**we will discuss**

* What is Repository Pattern
* Benefits of Repository Pattern
* An example that uses repository pattern to store and retrieve data from sql server database using entity framework core

**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.

public class Employee

{

    public int Id { get; set; }

    public string Name { get; set; }

    public string Email { get; set; }

    public Dept? Department { get; set; }

}

**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

{

    Employee GetEmployee(int Id);

    IEnumerable<Employee> GetAllEmployee();

    Employee Add(Employee employee);

    Employee Update(Employee employeeChanges);

    Employee Delete(int Id);

}

**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 - In-memory Implementation**  
  
The following MockEmployeeRepository class provides an implementation for IEmployeeRepository. This specific implementation stores and retrieves employees from an in-memory collection.

public class MockEmployeeRepository : IEmployeeRepository

{

    private List<Employee> \_employeeList;

    public MockEmployeeRepository()

    {

        \_employeeList = new List<Employee>()

        {

            new Employee() { Id = 1, Name = "Mary", Department = Dept.HR, Email = "mary@pragimtech.com" },

            new Employee() { Id = 2, Name = "John", Department = Dept.IT, Email = "john@pragimtech.com" },

            new Employee() { Id = 3, Name = "Sam", Department = Dept.IT, Email = "sam@pragimtech.com" },

        };

    }

    public Employee Add(Employee employee)

    {

        employee.Id = \_employeeList.Max(e => e.Id) + 1;

        \_employeeList.Add(employee);

        return employee;

    }

    public Employee Delete(int Id)

    {

        Employee employee = \_employeeList.FirstOrDefault(e => e.Id == Id);

        if (employee != null)

        {

            \_employeeList.Remove(employee);

        }

        return employee;

    }

    public IEnumerable<Employee> GetAllEmployee()

    {

        return \_employeeList;

    }

    public Employee GetEmployee(int Id)

    {

        return this.\_employeeList.FirstOrDefault(e => e.Id == Id);

    }

    public Employee Update(Employee employeeChanges)

    {

        Employee employee = \_employeeList.FirstOrDefault(e => e.Id == employeeChanges.Id);

        if (employee != null)

        {

            employee.Name = employeeChanges.Name;

            employee.Email = employeeChanges.Email;

            employee.Department = employeeChanges.Department;

        }

        return employee;

    }

}

**Repository Pattern - SQL Server Implementation**  
  
The following SQLEmployeeRepository class provides another implementation for IEmployeeRepository. This specific implementation stores and retrieves employees from a sql server database using entity framework core.

public class SQLEmployeeRepository : IEmployeeRepository

{

    private readonly AppDbContext context;

    public SQLEmployeeRepository(AppDbContext context)

    {

        this.context = context;

    }

    public Employee Add(Employee employee)

    {

        context.Employees.Add(employee);

        context.SaveChanges();

        return employee;

    }

    public Employee Delete(int Id)

    {

        Employee employee = context.Employees.Find(Id);

        if (employee != null)

        {

            context.Employees.Remove(employee);

            context.SaveChanges();

        }

        return employee;

    }

    public IEnumerable<Employee> GetAllEmployee()

    {

        return context.Employees;

    }

    public Employee GetEmployee(int Id)

    {

        return context.Employees.Find(Id);

    }

    public Employee Update(Employee employeeChanges)

    {

        var employee = context.Employees.Attach(employeeChanges);

        employee.State = Microsoft.EntityFrameworkCore.EntityState.Modified;

        context.SaveChanges();

        return employeeChanges;

    }

}

**Which implementation to use**  
  
Take a look at the following HomeController in our application. ASP.NET Core Dependency injection system injects an instance of IEmployeeRepository.

public class HomeController : Controller

{

    private IEmployeeRepository \_employeeRepository;

    public HomeController(IEmployeeRepository employeeRepository)

    {

        \_employeeRepository = employeeRepository;

    }

    public ViewResult Index()

    {

        var model = \_employeeRepository.GetAllEmployee();

        return View(model);

    }

    // Rest of the code

}

There are 2 implementations for IEmployeeRepository interface. How does the application know which implementation to use. The answer to this is in Startup class in Startup.cs file. With the following line of code, ASP.NET Core provides an instance of SQLEmployeeRepository class when an instance of IEmployeeRepository is requested. 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 SQLEmployeeRepository 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.youtube.com/playlist?list=PL6n9fhu94yhVkdrusLaQsfERmL_Jh4XmU).

public void ConfigureServices(IServiceCollection services)

{

    // Rest of the code

    services.AddScoped<IEmployeeRepository, SQLEmployeeRepository>();

}

Throughout our entire application, in all the places where IEmployeeRepository is injected an instance of SQLEmployeeRepository 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, MockEmployeeRepository>();

**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.