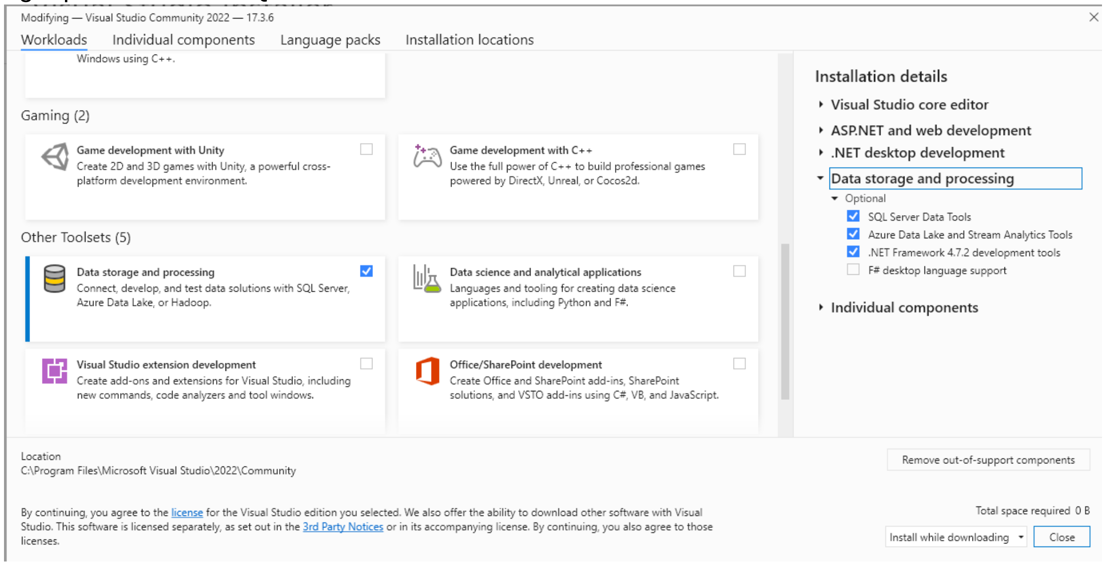
Lab Exercise 2: C# .NET 6 Class Library

For exercises 2-4, create a blogging site using ASP.NET Web API and SQL Server Database. The site must have log in, registration, and post functionality (adding a post, listing all posts, and showing details of one post).

This specific lab exercise will deal with backend functionality, using class libraries and connect through console UI with dependency injection.

The following should be downloaded:

- Visual Studio Community Edition
- SQL Server
  In Visual Studio, go to Tools > Get Tools and Features. See Data Storage and Processes in the right pane whether SQL Server Data Tools is checked.



Modifying — Visual Studio Community 2022 — 17.3.6

Workloads | Individual components | Language packs | Installation locations

Windows using C++.

Gaming (2)

Game development with Unity
Create 2D and 3D games with Unity, a powerful cross-platform development environment.

Game development with C++
Use the full power of C++ to build professional games powered by DirectX, Unreal, or Cocos2d.

Other Toolsets (5)

[✓] Data storage and processing
Connect, develop, and test data solutions with SQL Server, Azure Data Lake, or Hadoop.

Data science and analytical applications
Languages and tooling for creating data science applications, including Python and F#.

Visual Studio extension development
Create add-ons and extensions for Visual Studio, including new commands, code analyzers and tool windows.

Office/SharePoint development
Create Office and SharePoint add-ins, SharePoint solutions, and VSTO add-ins using C#, VB, and JavaScript.

Installation details
- Visual Studio core editor
- ASP.NET and web development
- .NET desktop development
- Data storage and processing
  - Optional
    - [✓] SQL Server Data Tools
    - [✓] Azure Data Lake and Stream Analytics Tools
    - [✓] .NET Framework 4.7.2 development tools
    - [ ] F# desktop language support
- Individual components

Location
C:\Program Files\Microsoft Visual Studio\2022\Community

Remove out-of-support components

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 0 B

Install while downloading | Close

Downloads per project

After setting up, right click each project and click Manage Nuget Packages. Browse for the following packages, ensuring that the 6.0 version is installed:

BlogDataLibrary (Class Library)

- Dapper
- Microsoft.Extensions.Configuration
- System.Data.SqlClient

BlogTestUI (Console App)

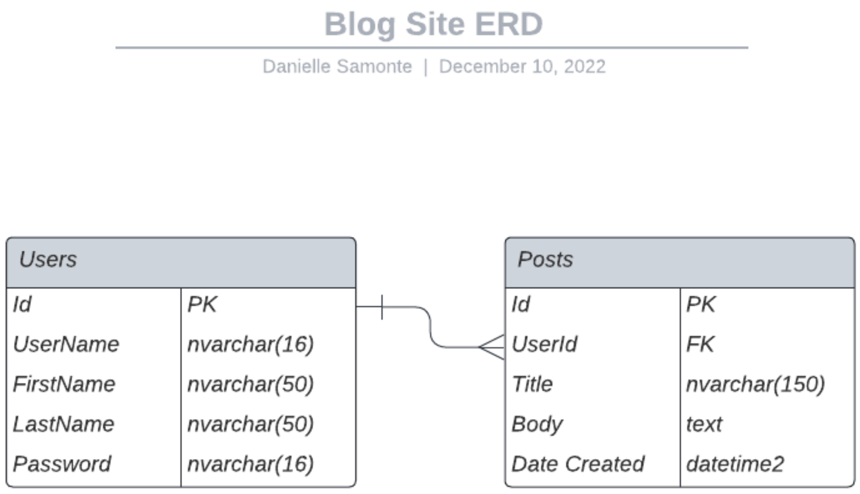
- Microsoft.Extensions.Configuration
- Microsoft.Extensions.Configuration.Json

BlogDB (SQL Server Database Project)

