Dapper - Code Generator

TABLE POHeader:

```
=================================================
====
= Stored procedures go in the database, not the app.    =
= You can copy them all into a SQL Management query and =
= select and execute them one at a time.                =
=================================================
====
-------------- Stored Proc for INSERT
CREATE PROCEDURE spPOHeader_Insert
--Parameters for Insert stored procedure
@POHeaderOrderNumber int,
@POHeaderOrderDate date,
@POHeaderSupplierID int,
@POHeaderSupplierAddress1 nvarchar(50),
@POHeaderSupplierAddress2 nvarchar(50),
@POHeaderSupplierAddress3 nvarchar(50),
@POHeaderSupplierPostCode nvarchar(10),
@POHeaderSupplierEmail nvarchar(256),
@POHeaderRequestedBy nvarchar(450),
@POHeaderIsArchived bit
AS
BEGIN
--SQL for Insert stored procedure
INSERT INTO POHeader(POHeaderOrderNumber, POHeaderOrderDate, PO
HeaderSupplierID, POHeaderSupplierAddress1, POHeaderSupplierAddress
2, POHeaderSupplierAddress3, POHeaderSupplierPostCode, POHeaderSup
plierEmail, POHeaderRequestedBy, POHeaderIsArchived) VALUES (@POHe
aderOrderNumber, @POHeaderOrderDate, @POHeaderSupplierID, @POHe
```

```
aderSupplierAddress1, @POHeaderSupplierAddress2, @POHeaderSupplier
Address3, @POHeaderSupplierPostCode, @POHeaderSupplierEmail, @PO
HeaderRequestedBy, @POHeaderIsArchived)
END

--------------  Stored Proc for SELECT (LIST, just first six fields but you can
change in final code.)
CREATE PROCEDURE spPOHeader_List
--No parameters required.
AS
BEGIN
--SQL for Select stored procedure.
SELECT TOP 30 POHeaderID, POHeaderOrderNumber, POHeaderOrderDat
e, POHeaderSupplierID, POHeaderSupplierAddress1, POHeaderSupplierAd
dress2 FROM POHeader ORDER BY POHeaderID DESC
END

--------------  Stored Proc for SELECT (one)
CREATE PROCEDURE spPOHeader_GetOne
-- Needs one parameter for primary key
@POHeaderID int
AS
BEGIN
-- SQL Select for one table row
SELECT POHeaderID, POHeaderOrderNumber, POHeaderOrderDate, POHe
aderSupplierID, POHeaderSupplierAddress1, POHeaderSupplierAddress2, P
OHeaderSupplierAddress3, POHeaderSupplierPostCode, POHeaderSupplie
rEmail, POHeaderRequestedBy, POHeaderIsArchived FROM POHeader WH
ERE POHeaderID= @POHeaderID
END


--------------  Stored Proc for SELECT LIKE
CREATE PROCEDURE spPOHeader_Search
-- One parameter required to identify row to delete.
@Param varchar(128)
AS
BEGIN
```

```
-- SQL for search looking for embedded content.
SELECT POHeaderOrderNumber, POHeaderOrderDate, POHeaderSupplierI
D, POHeaderSupplierAddress1, POHeaderSupplierAddress2 FROM POHead
er WHERE CAST(POHeaderOrderNumber AS varchar(20)) LIKE '%' + @par
am + '%' OR CONVERT(varchar(12),POHeaderOrderDate,101) LIKE + '%' +
@param + '%' OR CAST(POHeaderSupplierID AS varchar(20)) LIKE '%' +
@param + '%' OR POHeaderSupplierAddress1 LIKE '%' + @param + '%' O
R POHeaderSupplierAddress2 LIKE '%' + @param + '%' OR POHeaderSup
plierAddress3 LIKE '%' + @param + '%' OR POHeaderSupplierPostCode LI
KE '%' + @param + '%' OR POHeaderSupplierEmail LIKE '%' + @param +
'%' OR POHeaderRequestedBy LIKE '%' + @param + '%'
END

------------ Stored Proc for SELECT DATA RANGE
-- Another wild guess, but better than nothin'
CREATE PROCEDURE spPOHeader_DateRange
@StartDate date,
@EndDate date
AS
BEGIN
-- SQL for search looking range of dates
SELECT POHeaderOrderNumber, POHeaderOrderDate, POHeaderSupplierI
D, POHeaderSupplierAddress1, POHeaderSupplierAddress2 FROM POHead
er WHERE  CAST(POHeaderOrderDate AS Date) Between @startDate AND
@endDate
END

-------------  Stored Proc for UPDATE
CREATE PROCEDURE spPOHeader_Update
-- Parameters for Update stored procedure.
@POHeaderID int,
@POHeaderOrderNumber int,
@POHeaderOrderDate date,
@POHeaderSupplierID int,
@POHeaderSupplierAddress1 nvarchar(50),
@POHeaderSupplierAddress2 nvarchar(50),
@POHeaderSupplierAddress3 nvarchar(50),
@POHeaderSupplierPostCode nvarchar(10),
```

```
@POHeaderSupplierEmail nvarchar(256),
@POHeaderRequestedBy nvarchar(450),
@POHeaderIsArchived bit
AS
BEGIN
-- SQL for Update stored procedure
UPDATE POHeader SET POHeaderOrderNumber = @POHeaderOrderNumb
er, POHeaderOrderDate = @POHeaderOrderDate, POHeaderSupplierID =
@POHeaderSupplierID, POHeaderSupplierAddress1 = @POHeaderSupplier
Address1, POHeaderSupplierAddress2 = @POHeaderSupplierAddress2, PO
HeaderSupplierAddress3 = @POHeaderSupplierAddress3, POHeaderSuppli
erPostCode = @POHeaderSupplierPostCode, POHeaderSupplierEmail = @
POHeaderSupplierEmail, POHeaderRequestedBy = @POHeaderRequested
By, POHeaderIsArchived = @POHeaderIsArchived WHERE POHeaderID =
@POHeaderID
END

----------  Stored Proc for DELETE
CREATE PROCEDURE spPOHeader_Delete
-- One parameter required to identify row to delete.
@POHeaderID int
AS
BEGIN
-- SQL for Delete stored procedure (physically deletes, you may want to ch
ange this to mark inactive)
DELETE FROM POHeader WHERE POHeaderID = @POHeaderID
END


==========================================================
===
Back in Visual Studio, you need to add some classes
and an interface to the Data folder, with the
names shown below.


You should have three classes, per database table, in the Data folder.
