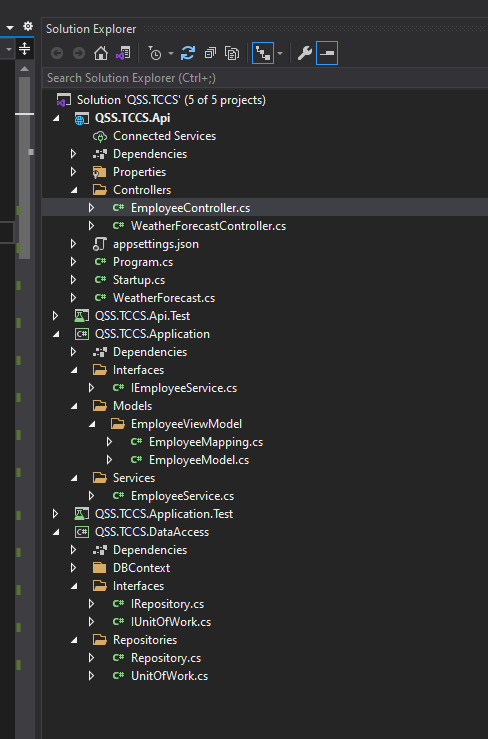
**EPAM -TCCS - Generic repository pattern in .net core web api**



1) Create **.Net core blank solution -> QSS.TCCS** project

2) Create **QSS.TCCS.DataAccess** .net core C# class library into solution

3) Install below library into **QSS.TCCS.DataAccess** classlibrary using **NuGet package manager**

**Microsoft.EntityFrameworkCore.SqlServer**

**Microsoft.EntityFrameworkCore.SqlServer.Design**

**Microsoft.EntityFrameworkCore.Tools**

4) Create **Models** folder in **QSS.TCCS.DataAccess** class library

5)Execute below query using **Package manager console** to generate **TCCSContext** class and **entities**

**PM> Scaffold-DbContext "Server=DESKTOP-PN1JAFF;Database=TCCS;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models**

After executing this command, it will generate files in Models folder

6) Open **TCCSContext.cs** file and comment the **OnConfiguring()** method

7) Add connection string in **appsettings.json** file

**"ConnectionStrings": {**

**"TCCSConnection": "Server=DESKTOP-PN1JAFF;Database=TCCS;Trusted\_Connection=True"**

**}**

8) Open **Startup.cs** file and add below into **ConfigureServices()** method

**services.AddDbContext<TCCSContext>(item => item.UseSqlServer**

**(Configuration.GetConnectionString("TCCSConnection")));**

9) Create **Interfaces** folder into **QSS.TCCS.DataAccess** class library and add below interface into this folder-

**IUnitOfWork.cs**

public interface IUnitOfWork : IDisposable

{

DbContext Context { get;}

}

**IRepository.cs**

public interface IRepository<TEntity, TType> where TEntity : class

{

Task<IEnumerable<TEntity>> GetAll();

Task<TEntity> GetById(TType id);

Task<IEnumerable<TEntity>> GetById(Expression<Func<TEntity, bool>> predicate);

Task<TEntity> AddAsync(TEntity entity);

TEntity Update(TEntity entity);

void Remove(TEntity entity);

Task RemoveById(TType id);

int SaveChanges();

Task<int> SaveChangesAsync();

Task<TEntity> SingleOrDefaultAsync(Expression<Func<TEntity, bool>> predicate);

Task<TEntity> FirstOrDefaultAsync(Expression<Func<TEntity, bool>> predicate);

void AddRange(IEnumerable<TEntity> entities);

Task AddRangeAsync(IEnumerable<TEntity> entities);

void UpdateRange(IEnumerable<TEntity> entities);

void RemoveRange(IEnumerable<TEntity> entities);

IEnumerable<TEntity> Include(params Expression<Func<TEntity, object>>[] includes);

Task<bool> CanSafeToRemove(TEntity entity);

}

10) Create **Repositories** folder into **QSS.TCCS.DataAccess** class library and add below **classes** into this folder-

**UnitOfWork.cs**

public class UnitOfWork : IUnitOfWork

{

public DbContext Context { get; }

public UnitOfWork(TCCSContext context)

{

Context = context;

}

public void Dispose()

{

Context.Dispose();

}

}

**Repository.cs**

public class Repository<TEntity, TType> : IRepository<TEntity, TType> where TEntity : class

{

protected IUnitOfWork \_unitOfWork;

protected DbSet<TEntity> \_entities { get; }

protected DbContext \_context;

public Repository(IUnitOfWork unitOfWork)

{

\_unitOfWork = unitOfWork;

if (\_unitOfWork.Context != null)

{

\_context = \_unitOfWork.Context;

\_entities = \_context.Set<TEntity>();

}

}

public async Task<IEnumerable<TEntity>> GetAll()

{

return await \_entities.ToListAsync();

}

public async Task<TEntity> GetById(TType id)

{

return await \_entities.FindAsync(id);

}

public async Task<IEnumerable<TEntity>> GetById(Expression<Func<TEntity, bool>> predicate)

{

return await \_entities.Where(predicate).ToListAsync();

}

public async Task<TEntity> AddAsync(TEntity entity)

{

var result = await \_entities.AddAsync(entity);

return result.Entity;

}

public TEntity Update(TEntity entity)

{

return \_entities.Update(entity).Entity;

}

public void Remove(TEntity entity)

{

\_entities.Remove(entity);

}

public async Task RemoveById(TType id)

{

var entity = await GetById(id);

Remove(entity);

}

public int SaveChanges()

{

return \_unitOfWork.Context.SaveChanges();

}

public async Task<int> SaveChangesAsync()

{

return await \_unitOfWork.Context.SaveChangesAsync();

}

public async Task<TEntity> SingleOrDefaultAsync(Expression<Func<TEntity, bool>> predicate)

{

return await \_entities.SingleOrDefaultAsync(predicate);

}

public async Task<TEntity> FirstOrDefaultAsync(Expression<Func<TEntity, bool>> predicate)