Setting up the project

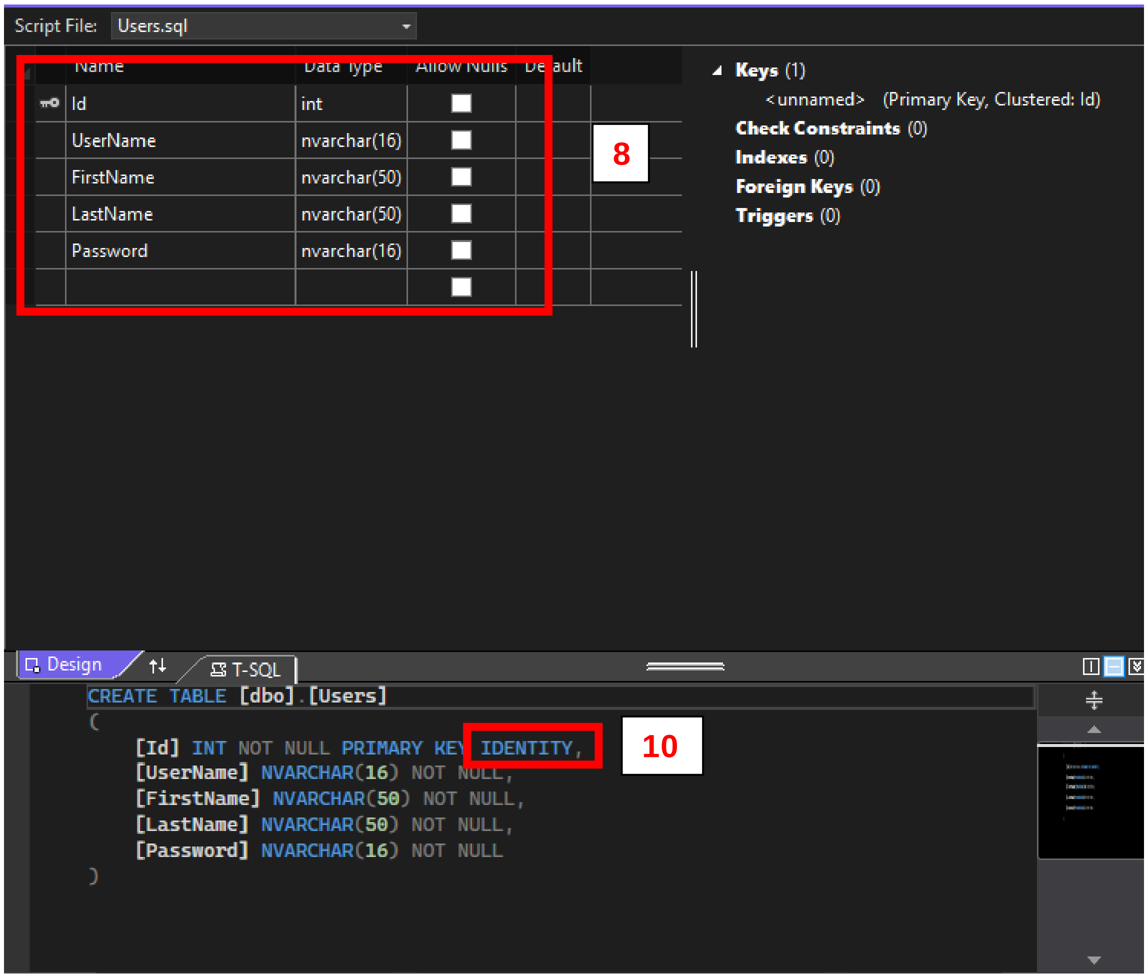
1. Open Visual Studio
2. Select Create New Project
3. Search for C# Class Library
4. Name the project as BlogDataLibrary
5. Name the Solution as LastNameLE2
6. Click Next
7. For Framework, choose .NET 6.0
8. Click Next
9. In Solution Explorer, in the right pane (View > Solution Explorer if not displayed), delete Class1.cs
10. Right click the Solution (LastNameLE2) > Add > New Project
11. Search for SQL Server Database Project
12. Name the project as LastNameBlogDB, hereon referred to as “DB Project”
13. Add another project, searching for Console App.
14. Name it as BlogTestUI
15. Use .NET 6.0, and check “Do not use top level statements”
16. Refer to the list of downloads per project

Setting up the database



1. In the DB Project, create new folder named dbo through Solution Explorer.
2. Under dbo, create a new folder named Tables.
3. Under Tables, right click > Add > Table
4. Name the first table as Users
5. Refer to the ERD for the columns and data type
   While it is not necessary to *copy* the capitalization of the column names, it is naming convention.
6. Uncheck Allow Nulls in all columns
7. Ensure in T-SQL, bottom pane, that Id has a keyword IDENTITY

   Kindly refer to the figure below for Users table configuration:



Script File: Users.sql

	Name	Data Type	Allow Nulls	Default
⚷	Id	int	☐	
	UserName	nvarchar(16)	☐	
	FirstName	nvarchar(50)	☐	
	LastName	nvarchar(50)	☐	
	Password	nvarchar(16)	☐	
			☐	

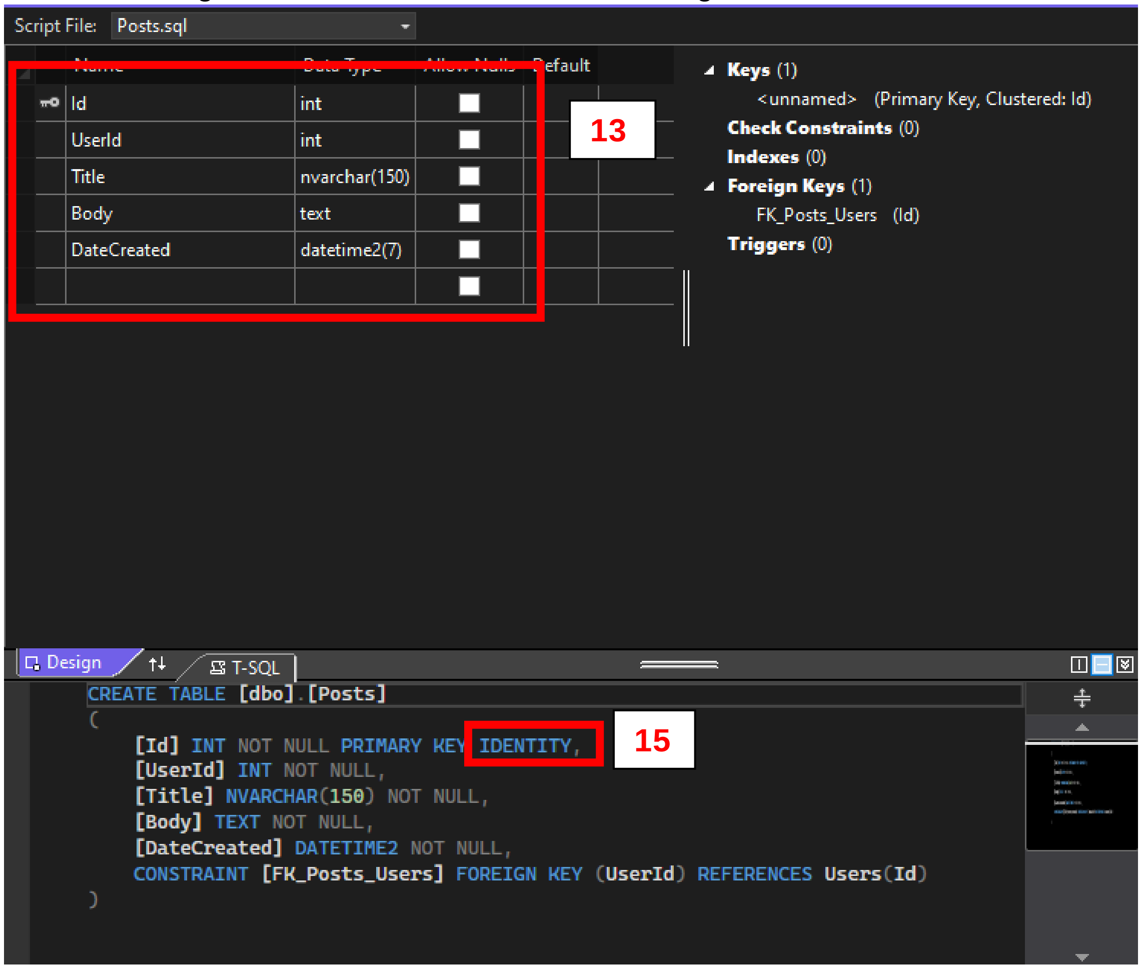
Keys (1)
<unnamed> (Primary Key, Clustered: Id)
Check Constraints (0)
Indexes (0)
Foreign Keys (0)
Triggers (0)

```
CREATE TABLE [dbo].[Users]
(
    [Id] INT NOT NULL PRIMARY KEY IDENTITY,
    [UserName] NVARCHAR(16) NOT NULL,
    [FirstName] NVARCHAR(50) NOT NULL,
    [LastName] NVARCHAR(50) NOT NULL,
    [Password] NVARCHAR(16) NOT NULL
)
```