```

```
---------------------------- /DATA/POHeader.cs
using System;
using System.ComponentModel.DataAnnotations;
// This is the model for one row in the database table. You may need to mak
e some adjustments.
namespace BlazorPurchaseOrders.Data
{
    public class POHeader
    {
    [Required]
public int POHeaderID { get; set; }
[Required]
public int POHeaderOrderNumber { get; set; }
[Required]
public DateTime POHeaderOrderDate { get; set; }
[Required]
public int POHeaderSupplierID { get; set; }
[StringLength(50)]
public string POHeaderSupplierAddress1 { get; set; }
[StringLength(50)]
public string POHeaderSupplierAddress2 { get; set; }
[StringLength(50)]
public string POHeaderSupplierAddress3 { get; set; }
[StringLength(10)]
public string POHeaderSupplierPostCode { get; set; }
[StringLength(256)]
public string POHeaderSupplierEmail { get; set; }
[StringLength(450)]
public string POHeaderRequestedBy { get; set; }
[Required]
public bool POHeaderIsArchived { get; set; }

    }
}

---------------------------- /DATA/POHeaderService.cs
// This is the service for the POHeader class.
using Dapper;
```

```
using Microsoft.Data.SqlClient;
using System;
using System.Collections.Generic;
using System.Data;
using System.Threading.Tasks;

namespace BlazorPurchaseOrders.Data
{
    public class POHeaderService : IPOHeaderService
    {
        // Database connection
        private readonly SqlConnectionConfiguration _configuration;
        public POHeaderService(SqlConnectionConfiguration configuration)
        {
            _configuration = configuration;
        }
        // Add (create) a POHeader table row (SQL Insert)
        // This only works if you're already created the stored procedure.
        public async Task<bool> POHeaderInsert(POHeader poheader)
        {
            using (var conn = new SqlConnection(_configuration.Value))
            {
                var parameters = new DynamicParameters();
            parameters.Add("POHeaderOrderNumber", poheader.POHeaderOrd
erNumber, DbType.Int32);
parameters.Add("POHeaderOrderDate", poheader.POHeaderOrderDate, Db
Type.Date);
parameters.Add("POHeaderSupplierID", poheader.POHeaderSupplierID, Db
Type.Int32);
parameters.Add("POHeaderSupplierAddress1", poheader.POHeaderSupplie
rAddress1, DbType.String);
parameters.Add("POHeaderSupplierAddress2", poheader.POHeaderSuppli
erAddress2, DbType.String);
parameters.Add("POHeaderSupplierAddress3", poheader.POHeaderSuppli
erAddress3, DbType.String);
parameters.Add("POHeaderSupplierPostCode", poheader.POHeaderSuppli
erPostCode, DbType.String);
parameters.Add("POHeaderSupplierEmail", poheader.POHeaderSupplierE
```

```
mail, DbType.String);
parameters.Add("POHeaderRequestedBy", poheader.POHeaderRequested
By, DbType.String);
parameters.Add("POHeaderIsArchived", poheader.POHeaderIsArchived, D
bType.Boolean);

            // Stored procedure method
            await conn.ExecuteAsync("spPOHeader_Insert", parameters, com
mandType: CommandType.StoredProcedure);
        }
        return true;
    }
    // Get a list of poheader rows (SQL Select)
    // This only works if you're already created the stored procedure.
    public async Task<IEnumerable<POHeader>> POHeaderList()
    {
        IEnumerable<POHeader> poheaders;
        using (var conn = new SqlConnection(_configuration.Value))
        {
            poheaders = await conn.QueryAsync<POHeader>("spPOHeader_
List", commandType: CommandType.StoredProcedure);
        }
        return poheaders;
    }
    //Search for data (very generic...you may need to adjust.
    public async Task<IEnumerable<POHeader>> POHeaderSearch(string
@Param)
    {
        var parameters = new DynamicParameters();
        parameters.Add("@Param", Param, DbType.String);
        IEnumerable<POHeader> poheaders;
        using (var conn = new SqlConnection(_configuration.Value))
        {
            poheaders = await conn.QueryAsync<POHeader>("spPOHeader_
Search", parameters, commandType: CommandType.StoredProcedure);
        }
        return poheaders;
    }
```

```
        // Search based on date range. Code generator makes wild guess, you
        // will likely need to adjust field names
        public async Task<IEnumerable<POHeader>> POHeaderDateRange(D
ateTime @StartDate, DateTime @EndDate)
        {
            var parameters = new DynamicParameters();
            parameters.Add("@StartDate", StartDate, DbType.Date);
            parameters.Add("@EndDate", EndDate, DbType.Date);
            IEnumerable<POHeader>  poheaders;
            using (var conn = new SqlConnection(_configuration.Value))
            {
                poheaders = await conn.QueryAsync<POHeader>("spPOHeader_
DateRange", parameters, commandType: CommandType.StoredProcedur
e);
            }
            return poheaders;
        }

        // Get one poheader based on its POHeaderID (SQL Select)
        // This only works if you're already created the stored procedure.
        public async Task<POHeader> POHeader_GetOne(int @POHeaderID)
        {
            POHeader poheader = new POHeader();
            var parameters = new DynamicParameters();
            parameters.Add("@POHeaderID", POHeaderID, DbType.Int32);
            using (var conn = new SqlConnection(_configuration.Value))
            {
                poheader = await conn.QueryFirstOrDefaultAsync<POHeader>("s
pPOHeader_GetOne",parameters,commandType: CommandType.StoredPro
cedure);
            }
            return poheader;
        }
        // Update one POHeader row based on its POHeaderID (SQL Update)
        // This only works if you're already created the stored procedure.
        public async Task<bool> POHeaderUpdate(POHeader poheader)
        {
            using (var conn = new SqlConnection(_configuration.Value))
```

```
        {
            var parameters = new DynamicParameters();
            parameters.Add("POHeaderID", poheader.POHeaderID, DbType.In
t32);

            parameters.Add("POHeaderOrderNumber", poheader.POHeaderO
rderNumber, DbType.Int32);
parameters.Add("POHeaderOrderDate", poheader.POHeaderOrderDate, Db
Type.Date);
parameters.Add("POHeaderSupplierID", poheader.POHeaderSupplierID, Db
Type.Int32);
parameters.Add("POHeaderSupplierAddress1", poheader.POHeaderSupplie
rAddress1, DbType.String);
parameters.Add("POHeaderSupplierAddress2", poheader.POHeaderSuppli
erAddress2, DbType.String);
parameters.Add("POHeaderSupplierAddress3", poheader.POHeaderSuppli
erAddress3, DbType.String);
parameters.Add("POHeaderSupplierPostCode", poheader.POHeaderSuppli
erPostCode, DbType.String);
parameters.Add("POHeaderSupplierEmail", poheader.POHeaderSupplierE
mail, DbType.String);
parameters.Add("POHeaderRequestedBy", poheader.POHeaderRequested
By, DbType.String);
parameters.Add("POHeaderIsArchived", poheader.POHeaderIsArchived, D
bType.Boolean);

            await conn.ExecuteAsync("spPOHeader_Update", parameters, co
mmandType: CommandType.StoredProcedure);
        }
        return true;
    }

    // Physically delete one POHeader row based on its POHeaderID (SQL
Delete)
    // This only works if you're already created the stored procedure.
