    return View(shift);
}

// POST: Shifts/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(Shift shift)
{
    var shiftRepo = _unitOfWork.GetRepository<Shift>();
    var shiftEntity = await shiftRepo.GetById(shift.ShiftId, shift.ComId);
    if (shiftEntity is null)
    {
        return NotFound();
    }
```

```
            await shiftRepo.Delete(shift.ShiftId, shift.ComId);
            await _unitOfWork.SaveAsync();
            return RedirectToAction(nameof(Index));
        }


    }
```



Here only ShiftName, ShiftIn, ShiftOut, ShiftLate is required and obviously created under the selected company.

Employee Controller

```
public class EmployeesController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IConfiguration _configuration;
```

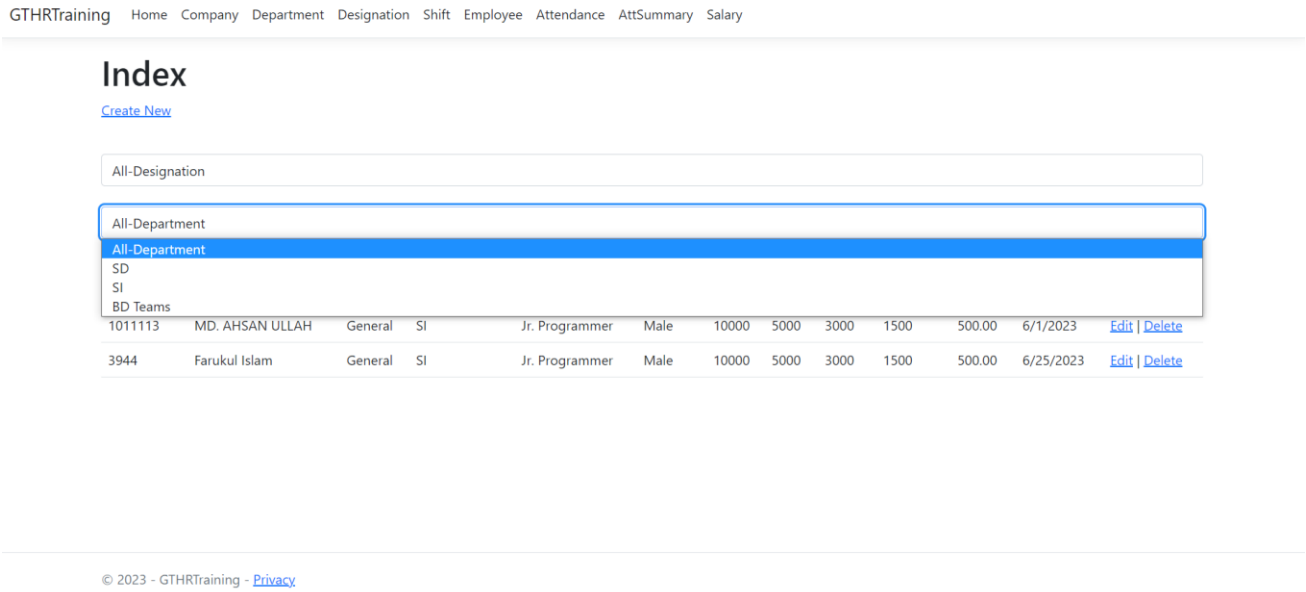
```
    public EmployeesController(IUnitOfWork unitOfWork, IConfiguration configuration)
    {
        _unitOfWork = unitOfWork;
        _configuration = configuration;
    }

    public async Task<IActionResult> Index()
    {
        var employeeRepo = _unitOfWork.GetRepository<Employee>();
        var shiftRepo = _unitOfWork.GetRepository<Shift>();
        var desigRepo = _unitOfWork.GetRepository<Designation>();
        var deptRepo = _unitOfWork.GetRepository<Department>();

        var data = await employeeRepo.GetAllAsync();
        var comid = Guid.Parse(Request.Cookies["comId"]);
        data = data.Where(x => x.ComId == comid);

        var departments = await deptRepo.Where(x => x.ComId == comid);
        ViewData["DeptId"] = new SelectList(departments, "DeptId", "DeptName");

        var designations = await desigRepo.Where(x => x.ComId == comid);
        ViewData["DesigId"] = new SelectList(designations, "DesigId", "DesigName");

        foreach (var flag in data)
        {
            flag.ShiftName = (await shiftRepo.Where(x => x.ComId == flag.ComId && x.ShiftId == flag.ShiftId)).Select(y =>
y.ShiftName).FirstOrDefault();
        }
        foreach (var flag in data)
        {
            flag.DesigName = (await desigRepo.Where(x => x.ComId == flag.ComId && x.DesigId == flag.DesigId)).Select(y =>
y.DesigName).FirstOrDefault();
        }
        foreach (var flag in data)
        {
            flag.DeptName = (await deptRepo.Where(x => x.ComId == flag.ComId && x.DeptId == flag.DeptId)).Select(y =>
y.DeptName).FirstOrDefault();
        }
        return View(data);
    }

    public IActionResult FilterEmployees(Guid desigId, Guid deptId)
    {
        string connectionString = _configuration.GetConnectionString("DefaultConnection");

        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();

            var command = new SqlCommand("Employee_List", connection);
            command.CommandType = CommandType.StoredProcedure;

            // Add parameters
            command.Parameters.AddWithValue("@ComId", Guid.Parse(Request.Cookies["comId"]));
            command.Parameters.AddWithValue("@Desig", (desigId == Guid.Empty) ? (object)DBNull.Value : desigId);
            command.Parameters.AddWithValue("@Dept", (deptId == Guid.Empty) ? (object)DBNull.Value : deptId);
```

```
        var employees = new List<Employee>();

        using (var reader = command.ExecuteReader())
        {
            while (reader.Read())
            {
                var employee = new Employee
                {
                    EmpId = reader.GetGuid(reader.GetOrdinal("EmpId")),
                    EmpCode = reader.GetString(reader.GetOrdinal("EmpCode")),
                    EmpName = reader.GetString(reader.GetOrdinal("EmpName")),
                    ShiftId = reader.GetGuid(reader.GetOrdinal("ShiftId")),

