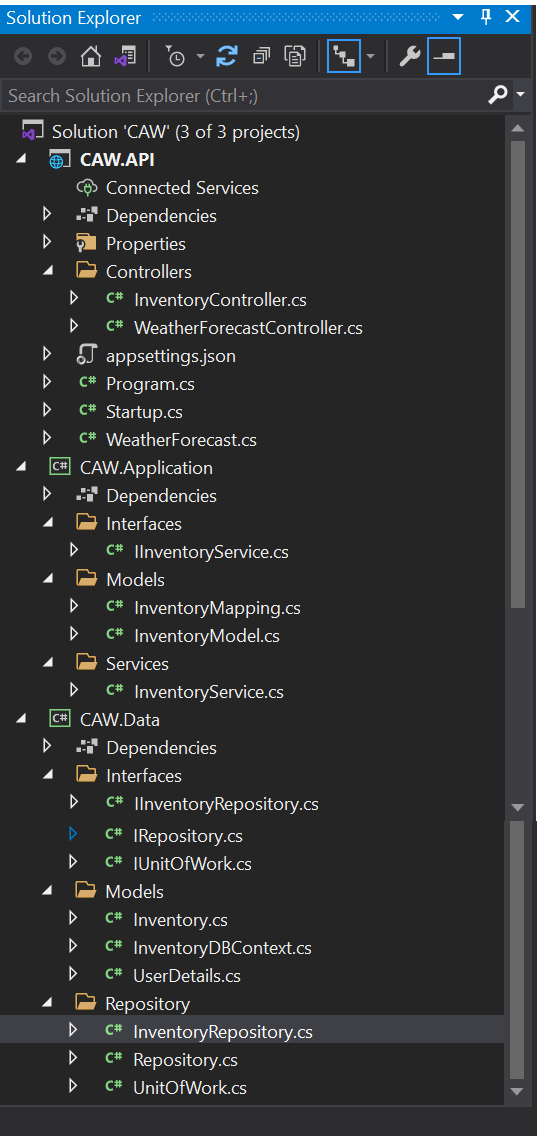
**Final Generic repository pattern in .net core web api and Xunit testing**



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

2) Create **CAW.Data** .net core C# class library into solution

3) Install below library into **CAW.Data** classlibrary using **NuGet package manager**

**Microsoft.EntityFrameworkCore.SqlServer**

**Microsoft.EntityFrameworkCore.SqlServer.Design**

**Microsoft.EntityFrameworkCore.Tools**

4) Create **Models** folder in **CAW**.**Data** class library

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

**PM> Scaffold-DbContext "Server=(localdb)\MSSQLLocalDB;Database=InventoryDB;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models**

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

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

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

"ConnectionStrings": {

"TCCSConnection": "Server=(localdb)\\MSSQLLocalDB;Database=InventoryDB;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 **CAW.Data** 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);

}

**IInventoryRepository.cs**

public interface IInventoryRepository

{

Task<IEnumerable<Inventory>> GetAll();

Task<Inventory> GetById(int id);

Task<IEnumerable<Inventory>> GetById(Expression<Func<Inventory,

bool>> predicate);

Task<Inventory> AddAsync(Inventory entity);

Inventory Update(Inventory entity);

void Remove(Inventory entity);

Task RemoveById(int id);

int SaveChanges();

Task<int> SaveChangesAsync();

Task<Inventory> SingleOrDefaultAsync(Expression<Func<Inventory, bool>>

predicate);

Task<Inventory> FirstOrDefaultAsync(Expression<Func<Inventory, bool>>

predicate);

void AddRange(IEnumerable<Inventory> entities);

Task AddRangeAsync(IEnumerable<Inventory> entities);

void UpdateRange(IEnumerable<Inventory> entities);

void RemoveRange(IEnumerable<Inventory> entities);

}

10) Create **Repositories** folder into **CAW.Data** 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);

}

}

**InventoryRepository.cs**

public class InventoryRepository : IInventoryRepository

{

IRepository<Inventory, int> \_repo;

public InventoryRepository(IRepository<Inventory, int> repo)

{

\_repo = repo;

}

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

{

return await \_repo.GetAll();

}

public async Task<Inventory> GetById(int id)

{

return await \_repo.GetById(id);

}

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

{

return await \_repo.GetById(predicate);

}

public async Task<Inventory> AddAsync(Inventory entity)

{

return await \_repo.AddAsync(entity);

}

public Inventory Update(Inventory entity)

{

return \_repo.Update(entity);

