Fixing DataTypes Null or empty string Throw exceptions using ADO.NET

SQL SERVER DATABASE:

Create DATABASE AuthenticationDB

USE AuthenticationDB

----------- drop TABLE SampleDataTypes

CREATE TABLE SampleDataTypes(

Id INT PRIMARY KEY IDENTITY(1,1),

UserName VARCHAR(100),

Age INT,

IsActive BIT,

DateOfBirth DATETIME,

Doublevalue FLOAT(53),

Profile VARBINARY(MAX) DEFAULT (0x0),

UserID UNIQUEIDENTIFIER,

TimeSpanValue TIME,

Salary FLOAT,

JoinDate DATE,

Price DECIMAL(18,2)

)

----------- DROP PROC InsertDataTypes

CREATE PROC InsertDataTypes(

@UserName VARCHAR(100),

@Age INT,

@IsActive BIT,

@DateOfBirth DATETIME,

@Doublevalue FLOAT(53),

@Profile VARBINARY(MAX),

@UserID UNIQUEIDENTIFIER,

@TimeSpanValue TIME,

@Salary FLOAT,

@JoinDate DATE,

@Price DECIMAL(18,2)

)

AS

BEGIN

INSERT INTO SampleDataTypes (UserName,Age,IsActive,DateOfBirth,Doublevalue,Profile,UserID,TimeSpanValue,Salary,JoinDate,Price)

VALUES(@UserName,@Age,@IsActive,@DateOfBirth,@Doublevalue,@Profile,@UserID,@TimeSpanValue,@Salary,@JoinDate,@Price)

END

-------- Testing procs ------

DECLARE @ProfileData VARBINARY(MAX)

DECLARE @Userid uniqueidentifier = NEWID();

SET @ProfileData = CONVERT(VARBINARY(MAX), 'Hellow this is saikumar')

EXEC InsertDataTypes

@UserName = 'saikumar p',

@Age = 77,

@IsActive = 0, ----- passing value '', 'false', 'true' , false, true, 0 , 1

@DateOfBirth = '',

@Doublevalue = 12330458.99893,

@Profile = null,

@UserID = @Userid,

@TimeSpanValue = '',

@Salary = '0',

@JoinDate ='',

@Price = '0'

AuthRepo.cs:

public async Task<ResponseDataTypes> InsertSampleData(SampleDataTypes model)

{

ResponseDataTypes responseDataTypes = new ResponseDataTypes();

try

{

// validate model before inserting a record:

//\_validationHelper.ValidateModel(model);

// string to byte[]

//byte[] profile = System.Text.Encoding.UTF8.GetBytes(model.ProfileName);

\_validationHelper.ValidateModelParameters(model);

var storedProcName = "InsertDataTypes";

var parameters = new SqlParameter[]

{

//new SqlParameter("@UserName",string.IsNullOrEmpty(model.UserName) ? (object)DBNull.Value : model.UserName),

//new SqlParameter("@UserName",string.IsNullOrEmpty(model.UserName) ? string.Empty : model.UserName),

new SqlParameter("@UserName",string.IsNullOrEmpty(model.UserName) ? string.Empty : model.UserName),

//new SqlParameter("@Age",model.Age == null ? 0 : model.Age),

new SqlParameter("@Age",string.IsNullOrEmpty(model.Age.ToString()) ? 0 : model.Age),

//new SqlParameter("@Age",model.Age == null ? (object)DBNull.Value : model.Age),

//new SqlParameter("@Age",model.Age == null ? 0 : (model.Age is string) ? int.Parse(model.Age.ToString()) : model.Age),

//new SqlParameter("@IsActive",model.IsActive == null ? false : model.IsActive),

// new SqlParameter("@IsActive",model.IsActive == null ?(object)DBNull.Value : model.IsActive),

new SqlParameter("@IsActive",string.IsNullOrEmpty(model.IsActive.ToString() ) ? false : model.IsActive),

// new SqlParameter("@DateOfBirth",model.DateOfBirth == null ? (object)DBNull.Value : model.DateOfBirth),

// new SqlParameter("@DateOfBirth",model.DateOfBirth == null? DateTime.Now : model.DateOfBirth),

//new SqlParameter("@DateOfBirth",string.IsNullOrEmpty(model.DateOfBirth.ToString()) ? (object)DBNull.Value : (object)DateTime.Parse(model.DateOfBirth.ToString())),

//new SqlParameter("@DateOfBirth",string.IsNullOrEmpty(model.DateOfBirth.ToString()) ? DateTime.Now : (object)DateTime.Parse(model?.DateOfBirth?.ToString())),

new SqlParameter("@DateOfBirth",string.IsNullOrEmpty(model.DateOfBirth.ToString()) ? new DateTime(1900,1,1) : model.DateOfBirth),