{

return await \_entities.FirstOrDefaultAsync(predicate);

}

public void AddRange(IEnumerable<TEntity> entities)

{

\_entities.AddRange(entities);

}

public async Task AddRangeAsync(IEnumerable<TEntity> entities)

{

await \_entities.AddRangeAsync(entities);

}

public void UpdateRange(IEnumerable<TEntity> entities)

{

\_entities.UpdateRange(entities);

}

public void RemoveRange(IEnumerable<TEntity> entities)

{

\_entities.RemoveRange(entities);

}

public IEnumerable<TEntity> Include(params Expression<Func<TEntity, object>>[] includeExpressions)

{

IQueryable<TEntity> query = null;

foreach (var includeExpression in includeExpressions)

{

query = \_entities.Include(includeExpression);

}

return query ?? \_entities;

}

public Task<bool> CanSafeToRemove(TEntity entity)

{

throw new NotImplementedException();

}

}

11) Create **QSS.TCCS.Application ->** .net core class library into solution

12) Create **Models** Folder in **QSS.TCCS.Application** library

13) Create **EmployeeViewModel** folder in **Models** folder

14) Install **AutoMapper.Extensions.Microsoft.DependencyInjection** into **QSS.TCCS.Application** Class library

15) Add below into **Startup.cs** class

public void ConfigureServices(IServiceCollection services)

{

services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());

}

16) Add **EmployeeModel.cs** and **EmployeeMapping.cs** class into **QSS.TCCS.Application -> Models ->**

**EmployeeViewModel** folder

**EmployeeModel.cs**

public class EmployeeModel

{

public int Id { get; set; }

public string Name { get; set; }

public string EmailId { get; set; }

public string Gender { get; set; }

public string MobileNumber { get; set; }

public int? Salary { get; set; }

}

**Employeemapping.cs**

public class EmployeeMapping : Profile

{

public EmployeeMapping()

{

CreateMap<EmployeeModel, Employee>()

.ForMember(dest => dest.Id, opt => opt.MapFrom(src => src.Id))

.ForMember(dest => dest.Name, opt => opt.MapFrom(src => src.Name))

.ForMember(dest => dest.EmailId, opt => opt.MapFrom(src => src.EmailId))

.ForMember(dest => dest.Gender, opt => opt.MapFrom(src => src.Gender))

.ForMember(dest => dest.MobileNumber, opt => opt.MapFrom(src => src.MobileNumber))

.ForMember(dest => dest.Salary, opt => opt.MapFrom(src => src.Salary));

CreateMap<Employee, EmployeeModel>();

}

}

17) Create **Interfaces** folder into **QSS.TCCS.Application** class library and add below interface into this folder-

public interface IEmployeeService

{

Task<IEnumerable<EmployeeModel>> GetAllEmployee();

Task<EmployeeModel> GetEmployeeById(int id);

Task<IEnumerable<EmployeeModel>> GetEmployeeById(Expression<Func<Employee, bool>> predicate);

Task<EmployeeModel> AddEmployeeAsync(EmployeeModel entity);

Task<EmployeeModel> UpdateEmployee(EmployeeModel entity);

Task<int> RemoveEmployee(EmployeeModel entity);

Task<int> RemoveEmployeeById(int id);

Task<EmployeeModel> SingleOrDefaultEmployeeAsync(Expression<Func<Employee, bool>> predicate);

Task<EmployeeModel> FirstOrDefaultEmployeeAsync(Expression<Func<Employee, bool>> predicate);

Task<int> AddEmployeeRange(IEnumerable<EmployeeModel> entities);

Task<int> AddEmployeeRangeAsync(IEnumerable<EmployeeModel> entities);

Task<int> UpdateEmployeeRange(IEnumerable<EmployeeModel> entities);

Task<int> RemoveEmployeeRange(IEnumerable<EmployeeModel> entities);

}

17) Create **Services** folder into **QSS.TCCS.Application** class library and add below service into this folder-

public class EmployeeService : IEmployeeService

{

private readonly IRepository<Employee, int> \_employeeRepository;

private readonly IMapper \_mapper;

public EmployeeService(IRepository<Employee, int> employeeRepository, IMapper mapper)

{

this.\_employeeRepository = employeeRepository;

this.\_mapper = mapper;

}

public async Task<IEnumerable<EmployeeModel>> GetAllEmployee()

{

try

{

IEnumerable<Employee> employees = await \_employeeRepository.GetAll();

return \_mapper.Map<IEnumerable<EmployeeModel>>(employees);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<EmployeeModel> GetEmployeeById(int id)

{

try

{

Employee employee = await \_employeeRepository.GetById(id);

return \_mapper.Map<EmployeeModel>(employee);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<IEnumerable<EmployeeModel>> GetEmployeeById(Expression<Func<Employee, bool>> predicate)

{

try

{

IEnumerable<Employee> employee = await \_employeeRepository.GetById(predicate);

return \_mapper.Map<IEnumerable<EmployeeModel>>(employee);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<EmployeeModel> AddEmployeeAsync(EmployeeModel entity)

{

try

{

var employee = \_mapper.Map<Employee>(entity);

var data = await \_employeeRepository.AddAsync(employee);

int result = await \_employeeRepository.SaveChangesAsync();

return \_mapper.Map<EmployeeModel>(employee);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<EmployeeModel> UpdateEmployee(EmployeeModel entity)

{

try

{

var employee = \_mapper.Map<Employee>(entity);

var data = \_employeeRepository.Update(employee);

int result = await \_employeeRepository.SaveChangesAsync();

return \_mapper.Map<EmployeeModel>(employee);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> RemoveEmployee(EmployeeModel entity)

{

try

{

var employee = \_mapper.Map<Employee>(entity);

\_employeeRepository.Remove(employee);

int result = await \_employeeRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> RemoveEmployeeById(int id)