8. Screenshot your own database design. Kindly show the Solution Explorer as well.

9. Create a new table under the same folder, named as Posts
10. Refer to the ERD for the column names and data types.
11. Uncheck Allow Nulls in all columns
12. Ensure in T-SQL, bottom pane, that Id has a keyword `IDENTITY`
13. In the pane next to the columns, right click Foreign Keys > Add New Foreign Key
14. Name it `FK_Posts_Users`
15. In T-SQL supply the following:
    Column: `UserId`
    ToTable: `Users`
    ToTableColumn: `Id`
    This will connect the Posts' UserId to Users' Id, making a one-to-many relationship.

    Refer to the figure below for the Posts table configuration:



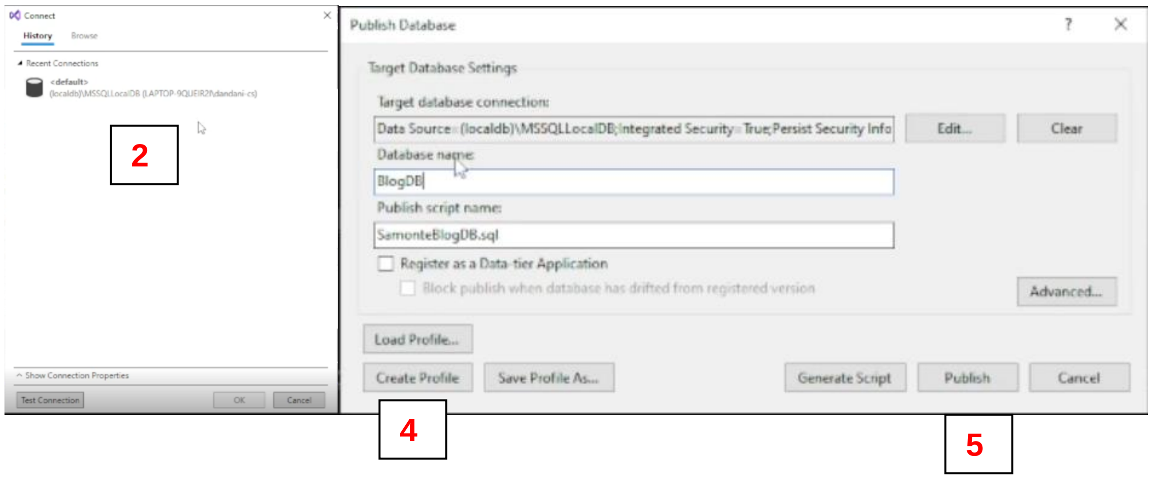
Name	Data Type	Allow Nulls	Default
Id	int	☐	
UserId	int	☐	
Title	nvarchar(150)	☐	
Body	text	☐	
DateCreated	datetime2(7)	☐	
		☐	

```
CREATE TABLE [dbo].[Posts]
(
    [Id] INT NOT NULL PRIMARY KEY IDENTITY,
    [UserId] INT NOT NULL,
    [Title] NVARCHAR(150) NOT NULL,
    [Body] TEXT NOT NULL,
    [DateCreated] DATETIME2 NOT NULL,
    CONSTRAINT [FK_Posts_Users] FOREIGN KEY (UserId) REFERENCES Users(Id)
)
```

Screenshot the database design for Posts. Ensure that the Solution Explorer is included.

Publishing the database

1. Right click the root of the DB Project and choose Publish
2. A Publish Database Pop up will show, click Edit > Browse for the MSSQL Local DB
3. Name the database as `BlogDB`, and leave the Publish script name as `LastNameBlogDB.sql`
4. Click Create Profile
5. Click Publish



Target database connection: Data Source=(localdb)\MSSQLLocalDB;Integrated Security=True;Persist Security Info

Database name: BlogDB

Publish script name: SamonteBlogDB.sql

☐ Register as a Data-tier Application

☐ Block publish when database has drifted from registered version

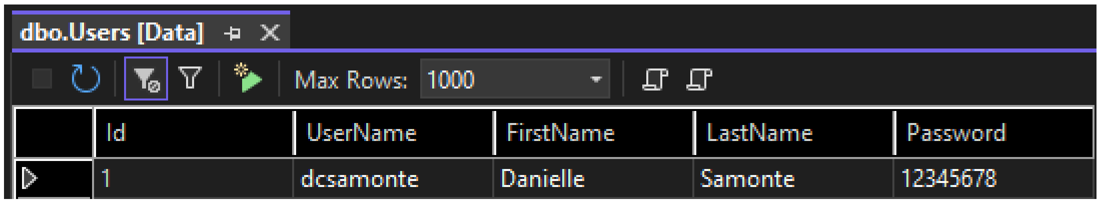
Verify whether the database is published

1. Go to View > SQL Server Object Explorer
2. Expand SQL Server > MSSQLLocalDB > Databases >
3. Ensure BlogDB is included in the list of databases

Adding rows through SQL Server Object Explorer

1. Expand BlogDB > Tables
2. Right click the Users table and choose View Data
3. Fill the first row with the following details:
   Username: `fmlastname` (f = given name initials; m: middle initial; eg:)
   Password: `Student Number`
   Do not put anything in ID. Put your first name and last name.