                    // ... map other properties accordingly
                    //Shift = new Shift { ShiftName = reader.GetString(reader.GetOrdinal("ShiftName")) },
                    //Department = new Department { DeptName = reader.GetString(reader.GetOrdinal("DeptName")) },
                    //Designation = new Designation { DesigName = reader.GetString(reader.GetOrdinal("DesigName")) }
                };

                employees.Add(employee);
            }
        }

        ViewBag.FilteredEmployees = employees;
    }

    return View("Index", ViewBag.FilteredEmployees);
}

public async Task<IActionResult> Create()
{
    var comid = Guid.Parse(Request.Cookies["comId"]);

    var shiftRepo = _unitOfWork.GetRepository<Shift>();
    var deptRepo = _unitOfWork.GetRepository<Department>();
    var desigRepo = _unitOfWork.GetRepository<Designation>();

    ViewData["ShiftId"] = new SelectList(await shiftRepo.Where(x => x.ComId == comid), "ShiftId", "ShiftName");
    ViewData["DeptId"] = new SelectList(await deptRepo.Where(x => x.ComId == comid), "DeptId", "DeptName");
    ViewData["DesigId"] = new SelectList(await desigRepo.Where(x => x.ComId == comid), "DesigId", "DesigName");

    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(Employee employee)
{

        employee.ComId = Guid.Parse(Request.Cookies["comId"]);
        var comRepo = _unitOfWork.GetRepository<Company>();
    Expression<Func<Company, bool>> predicate = x => x.ComId == employee.ComId;
    var data = (await comRepo.Where(predicate)).FirstOrDefault();
    var employeeRepo = _unitOfWork.GetRepository<Employee>();
    employee.Basic = data.Basic * employee.Gross;
```

```
            employee.Hrent = data.Hrent * employee.Gross;
            employee.Medical = data.Medical * employee.Gross;
            employee.Others = (1 - data.Basic - data.Hrent - data.Medical) * employee.Gross;
            employee.EmpId = Guid.NewGuid();
            employeeRepo.Create(employee);
                //await _unitOfWork.SaveAsync();
                return RedirectToAction(nameof(Index));



        }

        public async Task<IActionResult> Edit(Guid id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var companyRepo = _unitOfWork.GetRepository<Company>();
            var employeeRepo = _unitOfWork.GetRepository<Employee>();
            var shiftRepo = _unitOfWork.GetRepository<Shift>();
            var deptRepo = _unitOfWork.GetRepository<Department>();
            var desigRepo = _unitOfWork.GetRepository<Designation>();
            var comid = Guid.Parse(Request.Cookies["comId"]);

            var employee = await employeeRepo.GetById(id, comid);
            if (employee == null)
            {
                return NotFound();
            }

            ViewData["ComId"] = new SelectList(await companyRepo.Where(x => x.ComId == comid), "ComId", "ComName",
employee.ComId);
            ViewData["ShiftId"] = new SelectList(await shiftRepo.Where(x => x.ComId == comid), "ShiftId", "ShiftName",
employee.ShiftId);
            ViewData["DeptId"] = new SelectList(await deptRepo.Where(x => x.ComId == comid), "DeptId", "DeptName",
employee.DeptId);
            ViewData["DesigId"] = new SelectList(await desigRepo.Where(x => x.ComId == comid), "DesigId", "DesigName",
employee.DesigId);

            return View(employee);
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(Guid id, Employee employee)
        {
            if (id != employee.EmpId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
```

```
            var employeeRepo = _unitOfWork.GetRepository<Employee>();
            employeeRepo.Update(employee, employee.EmpId, employee.ComId);
            await _unitOfWork.SaveAsync();
        }
        catch (DbUpdateConcurrencyException)
        {

            throw;
        }
        return RedirectToAction(nameof(Index));
    }

    var shiftRepo = _unitOfWork.GetRepository<Shift>();
    var deptRepo = _unitOfWork.GetRepository<Department>();
    var desigRepo = _unitOfWork.GetRepository<Designation>();

    ViewData["ShiftId"] = new SelectList(await shiftRepo.GetAllAsync(), "ShiftId", "ShiftName", employee.ShiftId);
    ViewData["DeptId"] = new SelectList(await deptRepo.GetAllAsync(), "DeptId", "DeptName", employee.DeptId);
    ViewData["DesigId"] = new SelectList(await desigRepo.GetAllAsync(), "DesigId", "DesigName", employee.DesigId);

    return View(employee);
}

public async Task<IActionResult> Delete(Guid EmpId, Guid Comid)
{
    if (EmpId == null)
    {
        return NotFound();
    }

    var comid = Guid.Parse(Request.Cookies["comId"]);
    var employeeRepo = _unitOfWork.GetRepository<Employee>();
    var employee = await employeeRepo.GetById(EmpId, comid);
    if (employee == null)
    {
        return NotFound();
    }

    return View(employee);
}

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(Guid EmpId, Guid Comid)
{
    var employeeRepo = _unitOfWork.GetRepository<Employee>();

    var comid = Guid.Parse(Request.Cookies["comId"]);
    var employee = await employeeRepo.GetById(EmpId, comid);
    employeeRepo.Delete(employee.EmpId,employee.ComId);
    //await _unitOfWork.SaveAsync();
    return RedirectToAction(nameof(Index));
}

}
```

Here We can the Employee list using Department and Designation which will call a Controller IActionResult method And it's use a store procedure called Employee_list to filter the Employee list.



Attendance Controller

```
public class AttendancesController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IConfiguration _configuration;
        private readonly HRdbContext _context;


        public AttendancesController(IUnitOfWork unitOfWork, IConfiguration configuration, HRdbContext context)
        {
            _unitOfWork = unitOfWork;
            _configuration = configuration;
            _context = context;
        }