    public async Task<bool> POHeaderDelete(int POHeaderID)
    {
        var parameters = new DynamicParameters();
```

```
            parameters.Add("@POHeaderID", POHeaderID, DbType.Int32);
            using (var conn = new SqlConnection(_configuration.Value))
            {
                await conn.ExecuteAsync("spPOHeader_Delete",parameters, com
mandType: CommandType.StoredProcedure);
            }
            return true;
        }
    }
}

---------------------------- /Data/IPOHeaderService.cs
// This is the POHeader Interface
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace BlazorPurchaseOrders.Data
{
    // Each item below provides an interface to a method in POHeaderServic
es.cs
    public interface IPOHeaderService
    {
        Task<bool> POHeaderInsert(POHeader poheader);
        Task<IEnumerable<POHeader>> POHeaderList();
        Task<IEnumerable<POHeader>> POHeaderSearch(string Param);
        Task<IEnumerable<POHeader>> POHeaderDateRange(DateTime @St
artDate, DateTime @EndDate);
        Task<POHeader> POHeader_GetOne(int POHeaderID);
        Task<bool> POHeaderUpdate(POHeader poheader);
        Task<bool> POHeaderDelete(int POHeaderID);
    }
}
==========================================================
==
= With the classes finished, open up Startup.cs and
= add the services.AddScoprf linr below to the
=  public void ConfigureService{} block, perhaps just
```

```
=  above the comment and code that defines SqlConnectionConfiguration
================================================================
===

services.AddScoped<IPOHeaderService, POHeaderService>();
```

TABLE POLine:

```
================================================================
====
= Stored procedures go in the database, not the app.    =
= You can copy them all into a SQL Management query and =
= select and execute them one at a time.                =
================================================================
====
-------------- Stored Proc for INSERT
CREATE PROCEDURE spPOLine_Insert
--Parameters for Insert stored procedure
@POLineHeaderID int,
@POLineProductID int,
@POLineProductDescription nvarchar(50),
@POLineProductQuantity decimal(9, 3),
@POLineProductUnitPrice money,
@POLineTaxRate decimal(6, 4)
AS
BEGIN
--SQL for Insert stored procedure
INSERT INTO POLine(POLineHeaderID, POLineProductID, POLineProductDe
scription, POLineProductQuantity, POLineProductUnitPrice, POLineTaxRat
e) VALUES (@POLineHeaderID, @POLineProductID, @POLineProductDescr
iption, @POLineProductQuantity, @POLineProductUnitPrice, @POLineTaxR
ate)
```

```
END

--------------  Stored Proc for SELECT (LIST, just first six fields but you can change in final code.)
CREATE PROCEDURE spPOLine_List
--No parameters required.
AS
BEGIN
--SQL for Select stored procedure.
SELECT TOP 30 POLineID, POLineHeaderID, POLineProductID, POLineProductDescription, POLineProductQuantity, POLineProductUnitPrice FROM POLine ORDER BY POLineID DESC
END

--------------  Stored Proc for SELECT (one)
CREATE PROCEDURE spPOLine_GetOne
-- Needs one parameter for primary key
@POLineID int
AS
BEGIN
-- SQL Select for one table row
SELECT POLineID, POLineHeaderID, POLineProductID, POLineProductDescription, POLineProductQuantity, POLineProductUnitPrice, POLineTaxRate FROM POLine WHERE POLineID= @POLineID
END


--------------  Stored Proc for SELECT LIKE
CREATE PROCEDURE spPOLine_Search
-- One parameter required to identify row to delete.
@Param varchar(128)
AS
BEGIN
-- SQL for search looking for embedded content.
SELECT POLineHeaderID, POLineProductID, POLineProductDescription, POLineProductQuantity, POLineProductUnitPrice FROM POLine WHERE CAST(POLineHeaderID AS varchar(20)) LIKE '%' + @param + '%' OR CAST(POLineProductID AS varchar(20)) LIKE '%' + @param + '%' OR POLineProduct
```

```
Description LIKE '%' + @param + '%' OR CAST(POLineProductQuantity AS
varchar(20)) LIKE '%' + @param + '%' OR CAST(POLineProductUnitPrice A
S varchar(20)) LIKE '%' + @param + '%' OR CAST(POLineTaxRate AS varc
har(20)) LIKE '%' + @param + '%'
END

------------ Stored Proc for SELECT DATA RANGE
-- Another wild guess, but better than nothin'
CREATE PROCEDURE spPOLine_DateRange
@StartDate date,
@EndDate date
AS
BEGIN
-- SQL for search looking range of dates
SELECT POLineHeaderID, POLineProductID, POLineProductDescription, PO
LineProductQuantity, POLineProductUnitPrice FROM POLine WHERE
END

-------------  Stored Proc for UPDATE
CREATE PROCEDURE spPOLine_Update
-- Parameters for Update stored procedure.
@POLineID int,
@POLineHeaderID int,
@POLineProductID int,
@POLineProductDescription nvarchar(50),
@POLineProductQuantity decimal(9, 3),
@POLineProductUnitPrice money,
@POLineTaxRate decimal(6, 4)
AS
BEGIN
-- SQL for Update stored procedure
UPDATE POLine SET POLineHeaderID = @POLineHeaderID, POLineProduct
ID = @POLineProductID, POLineProductDescription = @POLineProductDes
cription, POLineProductQuantity = @POLineProductQuantity, POLineProdu
ctUnitPrice = @POLineProductUnitPrice, POLineTaxRate = @POLineTaxRat
e WHERE POLineID = @POLineID
END
```

```
-------------  Stored Proc for DELETE
CREATE PROCEDURE spPOLine_Delete
-- One parameter required to identify row to delete.
@POLineID int
AS
BEGIN
-- SQL for Delete stored procedure (physically deletes, you may want to ch
ange this to mark inactive)
DELETE FROM POLine WHERE POLineID = @POLineID
END



==========================================================
===
Back in Visual Studio, you need to add some classes
and an interface to the Data folder, with the
names shown below.


You should have three classes, per database table, in the Data folder.