{

try

{

await \_employeeRepository.RemoveById(id);

int result = await \_employeeRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<EmployeeModel> SingleOrDefaultEmployeeAsync(Expression<Func<Employee, bool>> predicate)

{

try

{

Employee employee = await \_employeeRepository.SingleOrDefaultAsync(predicate);

return \_mapper.Map<EmployeeModel>(employee);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<EmployeeModel> FirstOrDefaultEmployeeAsync(Expression<Func<Employee, bool>> predicate)

{

try

{

var employee = await \_employeeRepository.FirstOrDefaultAsync(predicate);

return \_mapper.Map<EmployeeModel>(employee);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> AddEmployeeRange(IEnumerable<EmployeeModel> entities)

{

try

{

var employees = \_mapper.Map<IEnumerable<Employee>>(entities);

\_employeeRepository.AddRange(employees);

int result = await \_employeeRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> AddEmployeeRangeAsync(IEnumerable<EmployeeModel> entities)

{

try

{

var employees = \_mapper.Map<IEnumerable<Employee>>(entities);

await \_employeeRepository.AddRangeAsync(employees);

int result = await \_employeeRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> UpdateEmployeeRange(IEnumerable<EmployeeModel> entities)

{

try

{

var employees = \_mapper.Map<IEnumerable<Employee>>(entities);

\_employeeRepository.UpdateRange(employees);

int result = await \_employeeRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> RemoveEmployeeRange(IEnumerable<EmployeeModel> entities)

{

try

{

var employees = this.\_mapper.Map<IEnumerable<Employee>>(entities);

this.\_employeeRepository.RemoveRange(employees);

int result = await \_employeeRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

}

18) Create .Net core api project with name **QSS.TCCS.API** and Add **EmployeeController.cs**

[Route("api/[controller]")]

[ApiController]

public class EmployeeController : ControllerBase

{

private readonly IEmployeeService \_employeeService;

private readonly IMapper \_mapper;

public EmployeeController(IEmployeeService employeeService, IMapper mapper)

{

this.\_employeeService = employeeService;

this.\_mapper = mapper;

}

[HttpGet("GetAllEmployee")]

public async Task<IActionResult> GetAllEmployee()

{

IEnumerable<EmployeeModel> result = await this.\_employeeService.GetAllEmployee();

return Ok(result);

}

[HttpGet("GetEmployeeById/{id}")]

public async Task<IActionResult> GetEmployeeById(int id)

{

EmployeeModel result = await this.\_employeeService.GetEmployeeById(id);

return Ok(result);

}

[HttpGet("GetByIdEmployee/{id}")]

public async Task<IActionResult> GetByIdEmployee(int id)

{

IEnumerable<EmployeeModel> result = await \_employeeService.GetEmployeeById(x => x.Id == id);

return Ok(result);

}

[HttpGet("GetEmployeeSignleOrDefault/{id}")]

public async Task<IActionResult> GetEmployeeSignleOrDefault(int id)

{

EmployeeModel result = await \_employeeService.SingleOrDefaultEmployeeAsync(x => x.Id == id);

return Ok(result);

}

[HttpGet("GetEmployeeFirstOrDefault/{id}")]

public async Task<IActionResult> GetEmployeeFirstOrDefault(int id)

{

EmployeeModel result = await \_employeeService.FirstOrDefaultEmployeeAsync(x => x.Id == id);

return Ok(result);

}

[HttpPost("AddEmployeeAsync")]

public async Task<IActionResult> AddEmployeeAsync(EmployeeModel employee)

{

EmployeeModel result = await \_employeeService.AddEmployeeAsync(employee);

return Ok(result);

}

[HttpPost("AddEmployeeRange")]

public async Task<IActionResult> AddEmployeeRange(IEnumerable<EmployeeModel> employees)

{

int result = await \_employeeService.AddEmployeeRange(employees);

return Ok(result);

}

[HttpPost("AddEmployeeRangeAsync")]

public async Task<IActionResult> AddEmployeeRangeAsync(IEnumerable<EmployeeModel> employees)

{

int result = await \_employeeService.AddEmployeeRangeAsync(employees);

return Ok(result);

}

[HttpPost("UpdateEmployee")]

public async Task<IActionResult> UpdateEmployee(EmployeeModel employee)

{

EmployeeModel result = await \_employeeService.UpdateEmployee(employee);

return Ok(result);

}

[HttpPost("UpdateEmployeeRange")]

public async Task<IActionResult> UpdateEmployeeRange(IEnumerable<EmployeeModel> employees)

{

int result = await \_employeeService.UpdateEmployeeRange(employees);

return Ok(result);

}

[HttpPost("RemoveEmployee")]

public async Task<IActionResult> RemoveEmployee(EmployeeModel employee)

{

int result = await \_employeeService.RemoveEmployee(employee);

return Ok(result);

}

[HttpPost("RemoveEmployeeById")]

public async Task<IActionResult> RemoveEmployeeById(int id)

{

int result = await \_employeeService.RemoveEmployeeById(id);

return Ok(result);

}

[HttpPost("RemoveEmployeeByRange")]

public async Task<IActionResult> RemoveEmployeeByRange(IEnumerable<EmployeeModel> employees)

{

int result = await \_employeeService.RemoveEmployeeRange(employees);

return Ok(result);

}

}

**XUnit Testing Using InMemory database**

22) Create **XUnit** project into **QSS.TCCS** with name **QSS.TCCS.UnitTest.Core**

23) Install **AutoMapper** and **Microsoft.EntityFrameworkCore.InMemory**

24) Create **AutoMapping.cs** class into **QSS.TCCS.UnitTest.Core**

public class AutoMapping

{

public IMapper GetMapper(Profile profile)

{

var mockMapper = new MapperConfiguration(cfg =>

{

cfg.AddProfile(profile);

});

var mapper = mockMapper.CreateMapper();

return mapper;

}

}

25) Create **TCCSDataFixture.cs** class into **QSS.TCCS.UnitTest.Core**

public class TCCSDataFixture : IDisposable

{

public TCCSContext tccsContext { get; private set; }

public DbContextOptions<TCCSContext> tccsContextOptions { get; private set; }

private const string Database = "TCCSInMemoryDatabase";

public TCCSDataFixture()