Sample:

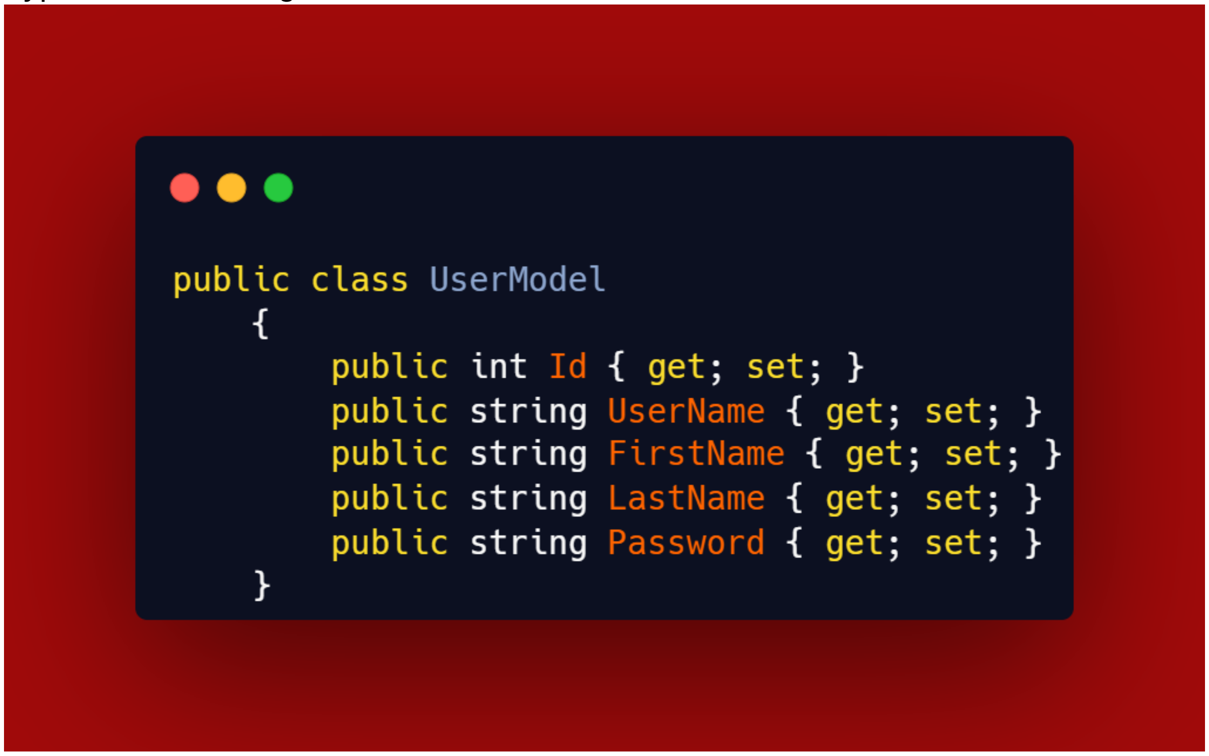


	Id	UserName	FirstName	LastName	Password
▷	1	dcsamonte	Danielle	Samonte	12345678

4. Screenshot your entry. Include the SQL Server Object Explorer pane with BlogDB expanded

Setting up Models

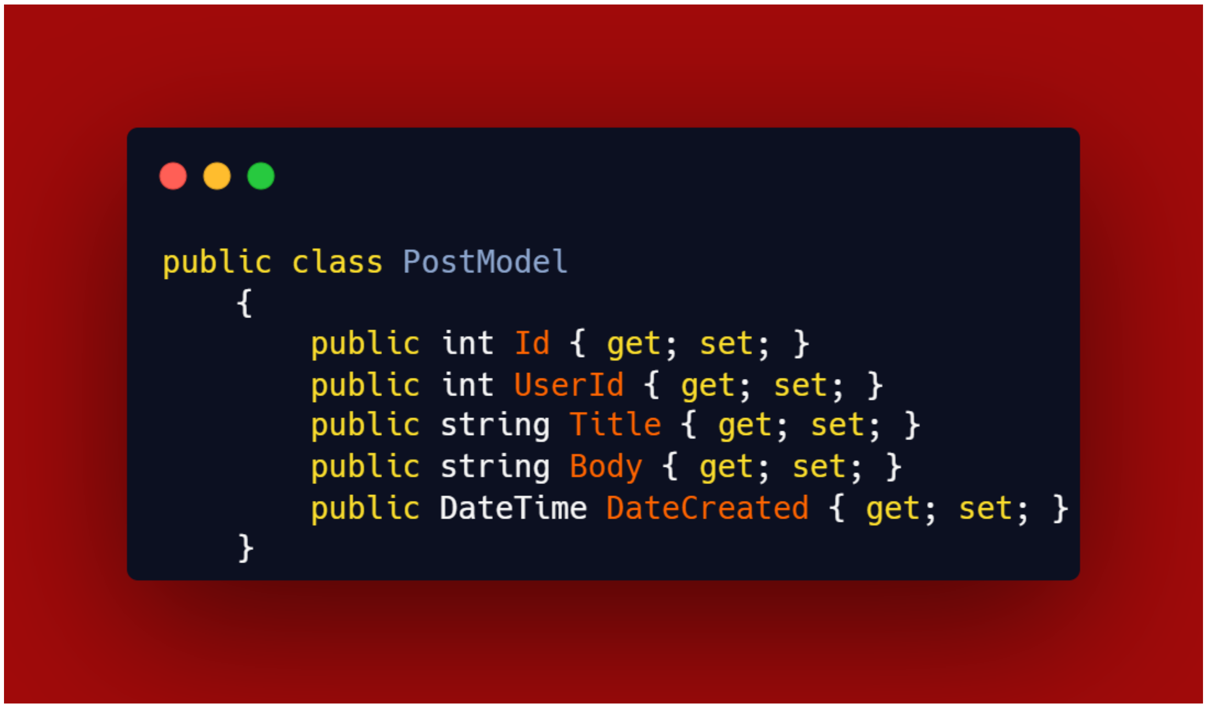
1. In the `BlogDataLibrary` project, add a new folder named `Models`
2. Under `Models`, right click and add a new class.
3. Name it `UserModel`, and ensure the class is `public`.
4. Type in the following:



```
public class UserModel
    {
        public int Id { get; set; }
        public string UserName { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Password { get; set; }
    }
```