}

public void Remove(Inventory entity)

{

\_repo.Remove(entity);

}

public async Task RemoveById(int id)

{

await \_repo.RemoveById(id);

}

public int SaveChanges()

{

return \_repo.SaveChanges();

}

public async Task<int> SaveChangesAsync()

{

return await \_repo.SaveChangesAsync();

}

public async Task<Inventory> FirstOrDefaultAsync(Expression<Func<Inventory,

bool>> predicate)

{

return await \_repo.FirstOrDefaultAsync(predicate);

}

public async Task<Inventory> SingleOrDefaultAsync(Expression<Func<Inventory,

bool>> predicate)

{

return await \_repo.SingleOrDefaultAsync(predicate);

}

public void AddRange(IEnumerable<Inventory> entities)

{

\_repo.AddRange(entities);

}

public async Task AddRangeAsync(IEnumerable<Inventory> entities)

{

await \_repo.AddRangeAsync(entities);

}

public void RemoveRange(IEnumerable<Inventory> entities)

{

\_repo.RemoveRange(entities);

}

public void UpdateRange(IEnumerable<Inventory> entities)

{

\_repo.UpdateRange(entities);

}

}

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

12) Create **Models** Folder in **CAW.Application** library

13) Create **InventoryViewModel** folder in **Models** folder

14) Install **AutoMapper.Extensions.Microsoft.DependencyInjection** into

**CAW.Application** Class libraryand **CAW.API** project

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

public void ConfigureServices(IServiceCollection services)

{

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

}

16) Add **InventoryModel.cs** and **InventoryMapping.cs** class into **CAW.Application ->**

**Models -> InventoryViewModel** folder

**InventoryModel.cs**

public class InventoryModel

{

public int InventoryId { get; set; }

public string Name { get; set; }

public int? Price { get; set; }

public int? Quantity { get; set; }

public DateTime? CreatedDate { get; set; }

}

**Inventorymapping.cs**

public class InventoryMapping : Profile

{

public InventoryMapping()

{

CreateMap<InventoryModel, Inventory>()

.ForMember(dest => dest.InventoryId, opt => opt.MapFrom(src =>

src.InventoryId))

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

src.Name))

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

.ForMember(dest => dest.Quantity, opt => opt.MapFrom(src =>

src.Quantity))

.ForMember(dest => dest.CreatedDate, opt => opt.MapFrom(src =>

src.CreatedDate));

CreateMap<Inventory, InventoryModel>();

}

}

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

**IInventoryService.cs**

public interface IInventoryService

{

Task<IEnumerable<InventoryModel>> GetAllInventory();

Task<InventoryModel> GetInventoryById(int id);

Task<IEnumerable<InventoryModel>> GetInventoryById(Expression<Func

<Inventory, bool>> predicate);

Task<int> AddInventoryAsync(InventoryModel entity);

Task<int> UpdateInventory(InventoryModel entity);

Task<int> RemoveInventory(InventoryModel entity);

Task<int> RemoveInventoryById(int id);

Task<InventoryModel> SingleOrDefaultAsync(Expression<Func<Inventory, bool>>

predicate);

Task<InventoryModel> FirstOrDefaultAsync(Expression<Func<Inventory, bool>>

predicate);

Task<int> AddRange(IEnumerable<InventoryModel> entities);

Task<int> AddRangeAsync(IEnumerable<InventoryModel> entities);

Task<int> UpdateRange(IEnumerable<InventoryModel> entities);

Task<int> RemoveRange(IEnumerable<InventoryModel> entities);

}

18) Create **Services** folder into **CAW.Application** class library and add below service into this folder-

**InventoryService.cs**

public class InventoryService : IInventoryService

{

private readonly IInventoryRepository \_inventoryRepository;

private readonly IMapper \_mapper;

public InventoryService(IInventoryRepository inventoryRepository,

IMapper mapper)

{

\_inventoryRepository = inventoryRepository;

\_mapper = mapper;

}

public async Task<IEnumerable<InventoryModel>> GetAllInventory()

{

try

{

var inventory = await \_inventoryRepository.GetAll();

var result = \_mapper.Map<IEnumerable<InventoryModel>>(inventory);

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<InventoryModel> GetInventoryById(int id)

{

try

{

var inventory = await \_inventoryRepository.GetById(id);

return \_mapper.Map<InventoryModel>(inventory);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<IEnumerable<InventoryModel>> GetInventoryById(Expression

<Func<Inventory, bool>> predicate)