{

tccsContextOptions = new DbContextOptionsBuilder<TCCSContext>()

.UseInMemoryDatabase(Database + DateTime.Now.ToFileTimeUtc())

.ConfigureWarnings(x => x.Ignore(InMemoryEventId.TransactionIgnoredWarning))

.UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking)

.EnableSensitiveDataLogging(true)

.Options;

tccsContext = new TCCSContext(tccsContextOptions);

tccsContext.Database.EnsureDeleted();

tccsContext.Database.EnsureCreated();

}

public void Dispose()

{

tccsContext.Database.EnsureDeleted();

tccsContext.Dispose();

}

}

26) Create **MoqData** Folder into **QSS.TCCS.UnitTest.Core**

27) Add **EmployeeMoq.cs** class into **QSS.TCCS.UnitTest.Core -> MoqData**

public class EmployeeMoq : IClassFixture<TCCSDataFixture>

{

TCCSDataFixture fixture;

public EmployeeMoq(TCCSDataFixture fixture)

{

this.fixture = fixture;

}

public void MoqData(Employee entity)

{

using (var qssContext = new TCCSContext(fixture.tccsContextOptions))

{

qssContext.Employees.Add(entity);

qssContext.SaveChanges();

}

}

public void MoqDataList(IEnumerable<Employee> entityList)

{

using (var qssContext = new TCCSContext(fixture.tccsContextOptions))

{

qssContext.Employees.AddRangeAsync(entityList);

qssContext.SaveChanges();

}

}

}

**XUnit testing for QSS.TCCS.DataAccess**

28) Create **XUnit** project with name **QSS.TCCS.DataAccess.UnitTests** into **QSS.TCCS**

29) Create **Repositories** folder and add **EmployeeRepositoryTest.cs** file into it and add below code

public class EmployeeRepositoryTest : IClassFixture<TCCSDataFixture>, IDisposable

{

TCCSDataFixture fixture;

public EmployeeRepositoryTest(TCCSDataFixture fixture)

{

this.fixture = fixture;

}

[Fact]

public async Task GetAllEmployee\_ShouldReturnList()

{

//Arrange

var employeeList = GetEmployeeList();

EmployeeMoq employeeMoq = new EmployeeMoq(fixture);

employeeMoq.MoqDataList(employeeList);

IUnitOfWork unitOfWork = new UnitOfWork(fixture.tccsContext);

IRepository<Employee,int> repository = new Repository<Employee, int>(unitOfWork);

IEmployeeRepository employeeRepository = new EmployeeRepository(repository);

//Act

var result = await employeeRepository.GetAllEmployee();

//Assert

Assert.IsAssignableFrom<IEnumerable<Employee>>(result);

Assert.NotNull(result);

Assert.Equal(4, result.Count());

}

[Fact]

public async Task GetEmployeeById\_ShouldReturnSpecificRecord()

{

//Arrange

int id = 1;

var employeeList = GetEmployeeList();

EmployeeMoq employeeMoq = new EmployeeMoq(fixture);

employeeMoq.MoqDataList(employeeList);

IUnitOfWork unitOfWork = new UnitOfWork(fixture.tccsContext);

IRepository<Employee, int> repository = new Repository<Employee, int>(unitOfWork);

IEmployeeRepository employeeRepository = new EmployeeRepository(repository);

//Act

var result = await employeeRepository.GetEmployeeById(id);

//Assert

Assert.IsAssignableFrom<Employee>(result);

Assert.NotNull(result);

Assert.Equal(id, result.Id);

}

[Fact]

public async Task GetEmployeeByIdUsingPredicate\_ShouldReturnNullIfIdNotMatch()

{

//Arrange

int id = 0;

var employeeList = GetEmployeeList();

EmployeeMoq employeeMoq = new EmployeeMoq(fixture);

employeeMoq.MoqDataList(employeeList);

IUnitOfWork unitOfWork = new UnitOfWork(fixture.tccsContext);

IRepository<Employee, int> repository = new Repository<Employee, int>(unitOfWork);

IEmployeeRepository employeeRepository = new EmployeeRepository(repository);

//Act

var result = await employeeRepository.GetEmployeeById(id);

//Assert

Assert.Null(result);

}

[Fact]

public async Task GetEmployeeByIdUsingPredicate\_ShouldReturnSpecificRecord()

{

//Arrange

int id = 1;

var employeeList = GetEmployeeList();

EmployeeMoq employeeMoq = new EmployeeMoq(fixture);

employeeMoq.MoqDataList(employeeList);

IUnitOfWork unitOfWork = new UnitOfWork(fixture.tccsContext);

IRepository<Employee, int> repository = new Repository<Employee, int>(unitOfWork);

IEmployeeRepository employeeRepository = new EmployeeRepository(repository);

//Act

var result = await employeeRepository.GetEmployeeById(x => x.Id == id);

//Assert

Assert.IsAssignableFrom<IEnumerable<Employee>>(result);

Assert.NotNull(result);

Assert.Single(result);

}

[Fact]

public async Task AddEmploeeAsync\_ShouldSaveEmployee()

{

//Arrange

var employee = GetEmployee();

EmployeeMoq employeeMoq = new EmployeeMoq(fixture);

employeeMoq.MoqData(employee);

var addEmployee = employee;

addEmployee.Id = 0;

IUnitOfWork unitOfWork = new UnitOfWork(fixture.tccsContext);

IRepository<Employee, int> repository = new Repository<Employee, int>(unitOfWork);

IEmployeeRepository employeeRepository = new EmployeeRepository(repository);

//Act

var result = await employeeRepository.AddEmployeeAsync(addEmployee);

//Assert

Assert.IsAssignableFrom<Employee>(result);

Assert.Equal(employee.Name, result.Name);

}

[Fact]

public async Task AddProductAsync\_ThrowException()