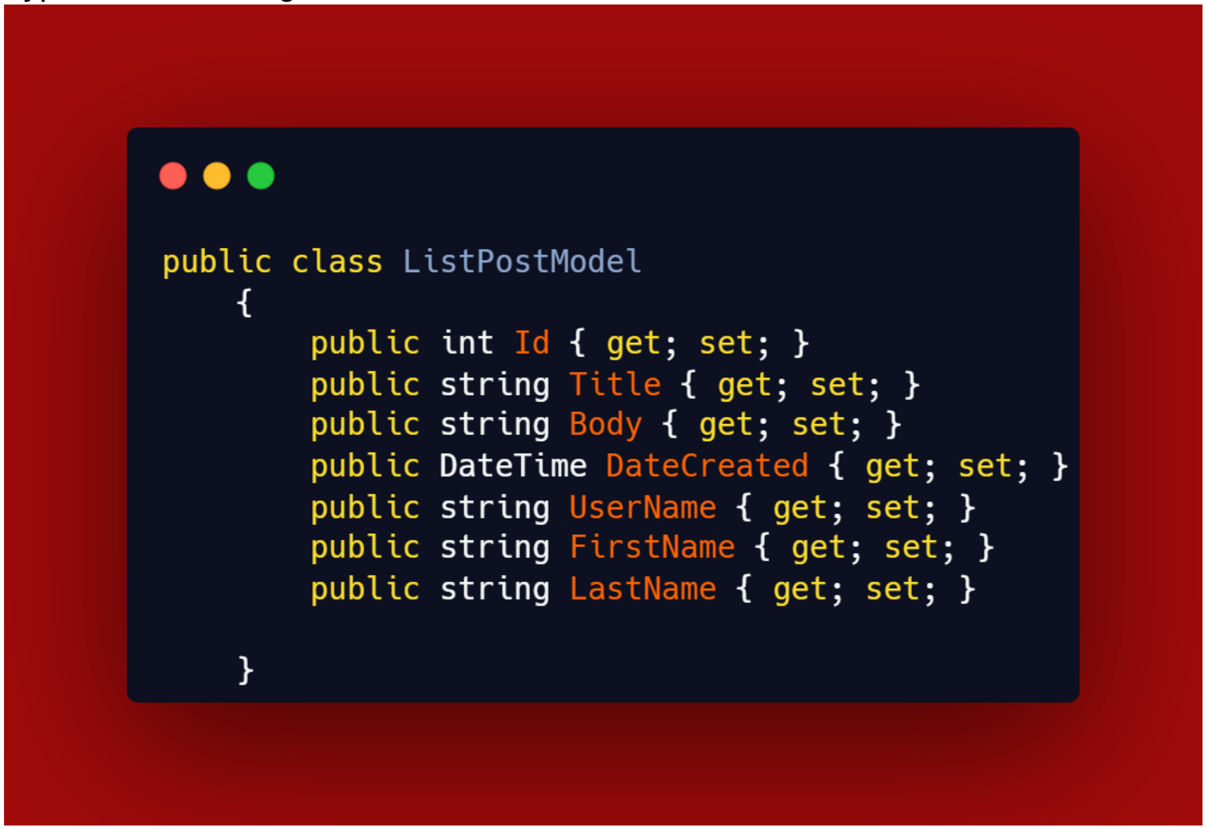
You can type in "prop" and tab twice to automate public int MyProperty and change data type and name.
Capitalization is not required, but is naming convention.

5. Create a new class named `PostModel`, and ensure the class is public.
6. Type in the following in the file:



```
public class PostModel
    {
        public int Id { get; set; }
        public int UserId { get; set; }
        public string Title { get; set; }
        public string Body { get; set; }
        public DateTime DateCreated { get; set; }
    }
```

7. Add another class named `ListPostModel`, and ensure the class is public.
8. Type in the following in the file:



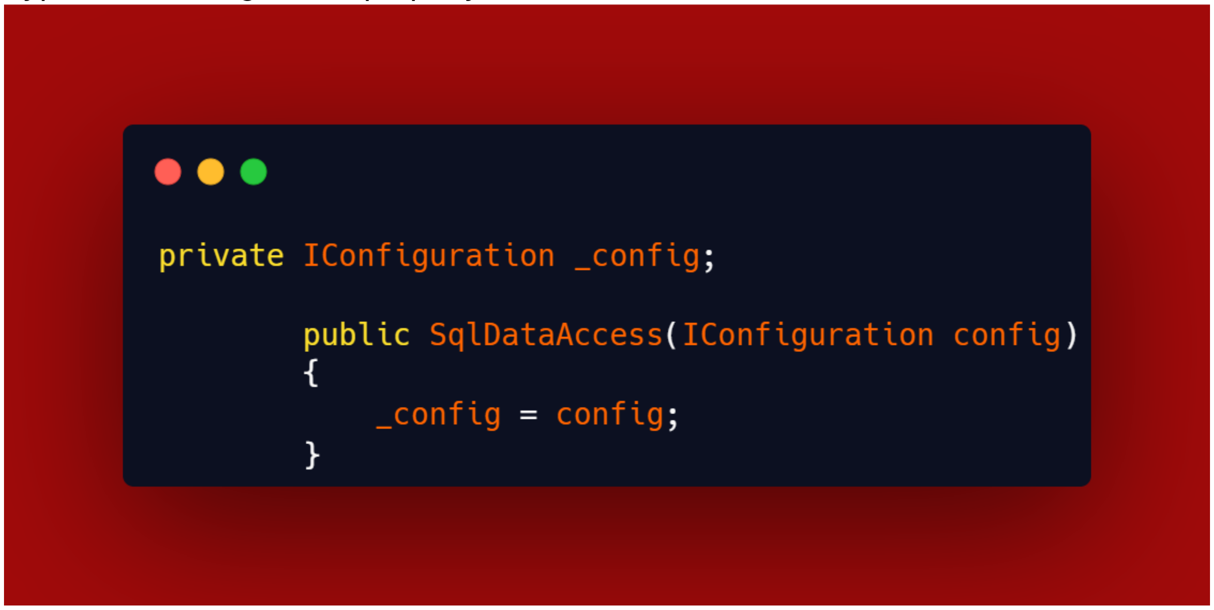
```
public class ListPostModel
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Body { get; set; }
        public DateTime DateCreated { get; set; }
        public string UserName { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }

    }
```

Setting up dependency injection

1. In the BlogDataLibary, add a new folder named Database.

Having a different project separated from the UI that interacts with the database allows us to reuse the functionality project in other UI. For this lab exercise, we'll be using a console app to test the functionality, while we can use it again for the ASP.NET Web API without needing to redo SQL commands.