{

try

{

var inventory = await \_inventoryRepository.GetById(predicate);

return \_mapper.Map<IEnumerable<InventoryModel>>(inventory);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> AddInventoryAsync(InventoryModel entity)

{

try

{

var inventory = \_mapper.Map<Inventory>(entity);

var res = await \_inventoryRepository.AddAsync(inventory);

int result = await \_inventoryRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> UpdateInventory(InventoryModel entity)

{

try

{

var inventory = \_mapper.Map<Inventory>(entity);

var res = \_inventoryRepository.Update(inventory);

int result = await \_inventoryRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> RemoveInventory(InventoryModel entity)

{

try

{

var inventory = \_mapper.Map<Inventory>(entity);

\_inventoryRepository.Remove(inventory);

int result = await \_inventoryRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> RemoveInventoryById(int id)

{

try

{

await \_inventoryRepository.RemoveById(id);

int result = await \_inventoryRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<InventoryModel> SingleOrDefaultAsync(Expression

<Func<Inventory, bool>> predicate)

{

try

{

var inventory = await \_inventoryRepository.SingleOrDefaultAsync(predicate);

return \_mapper.Map<InventoryModel>(inventory);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<InventoryModel> FirstOrDefaultAsync(Expression

<Func<Inventory, bool>> predicate)

{

try

{

var inventory = await \_inventoryRepository.FirstOrDefaultAsync(predicate);

return \_mapper.Map<InventoryModel>(inventory);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> AddRange(IEnumerable<InventoryModel> entities)

{

try

{

var inventories = \_mapper.Map<IEnumerable<Inventory>>(entities);

\_inventoryRepository.AddRange(inventories);

int result = await \_inventoryRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> AddRangeAsync(IEnumerable<InventoryModel> entities)

{

try

{

var inventories = \_mapper.Map<IEnumerable<Inventory>>(entities);

await \_inventoryRepository.AddRangeAsync(inventories);

int result = await \_inventoryRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> UpdateRange(IEnumerable<InventoryModel> entities)

{

try

{

var inventories = \_mapper.Map<IEnumerable<Inventory>>(entities);

\_inventoryRepository.UpdateRange(inventories);

int result = await \_inventoryRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<int> RemoveRange(IEnumerable<InventoryModel> entities)

{

try

{

var inventories = this.\_mapper.Map<IEnumerable<Inventory>>(entities);

this.\_inventoryRepository.RemoveRange(inventories);

int result = await \_inventoryRepository.SaveChangesAsync();

return result;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

}

19) Create .Net core api project with name **CAW.API** and Add **InventoryController.cs**

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

[ApiController]

public class InventoryController : ControllerBase

{

private readonly IInventoryService \_service;

public InventoryController(IInventoryService service)

{

\_service = service;

}

[HttpGet("GetAllInventory")]

public async Task<IActionResult> GetAll()

{

var result = await \_service.GetAllInventory();

return Ok(result);

}

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

public async Task<IActionResult> GetById(int id)

{

var result = await \_service.GetInventoryById(id);

if (result == null)

{

return NoContent();

}

return Ok(result);

}

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

public async Task<IActionResult> GetByIdUsingPredicate(int id)

{

var result = await \_service.GetInventoryById(x => x.InventoryId == id);

return Ok(result);

}

[HttpPost("AddInventoryAsync")]

public async Task<IActionResult> AddAsync(InventoryModel inventory)

{

if (inventory == null)

{

return BadRequest();

}

var result = await \_service.AddInventoryAsync(inventory);

return Ok(result);

}

[HttpPut("UpdateInventory")]

public async Task<IActionResult> Update(InventoryModel inventory)

{

if (inventory == null)

{

return BadRequest();

}

var result = await \_service.UpdateInventory(inventory);

return Ok(result);

}

[HttpDelete("RemoveInventory")]

public async Task<IActionResult> RemoveEmployee(InventoryModel inventory)

{

if (inventory == null)

{

return BadRequest();

}

int result = await \_service.RemoveInventory(inventory);

return Ok(result);

}

[HttpDelete("RemoveInventoryById")]

public async Task<IActionResult> RemoveInventoryById(int id)

{

if (id <= 0)

{

return BadRequest();

}

int result = await \_service.RemoveInventoryById(id);

return Ok(result);

}

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

public async Task<IActionResult> GetSingleOrDefault(int id)

{

var result = await \_service.SingleOrDefaultAsync(x => x.InventoryId == id);

return Ok(result);

}

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

public async Task<IActionResult> GetFirstOrDefault(int id)