        // GET: Attendances
        public async Task<IActionResult> Index()
        {
            var comid = Guid.Parse(Request.Cookies["comId"]);
            var attendanceRepo = _unitOfWork.GetRepository<Attendance>();
            var companyRepo = _unitOfWork.GetRepository<Company>();
```

```
    var employeeRepo = _unitOfWork.GetRepository<Employee>();
    var data = await attendanceRepo.GetAllAsync();
    data = data.Where(x => x.ComId == comid);

    foreach (var flag in data)
    {
        Expression<Func<Company, bool>> predicate = x => x.ComId == flag.ComId;
        flag.ComName = (await companyRepo.Where(predicate)).Select(y => y.ComName).FirstOrDefault();

        Expression<Func<Employee, bool>> predicates = x => x.ComId == comid && x.EmpId == flag.EmpId;
        flag.EmpName = (await employeeRepo.Where(predicates)).Select(y => y.EmpName).FirstOrDefault();
    }

    return View(data);
}

public async Task<IActionResult> FilterData(string criteria, DateTime date)
{
    string connectionString = _configuration.GetConnectionString("DefaultConnection");
    var attendances = new List<Attendance>();
    var comid = Guid.Parse(Request.Cookies["comId"]);
    var companyRepo = _unitOfWork.GetRepository<Company>();
    var employeeRepo = _unitOfWork.GetRepository<Employee>();

    using (var connection = new SqlConnection(connectionString))
    {
        connection.Open();

        var command = new SqlCommand("AttendanceList", connection);
        command.CommandType = CommandType.StoredProcedure;

        // Add parameters
        command.Parameters.AddWithValue("@ComId", Guid.Parse(Request.Cookies["comId"]));
```

```
            command.Parameters.AddWithValue("@status", criteria);
            command.Parameters.AddWithValue("@date", date);

            using (var reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    var attendance = new Attendance
                    {
                        ComId = reader.GetGuid(reader.GetOrdinal("ComId")),
                        EmpId = reader.GetGuid(reader.GetOrdinal("EmpId")),
                        dtDate = reader.GetDateTime(reader.GetOrdinal("dtDate")),
                        AttStatus = reader.GetString(reader.GetOrdinal("AttStatus")),
                        InTime = reader.IsDBNull(reader.GetOrdinal("InTime")) ? TimeSpan.MinValue :
reader.GetTimeSpan(reader.GetOrdinal("InTime")),
                        OutTime = reader.IsDBNull(reader.GetOrdinal("OutTime")) ? TimeSpan.MinValue :
reader.GetTimeSpan(reader.GetOrdinal("OutTime")),
                        // ... map other properties accordingly
                    };

                    attendances.Add(attendance);
                }
            }
        }
        foreach (var flag in attendances)
        {
            Expression<Func<Company, bool>> predicate = x => x.ComId == flag.ComId;
            flag.ComName = (await companyRepo.Where(predicate)).Select(y => y.ComName).FirstOrDefault();

            Expression<Func<Employee, bool>> predicates = x => x.ComId == comid && x.EmpId == flag.EmpId;
            flag.EmpName = (await employeeRepo.Where(predicates)).Select(y => y.EmpName).FirstOrDefault();

        }
```

```
    return Json(attendances);
}



public async Task<IActionResult> Create()
{

    var companyRepo = _unitOfWork.GetRepository<Company>();
    var employeeRepo = _unitOfWork.GetRepository<Employee>();
    ViewData["ComId"] = new SelectList(await companyRepo.Where(x => true), "ComId", "ComName");
    var comid = Guid.Parse(Request.Cookies["comId"]);
    ViewData["EmpId"] = new SelectList(await employeeRepo.Where(x => x.ComId == comid), "EmpId", "EmpName");
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("ComId,EmpId,dtDate,AttStatus,InTime,OutTime")] Attendance attendance)
{
    var shiftRepo = _unitOfWork.GetRepository<Shift>();
    var employeeRepo = _unitOfWork.GetRepository<Employee>();
    var attendanceRepo = _unitOfWork.GetRepository<Attendance>();
    var comid = Guid.Parse(Request.Cookies["comId"]);
    attendance.ComId = comid;
    Expression<Func<Employee, bool>> predicate = x => x.ComId == comid;
    var empShift = (await employeeRepo.Where(predicate)).Select(y => y.ShiftId).FirstOrDefault();

    Expression<Func<Shift, bool>> predicates = x => x.ShiftId == empShift;
    var timeIn = (await shiftRepo.Where(predicates)).Select(y => y.ShiftIn).FirstOrDefault();

    if(attendance.InTime == TimeSpan.Zero)
    {
```

```
    attendance.AttStatus = "A";
}
else
{
    if (timeIn >= attendance.InTime)
    {
        attendance.AttStatus = "P";
    }
    else
    {
        attendance.AttStatus = "L";
    }

}



await attendanceRepo.Create(attendance);
DateTime date = attendance.dtDate; // Your specific date

DateTime startDate = new DateTime(date.Year, date.Month, 1);
DateTime endDate = startDate.AddMonths(1).AddDays(-1);
string connectionString = _configuration.GetConnectionString("DefaultConnection");

using (SqlConnection connection = new SqlConnection(connectionString))
{
    using (SqlCommand command = new SqlCommand("AttendanceSum", connection))
    {
        command.CommandType = CommandType.StoredProcedure;