{

//Arrange

IUnitOfWork unitOfWork = new UnitOfWork(fixture.tccsContext);

IRepository<Employee, int> repository = new Repository<Employee, int>(unitOfWork);

IEmployeeRepository employeeRepository = new EmployeeRepository(repository);

//Act

Task act() => employeeRepository.AddEmployeeAsync(null);

//Assert

await Assert.ThrowsAsync<ArgumentNullException>(act);

}

[Fact]

public void UpdateEmployee\_ShouldUpdateEmployee()

{

//Arrange

var employee = GetEmployee();

EmployeeMoq employeeMoq = new EmployeeMoq(fixture);

employeeMoq.MoqData(employee);

employee.Name = "test111";

IUnitOfWork unitOfWork = new UnitOfWork(fixture.tccsContext);

IRepository<Employee, int> repository = new Repository<Employee, int>(unitOfWork);

IEmployeeRepository employeeRepository = new EmployeeRepository(repository);

//Act

var result = employeeRepository.UpdateEmployee(employee);

//Assert

Assert.IsAssignableFrom<Employee>(result);

Assert.Equal(employee.Name, result.Name);

}

[Fact]

public void UpdateEmployee\_ShouldThrowException()

{

//Arrange

IUnitOfWork unitOfWork = new UnitOfWork(fixture.tccsContext);

IRepository<Employee, int> repository = new Repository<Employee, int>(unitOfWork);

IEmployeeRepository employeeRepository = new EmployeeRepository(repository);

//Act

Employee act() => employeeRepository.UpdateEmployee(null);

//Assert

Assert.Throws<NullReferenceException>(act);

}

[Fact]

public void RemoveEmployee\_ShouldRemoveEmployee()

{

//Arrange

var employee = GetEmployee();

employee.Id = 5;

EmployeeMoq employeeMoq = new EmployeeMoq(fixture);

employeeMoq.MoqData(employee);

IUnitOfWork unitOfWork = new UnitOfWork(fixture.tccsContext);

IRepository<Employee, int> repository = new Repository<Employee, int>(unitOfWork);

IEmployeeRepository employeeRepository = new EmployeeRepository(repository);

//Act

employeeRepository.RemoveEmployee(employee);

}

[Fact]

public async Task RemoveEmployeeById\_ShouldRemoveEmployeeAsync()

{

//Arrange

int id = 1;

var employee = GetEmployee();

EmployeeMoq employeeMoq = new EmployeeMoq(fixture);

employeeMoq.MoqData(employee);

IUnitOfWork unitOfWork = new UnitOfWork(fixture.tccsContext);

IRepository<Employee, int> repository = new Repository<Employee, int>(unitOfWork);

IEmployeeRepository employeeRepository = new EmployeeRepository(repository);

//Act

await employeeRepository.RemoveEmployeeById(id);

}

public void Dispose()

{

fixture.tccsContext.Database.EnsureDeleted();

}

private IEnumerable<Employee> GetEmployeeList()

{

List<Employee> employeeList = new List<Employee>()

{

new Employee{Id=1,Name="test1",EmailId="test1@gmail.com",Gender="Male",MobileNumber="1234567890",Salary=30000},

new Employee{Id=2,Name="test2",EmailId="test2@gmail.com",Gender="Female",MobileNumber="1234567890",Salary=20000},

new Employee{Id=3,Name="test3",EmailId="test3@gmail.com",Gender="Female",MobileNumber="1234567890",Salary=80000},

new Employee{Id=4,Name="test4",EmailId="test4@gmail.com",Gender="Male",MobileNumber="1234567890",Salary=50000},

};

return employeeList;

}

private Employee GetEmployee()

{

Employee employee = new Employee()

{

Id = 1,

Name = "test1",

EmailId = "test1@gmail.com",

Gender = "Male",

MobileNumber = "1234567890",

Salary = 30000

};

return employee;

}

}

**XUnit testing for QSS.TCCS.Application**

30) Create **XUnit** project with name **QSS.TCCS.Application.UnitTests** into **CAW.RBC**

31) Create **Services** folder and add **EmployeeServiceTest.cs** file into it and add below code

public class ProductServiceTest : IClassFixture<CAWDataFixture>, IDisposable

{

CAWDataFixture fixture;

public ProductServiceTest(CAWDataFixture fixture)

{

this.fixture = fixture;

}

[Fact]

public async Task GetAllProduct\_ShouldReturnList()

{

//Arrange

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

var productModelList = GetProductModelList();

var productList = mapper.Map<IEnumerable<Product>>(productModelList);

ProductMoq productMoq = new ProductMoq(fixture);

productMoq.MoqDataList(productList);

IProductService service = new ProductService(

fixture.cawContext,new ProductRepository(fixture.cawContext), mapper);

//Act

var result = await service.GetAllProduct();

//Assert

Assert.IsAssignableFrom<IEnumerable<ProductModel>>(result);

}

[Fact]

public async Task GetProduct\_ShouldReturnProduct()

{

//Arrange

int id = 1;

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

var productModelList = GetProductModelList();

var productList = mapper.Map<IEnumerable<Product>>(productModelList);

ProductMoq productMoq = new ProductMoq(fixture);

productMoq.MoqDataList(productList);

IProductService service = new ProductService(

fixture.cawContext, new ProductRepository(fixture.cawContext), mapper);

//Act

var result = await service.GetProduct(id);

//Assert

Assert.IsAssignableFrom<ProductModel>(result);

}

[Fact]

public async Task GetProduct\_ShouldReturnNullIfIdNotMatch()

{

//Arrange

int id = 0;

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

var productModelList = GetProductModelList();

var productList = mapper.Map<IEnumerable<Product>>(productModelList);

ProductMoq productMoq = new ProductMoq(fixture);

productMoq.MoqDataList(productList);

IProductService service = new ProductService(

fixture.cawContext, new ProductRepository(fixture.cawContext), mapper);

//Act

var result = await service.GetProduct(id);

//Assert

Assert.Null(result);