{

var result = await \_service.FirstOrDefaultAsync(x => x.InventoryId == id);

return Ok(result);

}

[HttpPost("AddInventoryRange")]

public async Task<IActionResult> AddRange(IEnumerable<InventoryModel>

inventories)

{

if (inventories == null)

{

return BadRequest();

}

int result = await \_service.AddRange(inventories);

return Ok(result);

}

[HttpPost("AddInventoryRangeAsync")]

public async Task<IActionResult> AddRangeAsync(IEnumerable<InventoryModel>

inventories)

{

if (inventories == null)

{

return BadRequest();

}

int result = await \_service.AddRangeAsync(inventories);

return Ok(result);

}

[HttpPut("UpdateInventoryRange")]

public async Task<IActionResult> UpdateRange(IEnumerable<InventoryModel>

inventories)

{

if (inventories == null)

{

return BadRequest();

}

int result = await \_service.UpdateRange(inventories);

return Ok(result);

}

[HttpPut("RemoveInventoryRange")]

public async Task<IActionResult> RemoveRange(IEnumerable<InventoryModel>

inventories)

{

if (inventories == null)

{

return BadRequest();

}

int result = await \_service.RemoveRange(inventories);

return Ok(result);

}

}

**20) add code into startup.cs file**

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

services.AddDbContext<InventoryDBContext>(item => item.UseSqlServer

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

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

services.AddScoped<IUnitOfWork, UnitOfWork>();

services.AddScoped(typeof(IRepository<,>), typeof(Repository<,>));

services.AddScoped<IInventoryRepository, InventoryRepository>();

services.AddScoped<IInventoryService, InventoryService>();

services.AddSwaggerGen();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Version = "v1",

Title = "Implement Swagger UI",

Description = "A simple example to Implement Swagger UI",

});

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseSwagger();

app.UseSwaggerUI(c => {

c.SwaggerEndpoint("/swagger/v1/swagger.json", "Showing API V1");

});

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

**XUnit Testing Using InMemory database**

1) Create **XUnit** project into **CAW** with name **CAW.UnitTest.Core**

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

3) Create **AutoMapping.cs** class into **CAW.UnitTest.Core**

public class AutoMapping

{

public IMapper GetMapper(Profile profile)

{

var mockMapper = new MapperConfiguration(cfg =>

{

cfg.AddProfile(profile);

});

var mapper = mockMapper.CreateMapper();

return mapper;

}

}

4) Create **InventoryDBDataFixture.cs** class into **CAW.UnitTest.Core**

public class InventoryDBDataFixture : IDisposable

{

public InventoryDBContext inventoryDBContext { get; private set; }

public DbContextOptions<InventoryDBContext> inventoryDBContextOptions {

get; private set; }

private const string Database = "InventoryDBInMemoryDatabase";

public InventoryDBDataFixture()

{

inventoryDBContextOptions = new DbContextOptionsBuilder<InventoryDBContext>()

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

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

.UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking)

.EnableSensitiveDataLogging(true)

.Options;

inventoryDBContext = new InventoryDBContext(inventoryDBContextOptions);

inventoryDBContext.Database.EnsureDeleted();

inventoryDBContext.Database.EnsureCreated();

}

public void Dispose()

{

inventoryDBContext.Database.EnsureDeleted();

inventoryDBContext.Dispose();

}

}

5) Create **MoqData** Folder into **CAW.UnitTest.Core**

6) Add **InventoyMoq.cs** class into **CAW.UnitTest.Core -> MoqData**

public class InventoryMoq : IClassFixture<InventoryDBDataFixture>

{

InventoryDBDataFixture fixture;

public InventoryMoq(InventoryDBDataFixture fixture)

{

this.fixture = fixture;

}

public void MoqData(Inventory entity)

{

using (var qssContext = new InventoryDBContext(fixture.inventoryDBContextOptions))

{

qssContext.Inventory.Add(entity);

qssContext.SaveChanges();

}

}

public void MoqDataList(IEnumerable<Inventory> entityList)

{

using (var qssContext = new InventoryDBContext(fixture.inventoryDBContextOptions))

{

qssContext.Inventory.AddRangeAsync(entityList);

qssContext.SaveChanges();

}

}

}

**XUnit testing for CAW.API**

1) Create **XUnit** project with name **CAW.API.UnitTest** into **CAW**

2) Create **Controller** folder and add **InventoryControllerTest.cs** file into it and add below code

public class InventoryControllerTest

{

[Fact]

public async Task GetAllInventory\_ReturnOkResult()