        command.Parameters.AddWithValue("@ComId", comid);
        command.Parameters.AddWithValue("@dtTO", startDate);
        command.Parameters.AddWithValue("@dtFrom", endDate);
```

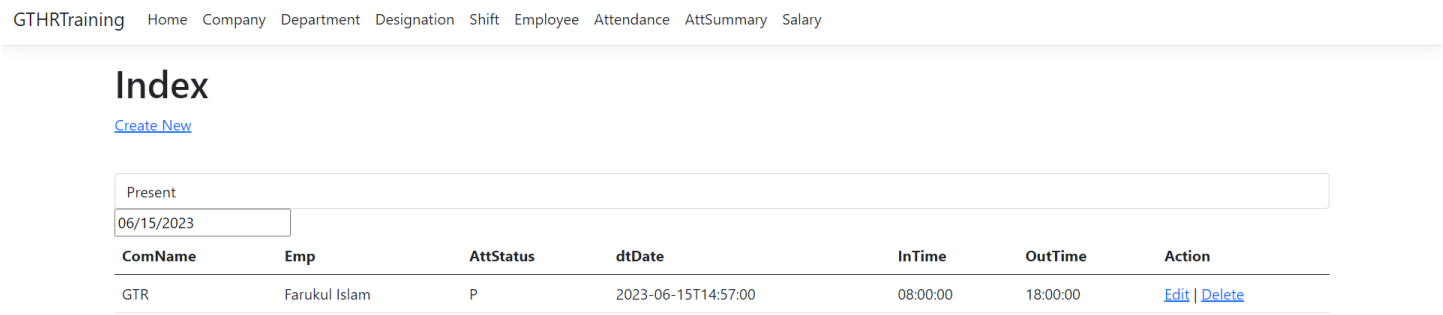
```
                connection.Open();
                command.ExecuteNonQuery();
            }
        }

        return RedirectToAction(nameof(Index));

    }

    // GET: Attendances/Edit/5
    public async Task<IActionResult> Edit(Guid ComId, Guid EmpId, DateTime dtDate)
    {
        if (EmpId == null )
        {
            return NotFound();
        }

        var attendanceRepo = _unitOfWork.GetRepository<Attendance>();

        var attendance = await attendanceRepo.GetById(ComId, EmpId, dtDate);
        if (attendance == null)
        {
            return NotFound();
        }
        return View(attendance);
    }

    // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(Guid id, [Bind("ComId,EmpId,dtDate,AttStatus,InTime,OutTime")] Attendance
attendance)
    {
```

```
    var attendanceRepo = _unitOfWork.GetRepository<Attendance>();
    try
    {
        await attendanceRepo.Update(attendance, attendance.ComId, attendance.EmpId, attendance.dtDate) ;
    }
    catch (DbUpdateConcurrencyException e)
    {
        return BadRequest(e.Message);
    }
    return RedirectToAction(nameof(Index));
}
public JsonResult GetEmployees(string companyId)
{

    List<Employee> filteredEmployees = GetFilteredEmployees(companyId);

    var filteredEmployeeData = filteredEmployees.Select(e => new { EmpId = e.EmpId, EmpName = e.EmpName });

    return Json(filteredEmployeeData);
}

private List<Employee> GetFilteredEmployees(string companyId)
{
    List<Employee> employees = new List<Employee>();

    Guid guid;

    guid = Guid.Parse(companyId);

    List<Employee> filteredEmployees = _context.Employee.Where(e => e.ComId == guid).ToList();

    return filteredEmployees;
```

```
}

public async Task<IActionResult> Delete(Guid? ComId, Guid? EmpId, DateTime dtDate)
{
    if (EmpId == null || _context.Attendance == null)
    {
        return NotFound();
    }

    var attendance = await _context.Attendance
        .Include(a => a.Com)
        .Include(a => a.Emp)
        .FirstOrDefaultAsync(m => m.ComId == ComId);
    if (attendance == null)
    {
        return NotFound();
    }

    return View(attendance);
}

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(Attendance attendances)
{
    var attendanceRepo = _unitOfWork.GetRepository<Attendance>();
    var company = await attendanceRepo.GetById(attendances.ComId, attendances.EmpId, attendances.dtDate);
    if (company is null)
    {
        return NotFound();
    }
    await attendanceRepo.Delete(attendances,attendances.ComId, attendances.EmpId, attendances.dtDate);
    return RedirectToAction(nameof(Index));
```

```
    }


    private bool AttendanceExists(Guid id)
    {
      return (_context.Attendance?.Any(e => e.ComId == id)).GetValueOrDefault();
    }
  }
```

Here we can also filter the Attendance list data based on Attendance status like All, Present, Late, Absent. It is also done by store procedure.



Attendance summary controller

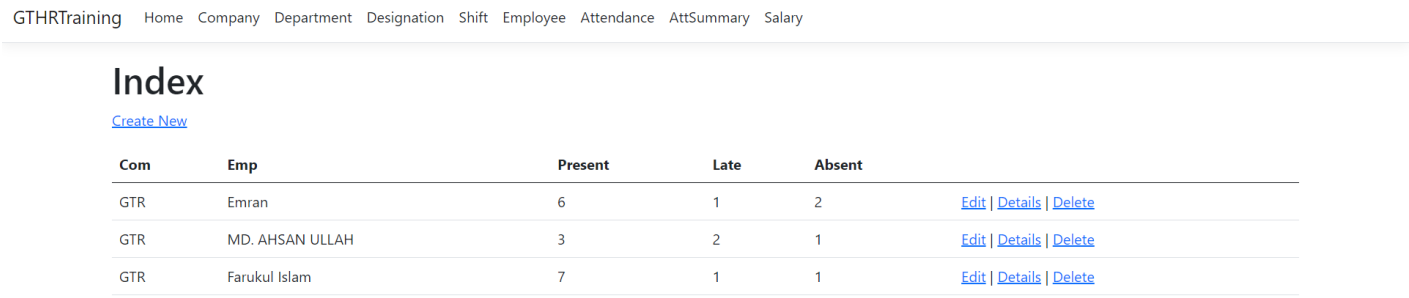
```
public class AttSummariesController : Controller
  {
    private readonly HRdbContext _context;
    private readonly IUnitOfWork _unitOfWork;

    public AttSummariesController(HRdbContext context, IUnitOfWork unitofwork)
    {
      _context = context;
      _unitOfWork = unitofwork;
    }