//new SqlParameter("@Doublevalue",model.Doublevalue == null ? 0.0 : model.Doublevalue),

//new SqlParameter("@Doublevalue",model.Doublevalue == null ? (object)DBNull.Value : model.Doublevalue),

new SqlParameter("@Doublevalue",string.IsNullOrEmpty(model.Doublevalue.ToString()) ? Double.MinValue : model.Doublevalue),

// new SqlParameter("@Profile",model.Profile == null ? new byte[0] : model.Profile),

new SqlParameter("@Profile",string.IsNullOrEmpty(model?.Profile?.ToString()) ? new byte[0] : model.Profile),

//new SqlParameter("@UserID",model.UserID == null ? Guid.NewGuid() : model.UserID),

//new SqlParameter("@UserID",model.UserID == null ? (object)DBNull.Value : model.UserID),

new SqlParameter("@UserID",string.IsNullOrEmpty(model?.UserID.ToString()) ? Guid.NewGuid() : model.UserID),

//new SqlParameter("@TimeSpanValue",model.TimeSpanValue == null ? TimeSpan.Zero : model.TimeSpanValue),

//new SqlParameter("@TimeSpanValue",model?.TimeSpanValue == null ? (object)DBNull.Value : model.TimeSpanValue),

new SqlParameter("@TimeSpanValue",string.IsNullOrEmpty(model?.TimeSpanValue.ToString() ) ? TimeSpan.Zero : model.TimeSpanValue),

// new SqlParameter("@Salary",model?.Salary == null ? 0.0 : model.Salary),

// new SqlParameter("@Salary",model?.Salary == null ? (object)DBNull.Value : model.Salary),

new SqlParameter("@Salary",string.IsNullOrEmpty(model?.Salary.ToString()) ? 0.0 : model.Salary),

new SqlParameter("@JoinDate",model?.JoinDate == null ? new DateTime(1900,1,1).ToString("yyyy-MM-dd") : model.JoinDate),

//new SqlParameter("@JoinDate",model.JoinDate == null ? DBNull.Value : model.JoinDate),

// new SqlParameter("@JoinDate",string.IsNullOrEmpty(model?.JoinDate.ToString()) ? (object)DBNull.Value : model.JoinDate),

//new SqlParameter("@Price",model.Price == null ? 0.00 : model.Price)

// new SqlParameter("@Price",model?.Price == null ? (object)DBNull.Value : model.Price)

new SqlParameter("@Price",string.IsNullOrEmpty(model?.Price.ToString()) ? 0.00 : model.Price)

};

int rowsAffected = await \_sqlHelpers.InsertTable(storedProcName, parameters);

if(rowsAffected > 0)

{

responseDataTypes.StatusCode = 200;

responseDataTypes.StatusMessage = "Record inserted successfully.";

responseDataTypes.SampleDataTypes = model;

}

else

{

responseDataTypes.StatusCode = 400;

responseDataTypes.StatusMessage = "failed.";

}

}

catch(SqlException ex)

{

responseDataTypes.StatusCode = 500;

responseDataTypes.StatusMessage = ex.Message.ToString();

}

catch(Exception ex)

{

responseDataTypes.StatusCode = 500;

responseDataTypes.StatusMessage =ex.Message.ToString();

}

return responseDataTypes;

}

ValidationHelper.cs:

using JWTRoleAuthentication.CommonLayer.Models;

using System.Reflection.Metadata;

namespace JWTRoleAuthentication.Controllers

{

public class ValidationHelper

{

public ValidationHelper()

{

}

// validate model:

public void ValidateModel(object model)

{

if (model == null)

{

throw new ArgumentNullException("Model can not be null.");

}

var properties = model.GetType().GetProperties();

properties.ToList().ForEach((prop) =>

{

var value = prop.GetValue(model);

ValidateParameters(value, prop.Name, prop.PropertyType);

});

}

public void ValidateParameters(object value, string propName, Type propType)

{

// null parameter value

if (value == null)

{

throw new ArgumentNullException($"{propName} Cannot be null.");

}

// empty string

if (propType == typeof(string) && string.IsNullOrWhiteSpace(value as string))

{

throw new ArgumentNullException($"{propName} can not be empty or whitespace. ");

}

// default datetime

if (propType == typeof(DateTime) && (DateTime)value == default)

{

throw new ArgumentNullException($"{propName} date value cannot be default date.");

}

// empty guid

if (propType == typeof(Guid) && (Guid)value == Guid.Empty)

{

throw new ArgumentNullException($"{propName} cannot be a empty Guid.");