--------------------------- /DATA/POLine.cs
using System;
using System.ComponentModel.DataAnnotations;
// This is the model for one row in the database table. You may need to mak
e some adjustments.
namespace BlazorPurchaseOrders.Data
{
   public class POLine
   {
   [Required]
public int POLineID { get; set; }
[Required]
public int POLineHeaderID { get; set; }
[Required]
public int POLineProductID { get; set; }
[Required]
[StringLength(50)]
```

```
public string POLineProductDescription { get; set; }
[Required]
public decimal POLineProductQuantity { get; set; }
[Required]
public decimal POLineProductUnitPrice { get; set; }
[Required]
public decimal POLineTaxRate { get; set; }

   }
}

--------------------------- /DATA/POLineService.cs
// This is the service for the POLine class.
using Dapper;
using Microsoft.Data.SqlClient;
using System;
using System.Collections.Generic;
using System.Data;
using System.Threading.Tasks;

namespace BlazorPurchaseOrders.Data
{
   public class POLineService : IPOLineService
   {
      // Database connection
      private readonly SqlConnectionConfiguration _configuration;
      public POLineService(SqlConnectionConfiguration configuration)
      {
         _configuration = configuration;
      }
      // Add (create) a POLine table row (SQL Insert)
      // This only works if you're already created the stored procedure.
      public async Task<bool> POLineInsert(POLine poline)
      {
         using (var conn = new SqlConnection(_configuration.Value))
         {
            var parameters = new DynamicParameters();
         parameters.Add("POLineHeaderID", poline.POLineHeaderID, DbTyp
```

```
e.Int32);
parameters.Add("POLineProductID", poline.POLineProductID, DbType.Int3
2);
parameters.Add("POLineProductDescription", poline.POLineProductDescri
ption, DbType.String);
parameters.Add("POLineProductQuantity", poline.POLineProductQuantity,
DbType.Decimal);
parameters.Add("POLineProductUnitPrice", poline.POLineProductUnitPrice,
DbType.Decimal);
parameters.Add("POLineTaxRate", poline.POLineTaxRate, DbType.Decimal);

            // Stored procedure method
            await conn.ExecuteAsync("spPOLine_Insert", parameters, comma
ndType: CommandType.StoredProcedure);
        }
        return true;
    }
    // Get a list of poline rows (SQL Select)
    // This only works if you're already created the stored procedure.
    public async Task<IEnumerable<POLine>> POLineList()
    {
        IEnumerable<POLine> polines;
        using (var conn = new SqlConnection(_configuration.Value))
        {
            polines = await conn.QueryAsync<POLine>("spPOLine_List", com
mandType: CommandType.StoredProcedure);
        }
        return polines;
    }
    //Search for data (very generic...you may need to adjust.
    public async Task<IEnumerable<POLine>> POLineSearch(string @Par
am)
    {
        var parameters = new DynamicParameters();
        parameters.Add("@Param", Param, DbType.String);
        IEnumerable<POLine> polines;
        using (var conn = new SqlConnection(_configuration.Value))
        {
```

```
            polines = await conn.QueryAsync<POLine>("spPOLine_Search",
parameters, commandType: CommandType.StoredProcedure);
        }
        return polines;
    }
    // Search based on date range. Code generator makes wild guess, you
    // will likely need to adjust field names
    public async Task<IEnumerable<POLine>> POLineDateRange(DateTim
e @StartDate, DateTime @EndDate)
    {
        var parameters = new DynamicParameters();
        parameters.Add("@StartDate", StartDate, DbType.Date);
        parameters.Add("@EndDate", EndDate, DbType.Date);
        IEnumerable<POLine>  polines;
        using (var conn = new SqlConnection(_configuration.Value))
        {
            polines = await conn.QueryAsync<POLine>("spPOLine_DateRang
e", parameters, commandType: CommandType.StoredProcedure);
        }
        return polines;
    }

    // Get one poline based on its POLineID (SQL Select)
    // This only works if you're already created the stored procedure.
    public async Task<POLine> POLine_GetOne(int @POLineID)
    {
        POLine poline = new POLine();
        var parameters = new DynamicParameters();
        parameters.Add("@POLineID", POLineID, DbType.Int32);
        using (var conn = new SqlConnection(_configuration.Value))
        {
            poline = await conn.QueryFirstOrDefaultAsync<POLine>("spPOLi
ne_GetOne",parameters,commandType: CommandType.StoredProcedure);
        }
        return poline;
    }
    // Update one POLine row based on its POLineID (SQL Update)
    // This only works if you're already created the stored procedure.
```

```
        public async Task<bool> POLineUpdate(POLine poline)
        {
            using (var conn = new SqlConnection(_configuration.Value))
            {
                var parameters = new DynamicParameters();
                parameters.Add("POLineID", poline.POLineID, DbType.Int32);

                parameters.Add("POLineHeaderID", poline.POLineHeaderID, DbTy
pe.Int32);
parameters.Add("POLineProductID", poline.POLineProductID, DbType.Int3
2);
parameters.Add("POLineProductDescription", poline.POLineProductDescri
ption, DbType.String);
parameters.Add("POLineProductQuantity", poline.POLineProductQuantity,
DbType.Decimal);
parameters.Add("POLineProductUnitPrice", poline.POLineProductUnitPrice,
DbType.Decimal);
parameters.Add("POLineTaxRate", poline.POLineTaxRate, DbType.Decimal);

                await conn.ExecuteAsync("spPOLine_Update", parameters, comm
andType: CommandType.StoredProcedure);
            }
            return true;
        }

        // Physically delete one POLine row based on its POLineID (SQL Delet
e)
        // This only works if you're already created the stored procedure.
        public async Task<bool> POLineDelete(int POLineID)
        {
            var parameters = new DynamicParameters();
            parameters.Add("@POLineID", POLineID, DbType.Int32);
            using (var conn = new SqlConnection(_configuration.Value))
            {
                await conn.ExecuteAsync("spPOLine_Delete",parameters, comma
ndType: CommandType.StoredProcedure);
            }
            return true;
```

```
      }
   }
}

--------------------------- /Data/IPOLineService.cs
// This is the POLine Interface
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace BlazorPurchaseOrders.Data
{
   // Each item below provides an interface to a method in POLineServices.
cs
   public interface IPOLineService
   {
      Task<bool> POLineInsert(POLine poline);
      Task<IEnumerable<POLine>> POLineList();
      Task<IEnumerable<POLine>> POLineSearch(string Param);
      Task<IEnumerable<POLine>> POLineDateRange(DateTime @StartDat
e, DateTime @EndDate);
      Task<POLine> POLine_GetOne(int POLineID);
      Task<bool> POLineUpdate(POLine poline);
      Task<bool> POLineDelete(int POLineID);
   }
}
```

TABLE Product:

```
==========================================================
====
= Stored procedures go in the database, not the app.    =
```

```
= You can copy them all into a SQL Management query and =
= select and execute them one at a time.              =
=================================================================
----------- Stored Proc for INSERT
CREATE PROCEDURE spProduct_Insert
--Parameters for Insert stored procedure
@ProductCode nvarchar(25),
@ProductDescription nvarchar(50),
@ProductUnitPrice money,
@ProductSupplierID int,
@ProductIsArchived bit
AS
BEGIN
--SQL for Insert stored procedure
INSERT INTO Product(ProductCode, ProductDescription, ProductUnitPrice,
ProductSupplierID, ProductIsArchived) VALUES (@ProductCode, @Product
Description, @ProductUnitPrice, @ProductSupplierID, @ProductIsArchived)
END

----------  Stored Proc for SELECT (LIST, just first six fields but you can
change in final code.)
CREATE PROCEDURE spProduct_List
--No parameters required.
AS
BEGIN
--SQL for Select stored procedure.
