To implement a repository pattern (IRepo) in your console application with **ADO.NET**, we can create a generic interface (IRepo<T>) and then implement it for specific entities like User, Facility, Booking, etc. The repository will handle common CRUD operations such as Add, Update, Remove, and GetAll.

**Step 1: Define the IRepo<T> Interface**

This interface will define the basic operations for all entities.

using System;

using System.Collections.Generic;

public interface IRepo<T> where T : class

{

void Add(T entity);

void Update(T entity);

void Remove(Guid id);

List<T> GetAll();

T GetById(Guid id);

}

**Step 2: Implement the UserRepo for User Entity**

Here’s an implementation of the repository pattern for the User entity. The implementation uses **ADO.NET** for database interactions.

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

public class UserRepo : IRepo<User>

{

private readonly string \_connectionString;

public UserRepo(string connectionString)

{

\_connectionString = connectionString;

}

public void Add(User user)

{

string query = "INSERT INTO Users (Id, UserName, Email, PasswordHash, FullName, PhoneNumber, IsVoyager, DateCreated) " +

"VALUES (@Id, @UserName, @Email, @PasswordHash, @FullName, @PhoneNumber, @IsVoyager, @DateCreated)";

var parameters = new[]

{

new SqlParameter("@Id", Guid.NewGuid()), // Generate new GUID for the user

new SqlParameter("@UserName", user.UserName),

new SqlParameter("@Email", user.Email),

new SqlParameter("@PasswordHash", user.PasswordHash),

new SqlParameter("@FullName", user.FullName),

new SqlParameter("@PhoneNumber", user.PhoneNumber),

new SqlParameter("@IsVoyager", user.IsVoyager),

new SqlParameter("@DateCreated", DateTime.Now)

};

ExecuteCommand(query, parameters);

}

public void Update(User user)

{

string query = "UPDATE Users SET UserName = @UserName, Email = @Email, PasswordHash = @PasswordHash, " +

"FullName = @FullName, PhoneNumber = @PhoneNumber, IsVoyager = @IsVoyager WHERE Id = @Id";

var parameters = new[]

{

new SqlParameter("@UserName", user.UserName),

new SqlParameter("@Email", user.Email),

new SqlParameter("@PasswordHash", user.PasswordHash),

new SqlParameter("@FullName", user.FullName),

new SqlParameter("@PhoneNumber", user.PhoneNumber),

new SqlParameter("@IsVoyager", user.IsVoyager),

new SqlParameter("@Id", user.Id)

};

ExecuteCommand(query, parameters);

}

public void Remove(Guid id)

{

string query = "DELETE FROM Users WHERE Id = @Id";

var parameters = new[]

{

new SqlParameter("@Id", id)

};

ExecuteCommand(query, parameters);

}

public List<User> GetAll()

{

string query = "SELECT \* FROM Users";

var users = new List<User>();

using (var connection = new SqlConnection(\_connectionString))

{

using (var command = new SqlCommand(query, connection))

{

connection.Open();

using (var reader = command.ExecuteReader())

{

while (reader.Read())

{

var user = new User

{

Id = reader.GetGuid(0),

UserName = reader.GetString(1),

Email = reader.GetString(2),

PasswordHash = reader.GetString(3),

FullName = reader.GetString(4),

PhoneNumber = reader.GetString(5),

IsVoyager = reader.GetBoolean(6),

DateCreated = reader.GetDateTime(7)

};

users.Add(user);

}

}

}

}

return users;

}

public User GetById(Guid id)

{

string query = "SELECT \* FROM Users WHERE Id = @Id";

User user = null;

var parameters = new[]

{

new SqlParameter("@Id", id)

};

using (var connection = new SqlConnection(\_connectionString))

{

using (var command = new SqlCommand(query, connection))

{

command.Parameters.AddRange(parameters);

connection.Open();

using (var reader = command.ExecuteReader())

{

if (reader.Read())

{

user = new User

{

Id = reader.GetGuid(0),

UserName = reader.GetString(1),

Email = reader.GetString(2),

PasswordHash = reader.GetString(3),

FullName = reader.GetString(4),

PhoneNumber = reader.GetString(5),

IsVoyager = reader.GetBoolean(6),

DateCreated = reader.GetDateTime(7)

};

}

}

}

}

return user;

}

private void ExecuteCommand(string query, SqlParameter[] parameters)

{

using (var connection = new SqlConnection(\_connectionString))

{

using (var command = new SqlCommand(query, connection))

{

command.Parameters.AddRange(parameters);

connection.Open();

command.ExecuteNonQuery();

}

}

}

}

**Step 3: Define the User Entity**

public class User

{

public Guid Id { get; set; }

public string UserName { get; set; }

public string Email { get; set; }

public string PasswordHash { get; set; }

public string FullName { get; set; }

public string PhoneNumber { get; set; }

public bool IsVoyager { get; set; }

public DateTime DateCreated { get; set; }

}

**Step 4: Using the Repository in the Console App**

Now, in your **Console App**, you can interact with the UserRepo class to perform CRUD operations.

class Program

{

static void Main(string[] args)

{

string connectionString = "your\_connection\_string\_here";

var userRepo = new UserRepo(connectionString);

// Adding a new user

var newUser = new User

{

UserName = "johndoe",

Email = "johndoe@example.com",

PasswordHash = "hashed\_password", // Ideally, you should hash the password

FullName = "John Doe",

PhoneNumber = "123-456-7890",

IsVoyager = true

};

userRepo.Add(newUser);

// Get all users

var users = userRepo.GetAll();

foreach (var user in users)

{

Console.WriteLine($"User: {user.UserName}, Email: {user.Email}");

}

// Update a user

var userToUpdate = users[0];

userToUpdate.PhoneNumber = "987-654-3210";

userRepo.Update(userToUpdate);

// Get a specific user by ID

var userById = userRepo.GetById(userToUpdate.Id);

Console.WriteLine($"Updated User: {userById.FullName}, Phone: {userById.PhoneNumber}");

// Remove a user

userRepo.Remove(userToUpdate.Id);

}

}

**Step 5: Implementing Other Repositories**

For other entities like **Facility**, **Booking**, etc., you can implement repositories in a similar way, creating their respective **repo classes** (FacilityRepo, BookingRepo, etc.) by following the same structure for Add, Update, Remove, GetAll, and GetById operations.

**Conclusion**

This setup uses the **Repository Pattern** with a generic IRepo<T> interface and provides basic CRUD operations for your entities (like User). You can extend this to other entities such as **Facility**, **Booking**, etc. by following the same pattern.

This pattern improves maintainability and testability, especially in larger applications, and helps you separate the database logic from the business logic.

Let me know if you need more assistance or if you'd like me to walk through additional examples!