}

// empty int

//if(propType == typeof(int) && (!int.TryParse(value.ToString(), out )))

//{

// throw new ArgumentNullException($"{propName} must be a integer value.");

//}

if (propType == typeof(int))

{

if (!int.TryParse(value.ToString(), out \_))

{

throw new ArgumentException($"{propName} must be a valid integer.");

}

}

if (propType == typeof(float))

{

if (!float.TryParse(value.ToString(), out \_))

{

throw new ArgumentException($"{propName} must be a valid float value.");

}

}

if (propType == typeof(double))

{

if (!double.TryParse(value.ToString(), out \_))

{

throw new ArgumentException($"{propName} must be a valid double value.");

}

}

if (propType == typeof(bool))

{

if (!bool.TryParse(value.ToString(), out \_))

{

throw new ArgumentException($"{propName} must be a valid bool value.");

}

}

if (propType == typeof(decimal))

{

if (!decimal.TryParse(value.ToString(), out \_))

{

throw new ArgumentException($"{propName} must be a valid decimal value.");

}

}

if (propType == typeof(TimeZone))

{

if (!TimeSpan.TryParse(value.ToString(), out \_))

{

throw new ArgumentException($"{propName} must be a valid Time value.");

}

}

}

public void ValidateModelParameters(SampleDataTypes model)

{

if(model.Profile != null)

{

if(model.Profile is string)

{

model.Profile = System.Text.Encoding.UTF8.GetBytes(model?.Profile.ToString());

}

else

{

model.Profile = model.Profile;

}

}

if (model.Age.HasValue)

{

if (model.Age is string)

{

model.Age = int.Parse(model.Age.Value.ToString());

}

else

{

model.Age = model.Age.Value;

}

}

if (model.Doublevalue.HasValue)

{

if (model.Doublevalue is string)

{

model.Doublevalue = double.Parse(model.Doublevalue.Value.ToString());

}

else

{

model.Doublevalue = model.Doublevalue;

}

}

if (model.Salary.HasValue)

{

if (model.Salary is string)

{

model.Salary = float.Parse(model.Salary.Value.ToString());

}

else

{

model.Salary = model.Salary;

}

}

if (model.Price.HasValue)

{

if (model.Price is string)

{

model.Price = decimal.Parse(model.Price.Value.ToString());

}

else

{

model.Price = model.Price;

}

}

if (model.TimeSpanValue.HasValue)

{

if (model.TimeSpanValue is string)

{

model.TimeSpanValue = TimeSpan.Parse(model.TimeSpanValue.Value.ToString());

}

else

{

model.TimeSpanValue = model.TimeSpanValue;

}

}

if (model.DateOfBirth.HasValue)

{

if (model.DateOfBirth is string)

{

model.DateOfBirth = DateTime.Parse(model.DateOfBirth.ToString());

}

else

{

model.DateOfBirth = model.DateOfBirth;

}

}

if (model.JoinDate.HasValue)

{

if (model.DateOfBirth is string)

{

model.JoinDate = DateTime.Parse(model.JoinDate.ToString());

}

else

{

model.JoinDate = model.JoinDate;

}

}

}

}

}

SqlHelper.cs:

private readonly string \_connectionString;

private SqlConnection \_connection;

public SqlHelpers(string connectionString)

{

\_connectionString = connectionString;

\_connection = new SqlConnection(connectionString);

}

//public void Dispose()

//{

// if (\_connection != null)

// {

// if(\_connection.State != System.Data.ConnectionState.Closed)

// {

// \_connection.Close();

// }

// \_connection.Dispose();

// \_connection = null;

// }

//}

public async ValueTask DisposeAsync()

{

if (\_connection != null)

{

if (\_connection.State != System.Data.ConnectionState.Closed)

{

await \_connection.CloseAsync();

}

\_connection.Dispose();

\_connection = null;

}

}

public async Task EnsureConnection()

{

if (\_connection != null)

{

if(\_connection.State != ConnectionState.Open)

{

await \_connection.OpenAsync();

}

}

}

// ExecuteNonQuery: for Update and Insert Data Records

// using following methods:

public async Task<int> ExecuteNonQueryAsync(SqlCommand command)

{

await EnsureConnection();

return await command.ExecuteNonQueryAsync();

}

public async Task<int> InsertTable(string storedProcName,params SqlParameter[] parameters)

{

await EnsureConnection();

using (var command = CreateStoredProcedureCommand(storedProcName))

{

if(parameters != null)

{

command.Parameters.AddRange(parameters);

}

int rowAffected = await ExecuteNonQueryAsync(command);

return rowAffected;

}

}