Stored Procedures Code

!.Update Price

CREATE PROCEDURE AddPriceChange

@Itemcode INT,

@Oldcost DECIMAL,

@Increase\_Decrease VARCHAR(255),

@PriceType VARCHAR(1),

@PriceUpdate VARCHAR(255),

@Newcost DECIMAL,

@Oldprice DECIMAL,

@Newprice DECIMAL

AS

BEGIN

INSERT INTO Pricechange (Itemcode, Oldcost, Increase\_Decrease, PriceType, PriceUpdate, Newcost, Oldprice, Newprice, CreateDate)

VALUES (@Itemcode, @Oldcost, @Increase\_Decrease, @PriceType, @PriceUpdate, @Newcost, @Oldprice, @Newprice, GETDATE());

IF (@PriceUpdate = 'Cost' OR @PriceUpdate = 'CostAndPrice')

BEGIN

UPDATE Item SET Cost = @Newcost WHERE Itemcode = @Itemcode;

END

IF (@PriceUpdate = 'Price' OR @PriceUpdate = 'CostAndPrice')

BEGIN

UPDATE Item SET Price = @Newprice WHERE Itemcode = @Itemcode;

END

END

2.Add ITem

CREATE PROCEDURE AddItem

@Itemcode INT,

@Barcode VARCHAR(255),

@Itemname VARCHAR(255),

@Cost DECIMAL,

@Price DECIMAL

AS

BEGIN

IF NOT EXISTS (SELECT 1 FROM Item WHERE Itemname = @Itemname OR Barcode = @Barcode)

BEGIN

INSERT INTO Item (Itemcode, Barcode, Itemname, Cost, Price, CreateDate, UpdateDate)

VALUES (@Itemcode, @Barcode, @Itemname, @Cost, @Price, GETDATE(), GETDATE());

END

ELSE

BEGIN

RAISERROR('Item with the same Itemname or Barcode already exists.', 16, 1);

END

END

3.List Items

CREATE PROCEDURE ListItems

AS

BEGIN

SELECT \* FROM Item;

END

4.List Price Change

CREATE PROCEDURE ListPriceChanges

AS

BEGIN

SELECT \* FROM Pricechange;

END

Queries for database generation

-- Create the Item table

CREATE TABLE Item (

Srno INT IDENTITY(1,1),

Itemcode INT PRIMARY KEY,

Barcode VARCHAR(255),

Itemname VARCHAR(255),

Cost DECIMAL,

Price DECIMAL,

CreateDate DATETIME,

UpdateDate DATETIME

);

-- Create the Pricechange table

CREATE TABLE Pricechange (

Srno INT IDENTITY(1,1) PRIMARY KEY,

Itemcode INT,

Oldcost DECIMAL,

Increase\_Decrease VARCHAR(255),

PriceType VARCHAR(1),

PriceUpdate VARCHAR(255),

Newcost DECIMAL,

Oldprice DECIMAL,

Newprice DECIMAL,

CreateDate DATETIME,

CONSTRAINT FK\_Pricechange\_Itemcode FOREIGN KEY (Itemcode) REFERENCES Item(Itemcode)

);

Code for Data Access Layer

I have made two repositories and corresponding interfaces and done dependency injection of both interfaces in my controller

Code in ItemRepository

using DataAccessLayer.Interfaces;

using System.Data.SqlClient;

using System.Data;

using Entities;

namespace DataAccessLayer.Repositories

{

public class ItemRepository:IItemInterface

{

private readonly string connectionString;

public ItemRepository(string connectionString)

{

this.connectionString = connectionString;

}

public List<ItemDTO> ListItems()

{

List<ItemDTO> items = new List<ItemDTO>();

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using SqlCommand command = new SqlCommand("ListItems", connection);

command.CommandType = CommandType.StoredProcedure;

using SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

ItemDTO item = new()

{

Srno = reader.GetInt32(reader.GetOrdinal("Srno")),

Itemcode = reader.GetInt32(reader.GetOrdinal("Itemcode")),

Barcode = reader.GetString(reader.GetOrdinal("Barcode")),

Itemname = reader.GetString(reader.GetOrdinal("Itemname")),

Cost = reader.GetDecimal(reader.GetOrdinal("Cost")),

Price = reader.GetDecimal(reader.GetOrdinal("Price")),

CreateDate = reader.GetDateTime(reader.GetOrdinal("CreateDate")),

UpdateDate = reader.GetDateTime(reader.GetOrdinal("UpdateDate"))

};

items.Add(item);

}

}

return items;

}

public string AddItem(ItemDTO itemDTO)

{

using SqlConnection connection = new SqlConnection(connectionString);

connection.Open();

using SqlCommand command = new SqlCommand("AddItem", connection);

command.CommandType = CommandType.StoredProcedure;

// Add the parameters

command.Parameters.AddWithValue("@Itemcode", itemDTO.Itemcode);

command.Parameters.AddWithValue("@Barcode", itemDTO.Barcode);

command.Parameters.AddWithValue("@Itemname", itemDTO.Itemname);

command.Parameters.AddWithValue("@Cost", itemDTO.Cost);

command.Parameters.AddWithValue("@Price", itemDTO.Price);

try

{

// Execute the stored procedure

command.ExecuteNonQuery();

return "Added Successfully";