}

[Fact]

public async Task GetProductById\_ShouldReturnProduct()

{

//Arrange

int id = 1;

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

var productModelList = GetProductModelList();

var productList = mapper.Map<IEnumerable<Product>>(productModelList);

ProductMoq productMoq = new ProductMoq(fixture);

productMoq.MoqDataList(productList);

IProductService service = new ProductService(

fixture.cawContext, new ProductRepository(fixture.cawContext), mapper);

//Act

var result = await service.GetProductById(x => x.Id == id);

//Assert

Assert.IsAssignableFrom<IEnumerable<ProductModel>>(result);

}

[Fact]

public async Task AddProductAsync\_ShouldSaveProduct()

{

//Arrange

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

var productModel = GetProductModel();

var product = mapper.Map<Product>(productModel);

ProductMoq productMoq = new ProductMoq(fixture);

productMoq.MoqData(product);

var addProduct = GetAddProductModel();

addProduct.Id = 0;

IProductService service = new ProductService(

fixture.cawContext, new ProductRepository(fixture.cawContext), mapper);

//Act

var result = await service.AddProductAsync(addProduct);

//Assert

Assert.IsAssignableFrom<ProductModel>(result);

//Assert.Equal(product.Name, result.Name);

}

[Fact]

public async Task AddProductAsync\_ShouldthrowException()

{

//Arrange

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

IProductService service = new ProductService(

fixture.cawContext, new ProductRepository(fixture.cawContext), mapper);

//Act

Task act() => service.AddProductAsync(null);

//Assert

await Assert.ThrowsAsync<System.Exception>(act);

}

[Fact]

public async Task UpdateProduct\_ShouldUpdateProduct()

{

//Arrange

string updatedValue = "test1111";

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

var productModel = GetProductModel();

var product = mapper.Map<Product>(productModel);

ProductMoq productMoq = new ProductMoq(fixture);

productMoq.MoqData(product);

productModel.Name = updatedValue;

IProductService service = new ProductService(

fixture.cawContext, new ProductRepository(fixture.cawContext), mapper);

//Act

var result = await service.UpdateProduct(productModel);

//Assert

Assert.IsAssignableFrom<ProductModel>(result);

Assert.Equal(updatedValue, result.Name);

}

[Fact]

public async Task UpdateProduct\_ShouldthrowException()

{

//Arrange

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

IProductService service = new ProductService(

fixture.cawContext, new ProductRepository(fixture.cawContext), mapper);

//Act

Task act() => service.UpdateProduct(null);

//Assert

await Assert.ThrowsAsync<System.Exception>(act);

}

[Fact]

public async Task RemoveProduct\_ShouldRemoveProduct()

{

//Arrange

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

var productModel = GetProductModel();

var product = mapper.Map<Product>(productModel);

ProductMoq productMoq = new ProductMoq(fixture);

productMoq.MoqData(product);

IProductService service = new ProductService(

fixture.cawContext, new ProductRepository(fixture.cawContext), mapper);

//Act

var result = await service.RemoveProduct(productModel);

//Assert

Assert.IsAssignableFrom<int>(result);

Assert.Equal(1, result);

}

[Fact]

public async Task RemoveProduct\_ShouldThrowExceptionWhenWePassNull()

{

//Arrange

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

IProductService service = new ProductService(

fixture.cawContext, new ProductRepository(fixture.cawContext), mapper);

//Act

Task act() => service.RemoveProduct(null);

//Assert

await Assert.ThrowsAsync<System.Exception>(act);

}

[Fact]

public async Task RemoveProductById\_ShouldRemoveProduct()

{

//Arrange

int id = 1;

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

var productModel = GetProductModel();

var product = mapper.Map<Product>(productModel);

ProductMoq productMoq = new ProductMoq(fixture);

productMoq.MoqData(product);

IProductService service = new ProductService(

fixture.cawContext, new ProductRepository(fixture.cawContext), mapper);

//Act

var result = await service.RemoveProductById(id);

//Assert

Assert.IsAssignableFrom<int>(result);

Assert.Equal(1, result);

}

[Fact]

public async Task RemoveProductById\_ShouldThrowExceptionWhenIdNotMatch()

{

//Arrange

int id = 0;

AutoMapping autoMapping = new AutoMapping();

var mapper = autoMapping.GetMapper(new ProductMapping());

var productModel = GetProductModel();

var product = mapper.Map<Product>(productModel);

ProductMoq productMoq = new ProductMoq(fixture);

productMoq.MoqData(product);

IProductService service = new ProductService(

fixture.cawContext, new ProductRepository(fixture.cawContext), mapper);

//Act

Task act() => service.RemoveProductById(id);

//Assert

await Assert.ThrowsAsync<System.Exception>(act);

}

private IEnumerable<ProductModel> GetProductModelList()

{

List<ProductModel> productList = new List<ProductModel>()

{

new ProductModel{Id=1,Name="test1",Description="desc1",Quantity=10},

new ProductModel{Id=2,Name="test2",Description="desc2",Quantity=20},

new ProductModel{Id=3,Name="test3",Description="desc3",Quantity=30},

new ProductModel{Id=4,Name="test4",Description="desc4",Quantity=40},

};

return productList;

}

private ProductModel GetProductModel()

{

ProductModel product = new ProductModel()

{

Id = 1,

Name = "test1",

Description = "des1",

Quantity = 10

};

return product;

}

private ProductModel GetAddProductModel()

{

ProductModel product = new ProductModel()

{

Id = 0,

Name = "abc",

Description = "des1",

Quantity = 10

};

return product;

}

public void Dispose()

{

fixture.cawContext.Database.EnsureDeleted();

//fixture.cawContext.Dispose();

//fixture.Dispose();

}

}

**XUnit testing for QSS.TCCS.API**

32) Create **XUnit** project with name **QSS.TCCS.API.UnitTests** into **QSS.TCCS**

33) Create **Controller** folder and add **EmployeeControllerTest.cs** file into it and add below code

public class EmployeeControllerTest

{

[Fact]

public async Task GetAllEmployee\_ReturnOkResult()