{

// Arrange

var mockRepo = new Mock<IInventoryService>();

mockRepo.Setup(repo => repo.GetAllInventory())

.ReturnsAsync(GetInventoryList());

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.GetAllInventory();

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task GetInventoryById\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IInventoryService>();

mockRepo.Setup(repo => repo.GetInventoryById(id))

.ReturnsAsync(GetInventoryById(id));

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.GetInventoryById(id);

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task GetInventoryByIdUsingPredicate\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IInventoryService>();

mockRepo.Setup(repo => repo.GetInventoryById(x=>x.InventoryId == id))

.ReturnsAsync(GetInventoryList());

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.GetInventoryByIdUsingPredicate(id);

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task AddInventoryAsync\_ReturnOkResult()

{

// Arrange

var mockRepo = new Mock<IInventoryService>();

var addInventory = AddInventoryModel();

mockRepo.Setup(repo => repo.AddInventoryAsync(addInventory))

.ReturnsAsync(1);

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.AddInventoryAsync(addInventory);

//// 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<IInventoryService>();

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.AddInventoryAsync(null);

//// Assert

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

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task UpdateInventory\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IInventoryService>();

var updateInventory = GetInventoryById(id);

mockRepo.Setup(repo => repo.UpdateInventory(updateInventory))

.ReturnsAsync(1);

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.UpdateInventory(updateInventory);

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task UpdateInventory\_ReturnBadRequest()

{

// Arrange

var mockRepo = new Mock<IInventoryService>();

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.UpdateInventory(null);

//// Assert

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

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task RemoveInventory\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IInventoryService>();

var removeInventory = GetInventoryById(id);

mockRepo.Setup(repo => repo.RemoveInventory(removeInventory))

.ReturnsAsync(id);

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.RemoveInventory(removeInventory);

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task RemoveInventory\_ReturnBadRequest()

{

// Arrange

var mockRepo = new Mock<IInventoryService>();

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.RemoveInventory(null);

//// Assert

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

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task RemoveInventoryById\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IInventoryService>();

mockRepo.Setup(repo => repo.RemoveInventoryById(id))

.ReturnsAsync(id);

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.RemoveInventoryById(id);

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task RemoveInventoryById\_ReturnBadRequest()

{

// Arrange

var mockRepo = new Mock<IInventoryService>();

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.RemoveInventoryById(0);

//// Assert

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

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task GetInventorySingleOrDefaultAsync\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IInventoryService>();

mockRepo.Setup(repo => repo.SingleOrDefaultAsync(x => x.InventoryId == id))

.ReturnsAsync(GetInventoryById(id));

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.GetInventorySingleOrDefaultAsync(id);

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task GetInventoryFirstOrDefaultAsync\_ReturnOkResult()

{

// Arrange

int id = 1;

var mockRepo = new Mock<IInventoryService>();

mockRepo.Setup(repo => repo.FirstOrDefaultAsync(x => x.InventoryId == id))

.ReturnsAsync(GetInventoryById(id));

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.GetInventoryFirstOrDefaultAsync(id);

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task AddInventoryRange\_ReturnOkResult()

{

// Arrange

int res = 1;

var mockRepo = new Mock<IInventoryService>();

var inventoryList = GetInventoryList();

mockRepo.Setup(repo => repo.AddRange(inventoryList))

.ReturnsAsync(res);

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.AddInventoryRange(inventoryList);

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task AddInventoryRange\_ReturnBadRequest()

{

// Arrange

var mockRepo = new Mock<IInventoryService>();

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.AddInventoryRange(null);

//// Assert

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

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task AddInventoryRangeAsync\_ReturnOkResult()

{

// Arrange

int res = 1;

var mockRepo = new Mock<IInventoryService>();

var inventoryList = GetInventoryList();

mockRepo.Setup(repo => repo.AddRangeAsync(inventoryList))

.ReturnsAsync(res);

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.AddInventoryRangeAsync(inventoryList);

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task AddInventoryRangeAsync\_ReturnBadRequest()

{

// Arrange

var mockRepo = new Mock<IInventoryService>();

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.AddInventoryRangeAsync(null);

//// Assert

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

.StatusCode);

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task UpdateInventoryRange\_ReturnOkResult()

{

// Arrange

int res = 1;

var mockRepo = new Mock<IInventoryService>();

var inventoryList = GetInventoryList();

mockRepo.Setup(repo => repo.UpdateRange(inventoryList))

.ReturnsAsync(res);