    // GET: AttSummaries
    public async Task<IActionResult> Index()
    {
      var attendanceRepo = _unitOfWork.GetRepository<AttSummary>();
      var comid = Guid.Parse(Request.Cookies["comId"]);
      var hRdbContext = await attendanceRepo.Where(x => x.ComId == comid,
                              a => a.Com,
                              a => a.Emp);
      return View(hRdbContext);
    }


    public async Task<IActionResult> Create()
```

```
    {
        var companyRepo = _unitOfWork.GetRepository<Company>();
        var employeeRepo = _unitOfWork.GetRepository<Employee>();
        ViewData["ComId"] = new SelectList(await companyRepo.Where(x => true), "ComId", "ComName");
        var comid = Guid.Parse(Request.Cookies["comId"]);
        ViewData["EmpId"] = new SelectList(await employeeRepo.Where(x => x.ComId == comid), "EmpId", "EmpName");
        return View();
    }
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create([Bind("ComId,EmpId,dtYear,dtMonth,Present,Late,Absent,WDay")] AttSummary
attSummary)
    {
        var companyRepo = _unitOfWork.GetRepository<Company>();
        var employeeRepo = _unitOfWork.GetRepository<Employee>();

            attSummary.ComId = Guid.NewGuid();
            _context.Add(attSummary);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
    }

    // GET: AttSummaries/Edit/5
    public async Task<IActionResult> Edit(Guid? id)
    {
        if (id == null || _context.AttSummary == null)
        {
            return NotFound();
        }

        var attSummary = await _context.AttSummary.FindAsync(id);
        if (attSummary == null)
        {
            return NotFound();
        }
        var companyRepo = _unitOfWork.GetRepository<Company>();
        var employeeRepo = _unitOfWork.GetRepository<Employee>();
        ViewData["ComId"] = new SelectList(await companyRepo.Where(x => true), "ComId", "ComName");
        var comid = Guid.Parse(Request.Cookies["comId"]);
        ViewData["EmpId"] = new SelectList(await employeeRepo.Where(x => x.ComId == comid), "EmpId", "EmpName");
        return View(attSummary);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(Guid id, [Bind("ComId,EmpId,dtYear,dtMonth,Present,Late,Absent,WDay")]
AttSummary attSummary)
    {
        if (id != attSummary.ComId)
        {
            return NotFound();
        }

        if (ModelState.IsValid)
        {
            try
```

```
            {
                _context.Update(attSummary);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!AttSummaryExists(attSummary.ComId))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        var companyRepo = _unitOfWork.GetRepository<Company>();
        var employeeRepo = _unitOfWork.GetRepository<Employee>();
        ViewData["ComId"] = new SelectList(await companyRepo.Where(x => true), "ComId", "ComName");
        var comid = Guid.Parse(Request.Cookies["comId"]);
        ViewData["EmpId"] = new SelectList(await employeeRepo.Where(x => x.ComId == comid), "EmpId", "EmpName");
        return View(attSummary);
    }

    // GET: AttSummaries/Delete/5
    public async Task<IActionResult> Delete(Guid? id)
    {
        if (id == null || _context.AttSummary == null)
        {
            return NotFound();
        }

        var attSummary = await _context.AttSummary
            .Include(a => a.Com)
            .Include(a => a.Emp)
            .FirstOrDefaultAsync(m => m.ComId == id);
        if (attSummary == null)
        {
            return NotFound();
        }

        return View(attSummary);
    }

    // POST: AttSummaries/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(Guid id)
    {
        if (_context.AttSummary == null)
        {
            return Problem("Entity set 'HRdbContext.AttSummary'  is null.");
        }
        var attSummary = await _context.AttSummary.FindAsync(id);
        if (attSummary != null)
```

```
        {
            _context.AttSummary.Remove(attSummary);
        }

        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    private bool AttSummaryExists(Guid id)
    {
      return (_context.AttSummary?.Any(e => e.ComId == id)).GetValueOrDefault();
    }
}
```

It will be auto created and updated based on Insert and update on Attendance Table which is maintained by trigger.



Salary Controller

```
public class SalariesController : Controller
  {
    private readonly IUnitOfWork _unitOfWork;
    private readonly HRdbContext _context;
    private readonly IConfiguration _configuration;

    public SalariesController( HRdbContext context, IConfiguration configuration ,IUnitOfWork unitOfWork)
    {
        _context = context;
        _configuration = configuration;
        _unitOfWork = unitOfWork;
    }

    // GET: Salaries
    public async Task<IActionResult> Index()
    {
        var salaryRepo = _unitOfWork.GetRepository<Salary>();
        var companyRepo = _unitOfWork.GetRepository<Company>();
        var _empRepo = _unitOfWork.GetRepository<Employee>();
        var _deptRepo = _unitOfWork.GetRepository<Department>();