{

// Arrange

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.GetAllEmployee()).ReturnsAsync(GetEmployeeModelList());

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.GetAllEmployee();

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task GetEmployeeById\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.GetEmployeeById(id)).ReturnsAsync(GetEmployeeModelById(id));

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.GetEmployeeById(id);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task GetEmployeeById\_ReturnNoContent()

{

// Arrange

int id = 10;

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.GetEmployeeById(id)).ReturnsAsync(GetEmployeeModelById(id));

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.GetEmployeeById(id);

//// Assert

Assert.Equal(204, ((Microsoft.AspNetCore.Mvc.StatusCodeResult)result).StatusCode);

Assert.IsType<NoContentResult>(result);

}

[Fact]

public async Task GetEmployeeByIdUsingPredicate\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.GetEmployeeById(x => x.Id == id)).ReturnsAsync(GetEmployeeModelUsingPredicate(id));

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.GetEmployeeByIdUsingPredicate(id);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task GetEmployeeSignleOrDefault\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.SingleOrDefaultEmployeeAsync(x => x.Id == id)).ReturnsAsync(GetEmployeeSingleOrDefault(id));

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.GetEmployeeSignleOrDefault(id);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task GetEmployeeSignleOrDefault\_ReturnNoContent()

{

// Arrange

int id = 10;

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.SingleOrDefaultEmployeeAsync(x => x.Id == id)).ReturnsAsync(GetEmployeeSingleOrDefault(id));

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.GetEmployeeSignleOrDefault(id);

//// Assert

Assert.Equal(204, ((Microsoft.AspNetCore.Mvc.StatusCodeResult)result).StatusCode);

Assert.IsType<NoContentResult>(result);

}

[Fact]

public async Task GetEmployeeFirstOrDefault\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.FirstOrDefaultEmployeeAsync(x => x.Id == id)).ReturnsAsync(GetEmployeeFirstOrDerfault(id));

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.GetEmployeeFirstOrDefault(id);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task GetEmployeeFirstOrDefault\_ReturnNoContent()

{

// Arrange

int id = 10;

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.FirstOrDefaultEmployeeAsync(x => x.Id == id)).ReturnsAsync(GetEmployeeFirstOrDerfault(id));

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.GetEmployeeFirstOrDefault(id);

//// Assert

Assert.Equal(204, ((Microsoft.AspNetCore.Mvc.StatusCodeResult)result).StatusCode);

Assert.IsType<NoContentResult>(result);

}

[Fact]

public async Task AddEmployeeAsync\_ReturnOkResult()

{

// Arrange

var mockRepo = new Mock<IEmployeeService>();

var addProduct = AddEmployeeModel();

mockRepo.Setup(repo => repo.AddEmployeeAsync(addProduct)).ReturnsAsync(addProduct);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.AddEmployeeAsync(addProduct);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task AddEmployeeAsync\_ReturnBadRequest()

{

// Arrange

var mockRepo = new Mock<IEmployeeService>();

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.AddEmployeeAsync(null);

//// Assert

Assert.Equal(400, ((Microsoft.AspNetCore.Mvc.StatusCodeResult)result).StatusCode);

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task AddEmployeeRange\_ReturnOkResult()

{

// Arrange

int output = 1;

var mockRepo = new Mock<IEmployeeService>();

var addProductList = GetEmployeeModelList();

mockRepo.Setup(repo => repo.AddEmployeeRange(addProductList)).ReturnsAsync(output);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.AddEmployeeRange(addProductList);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task AddEmployeeRange\_ReturnBadRequest()

{

// Arrange

int output = 1;

var mockRepo = new Mock<IEmployeeService>();

var addProductList = GetEmployeeModelList();

mockRepo.Setup(repo => repo.AddEmployeeRange(addProductList)).ReturnsAsync(output);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.AddEmployeeRange(null);

//// Assert

Assert.Equal(400, ((Microsoft.AspNetCore.Mvc.StatusCodeResult)result).StatusCode);

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task AddEmployeeRangeAsync\_ReturnOkResult()

{

// Arrange

int output = 1;

var mockRepo = new Mock<IEmployeeService>();

var addProductList = GetEmployeeModelList();

mockRepo.Setup(repo => repo.AddEmployeeRangeAsync(addProductList)).ReturnsAsync(output);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.AddEmployeeRangeAsync(addProductList);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task AddEmployeeRangeAsync\_ReturnBadRequest()

{

// Arrange

int output = 1;

var mockRepo = new Mock<IEmployeeService>();

var addProductList = GetEmployeeModelList();

mockRepo.Setup(repo => repo.AddEmployeeRangeAsync(addProductList)).ReturnsAsync(output);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.AddEmployeeRangeAsync(null);

//// Assert

Assert.Equal(400, ((Microsoft.AspNetCore.Mvc.StatusCodeResult)result).StatusCode);

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task UpdateEmployeeAsync\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IEmployeeService>();

var updateProduct = GetEmployeeModelById(id);

mockRepo.Setup(repo => repo.UpdateEmployee(updateProduct)).ReturnsAsync(updateProduct);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.UpdateEmployee(updateProduct);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task UpdateEmployeeAsync\_ReturnBadRequest()

{

// Arrange

var mockRepo = new Mock<IEmployeeService>();

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.UpdateEmployee(null);

//// Assert

Assert.Equal(400, ((Microsoft.AspNetCore.Mvc.StatusCodeResult)result).StatusCode);

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task UpdateEmployeeRange\_ReturnOkResult()

{

// Arrange

int output = 1;

var mockRepo = new Mock<IEmployeeService>();

var updateProductList = GetEmployeeModelList();

mockRepo.Setup(repo => repo.UpdateEmployeeRange(updateProductList)).ReturnsAsync(output);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.UpdateEmployeeRange(updateProductList);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task UpdateEmployeeRange\_ReturnBadRequest()

