RSAECRSAPI PROJECT

Install .NET 8 :

To download : <https://dotnet.microsoft.com/en-us/download/dotnet/8.0>

Choose Current or LTS(Long Term Support) version:

**.NET 8.0 SDK (v8.0.303) - Windows x64 Installer!**

Visual Studio support

Visual Studio 2022 (v17.10)

Included in

Visual Studio 17.10.3

Included runtimes

.NET Runtime 8.0.7

ASP.NET Core Runtime 8.0.7

.NET Desktop Runtime 8.0.7

Language support

C# 12.0

F# 8.0

Visual Basic 16.9

Project folder structures:

Requirements for the project:

1. To add sqlHelpers.cs used for the custom Data Access Layer .
2. Install packages for project.

System.Data.Client ------ version ="4.8.6"

<PackageReference Include="System.Data.SqlClient" Version="4.8.6" />

**SqlHelpers.cs:**

using System.Data.SqlClient;

using System.Data;

namespace RSAECRSAPI.ECRSDAL.Models

{

//public class SqlHelpers : IDisposable

public class SqlHelpers : IAsyncDisposable

{

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();

}

}

}

// sql command:

public SqlCommand CreateCommand(string commandText, CommandType commandType = CommandType.StoredProcedure)

{

var command = \_connection.CreateCommand();

command.CommandText = commandText;

command.CommandType = commandType;

return command;

}

// Create StoredProcedure command:

//public SqlCommand CreateStoredProcedureCommand(string storedProcedureName)

//{

// return CreateCommand(storedProcedureName,CommandType.StoredProcedure);

//}

public SqlCommand CreateStoredProcedureCommand(string storedProcedureName)

{

//var command = \_connection.CreateCommand();

//command.CommandText = storedProcedureName;

//command.CommandType = CommandType.StoredProcedure;

//return command;

var command = new SqlCommand(storedProcedureName, \_connection);

command.CommandType = CommandType.StoredProcedure;

return command;

}

// SqlDataAdapter and SqlDataReader

// To Read Table Data using SqlDataAdapter and SqlDataReader for

public async Task<DataTable> ExecuteDataTableAsync(SqlCommand command)

{

DataTable dataTable = new DataTable();

await EnsureConnection();

using (var adapter = new SqlDataAdapter(command))

{

await Task.Run(() => adapter.Fill(dataTable));

return dataTable;

}

}

public async Task<DataTable> ReadDataTable(SqlCommand command)

{

DataTable dataTable = new DataTable();

await EnsureConnection();

using (var reader = await ExecuteReader(command))

{

dataTable.Load(reader);

return dataTable;

}

}

public async Task<DataSet> GetMultipleTablesData(string[] storedProcNames, SqlParameter[] parameters = null)

{

await EnsureConnection();

var dataSet = new DataSet();

var tasks = storedProcNames.Select(async procName =>

{

using (var cmd = CreateStoredProcedureCommand(procName))

{

if (parameters != null)

{

cmd.Parameters.AddRange(parameters);

}

using (var adapter = new SqlDataAdapter(cmd))

{

var dataTable = new DataTable();

await Task.Run(() => adapter.Fill(dataTable));

dataSet.Tables.Add(dataTable);

}

}

});

await Task.WhenAll(tasks);

return dataSet;

}

// To Read database records using SqlDataReader: for read single row and multiple rows and datarowcollections.

// use following methods:

public async Task<SqlDataReader> ExecuteReader(SqlCommand command)

{

// await EnsureConnection();

//return await command.ExecuteReaderAsync(CommandBehavior.CloseConnection);

return await command.ExecuteReaderAsync();

}

public async Task<DataRow> GetSingleRow(string storedProcedureName, params SqlParameter[] parameters)

{

await EnsureConnection();

using (var command = CreateStoredProcedureCommand(storedProcedureName))

{

command.Parameters.AddRange(parameters);

using (var reader = await ExecuteReader(command))

{

var dataTable = new DataTable();

dataTable.Load(reader);

if (dataTable.Rows.Count > 0)

{

return dataTable.Rows[0];

}

}

}

return null;