SELECT TOP 30 ProductID, ProductCode, ProductDescription, ProductUnit
Price, ProductSupplierID, ProductIsArchived FROM Product ORDER BY Pro
ductID DESC
END

----------  Stored Proc for SELECT (one)
CREATE PROCEDURE spProduct_GetOne
-- Needs one parameter for primary key
@ProductID int
AS
BEGIN
```

```
-- SQL Select for one table row
SELECT ProductID, ProductCode, ProductDescription, ProductUnitPrice, Pr
oductSupplierID, ProductIsArchived FROM Product WHERE ProductID= @P
roductID
END


--------------  Stored Proc for SELECT LIKE
CREATE PROCEDURE spProduct_Search
-- One parameter required to identify row to delete.
@Param varchar(128)
AS
BEGIN
-- SQL for search looking for embedded content.
SELECT ProductCode, ProductDescription, ProductUnitPrice, ProductSuppl
ierID, ProductIsArchived FROM Product WHERE ProductCode LIKE '%' + @
param + '%' OR ProductDescription LIKE '%' + @param + '%' OR CAST(Pr
oductUnitPrice AS varchar(20)) LIKE '%' + @param + '%' OR CAST(Produc
tSupplierID AS varchar(20)) LIKE '%' + @param + '%'
END

------------- Stored Proc for SELECT DATA RANGE
-- Another wild guess, but better than nothin'
CREATE PROCEDURE spProduct_DateRange
@StartDate date,
@EndDate date
AS
BEGIN
-- SQL for search looking range of dates
SELECT ProductCode, ProductDescription, ProductUnitPrice, ProductSuppl
ierID, ProductIsArchived FROM Product WHERE
END

--------------  Stored Proc for UPDATE
CREATE PROCEDURE spProduct_Update
-- Parameters for Update stored procedure.
@ProductID int,
@ProductCode nvarchar(25),
```

```
@ProductDescription nvarchar(50),
@ProductUnitPrice money,
@ProductSupplierID int,
@ProductIsArchived bit
AS
BEGIN
-- SQL for Update stored procedure
UPDATE Product SET ProductCode = @ProductCode, ProductDescription =
@ProductDescription, ProductUnitPrice = @ProductUnitPrice, ProductSupp
lierID = @ProductSupplierID, ProductIsArchived = @ProductIsArchived WH
ERE ProductID = @ProductID
END

----------  Stored Proc for DELETE
CREATE PROCEDURE spProduct_Delete
-- One parameter required to identify row to delete.
@ProductID int
AS
BEGIN
-- SQL for Delete stored procedure (physically deletes, you may want to ch
ange this to mark inactive)
DELETE FROM Product WHERE ProductID = @ProductID
END



==========================================================
===
Back in Visual Studio, you need to add some classes
and an interface to the Data folder, with the
names shown below.


You should have three classes, per database table, in the Data folder.

---------------------------- /DATA/Product.cs
using System;
using System.ComponentModel.DataAnnotations;
// This is the model for one row in the database table. You may need to mak
```

```
e some adjustments.
namespace BlazorPurchaseOrders.Data
{
    public class Product
    {
    [Required]
public int ProductID { get; set; }
[Required]
[StringLength(25)]
public string ProductCode { get; set; }
[Required]
[StringLength(50)]
public string ProductDescription { get; set; }
[Required]
public decimal ProductUnitPrice { get; set; }
public int ProductSupplierID { get; set; }
[Required]
public bool ProductIsArchived { get; set; }

    }
}

---------------------------- /DATA/ProductService.cs
// This is the service for the Product class.
using Dapper;
using Microsoft.Data.SqlClient;
using System;
using System.Collections.Generic;
using System.Data;
using System.Threading.Tasks;

namespace BlazorPurchaseOrders.Data
{
    public class ProductService : IProductService
    {
        // Database connection
        private readonly SqlConnectionConfiguration _configuration;
        public ProductService(SqlConnectionConfiguration configuration)
```

```
        {
            _configuration = configuration;
        }
        // Add (create) a Product table row (SQL Insert)
        // This only works if you're already created the stored procedure.
        public async Task<bool> ProductInsert(Product product)
        {
            using (var conn = new SqlConnection(_configuration.Value))
            {
                var parameters = new DynamicParameters();
            parameters.Add("ProductCode", product.ProductCode, DbType.Stri
ng);
parameters.Add("ProductDescription", product.ProductDescription, DbTyp
e.String);
parameters.Add("ProductUnitPrice", product.ProductUnitPrice, DbType.De
cimal);
parameters.Add("ProductSupplierID", product.ProductSupplierID, DbType.I
nt32);
parameters.Add("ProductIsArchived", product.ProductIsArchived, DbType.
Boolean);

                // Stored procedure method
                await conn.ExecuteAsync("spProduct_Insert", parameters, comm
andType: CommandType.StoredProcedure);
            }
            return true;
        }
        // Get a list of product rows (SQL Select)
        // This only works if you're already created the stored procedure.
        public async Task<IEnumerable<Product>> ProductList()
        {
            IEnumerable<Product> products;
            using (var conn = new SqlConnection(_configuration.Value))
            {
                products = await conn.QueryAsync<Product>("spProduct_List", c
ommandType: CommandType.StoredProcedure);
            }
            return products;
```

```
        }
        //Search for data (very generic...you may need to adjust.
        public async Task<IEnumerable<Product>> ProductSearch(string @Pa
ram)
        {
            var parameters = new DynamicParameters();
            parameters.Add("@Param", Param, DbType.String);
            IEnumerable<Product> products;
            using (var conn = new SqlConnection(_configuration.Value))
            {
                products = await conn.QueryAsync<Product>("spProduct_Searc
h", parameters, commandType: CommandType.StoredProcedure);
            }
            return products;
        }
        // Search based on date range. Code generator makes wild guess, you
        // will likely need to adjust field names
        public async Task<IEnumerable<Product>> ProductDateRange(DateTi
me @StartDate, DateTime @EndDate)
        {
            var parameters = new DynamicParameters();
            parameters.Add("@StartDate", StartDate, DbType.Date);
            parameters.Add("@EndDate", EndDate, DbType.Date);
            IEnumerable<Product>  products;
            using (var conn = new SqlConnection(_configuration.Value))
            {
                products = await conn.QueryAsync<Product>("spProduct_DateRa
nge", parameters, commandType: CommandType.StoredProcedure);
            }
            return products;
        }

        // Get one product based on its ProductID (SQL Select)
        // This only works if you're already created the stored procedure.
        public async Task<Product> Product_GetOne(int @ProductID)
        {
            Product product = new Product();
            var parameters = new DynamicParameters();
```

```
            parameters.Add("@ProductID", ProductID, DbType.Int32);
            using (var conn = new SqlConnection(_configuration.Value))
            {
                product = await conn.QueryFirstOrDefaultAsync<Product>("spPr
oduct_GetOne",parameters,commandType: CommandType.StoredProcedur
e);
            }
            return product;
        }
        // Update one Product row based on its ProductID (SQL Update)
        // This only works if you're already created the stored procedure.
        public async Task<bool> ProductUpdate(Product product)
        {
            using (var conn = new SqlConnection(_configuration.Value))
            {
                var parameters = new DynamicParameters();
                parameters.Add("ProductID", product.ProductID, DbType.Int32);

                parameters.Add("ProductCode", product.ProductCode, DbType.S
tring);
parameters.Add("ProductDescription", product.ProductDescription, DbTyp
e.String);
parameters.Add("ProductUnitPrice", product.ProductUnitPrice, DbType.De
cimal);
parameters.Add("ProductSupplierID", product.ProductSupplierID, DbType.I
nt32);
parameters.Add("ProductIsArchived", product.ProductIsArchived, DbType.
