Repository Pattern in ASP.NET Core REST API

In this video, we will learn

1. What is Repository Pattern
2. Benefits of Repository Pattern
3. An example that uses repository pattern to store and retrieve data from sql server database

What is Repository Pattern

Repository Pattern is an abstraction of the Data Access Layer. It hides the details of how exactly the data is saved or retrieved from the underlying data source. The details of how the data is stored and retrieved is in the respective repository. For example, you may have a repository that stores and retrieves data from an in-memory collection. You may have another repository that stores and retrieves data from a database like SQL Server. Yet another repository that stores and retrieves data from an XML file.

Repository Pattern Interface

The interface in the repository pattern specifies

* What operations (i.e methods) are supported by the repository
* The data required for each of the operations i.e the parameters that need to be passed to the method and the data the method returns
* The repository interface contains what it can do, but not, how it does, what it can do
* The implementation details are in the respective repository class that implements the repository Interface

public interface IEmployeeRepository

{

Task<IEnumerable<Employee>> GetEmployees();

Task<Employee> GetEmployee(int employeeId);

Task<Employee> AddEmployee(Employee employee);

Task<Employee> UpdateEmployee(Employee employee);

void DeleteEmployee(int employeeId);

}

Copy

IEmployeeRepository interface supports the following operations

* Get all the employees
* Get a single employee by id
* Add a new employee
* Updat an employee
* Delete an employee

The details of how these operations are implmented are in the repository class that implments this IEmployeeRepository interface.

Repository Pattern - SQL Server Implementation

The following EmployeeRepository class provides an implementation for IEmployeeRepository. This specific implementation stores and retrieves employees from a sql server database.

using EmployeeManagement.Models;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace EmployeeManagement.Api.Models

{

public class EmployeeRepository : IEmployeeRepository

{

private readonly AppDbContext appDbContext;

public EmployeeRepository(AppDbContext appDbContext)

{

this.appDbContext = appDbContext;

}

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

{

return await appDbContext.Employees.ToListAsync();

}

public async Task<Employee> GetEmployee(int employeeId)

{

return await appDbContext.Employees

.FirstOrDefaultAsync(e => e.EmployeeId == employeeId);

}

public async Task<Employee> AddEmployee(Employee employee)

{

var result = await appDbContext.Employees.AddAsync(employee);

await appDbContext.SaveChangesAsync();

return result.Entity;

}

public async Task<Employee> UpdateEmployee(Employee employee)

{

var result = await appDbContext.Employees

.FirstOrDefaultAsync(e => e.EmployeeId == employee.EmployeeId);

if (result != null)

{

result.FirstName = employee.FirstName;

result.LastName = employee.LastName;

result.Email = employee.Email;

result.DateOfBrith = employee.DateOfBrith;

result.Gender = employee.Gender;

result.DepartmentId = employee.DepartmentId;

result.PhotoPath = employee.PhotoPath;

await appDbContext.SaveChangesAsync();

return result;

}

return null;

}

public async void DeleteEmployee(int employeeId)

{

var result = await appDbContext.Employees

.FirstOrDefaultAsync(e => e.EmployeeId == employeeId);

if (result != null)

{

appDbContext.Employees.Remove(result);

await appDbContext.SaveChangesAsync();

}

}

}

}

Copy

DepartmentRepository Interface

public interface IDepartmentRepository

{

IEnumerable<Department> GetDepartments();

Department GetDepartment(int departmentId);

}

Copy

DepartmentRepository Implementation

using EmployeeManagement.Models;

using System.Collections.Generic;

using System.Linq;

namespace EmployeeManagement.Api.Models

{

public class DepartmentRepository : IDepartmentRepository

{

private readonly AppDbContext appDbContext;

public DepartmentRepository(AppDbContext appDbContext)

{

this.appDbContext = appDbContext;

}

public Department GetDepartment(int departmentId)

{

return appDbContext.Departments

.FirstOrDefault(d => d.DepartmentId == departmentId);

}

public IEnumerable<Department> GetDepartments()

{

return appDbContext.Departments;

}

}

}

Copy

Which implementation to use

With the following 2 lines of code, ASP.NET Core provides an instance of EmployeeRepository class when an instance of IEmployeeRepository is requested. Similarly an instance of DepartmentRepository class is provided when an instance of IDepartmentRepository is requested.

public void ConfigureServices(IServiceCollection services)

{

// Rest of the code

services.AddScoped<IDepartmentRepository, DepartmentRepository>();

services.AddScoped<IEmployeeRepository, EmployeeRepository>();

}

Copy

We are using AddScoped() method because we want the instance to be alive and available for the entire scope of the given HTTP request. For another new HTTP request, a new instance of EmployeeRepository class will be provided and it will be available throughout the entire scope of that HTTP request.

We discussed the difference between AddSingleton(), AddScoped() and AddTransient() methods in detail in [Part 44](https://youtu.be/v6Nr7Zman_Y) of [ASP.NET Core Tutorial](https://www.pragimtech.com/courses/asp-net-core-mvc-tutorial-for-beginners/).

Throughout our entire application, in all the places where IEmployeeRepository is injected an instance of EmployeeRepository is provided. If you want your application to use a different implementation instead, all you need to change is the following one line of code.

services.AddScoped<IEmployeeRepository, EmployeeRepository>();

Copy

Benefits of Repository Pattern

* The code is cleaner, and easier to reuse and maintain.
* Enables us to create loosely coupled systems. For example, if we want our application to work with oracle instead of sql server, implement an OracleRepository that knows how to read and write to Oracle database and register OracleRepository with the dependency injection system.
* In an unit testing project, it is easy to replace a real repository with a fake implementation for testing.

Slides