}

public async Task<List<DataRow>> GetMultipleRows(string storedProcedureName, SqlParameter[] parameters)

{

await EnsureConnection();

var result = new List<DataRow>();

using (var command = CreateStoredProcedureCommand(storedProcedureName))

{

if (parameters != null)

{

command.Parameters.AddRange(parameters);

}

using (var reader = await ExecuteReader(command))

{

var dataTable = new DataTable();

dataTable.Load(reader);

if (dataTable.Rows.Count > 0)

{

result = dataTable.AsEnumerable().Select((row) => row).ToList();

}

}

}

return result;

}

public async Task<DataRowCollection> GetMultipleDataRows(string storedProcedureName, SqlParameter[] parameters)

{

await EnsureConnection();

using (var command = CreateStoredProcedureCommand(storedProcedureName))

{

if (parameters != null)

{

command.Parameters.AddRange(parameters);

}

using (var reader = await ExecuteReader(command))

{

var dataTable = new DataTable();

dataTable.Load(reader);

if (dataTable.Rows.Count > 0)

{

return dataTable.Rows;

}

}

}

return null;

}

// 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;

}

}

public async Task<int> UpdateTable(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;

}

}

// ExecuteScalar methods:

//public async Task<object> ExecuteScalarAsync(SqlCommand command)

//{

// await EnsureConnection();

// return await command.ExecuteScalarAsync();

//}

public async Task<String> ExecuteScalarString(SqlCommand command)

{

await EnsureConnection();

return (string)await command.ExecuteScalarAsync();

}

public async Task<int> ExecuteScalarInt(SqlCommand command)

{

await EnsureConnection();

return (int)await command.ExecuteScalarAsync();

}

public async Task<int> ExecuteIntScalar(string storeProcName, params SqlParameter[] parameters)

{

await EnsureConnection();

using (var cmd = CreateStoredProcedureCommand(storeProcName))

{

if (parameters != null)

{

cmd.Parameters.AddRange(parameters);

}

int variable = (int)await ExecuteScalarInt(cmd);

return variable;

}

}

public async Task<string> ExecuteStringScalar(string storeProcName, params SqlParameter[] parameters)

{

await EnsureConnection();

using (var cmd = CreateStoredProcedureCommand(storeProcName))

{

if (parameters != null)

{

cmd.Parameters.AddRange(parameters);

}

string variable = (string)await ExecuteScalarString(cmd);

return variable;

}

}

// IDataReader Methods:

// use following methods:

public async Task<IDataReader> ExecuteIDataReader(SqlCommand command)

{

await EnsureConnection();

return await command.ExecuteReaderAsync(CommandBehavior.CloseConnection);

}

public async Task<IDataReader> IDataReaderAsync(string storeProcName, params SqlParameter[] parameters)

{

await EnsureConnection();

using (var cmd = CreateStoredProcedureCommand(storeProcName))

{

if (parameters != null)

{

cmd.Parameters.AddRange(parameters);

}

return await ExecuteIDataReader(cmd);

}

}

public async Task<DataRowCollection> IDataReaderMultiRow(string storeProcName, params SqlParameter[] parameters)

{

using (IDataReader reader = await IDataReaderAsync(storeProcName, parameters))

{

var dataTable = new DataTable();

dataTable.Load(reader);

return dataTable.Rows;

}

}

public async Task<List<DataRow>> IDataReaderMultiRows(string storeProcName, params SqlParameter[] parameters)

{

using (IDataReader reader = await IDataReaderAsync(storeProcName, parameters))

{

var dataTable = new DataTable();

dataTable.Load(reader);

return dataTable.Rows.Cast<DataRow>().ToList();

}

}

public async Task<List<IDataRecord>> IDataRecordsData(string storedProcName, params SqlParameter[] parameters)

{

var records = new List<IDataRecord>();

await EnsureConnection();

using (var cmd = CreateStoredProcedureCommand(storedProcName))

{

if (parameters != null)

{

cmd.Parameters.AddRange(parameters);

}

using (IDataReader reader = await ExecuteIDataReader(cmd))

{

while (reader.Read())

{

records.Add(reader);

}

}

}

return records;

}

}

}