Boolean);

                await conn.ExecuteAsync("spProduct_Update", parameters, com
mandType: CommandType.StoredProcedure);
            }
            return true;
        }

        // Physically delete one Product row based on its ProductID (SQL Delet
e)
        // This only works if you're already created the stored procedure.
```

```
        public async Task<bool> ProductDelete(int ProductID)
        {
            var parameters = new DynamicParameters();
            parameters.Add("@ProductID", ProductID, DbType.Int32);
            using (var conn = new SqlConnection(_configuration.Value))
            {
                await conn.ExecuteAsync("spProduct_Delete",parameters, comm
andType: CommandType.StoredProcedure);
            }
            return true;
        }
    }
}

---------------------------- /Data/IProductService.cs
// This is the Product Interface
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace BlazorPurchaseOrders.Data
{
    // Each item below provides an interface to a method in ProductServices.
cs
    public interface IProductService
    {
        Task<bool> ProductInsert(Product product);
        Task<IEnumerable<Product>> ProductList();
        Task<IEnumerable<Product>> ProductSearch(string Param);
        Task<IEnumerable<Product>> ProductDateRange(DateTime @StartDa
te, DateTime @EndDate);
        Task<Product> Product_GetOne(int ProductID);
        Task<bool> ProductUpdate(Product product);
        Task<bool> ProductDelete(int ProductID);
    }
}
==========================================================
==
```

```
= With the classes finished, open up Startup.cs and
= add the services.AddScoprf linr below to the
=  public void ConfigureService{} block, perhaps just
=  above the comment and code that defines SqlConnectionConfiguration
================================================================
===

services.AddScoped<IProductService, ProductService>();
```

TABLE Supplier:

```
================================================================
====
= Stored procedures go in the database, not the app.    =
= You can copy them all into a SQL Management query and =
= select and execute them one at a time.                =
================================================================
====
---------- Stored Proc for INSERT
CREATE PROCEDURE spSupplier_Insert
--Parameters for Insert stored procedure
@SupplierName nvarchar(50),
@SupplierAddress1 nvarchar(50),
@SupplierAddress2 nvarchar(50),
@SupplierAddress3 nvarchar(50),
@SupplierPostCode nvarchar(10),
@SupplierEmail nvarchar(256),
@SupplierIsArchived bit
AS
BEGIN
```

```
--SQL for Insert stored procedure
INSERT INTO Supplier(SupplierName, SupplierAddress1, SupplierAddress2,
SupplierAddress3, SupplierPostCode, SupplierEmail, SupplierIsArchived) V
ALUES (@SupplierName, @SupplierAddress1, @SupplierAddress2, @Suppli
erAddress3, @SupplierPostCode, @SupplierEmail, @SupplierIsArchived)
END

GO

----------  Stored Proc for SELECT (LIST, just first six fields but you can
change in final code.)
CREATE PROCEDURE spSupplier_List
--No parameters required.
AS
BEGIN
--SQL for Select stored procedure.
SELECT SupplierID, SupplierName, SupplierAddress1, SupplierAddress2, S
upplierAddress3, SupplierPostCode, SupplierEmail, SupplierIsArchived FRO
M Supplier ORDER BY SupplierID DESC
END

GO

----------  Stored Proc for SELECT (one)
CREATE PROCEDURE spSupplier_GetOne
-- Needs one parameter for primary key
@SupplierID int
AS
BEGIN
-- SQL Select for one table row
SELECT SupplierID, SupplierName, SupplierAddress1, SupplierAddress2, S
upplierAddress3, SupplierPostCode, SupplierEmail, SupplierIsArchived FRO
M Supplier WHERE SupplierID= @SupplierID
END

GO

----------  Stored Proc for UPDATE
```

```
CREATE PROCEDURE spSupplier_Update
-- Parameters for Update stored procedure.
@SupplierID int,
@SupplierName nvarchar(50),
@SupplierAddress1 nvarchar(50),
@SupplierAddress2 nvarchar(50),
@SupplierAddress3 nvarchar(50),
@SupplierPostCode nvarchar(10),
@SupplierEmail nvarchar(256),
@SupplierIsArchived bit
AS
BEGIN
-- SQL for Update stored procedure
UPDATE Supplier SET SupplierName = @SupplierName, SupplierAddress1
= @SupplierAddress1, SupplierAddress2 = @SupplierAddress2, SupplierAd
dress3 = @SupplierAddress3, SupplierPostCode = @SupplierPostCode, Su
pplierEmail = @SupplierEmail, SupplierIsArchived = @SupplierIsArchived W
HERE SupplierID = @SupplierID
END

GO



==========================================================
===
Back in Visual Studio, you need to add some classes
and an interface to the Data folder, with the
names shown below.



You should have three classes, per database table, in the Data folder.

---------------------------- /DATA/Supplier.cs
using System;
using System.ComponentModel.DataAnnotations;
// This is the model for one row in the database table. You may need to mak
e some adjustments.
namespace BlazorPurchaseOrders.Data
```

```
{
    public class Supplier
    {
    [Required]
public int SupplierID { get; set; }
[Required]
[StringLength(50)]
public string SupplierName { get; set; }
[StringLength(50)]
public string SupplierAddress1 { get; set; }
[StringLength(50)]
public string SupplierAddress2 { get; set; }
[StringLength(50)]
public string SupplierAddress3 { get; set; }
[StringLength(10)]
public string SupplierPostCode { get; set; }
[StringLength(256)]
public string SupplierEmail { get; set; }
[Required]
public bool SupplierIsArchived { get; set; }

    }
}
--------------------------- /DATA/SupplierService.cs
using Dapper;
using Microsoft.Data.SqlClient;
using System;
using System.Collections.Generic;
using System.Data;
using System.Threading.Tasks;

namespace BlazorPurchaseOrders.Data
{
    public class SupplierService : ISupplierService
    {
        // Database connection
        private readonly SqlConnectionConfiguration _configuration;
        public SupplierService(SqlConnectionConfiguration configuration)
```

```
    {
        _configuration = configuration;
    }
    // Add (create) a Supplier table row (SQL Insert)
    // This only works if you're already created the stored procedure.
    public async Task<bool> SupplierInsert(Supplier supplier)
    {
        using (var conn = new SqlConnection(_configuration.Value))
        {
            var parameters = new DynamicParameters();
            parameters.Add("SupplierName", supplier.SupplierName, DbTyp
e.String);
            parameters.Add("SupplierAddress1", supplier.SupplierAddress1, D
bType.String);
            parameters.Add("SupplierAddress2", supplier.SupplierAddress2,
DbType.String);
            parameters.Add("SupplierAddress3", supplier.SupplierAddress3,
DbType.String);
            parameters.Add("SupplierPostCode", supplier.SupplierPostCode,
DbType.String);
            parameters.Add("SupplierEmail", supplier.SupplierEmail, DbType.