        var data = await salaryRepo.GetAllAsync();
        var comid = Guid.Parse(Request.Cookies["comId"]);
        data = data.Where(x => x.ComId == comid);
        foreach (var flag in data)
```

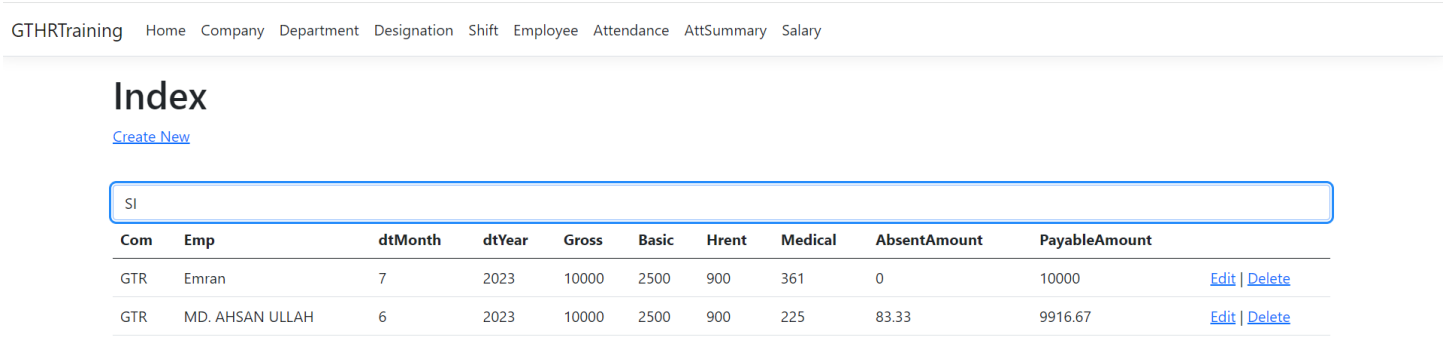
```
    {
        Expression<Func<Company, bool>> predicate = x => x.ComId == flag.ComId;
        flag.ComName = (await companyRepo.Where(predicate)).Select(y => y.ComName).FirstOrDefault();
    }
    foreach (var flag in data)
    {
        Expression<Func<Employee, bool>> predicate = x => x.ComId == comid && x.EmpId == flag.EmpId;
        flag.EmpName = (await _empRepo.Where(predicate)).Select(y => y.EmpName).FirstOrDefault();
    }
    ViewData["DeptId"] = new SelectList(await _deptRepo.Where(x => x.ComId == comid), "DeptId", "DeptName");
    return View(data);
}

public async Task<IActionResult> FilterData(Guid Dept)
{
    string connectionString = _configuration.GetConnectionString("DefaultConnection");
    var salaries = new List<Salary>();
    var comid = Guid.Parse(Request.Cookies["comId"]);

    using (var connection = new SqlConnection(connectionString))
    {
        connection.Open();

        var command = new SqlCommand("SalaryList", connection);
        command.CommandType = CommandType.StoredProcedure;

        // Add parameters
        command.Parameters.AddWithValue("@ComId", Guid.Parse(Request.Cookies["comId"]));
        command.Parameters.AddWithValue("@DeptId", Dept);
        command.Parameters.AddWithValue("@Criteria", 0);

        using (var reader = command.ExecuteReader())
        {
            while (reader.Read())
            {
                var salary = new Salary
                {
                    ComId = reader.GetGuid(reader.GetOrdinal("ComId")),
                    EmpId = reader.GetGuid(reader.GetOrdinal("EmpId")),
                    dtYear = reader.GetString(reader.GetOrdinal("dtYear")),
                    dtMonth = reader.GetString(reader.GetOrdinal("dtMonth")),
                    Gross = reader.GetDouble(reader.GetOrdinal("Gross")),
                    Basic = reader.GetDouble(reader.GetOrdinal("Basic")),
                    Hrent = reader.GetDouble(reader.GetOrdinal("Hrent")),
                    Medical = reader.GetDouble(reader.GetOrdinal("Medical")),
                    AbsentAmount = Math.Round(reader.GetDouble(reader.GetOrdinal("AbsentAmount")), 2),
                    PayableAmount = Math.Round(reader.GetDouble(reader.GetOrdinal("PayableAmount")), 2)
                    // ... map other properties accordingly
                };

                salaries.Add(salary);
            }
        }
    }

    var companyRepo = _unitOfWork.GetRepository<Company>();
```

```
            var _empRepo = _unitOfWork.GetRepository<Employee>();
            foreach (var flag in salaries)
            {
                Expression<Func<Company, bool>> predicate = x => x.ComId == flag.ComId;
                flag.ComName = (await companyRepo.Where(predicate)).Select(y => y.ComName).FirstOrDefault();

                Expression<Func<Employee, bool>> predicates = x => x.ComId == comid && x.EmpId == flag.EmpId;
                flag.EmpName = (await _empRepo.Where(predicates)).Select(y => y.EmpName).FirstOrDefault();
            }

            return Json(salaries);
        }

        public async Task<IActionResult> Create()
        {

            var companyRepo = _unitOfWork.GetRepository<Company>();
            var _empRepo = _unitOfWork.GetRepository<Employee>();
            ViewData["ComId"] = new SelectList(await companyRepo.Where(x => true), "ComId", "ComName");
            var comid = Guid.Parse(Request.Cookies["comId"]);
            ViewData["EmpId"] = new SelectList(await _empRepo.Where(x => x.ComId == comid), "EmpId", "EmpName");
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("ComId,EmpId,dtYear,dtMonth,Gross,Basic,Hrent,Medical,AbsentAmount,PayableAmount")] Salary salary)
        {

            var companyRepo = _unitOfWork.GetRepository<Company>();
            salary.ComId = Guid.Parse(Request.Cookies["comId"]);
            var Basic = await companyRepo.GetById(salary.ComId);
            var empBasic = _context.Employee.Where(x=>x.ComId ==  salary.ComId && x.EmpId ==  salary.EmpId).FirstOrDefault();
            salary.Basic = Basic.Basic * empBasic.Basic;

            salary.Hrent = Basic.Hrent * empBasic.Hrent;
            salary.Medical = Basic.Medical * empBasic.Medical;
            salary.Gross = empBasic.Gross;

            var absentamnt = _context.AttSummary.Where(x=> x.ComId == salary.ComId && x.EmpId == salary.EmpId &&
x.dtMonth == salary.dtMonth&& x.dtYear == salary.dtYear).FirstOrDefault();

            if (absentamnt != null)
            {
                salary.AbsentAmount = salary.Basic / 30;
                salary.AbsentAmount = salary.AbsentAmount * absentamnt.Absent;
            }
            else
            {
                salary.AbsentAmount = 0;
            }
            salary.PayableAmount = salary.Gross - salary.AbsentAmount;
                _context.Add(salary);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
```