{

// Arrange

int output = 1;

var mockRepo = new Mock<IEmployeeService>();

var updateProductList = GetEmployeeModelList();

mockRepo.Setup(repo => repo.UpdateEmployeeRange(updateProductList)).ReturnsAsync(output);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.UpdateEmployeeRange(null);

//// Assert

Assert.Equal(400, ((Microsoft.AspNetCore.Mvc.StatusCodeResult)result).StatusCode);

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task RemoveEmployeeAsync\_ReturnOkResultWithSucess()

{

// Arrange

int id = 1;

int returnData = 1;

var mockRepo = new Mock<IEmployeeService>();

var employee = GetEmployeeModelById(id);

mockRepo.Setup(repo => repo.RemoveEmployee(employee)).ReturnsAsync(returnData);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.RemoveEmployee(employee);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

Assert.Equal(1, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).Value);

}

[Fact]

public async Task RemoveEmployeeAsync\_ReturnOkResultWithNotSucess()

{

// Arrange

int id = 1;

int returnData = 0;

var mockRepo = new Mock<IEmployeeService>();

var employee = GetEmployeeModelById(id);

mockRepo.Setup(repo => repo.RemoveEmployee(employee)).ReturnsAsync(returnData);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.RemoveEmployee(employee);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

Assert.Equal(0, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).Value);

}

[Fact]

public async Task RemoveEmployeeAsync\_ReturnBadRequest()

{

// Arrange

var mockRepo = new Mock<IEmployeeService>();

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.RemoveEmployee(null);

//// Assert

Assert.Equal(400, ((Microsoft.AspNetCore.Mvc.StatusCodeResult)result).StatusCode);

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task RemoveProductById\_ReturnOkResultWithSucess()

{

// Arrange

int returnData = 1;

int id = 1;

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.RemoveEmployeeById(id)).ReturnsAsync(returnData);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.RemoveEmployeeById(id);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

Assert.Equal(1, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).Value);

}

[Fact]

public async Task RemoveProductById\_ReturnOkResultWithNotSucess()

{

// Arrange

int returnData = 0;

int id = 1;

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.RemoveEmployeeById(id)).ReturnsAsync(returnData);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.RemoveEmployeeById(id);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

Assert.Equal(0, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).Value);

}

[Fact]

public async Task RemoveProductById\_ReturnBadRequest()

{

// Arrange

int id = 0;

var mockRepo = new Mock<IEmployeeService>();

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.RemoveEmployeeById(id);

//// Assert

Assert.Equal(400, ((Microsoft.AspNetCore.Mvc.StatusCodeResult)result).StatusCode);

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task RemoveEmployeeByRange\_ReturnOkResultWithSucess()

{

// Arrange

int returnData = 1;

var employeeList = GetEmployeeModelList();

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.RemoveEmployeeRange(employeeList)).ReturnsAsync(returnData);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.RemoveEmployeeByRange(employeeList);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

Assert.Equal(1, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).Value);

}

[Fact]

public async Task RemoveEmployeeByRange\_ReturnOkResultWithNotSucess()

{

// Arrange

int returnData = 0;

var employeeList = GetEmployeeModelList();

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.RemoveEmployeeRange(employeeList)).ReturnsAsync(returnData);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.RemoveEmployeeByRange(employeeList);

//// Assert

Assert.Equal(200, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).StatusCode);

Assert.IsType<OkObjectResult>(result);

Assert.Equal(0, ((Microsoft.AspNetCore.Mvc.ObjectResult)result).Value);

}

[Fact]

public async Task RemoveEmployeeByRange\_ReturnBadRequest()

{

// Arrange

int returnData = 0;

var employeeList = GetEmployeeModelList();

var mockRepo = new Mock<IEmployeeService>();

mockRepo.Setup(repo => repo.RemoveEmployeeRange(employeeList)).ReturnsAsync(returnData);

var controller = new EmployeeController(mockRepo.Object);

//// Act

var result = await controller.RemoveEmployeeByRange(null);

//// Assert

Assert.Equal(400, ((Microsoft.AspNetCore.Mvc.StatusCodeResult)result).StatusCode);

Assert.IsType<BadRequestResult>(result);

}

private IEnumerable<EmployeeModel> GetEmployeeModelList()

{

List<EmployeeModel> employeeList = new List<EmployeeModel>()

{

new EmployeeModel{Id=1,Name="test1",EmailId="test1@gmail.com",Gender="Male",MobileNumber="1234567890",Salary=30000},

new EmployeeModel{Id=2,Name="test2",EmailId="test2@gmail.com",Gender="Female",MobileNumber="1234567890",Salary=20000},

new EmployeeModel{Id=3,Name="test3",EmailId="test3@gmail.com",Gender="Female",MobileNumber="1234567890",Salary=80000},

new EmployeeModel{Id=4,Name="test4",EmailId="test4@gmail.com",Gender="Male",MobileNumber="1234567890",Salary=50000},

};

return employeeList;

}

private EmployeeModel GetEmployeeModelById(int emplyeeId)

{

var employee = GetEmployeeModelList().FirstOrDefault(x => x.Id == emplyeeId); ;

return employee;

}

private IEnumerable<EmployeeModel> GetEmployeeModelUsingPredicate(int employeeId)

{

var productList = GetEmployeeModelList().Where(x => x.Id == employeeId);

return productList;

}

private EmployeeModel GetEmployeeFirstOrDerfault(int employeeId)

{

var employee = GetEmployeeModelList().FirstOrDefault(x => x.Id == employeeId);

return employee;

}

private EmployeeModel GetEmployeeSingleOrDefault(int employeeId)

{

var employee = GetEmployeeModelList().SingleOrDefault(x => x.Id == employeeId);

return employee;

}

private EmployeeModel AddEmployeeModel()

{

EmployeeModel employee = new EmployeeModel()

{

Id = 0,

Name = "abc",

EmailId = "abc@gmail.com",

Gender = "Male",

MobileNumber = "1234567890",

Salary = 30000

};

return employee;

}

}