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.UpdateInventoryRange(inventoryList);

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task UpdateInventoryRange\_ReturnBadRequest()

{

// Arrange

var mockRepo = new Mock<IInventoryService>();

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.UpdateInventoryRange(null);

//// Assert

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

Assert.IsType<BadRequestResult>(result);

}

[Fact]

public async Task RemoveInventoryRange\_ReturnOkResult()

{

// Arrange

int res = 1;

var mockRepo = new Mock<IInventoryService>();

var inventoryList = GetInventoryList();

mockRepo.Setup(repo => repo.RemoveRange(inventoryList))

.ReturnsAsync(res);

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.RemoveInventoryRange(inventoryList);

//// Assert

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

Assert.IsType<OkObjectResult>(result);

}

[Fact]

public async Task RemoveInventoryRange\_ReturnBadRequest()

{

// Arrange

var mockRepo = new Mock<IInventoryService>();

var controller = new InventoryController(mockRepo.Object);

//// Act

var result = await controller.RemoveInventoryRange(null);

//// Assert

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

Assert.IsType<BadRequestResult>(result);

}

private IEnumerable<InventoryModel> GetInventoryList()

{

List<InventoryModel> list = new List<InventoryModel>()

{

new InventoryModel {

InventoryId=1,

Name="Test1",

Price=10,

Quantity=10,

CreatedDate=DateTime.Now

},

new InventoryModel {

InventoryId=2,

Name="Test2",

Price=20,

Quantity=20,

CreatedDate=DateTime.Now

},

};

return list;

}

private InventoryModel GetInventoryById(int id)

{

var inventory = GetInventoryList().FirstOrDefault(x=>x.InventoryId == id);

return inventory;

}

private InventoryModel AddInventoryModel()

{

InventoryModel obj = new InventoryModel()

{

InventoryId = 1,

Name = " Test 1",

Price = 10,

Quantity = 10,

CreatedDate = DateTime.Now

};

return obj;

}

}

**XUnit testing for CAW.Application**

1) Create **XUnit** project with name **CAW.Application.UnitTest** into **CAW**

2) Create **Services** folder and add **InventoryServicesTest.cs** file into it and add below code

public class InventoryRepositoryTest : IClassFixture<InventoryDBDataFixture>

{

InventoryDBDataFixture fixture;

public InventoryRepositoryTest(InventoryDBDataFixture fixture)

{

this.fixture = fixture;

}

[Fact]

public async Task GetAll\_ShouldReturnList()

{

//Arrange

var repository = new Mock<IRepository<Inventory, int>>();

repository.Setup(x => x.GetAll()).ReturnsAsync(GetInventoryList());

var inventoryRepo = new InventoryRepository(repository.Object);

//act

var result = inventoryRepo.GetAll();

//Assert

Assert.IsAssignableFrom<IEnumerable<Inventory>>(result.Result);

}

[Fact]

public async Task GetById\_ShouldReturnInventory()

{

//Arrange

int id = 1;

var repository = new Mock<IRepository<Inventory, int>>();

repository.Setup(x => x.GetById(id)).ReturnsAsync(GetInventoryById(id));

var inventoryRepo = new InventoryRepository(repository.Object);

//act

var result = inventoryRepo.GetById(id);

//Assert

Assert.IsAssignableFrom<Inventory>(result.Result);

}

[Fact]

public async Task GetByIdUsingPredicate\_ShouldReturnInventory()

{

//Arrange

int id = 1;

var repository = new Mock<IRepository<Inventory, int>>();

repository.Setup(x => x.GetById(id)).ReturnsAsync(GetInventoryById(id));

var inventoryRepo = new InventoryRepository(repository.Object);

//act

var result = inventoryRepo.GetById(x => x.InventoryId == id);

//Assert

Assert.IsAssignableFrom<IEnumerable<Inventory>>(result.Result);

}

[Fact]

public async Task AddAsync\_ShouldSaveRecord()

{

//Arrange

var addInventory = GetInventoryById(1);

addInventory.InventoryId = 0;

IUnitOfWork unitOfWork = new UnitOfWork(fixture.inventoryDBContext);

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

IInventoryRepository inventoryRepo = new InventoryRepository(repository);

//act

var result = await inventoryRepo.AddAsync(addInventory);

//Assert

Assert.IsAssignableFrom<Inventory>(result);

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

}

[Fact]

public async Task Update\_ShouldUpdateRecord()

{

var updateInventory = GetInventoryById(1);

updateInventory.Name = "Test11";