String);
            parameters.Add("SupplierIsArchived", supplier.SupplierIsArchive
d, DbType.Boolean);

            // Stored procedure method
            await conn.ExecuteAsync("spSupplier_Insert", parameters, comm
andType: CommandType.StoredProcedure);
        }
        return true;
    }
    // Get a list of supplier rows (SQL Select)
    // This only works if you're already created the stored procedure.
    public async Task<IEnumerable<Supplier>> SupplierList()
    {
        IEnumerable<Supplier> suppliers;
        using (var conn = new SqlConnection(_configuration.Value))
        {
```

```
            suppliers = await conn.QueryAsync<Supplier>("spSupplier_List",
commandType: CommandType.StoredProcedure);
        }
        return suppliers;
    }
    // Get one supplier based on its SupplierID (SQL Select)
    // This only works if you're already created the stored procedure.
    public async Task<Supplier> Supplier_GetOne(int @SupplierID)
    {
        Supplier supplier = new Supplier();
        var parameters = new DynamicParameters();
        parameters.Add("@SupplierID", SupplierID, DbType.Int32);
        using (var conn = new SqlConnection(_configuration.Value))
        {
            supplier = await conn.QueryFirstOrDefaultAsync<Supplier>("spS
upplier_GetOne", parameters, commandType: CommandType.StoredProce
dure);
        }
        return supplier;
    }
    // Update one Supplier row based on its SupplierID (SQL Update)
    // This only works if you're already created the stored procedure.
    public async Task<bool> SupplierUpdate(Supplier supplier)
    {
        using (var conn = new SqlConnection(_configuration.Value))
        {
            var parameters = new DynamicParameters();
            parameters.Add("SupplierID", supplier.SupplierID, DbType.Int32);

            parameters.Add("SupplierName", supplier.SupplierName, DbTyp
e.String);
            parameters.Add("SupplierAddress1", supplier.SupplierAddress1, D
bType.String);
            parameters.Add("SupplierAddress2", supplier.SupplierAddress2,
DbType.String);
            parameters.Add("SupplierAddress3", supplier.SupplierAddress3,
DbType.String);
            parameters.Add("SupplierPostCode", supplier.SupplierPostCode,
```

```
DbType.String);
            parameters.Add("SupplierEmail", supplier.SupplierEmail, DbType.
String);
            parameters.Add("SupplierIsArchived", supplier.SupplierIsArchive
d, DbType.Boolean);

            await conn.ExecuteAsync("spSupplier_Update", parameters, com
mandType: CommandType.StoredProcedure);
         }
         return true;
      }
   }
}

---------------------------- /Data/ISupplierService.cs
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace BlazorPurchaseOrders.Data
{
   // Each item below provides an interface to a method in SupplierService
s.cs
   public interface ISupplierService
   {
      Task<bool> SupplierInsert(Supplier supplier);
      Task<IEnumerable<Supplier>> SupplierList();
      Task<Supplier> Supplier_GetOne(int SupplierID);
      Task<bool> SupplierUpdate(Supplier supplier);
   }
}
==========================================================
==
= With the classes finished, open up Startup.cs and
= add the services.AddScoprf linr below to the
=  public void ConfigureService{} block, perhaps just
=  above the comment and code that defines SqlConnectionConfiguration
==========================================================
```

```
===

services.AddScoped<ISupplierService, SupplierService>();



=================================================
====
= Next comes the pages. Each goes in the Pages folder
= and each should be defined as a Razor Component.
=================================================
====



=================================================
=========
End of generated code.
```

TABLA Tax:

```


=================================================
====
= Stored procedures go in the database, not the app.    =
= You can copy them all into a SQL Management query and =
= select and execute them one at a time.                =
=================================================
====
---------- Stored Proc for INSERT
CREATE PROCEDURE spTax_Insert
--Parameters for Insert stored procedure
@TaxDescription nvarchar(50),
@TaxRate decimal(6, 4),
@TaxIsArchived bit
AS
BEGIN
```

```
--SQL for Insert stored procedure
INSERT INTO Tax(TaxDescription, TaxRate, TaxIsArchived) VALUES (@Tax
Description, @TaxRate, @TaxIsArchived)
END

----------  Stored Proc for SELECT (LIST, just first six fields but you can
change in final code.)
CREATE PROCEDURE spTax_List
--No parameters required.
AS
BEGIN
--SQL for Select stored procedure.
SELECT TOP 30 TaxID, TaxDescription, TaxRate, TaxIsArchived FROM Tax
ORDER BY TaxID DESC
END

----------  Stored Proc for SELECT (one)
CREATE PROCEDURE spTax_GetOne
-- Needs one parameter for primary key
@TaxID int
AS
BEGIN
-- SQL Select for one table row
SELECT TaxID, TaxDescription, TaxRate, TaxIsArchived FROM Tax WHERE
TaxID= @TaxID
END


----------  Stored Proc for SELECT LIKE
CREATE PROCEDURE spTax_Search
-- One parameter required to identify row to delete.
@Param varchar(128)
AS
BEGIN
-- SQL for search looking for embedded content.
SELECT TaxDescription, TaxRate, TaxIsArchived FROM Tax WHERE TaxDes
cription LIKE '%' + @param + '%' OR CAST(TaxRate AS varchar(20)) LIKE
'%' + @param + '%'
```

```
END

------------ Stored Proc for SELECT DATA RANGE
-- Another wild guess, but better than nothin'
CREATE PROCEDURE spTax_DateRange
@StartDate date,
@EndDate date
AS
BEGIN
-- SQL for search looking range of dates
SELECT TaxDescription, TaxRate, TaxIsArchived FROM Tax WHERE
END

-------------  Stored Proc for UPDATE
CREATE PROCEDURE spTax_Update
-- Parameters for Update stored procedure.
@TaxID int,
@TaxDescription nvarchar(50),
@TaxRate decimal(6, 4),
@TaxIsArchived bit
AS
BEGIN
-- SQL for Update stored procedure
UPDATE Tax SET TaxDescription = @TaxDescription, TaxRate = @TaxRate,
TaxIsArchived = @TaxIsArchived WHERE TaxID = @TaxID
END

-------------  Stored Proc for DELETE
CREATE PROCEDURE spTax_Delete
-- One parameter required to identify row to delete.