```
    }
    public JsonResult GetEmployees(string companyId)
    {

        List<Employee> filteredEmployees = GetFilteredEmployees(companyId);

        var filteredEmployeeData = filteredEmployees.Select(e => new { EmpId = e.EmpId, EmpName = e.EmpName });

        return Json(filteredEmployeeData);
    }

    private List<Employee> GetFilteredEmployees(string companyId)
    {
        List<Employee> employees = new List<Employee>();
        Guid guid;

        guid = Guid.Parse(companyId);

        List<Employee> filteredEmployees = _context.Employee.Where(e => e.ComId == guid).ToList();

        return filteredEmployees;
    }
    // GET: Salaries/Edit/5
    public async Task<IActionResult> Edit(Guid? id)
    {
        if (id == null || _context.Salary == null)
        {
            return NotFound();
        }

        var salary = await _context.Salary.FindAsync(id);
        if (salary == null)
        {
            return NotFound();
        }

        var companyRepo = _unitOfWork.GetRepository<Company>();
        var _empRepo = _unitOfWork.GetRepository<Employee>();
        var companies = await companyRepo.Where(x => x.ComId == salary.ComId);
        var employees = await _empRepo.Where(x => x.EmpId == salary.EmpId);

        ViewData["ComId"] = new SelectList(companies, "ComId", "ComName", salary.ComId);
        ViewData["EmpId"] = new SelectList(employees, "EmpId", "EmpName", salary.EmpId);
        return View(salary);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(Guid id,
[Bind("ComId,EmpId,dtYear,dtMonth,Gross,Basic,Hrent,Medical,AbsentAmount,PayableAmount")] Salary salary)
    {
        var _salaryRepo = _unitOfWork.GetRepository<Salary>();
        try
        {
            await _salaryRepo.Update(salary, salary.ComId, salary.EmpId, salary.dtMonth, salary.dtYear);
```

```
            }
            catch (DbUpdateConcurrencyException e)
            {
                return BadRequest(e.Message);
            }
            return RedirectToAction(nameof(Index));
        }

        // GET: Salaries/Delete/5
        public async Task<IActionResult> Delete(Guid? ComId, Guid? EmpId, string dtMonth, string dtYear)
        {

            var _salaryRepo = _unitOfWork.GetRepository<Salary>();
            var salaries = await _salaryRepo.Where(s => s.ComId == ComId, s => s.Com, s => s.Emp);
            var salary = salaries.FirstOrDefault();



            if (salary == null)
            {
                return NotFound();
            }

            return View(salary);
        }

        // POST: Salaries/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(Salary salary)
        {
            var _salaryRepo = _unitOfWork.GetRepository<Salary>();
            var company = await _salaryRepo.GetById( salary.ComId, salary.EmpId, salary.dtMonth, salary.dtYear);
            if (company is null)
            {
                return NotFound();
            }
            await _salaryRepo.Delete(salary, salary.ComId, salary.EmpId, salary.dtMonth, salary.dtYear);
            return RedirectToAction(nameof(Index));
        }

        private bool SalaryExists(Guid id)
        {
          return (_context.Salary?.Any(e => e.ComId == id)).GetValueOrDefault();
        }
    }
```

It is also show salary report based on Department.



Store Procedure

- It will shift data from attendance table to attendance summary which in initialized in Create method in Attendance Controller

```
USE [MVCTask]
GO
/****** Object:  StoredProcedure [dbo].[AttendanceSum]    Script Date: 6/26/2023 1:31:32 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- exec AttendanceSum 'AC462C54-9DE2-48AE-B022-D727E62AA1D9', '01-Jun-2023', '30-Jun-2023',NULL
ALTER PROCEDURE [dbo].[AttendanceSum] @ComId UNIQUEIDENTIFIER, @dtTO DateTime, @dtFrom DateTime, @EmpId
UNIQUEIDENTIFIER
AS
BEGIN
                    DELETE FROM AttSummary
                    WHERE ComId = @ComId AND dtMonth = MONTH(@dtTO) AND dtYear = YEAR(@dtTO)

                    INSERT INTO AttSummary (ComId, EmpId, dtMonth, dtYear,Present, Late, Absent)
                    SELECT DISTINCT s.ComId, s.EmpId, MONTH(@dtTO), YEAR(@dtTO),0 ,0 ,0
                    FROM Attendance AS s
                    INNER JOIN Employee AS e ON e.Empid = s.EmpId AND e.ComId = s.ComId
                    WHERE e.Empid = s.EmpId
                    AND e.ComId = s.ComId
                    AND s.dtDate BETWEEN @dtTO AND @dtFrom and s.ComId = @ComId


    UPDATE att
    SET att.Present = COALESCE(sub.Present, 0),
        att.Late =  COALESCE(sub.Late, 0),
        att.Absent = COALESCE(sub.Absent, 0)
    FROM AttSummary att
    LEFT JOIN (
        SELECT EmpId,
            COUNT(CASE WHEN AttStatus = 'P' THEN 1 END) AS Present,
            COUNT(CASE WHEN AttStatus = 'L' THEN 1 END) AS Late,
            COUNT(CASE WHEN AttStatus = 'A' THEN 1 END) AS Absent
        FROM Attendance
        WHERE AttStatus IN ('P', 'L', 'A') and dtDate between @dtTO and @dtFrom
        GROUP BY EmpId
    ) AS sub ON att.EmpId = sub.EmpId;
```

```
END
```

- **It filter's the Attendance list based on Attendance Status used in FilterData of AttendanceController**