InventoryMoq moq = new InventoryMoq(fixture);

moq.MoqData(updateInventory);

IUnitOfWork unitOfWork = new UnitOfWork(fixture.inventoryDBContext);

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

IInventoryRepository inventoryRepo = new InventoryRepository(repository);

//Act

var result = inventoryRepo.Update(updateInventory);

//Assert

Assert.IsAssignableFrom<Inventory>(result);

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

}

[Fact]

public async Task Remove\_ShouldRemoveRecord()

{

//Arrange

var removeInventory = GetInventoryById(1);

removeInventory.InventoryId = 2;

removeInventory.Name = "Test22";

InventoryMoq moq = new InventoryMoq(fixture);

moq.MoqData(removeInventory);

IUnitOfWork unitOfWork = new UnitOfWork(fixture.inventoryDBContext);

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

IInventoryRepository inventoryRepository = new InventoryRepository(repository);

//Act

inventoryRepository.Remove(removeInventory);

var res = await inventoryRepository.SaveChangesAsync();

var list = inventoryRepository.GetAll();

Assert.Equal(list.Result.Count(), 5);

Assert.Equal(res, 1);

}

[Fact]

public async Task RemoveById\_ShouldRemoveRecord()

{

//Arrange

int id = 3;

var removeInventory = GetInventoryById(1);

removeInventory.Name = "Test33";

removeInventory.InventoryId = 3;

InventoryMoq moq = new InventoryMoq(fixture);

moq.MoqData(removeInventory);

IUnitOfWork unitOfWork = new UnitOfWork(fixture.inventoryDBContext);

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

IInventoryRepository inventoryRepository = new InventoryRepository(repository);

//Act

await inventoryRepository.RemoveById(id);

var res = await inventoryRepository.SaveChangesAsync();

var list = inventoryRepository.GetAll();

Assert.Equal(list.Result.Count(), 3);

Assert.Equal(res, 2);

}

[Fact]

public async Task SaveChanges\_ShouldSaveRecord()

{

//Arrange

var repository = new Mock<IRepository<Inventory, int>>();

repository.Setup(x => x.SaveChanges()).Returns(1);

var inventoryRepo = new InventoryRepository(repository.Object);

//act

int result = inventoryRepo.SaveChanges();

//Assert

Assert.Equal(result, 1);

}

[Fact]

public async Task SaveChangesAsync\_ShouldSaveRecord()

{

//Arrange

var repository = new Mock<IRepository<Inventory, int>>();

repository.Setup(x => x.SaveChangesAsync()).ReturnsAsync(1);

var inventoryRepo = new InventoryRepository(repository.Object);

//act

var result = inventoryRepo.SaveChangesAsync();

//Assert

Assert.Equal(result.Result, 1);

}

[Fact]

public async Task FirstOrDefaultAsync\_ShouldReturnInventory()

{

//Arrange

int id = 1;

var repository = new Mock<IRepository<Inventory, int>>();

repository.Setup(x => x.FirstOrDefaultAsync(x => x.InventoryId == id)).ReturnsAsync(GetInventoryById(id));

var inventoryRepo = new InventoryRepository(repository.Object);

//act

var result = inventoryRepo.FirstOrDefaultAsync(x => x.InventoryId == id);

//Assert

Assert.IsAssignableFrom<Inventory>(result.Result);

}

[Fact]

public async Task SingleOrDefaultAsync\_ShouldReturnInventory()

{

//Arrange

int id = 1;

var repository = new Mock<IRepository<Inventory, int>>();

repository.Setup(x => x.SingleOrDefaultAsync(x => x.InventoryId == id)).ReturnsAsync(GetInventoryById(id));

var inventoryRepo = new InventoryRepository(repository.Object);

//act

var result = inventoryRepo.SingleOrDefaultAsync(x => x.InventoryId == id);

//Assert

Assert.IsAssignableFrom<Inventory>(result.Result);

}

[Fact]

public async Task AddRange\_ShouldSaveRecord()

{

//Arrange

var addInventoryRangeList = AddRangeInventoryList();

IUnitOfWork unitOfWork = new UnitOfWork(fixture.inventoryDBContext);

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

IInventoryRepository inventoryRepository = new InventoryRepository(repository);

//Act

inventoryRepository.AddRange(addInventoryRangeList);

var res = await inventoryRepository.SaveChangesAsync();

var list = inventoryRepository.GetAll();

//Assert

Assert.Equal(list.Result.Count(), 7);

Assert.Equal(res, 2);