@TaxID int
AS
BEGIN
-- SQL for Delete stored procedure (physically deletes, you may want to ch
ange this to mark inactive)
DELETE FROM Tax WHERE TaxID = @TaxID
END
```

```
================================================================
===
Back in Visual Studio, you need to add some classes
and an interface to the Data folder, with the
names shown below.


You should have three classes, per database table, in the Data folder.

---------------------------- /DATA/Tax.cs
using System;
using System.ComponentModel.DataAnnotations;
// This is the model for one row in the database table. You may need to mak
e some adjustments.
namespace BlazorPurchaseOrders.Data
{
    public class Tax
    {
    [Required]
public int TaxID { get; set; }
[Required]
[StringLength(50)]
public string TaxDescription { get; set; }
[Required]
public decimal TaxRate { get; set; }
[Required]
public bool TaxIsArchived { get; set; }

    }
}

---------------------------- /DATA/TaxService.cs
// This is the service for the Tax class.
using Dapper;
using Microsoft.Data.SqlClient;
using System;
using System.Collections.Generic;
```

```
using System.Data;
using System.Threading.Tasks;

namespace BlazorPurchaseOrders.Data
{
   public class TaxService : ITaxService
   {
      // Database connection
      private readonly SqlConnectionConfiguration _configuration;
      public TaxService(SqlConnectionConfiguration configuration)
      {
         _configuration = configuration;
      }
      // Add (create) a Tax table row (SQL Insert)
      // This only works if you're already created the stored procedure.
      public async Task<bool> TaxInsert(Tax tax)
      {
         using (var conn = new SqlConnection(_configuration.Value))
         {
            var parameters = new DynamicParameters();
         parameters.Add("TaxDescription", tax.TaxDescription, DbType.Strin
g);
parameters.Add("TaxRate", tax.TaxRate, DbType.Decimal);
parameters.Add("TaxIsArchived", tax.TaxIsArchived, DbType.Boolean);

            // Stored procedure method
            await conn.ExecuteAsync("spTax_Insert", parameters, commandT
ype: CommandType.StoredProcedure);
         }
         return true;
      }
      // Get a list of tax rows (SQL Select)
      // This only works if you're already created the stored procedure.
      public async Task<IEnumerable<Tax>> TaxList()
      {
         IEnumerable<Tax> taxes;
         using (var conn = new SqlConnection(_configuration.Value))
         {
```

```
            taxes = await conn.QueryAsync<Tax>("spTax_List", commandTyp
e: CommandType.StoredProcedure);
         }
         return taxes;
      }
      //Search for data (very generic...you may need to adjust.
      public async Task<IEnumerable<Tax>> TaxSearch(string @Param)
      {
         var parameters = new DynamicParameters();
         parameters.Add("@Param", Param, DbType.String);
         IEnumerable<Tax> taxes;
         using (var conn = new SqlConnection(_configuration.Value))
         {
            taxes = await conn.QueryAsync<Tax>("spTax_Search", parameter
s, commandType: CommandType.StoredProcedure);
         }
         return taxes;
      }
      // Search based on date range. Code generator makes wild guess, you
      // will likely need to adjust field names
      public async Task<IEnumerable<Tax>> TaxDateRange(DateTime @Sta
rtDate, DateTime @EndDate)
      {
         var parameters = new DynamicParameters();
         parameters.Add("@StartDate", StartDate, DbType.Date);
         parameters.Add("@EndDate", EndDate, DbType.Date);
         IEnumerable<Tax>  taxes;
         using (var conn = new SqlConnection(_configuration.Value))
         {
            taxes = await conn.QueryAsync<Tax>("spTax_DateRange", param
eters, commandType: CommandType.StoredProcedure);
         }
         return taxes;
      }

      // Get one tax based on its TaxID (SQL Select)
      // This only works if you're already created the stored procedure.
      public async Task<Tax> Tax_GetOne(int @TaxID)
```

```
        {
            Tax tax = new Tax();
            var parameters = new DynamicParameters();
            parameters.Add("@TaxID", TaxID, DbType.Int32);
            using (var conn = new SqlConnection(_configuration.Value))
            {
                tax = await conn.QueryFirstOrDefaultAsync<Tax>("spTax_GetOn
e",parameters,commandType: CommandType.StoredProcedure);
            }
            return tax;
        }
        // Update one Tax row based on its TaxID (SQL Update)
        // This only works if you're already created the stored procedure.
        public async Task<bool> TaxUpdate(Tax tax)
        {
            using (var conn = new SqlConnection(_configuration.Value))
            {
                var parameters = new DynamicParameters();
                parameters.Add("TaxID", tax.TaxID, DbType.Int32);

                parameters.Add("TaxDescription", tax.TaxDescription, DbType.Str
ing);
parameters.Add("TaxRate", tax.TaxRate, DbType.Decimal);
parameters.Add("TaxIsArchived", tax.TaxIsArchived, DbType.Boolean);

                await conn.ExecuteAsync("spTax_Update", parameters, command
Type: CommandType.StoredProcedure);
            }
            return true;
        }

        // Physically delete one Tax row based on its TaxID (SQL Delete)
        // This only works if you're already created the stored procedure.
        public async Task<bool> TaxDelete(int TaxID)
        {
            var parameters = new DynamicParameters();
            parameters.Add("@TaxID", TaxID, DbType.Int32);
            using (var conn = new SqlConnection(_configuration.Value))
```

```
        {
            await conn.ExecuteAsync("spTax_Delete",parameters, commandT
ype: CommandType.StoredProcedure);
        }
        return true;
      }
   }
}

---------------------------- /Data/ITaxService.cs
// This is the Tax Interface
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace BlazorPurchaseOrders.Data
{
   // Each item below provides an interface to a method in TaxServices.cs
   public interface ITaxService
   {
      Task<bool> TaxInsert(Tax tax);
      Task<IEnumerable<Tax>> TaxList();
      Task<IEnumerable<Tax>> TaxSearch(string Param);
      Task<IEnumerable<Tax>> TaxDateRange(DateTime @StartDate, DateT
ime @EndDate);
      Task<Tax> Tax_GetOne(int TaxID);
      Task<bool> TaxUpdate(Tax tax);
      Task<bool> TaxDelete(int TaxID);
   }
}
==========================================================
==
= With the classes finished, open up Startup.cs and
= add the services.AddScoprf linr below to the
=  public void ConfigureService{} block, perhaps just
=  above the comment and code that defines SqlConnectionConfiguration
==========================================================
===
```

```
services.AddScoped<ITaxService, TaxService>();


================================================================
= Next comes the pages. Each goes in the Pages folder
= and each should be defined as a Razor Component.
================================================================


================================================================
End of generated code.
```