}

catch (SqlException ex)

{

// Handle the exception

if (ex.Number == 50000) // Custom error number for "Item with the same Itemname or Barcode already exists"

{

return "Item with the same Itemname or Barcode already exists.";

}

else

{

return "Error Occured";

}

}

}

}

}

Code in PriceChangeRepository

using DataAccessLayer.Interfaces;

using Entities;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer.Repositories

{

public class PriceChangeRepository:IPriceChangeInterface

{

private readonly string connectionString;

public PriceChangeRepository(string connectionString)

{

this.connectionString = connectionString;

}

public string AddPriceChangeAndUpdateItem(PriceChangeDTO pricedata)

{

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (SqlCommand command = new SqlCommand("AddPriceChangeAndUpdateItem", connection))

{

command.CommandType = CommandType.StoredProcedure;

command.Parameters.AddWithValue("@Itemcode", pricedata.itemcode);

command.Parameters.AddWithValue("@Increase\_Decrease", pricedata.increaseDecrease);

command.Parameters.AddWithValue("@PriceType", pricedata.priceType);

command.Parameters.AddWithValue("@PriceUpdate", pricedata.priceUpate);

command.Parameters.AddWithValue("@Amount", pricedata.amount);

command.ExecuteNonQuery();

}

}

return "Price change and item update successful.";

}

catch (SqlException ex)

{

string errorMessage = "An error occurred while adding price change and updating the item.";

return errorMessage;

}

}

public List<PriceChangeDTO> ListPriceChanges()

{

List<PriceChangeDTO> priceChanges = new();

using (SqlConnection connection = new(connectionString))

{

connection.Open();

using SqlCommand command = new("ListPriceChanges", connection);

command.CommandType = CommandType.StoredProcedure;

using SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

PriceChangeDTO priceChange = new()

{

itemcode = (int)reader["Itemcode"],

oldcost = (decimal)reader["Oldcost"],

increaseDecrease = (string)reader["Increase\_Decrease"],

priceType = (string)reader["PriceType"],

priceUpate = (string)reader["PriceUpdate"],

newcost = (decimal)reader["Newcost"],

oldprice = (decimal)reader["Oldprice"],

newprice = (decimal)reader["Newprice"],

Created = (DateTime)reader["CreateDate"]

};

priceChanges.Add(priceChange);

}

}

return priceChanges;

}

}

}

Code in Interfaces :

using Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer.Interfaces

{

public interface IItemInterface

{

public List<ItemDTO> ListItems();

public string AddItem(ItemDTO itemDTO);

}

}

using Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer.Interfaces

{

public interface IPriceChangeInterface

{

public string AddPriceChangeAndUpdateItem(PriceChangeDTO pricedata);

public List<PriceChangeDTO> ListPriceChanges();

}

}

Then I have used these all methods in ItemController

Code in ItemController

using DataAccessLayer.Interfaces;

using Entities;

using Microsoft.AspNetCore.Mvc;

namespace PracticalTest.Controllers

{

public class ItemController : Controller

{

private readonly IItemInterface \_itemInterface;

private readonly IPriceChangeInterface \_priceChangeInterface;

public ItemController(IItemInterface itemInterface,IPriceChangeInterface priceChangeInterface)

{

\_itemInterface = itemInterface;

\_priceChangeInterface = priceChangeInterface;

}

public IActionResult GetItems()

{

List<ItemDTO> items = \_itemInterface.ListItems();

return View(items);

}

public IActionResult AddItem(ItemDTO itemDTO)

{

TempData["status"]=\_itemInterface.AddItem(itemDTO);

return View();

}

public IActionResult GetPriceChanges()

{

List<PriceChangeDTO> changeList=\_priceChangeInterface.ListPriceChanges();

return View(changeList);

}

public IActionResult AddPriceChange(PriceChangeDTO priceChangeDTO)

{

TempData["status"] = \_priceChangeInterface.AddPriceChangeAndUpdateItem(priceChangeDTO);

return View();

}

}

}

Code in Program.cs

using DataAccessLayer.Interfaces;

using DataAccessLayer.Repositories;

using Microsoft.Extensions.Configuration;

using System.Configuration;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();

var configuration = new ConfigurationBuilder()

.SetBasePath(builder.Environment.ContentRootPath)

.AddJsonFile("appsettings.json")

.Build();

string connectionString = configuration.GetConnectionString("Data");

builder.Services.AddScoped<IItemInterface, ItemRepository>(provider =>

new ItemRepository(connectionString));

builder.Services.AddScoped<IPriceChangeInterface,PriceChangeRepository>(provider =>new PriceChangeRepository(connectionString));

var app = builder.Build();

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();

Code in appsetting.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"Data": "Source=MSIGF63\\SQLEXPRESS;Initial Catalog=PracticalTest;Integrated Security=True;"

}

}

So these are all the documented answers of the test.