}

[Fact]

public async Task AddRangeAsync\_ShouldSaveRecord()

{

//Arrange

var addInventoryRangeAsynList = AddRangeAsyncInventoryList();

IUnitOfWork unitOfWork = new UnitOfWork(fixture.inventoryDBContext);

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

IInventoryRepository inventoryRepository = new InventoryRepository(repository);

//Act

await inventoryRepository.AddRangeAsync(addInventoryRangeAsynList);

var res = await inventoryRepository.SaveChangesAsync();

var list = inventoryRepository.GetAll();

//Assert

Assert.Equal(list.Result.Count(), 5);

Assert.Equal(res, 2);

}

[Fact]

public async Task RemoveRange\_ShouldSaveRecord()

{

//Arrange

var removeRangeList = GetRemoveRangeInventoryList();

InventoryMoq moq = new InventoryMoq(fixture);

moq.MoqDataList(removeRangeList);

IUnitOfWork unitOfWork = new UnitOfWork(fixture.inventoryDBContext);

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

IInventoryRepository inventoryRepository = new InventoryRepository(repository);

//Act

inventoryRepository.RemoveRange(removeRangeList);

var res = await inventoryRepository.SaveChangesAsync();

var list = inventoryRepository.GetAll();

//Assert

Assert.Equal(list.Result.Count(), 2);

Assert.Equal(res,2);

}

[Fact]

public async Task UpdateRange\_ShouldSaveRecord()

{

//Arrange

var updateInventoryList = GetUpdateRangeInventoryList().ToList();

InventoryMoq moq = new InventoryMoq(fixture);

moq.MoqDataList(updateInventoryList);

updateInventoryList[0].Name = updateInventoryList[0].Name + " Test";

updateInventoryList[1].Name = updateInventoryList[1].Name + " Test";

IUnitOfWork unitOfWork = new UnitOfWork(fixture.inventoryDBContext);

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

IInventoryRepository inventoryRepository = new InventoryRepository(repository);

//Act

inventoryRepository.UpdateRange(updateInventoryList);

var res = await inventoryRepository.SaveChangesAsync();

var list = inventoryRepository.GetAll();

//Assert

Assert.Equal(list.Result.Count(), 2);

Assert.Equal(res, 2);

}

private IEnumerable<Inventory> GetInventoryList()

{

List<Inventory> list = new List<Inventory>()

{

new Inventory {

InventoryId=1,

Name="Test1",

Price=10,

Quantity=10,

CreatedDate=DateTime.Now

},

new Inventory {

InventoryId=2,

Name="Test2",

Price=20,

Quantity=20,

CreatedDate=DateTime.Now

},

};

return list;

}

private Inventory GetInventoryById(int id)

{

var inventory = GetInventoryList().FirstOrDefault(x => x.InventoryId == id);

return inventory;

}

private IEnumerable<Inventory> AddRangeInventoryList()

{

List<Inventory> list = new List<Inventory>()

{

new Inventory {

InventoryId=4,

Name="Test44",

Price=40,

Quantity=40,

CreatedDate=DateTime.Now

},

new Inventory {

InventoryId=5,

Name="Test55",

Price=50,

Quantity=50,

CreatedDate=DateTime.Now

},

};

return list;

}

private IEnumerable<Inventory> AddRangeAsyncInventoryList()

{

List<Inventory> list = new List<Inventory>()

{

new Inventory {

InventoryId=6,

Name="Test66",

Price=40,

Quantity=40,

CreatedDate=DateTime.Now

},

new Inventory {

InventoryId=7,

Name="Test77",

Price=50,

Quantity=50,

CreatedDate=DateTime.Now

},

};

return list;

}

private IEnumerable<Inventory> GetRemoveRangeInventoryList()

{

List<Inventory> list = new List<Inventory>()

{

new Inventory {

InventoryId=8,

Name="Test88",

Price=80,

Quantity=80,

CreatedDate=DateTime.Now

},

new Inventory {

InventoryId=9,

Name="Test99",

Price=90,

Quantity=90,

CreatedDate=DateTime.Now

}

};

return list;

}

private IEnumerable<Inventory> GetUpdateRangeInventoryList()

{

List<Inventory> list = new List<Inventory>()

{

new Inventory {

InventoryId=10,

Name="Test10",

Price=100,

Quantity=100,

CreatedDate=DateTime.Now

},

new Inventory {

InventoryId=11,

Name="Test11",

Price=110,

Quantity=110,

CreatedDate=DateTime.Now

}

};

return list;

}

}