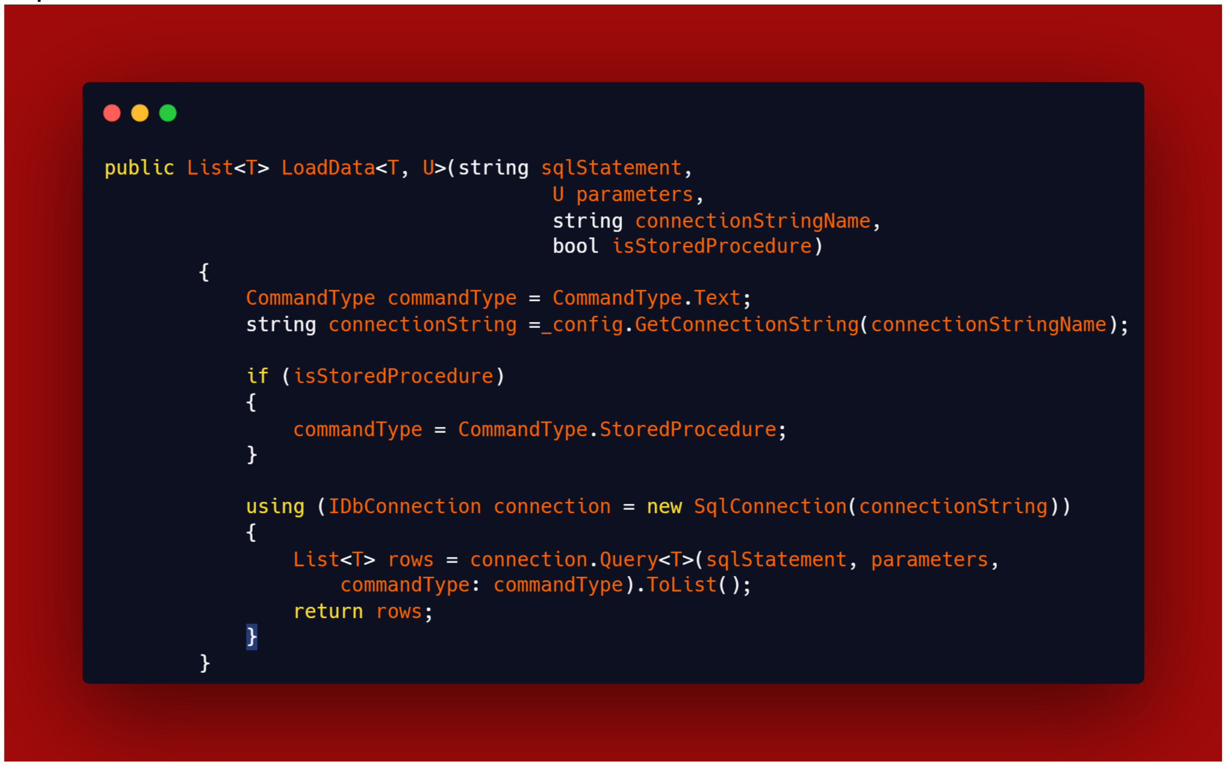
2. Create a new class called SqlDataAccess, and ensure the class is public.
3. Type in the configuration property and the constructor:



```
private IConfiguration _config;

        public SqlDataAccess(IConfiguration config)
        {
            _config = config;
        }
```

_config contains the connection string to the database. Passing it as a parameter rather than hardcoding it in the program allows it to be easily modified and therefore have loosely coupled.

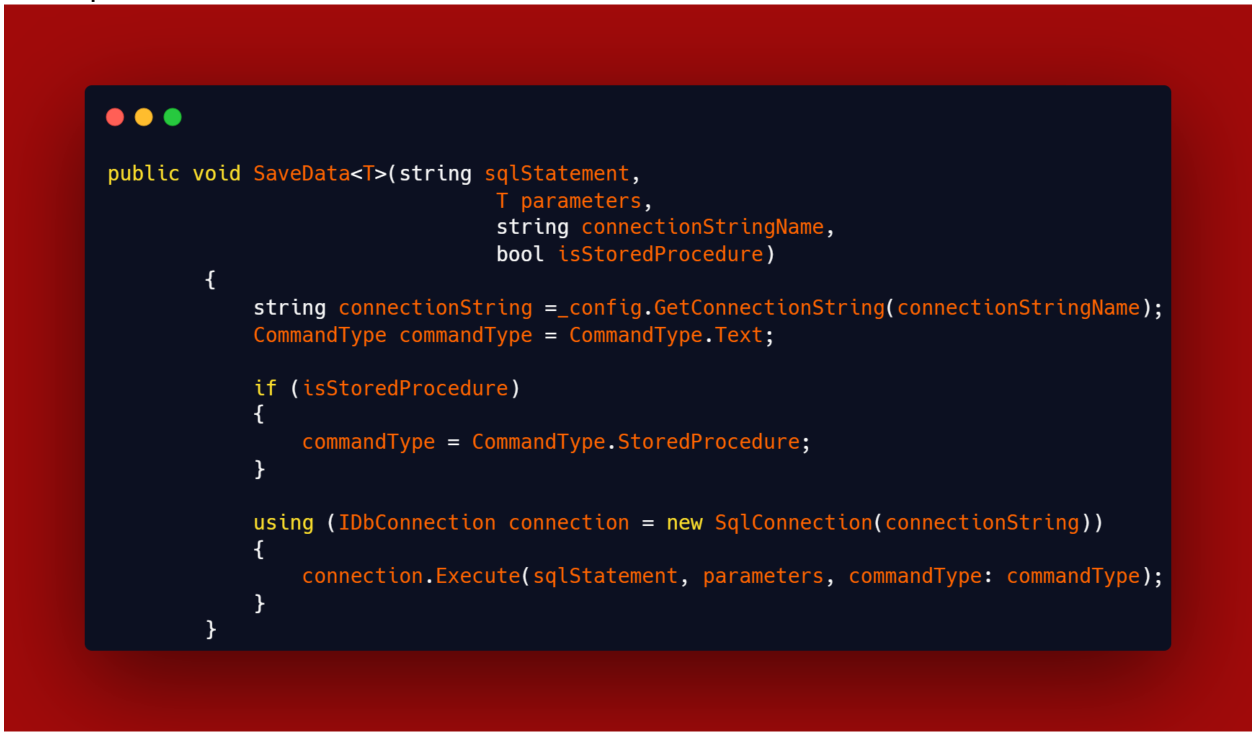
4. For SQL commands that require return of data, type in the following inside the SqlDataAccess class as another method:



```
public List<T> LoadData<T, U>(string sqlStatement,
                                      U parameters,
                                      string connectionStringName,
                                      bool isStoredProcedure)
        {
            CommandType commandType = CommandType.Text;
            string connectionString =_config.GetConnectionString(connectionStringName);

            if (isStoredProcedure)
            {
                commandType = CommandType.StoredProcedure;
            }

            using (IDbConnection connection = new SqlConnection(connectionString))
            {
                List<T> rows = connection.Query<T>(sqlStatement, parameters,
                    commandType: commandType).ToList();
                return rows;
            }
        }
```