```
USE [MVCTask]
GO
/****** Object:  StoredProcedure [dbo].[AttendanceList]    Script Date: 6/28/2023 3:15:58 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- exec AttendanceList 'AC462C54-9DE2-48AE-B022-D727E62AA1D9', 'All', '12-Jun-2023'
ALTER PROCEDURE [dbo].[AttendanceList] @ComID uniqueidentifier, @status nvarchar(50), @date Date
AS
BEGIN
	if @status = 'All'
	BEGIN
		select * from Attendance at
		left outer join Employee e on at.EmpId = e.EmpId
		Where CONVERT(DATE, at.dtDate) = @date and at.ComId = @ComID
	END
	ELSE
	if @status = 'L'
	BEGIN
		select * from Attendance at
		left outer join Employee e on at.EmpId = e.EmpId
		Where at.AttStatus = 'L' and CONVERT(DATE, at.dtDate) = @date and at.ComId = @ComID
	END
	ELSE
	if @status = 'A'
	BEGIN
		select * from Attendance at
		left outer join Employee e on at.EmpId = e.EmpId
		Where at.AttStatus = 'A'  and CONVERT(DATE, at.dtDate) = @date and at.ComId = @ComID
	END
	ELSE
	if @status = 'P'
	BEGIN
		select * from Attendance at
		left outer join Employee e on at.EmpId = e.EmpId
		Where at.AttStatus = 'P'  and CONVERT(DATE, at.dtDate) = @date and at.ComId = @ComID
	END


END
```

- **It shows the reports of Employee salary based on department and also returns the salary summary in Case of Criteria 1 used in FilterData In SalariesController.**

```
USE [MVCTask]
GO
/****** Object:  StoredProcedure [dbo].[SalaryList]    Script Date: 6/28/2023 3:18:14 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- exec SalaryList 'AC462C54-9DE2-48AE-B022-D727E62AA1D9', '0A150D0E-E14D-4C1F-8A16-C35D59D1D4B4', 1

ALTER PROCEDURE [dbo].[SalaryList] @ComId uniqueidentifier, @DeptId uniqueidentifier , @Criteria Int
```

```
AS
Begin
if @Criteria = 0

	BEGIN
		select * from Salary
		inner join Employee on Salary.EmpId = Employee.EmpId
		where  Salary.ComId = @ComId and Employee.DeptId = @DeptId
	END

Else
	Begin
		SELECT
				SUM(Salary.Gross) AS TotalGross,
				SUM(Salary.Basic) AS TotalBasic,
				SUM(Salary.PayableAmount) AS TotalPayable
			FROM
				Salary
				INNER JOIN Employee ON Salary.EmpId = Employee.EmpId
			WHERE
				Employee.ComId = @ComId
				AND Employee.DeptId = @DeptId
			GROUP BY
				Employee.ComId,
				Employee.DeptId;

	END



END
```

Trigger

- **It will update attSummary table on update and insert of attendance**

```
USE [MVCTask]
GO
/****** Object:  Trigger [dbo].[UpdateAttendanceSummary]    Script Date: 6/26/2023 1:33:26 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[UpdateAttendanceSummary]
ON [MVCTask].[dbo].[Attendance]
AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;

  -- Update the corresponding rows in [AttendanceSummary] for INSERT operation
  UPDATE ASummary
  SET ASummary.Present = ASummary.Present +
      CASE WHEN I.AttStatus = 'P' THEN 1 ELSE 0 END,
      ASummary.Late = ASummary.Late +
      CASE WHEN I.AttStatus = 'L' THEN 1 ELSE 0 END,
      ASummary.Absent = ASummary.Absent +
      CASE WHEN I.AttStatus = 'A' THEN 1 ELSE 0 END
  FROM [MVCTask].[dbo].[AttSummary] ASummary
  INNER JOIN inserted I ON ASummary.ComId = I.ComId
                        AND ASummary.EmpId = I.EmpId
                        AND ASummary.dtYear = YEAR(I.dtDate)
                        AND ASummary.dtMonth = MONTH(I.dtDate);
```

```
  -- Update the corresponding rows in [AttendanceSummary] for UPDATE operation
  UPDATE ASummary
  SET ASummary.Present = ASummary.Present +
      CASE WHEN I.AttStatus = 'P' THEN 1 ELSE 0 END -
      CASE WHEN D.AttStatus = 'P' THEN 1 ELSE 0 END,
      ASummary.Late = ASummary.Late +
      CASE WHEN I.AttStatus = 'L' THEN 1 ELSE 0 END -
      CASE WHEN D.AttStatus = 'L' THEN 1 ELSE 0 END,
      ASummary.Absent = ASummary.Absent +
      CASE WHEN I.AttStatus = 'A' THEN 1 ELSE 0 END -
      CASE WHEN D.AttStatus = 'A' THEN 1 ELSE 0 END
  FROM [MVCTask].[dbo].[AttSummary] ASummary
  INNER JOIN inserted I ON ASummary.ComId = I.ComId
                       AND ASummary.EmpId = I.EmpId
                       AND ASummary.dtYear = YEAR(I.dtDate)
                       AND ASummary.dtMonth = MONTH(I.dtDate)
  INNER JOIN deleted D ON ASummary.ComId = D.ComId
                       AND ASummary.EmpId = D.EmpId
                       AND ASummary.dtYear = YEAR(D.dtDate)
                       AND ASummary.dtMonth = MONTH(D.dtDate);
END;
```

- It will update salary table on insert , update of Attendance and attsummary table also

```
USE [MVCTask]
GO
/****** Object:  Trigger [dbo].[UpdateSalaryAbsentAmount]    Script Date: 6/26/2023 1:34:55 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[UpdateSalaryAbsentAmount]
ON [MVCTask].[dbo].[AttSummary]
AFTER UPDATE
AS
BEGIN
  SET NOCOUNT ON;

  -- Check if the relevant columns were updated
  IF UPDATE(Present) OR UPDATE(Late) OR UPDATE(Absent)
  BEGIN
    -- Update the corresponding rows in [Salary]
    UPDATE TOP (10000) Salary
    SET Salary.AbsentAmount = ASummary.Absent * (Salary.Basic /30),
       Salary.PayableAmount = Salary.Gross - Salary.AbsentAmount
    FROM [MVCTask].[dbo].[Salary] Salary
    INNER JOIN inserted ASummary ON Salary.ComId = ASummary.ComId
                                AND Salary.EmpId = ASummary.EmpId
                                AND Salary.dtYear = ASummary.dtYear
                                AND Salary.dtMonth = ASummary.dtMonth;
  END
END;
```