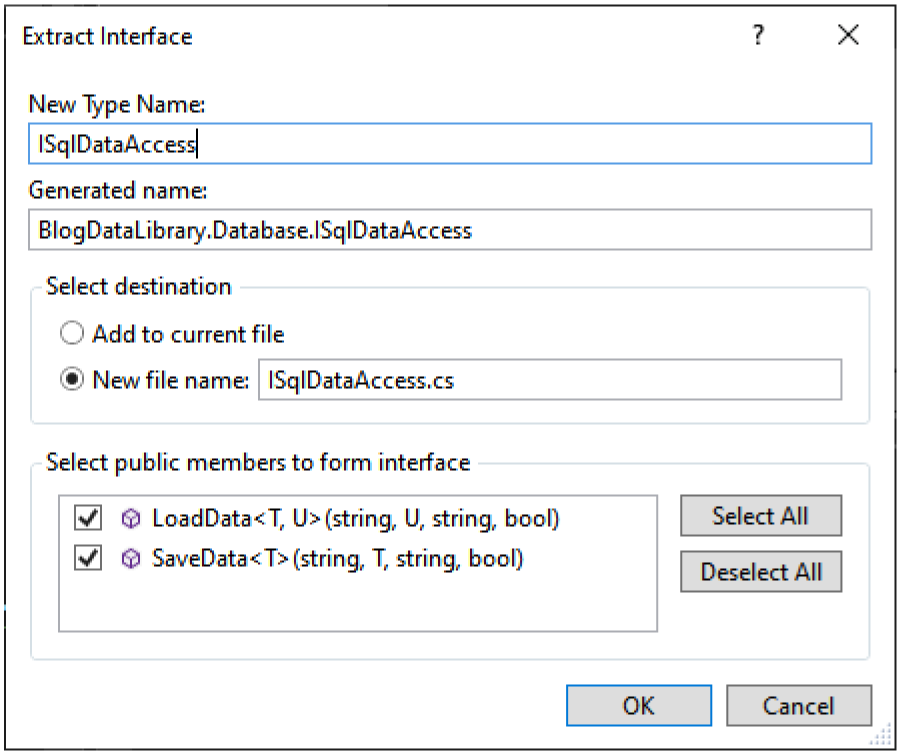
Stored procedures are a set of SQL code. They are usually used for commands that are frequently run, as the databased stores it tokenized in memory unlike queries, that has to be compiled every time.
T, U are generics, which is a concept that is a replacement of a specific data type.

5. For SQL commands that save data into the database, type in the following inside the SqlDataAccess class as another method:



```
public void SaveData<T>(string sqlStatement,
                                T parameters,
                                string connectionStringName,
                                bool isStoredProcedure)
        {
            string connectionString =_config.GetConnectionString(connectionStringName);
            CommandType commandType = CommandType.Text;

            if (isStoredProcedure)
            {
                commandType = CommandType.StoredProcedure;
            }

            using (IDbConnection connection = new SqlConnection(connectionString))
            {
                connection.Execute(sqlStatement, parameters, commandType: commandType);
            }
        }
```

6. Right click “public class SqlDataAccess” > Quick Actions and Refactoring > Extract Interface
7. Keep the default settings. For reference:


Extract Interface

New Type Name: ISqlDataAccess

Generated name: BlogDataLibrary.Database.ISqlDataAccess

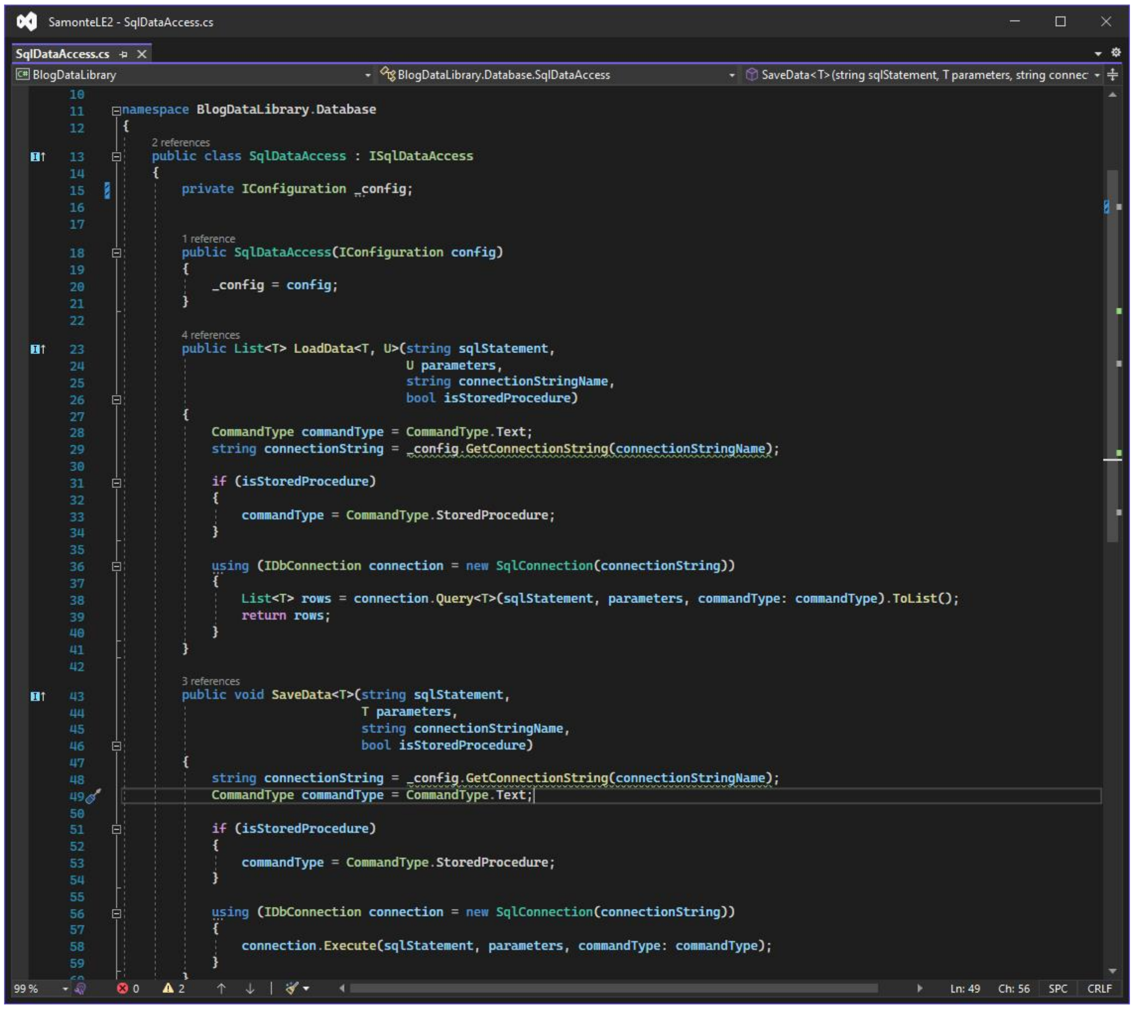
Select destination

- ○ Add to current file
- ● New file name: ISqlDataAccess.cs

Select public members to form interface

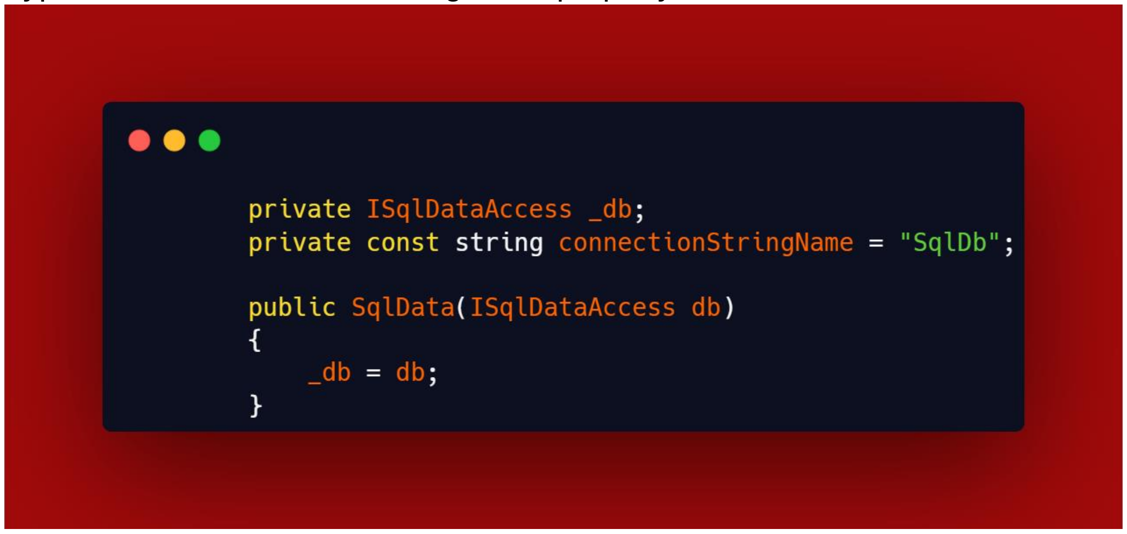
- ✓ LoadData<T, U>(string, U, string, bool)
- ✓ SaveData<T>(string, T, string, bool)

8. Your SqlDataAccess.cs should look like this:


SamonteLE2 - SqlDataAccess.cs
SqlDataAccess.cs
BlogDataLibrary
BlogDataLibrary.Database.SqlDataAccess
SaveData<T>(string sqlStatement, T parameters, string connec
99 %
0
2
Ln: 49
Ch: 56
SPC
CRLF

```csharp
namespace BlogDataLibrary.Database
{
    2 references
    public class SqlDataAccess : ISqlDataAccess
    {
        private IConfiguration _config;


        1 reference
        public SqlDataAccess(IConfiguration config)
        {
            _config = config;
        }

        4 references
        public List<T> LoadData<T, U>(string sqlStatement,
                                      U parameters,
                                      string connectionStringName,
                                      bool isStoredProcedure)
        {
            CommandType commandType = CommandType.Text;
            string connectionString = _config.GetConnectionString(connectionStringName);

            if (isStoredProcedure)
            {
                commandType = CommandType.StoredProcedure;
            }

            using (IDbConnection connection = new SqlConnection(connectionString))
            {
                List<T> rows = connection.Query<T>(sqlStatement, parameters, commandType: commandType).ToList();
                return rows;
            }
        }

        3 references
        public void SaveData<T>(string sqlStatement,
                                T parameters,
                                string connectionStringName,
                                bool isStoredProcedure)
        {
            string connectionString = _config.GetConnectionString(connectionStringName);
            CommandType commandType = CommandType.Text;

            if (isStoredProcedure)
            {
                commandType = CommandType.StoredProcedure;
            }

            using (IDbConnection connection = new SqlConnection(connectionString))
            {
                connection.Execute(sqlStatement, parameters, commandType: commandType);
            }
```

9. On the BlogDataLibrary Project, add a new folder named Data.
10. Create a new class named SqlData.
11. Type in the db, connectionStringName property and the constructor:



```csharp
private ISqlDataAccess _db;
private const string connectionStringName = "SqlDb";

public SqlData(ISqlDataAccess db)
{
    _db = db;
}
```

ISqlDataAccess is used as a parameter so that it accepts anything that implements the interface. This lab exercise uses SQL Server, but if ever other SQL DBMS is used (eg. MySQL), there's no need to change this code as long as the MySQL class implements ISqlDataAccess.

Put all screenshots of this activity with the link of your GitHub repository in a MS Word file and submit it in the Blackboard link provided.