|  |  |
| --- | --- |
|  | LAB | Dapper |
|  | WORKSHOP | Design Patterns |
|  | Crystal Tenn  crtenn@microsoft.com |

# SETUP LAB ENVRIOMENT

## Prerequisites

To perform the tasks in this lab you need following:

* Visual Studio 2015 or 2017, any version

## Objectives

In this lab you will complete following tasks:

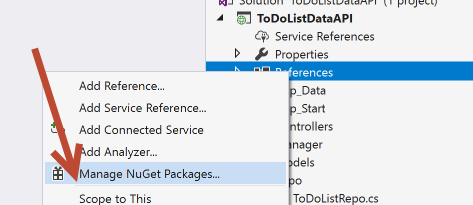
* Add Dapper to a solution and implement it.
* Understand how Dapper works
* Understand the benefits of Dapper.

### Estimated Completion Time: 30 minutes.

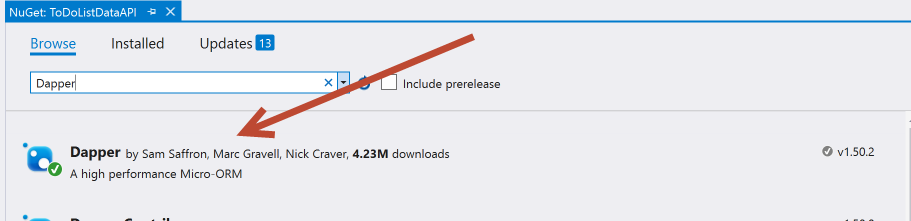
## Task: Adding Dapper to an API solution

In this task you will create a new Console application.

1. Open 01 ToDoListDataAPI [begin] where your work should have been completed from Lab 1 which was on Swagger. We will continue from this project. If you did not finish the first lab, you can either go back and work on that or open up this solution and start from here: 01 ToDoListDataAPI Swagger [end]
2. Make sure the project Builds and Runs, test out the Get All API to ensure everything is in working order before we begin.
3. Right click on the References, choose Manage NuGet packages.



1. Under Browse, type in Dapper.



1. Install the latest version of Dapper to this project.

Note: Dapper is an object-relational mapping (ORM) product for the Microsoft .NET platform: it provides a framework for mapping an object-oriented domain model to a traditional relational database. Its purpose is to relieve the developer from a significant portion of relational data persistence-related programming tasks. Dapper is free as open source software that is distributed under dual license, either the Apache License 2.0 or the MIT License.

You can use Dapper with Stored Procedures, as we will do in this example. Or there is also support for Dapper and LINQ extensions.

The following are the key features of Dapper:

* Speedy and fast performance
* Fewer lines of code
* Object mapper
* Choice of static/dynamic object binding
* Easy handling of SQL query
* Multiple query support
* Support and easy handling of stored procedures
* Operating directly on IDBConnection class
* Bulk data insert functionality

1. Navigate to the Repo folder, open the ToDoListRepo.cs file.
2. Notice how we are using ADO.NET and that it is quite a lot of code to get a little bit done. Imagine how much more code this would be in something larger than a demo with more than 2 properties!
3. Add this generic method at the top of your ToDoListRepo:

public T DbConnection<T>(Func<IDbConnection, T> getData)

{

using (SqlConnection conn = new SqlConnection())

{

conn.ConnectionString = ConfigurationManager.ConnectionStrings["todoItems"].ConnectionString;

conn.Open();

return getData(conn);

}

}

1. This will allow us to avoid repeating a lot of code. You can extract this further for large projects to deal with additional tables / many repos. But for now, this is good for one repo with one table in our demo.
2. Make sure you have all of these usings on the top of your file:

using ToDoListDataAPI.Models;

using System.Configuration;

using System.Data;

using Dapper;

1. Replace your GetToDoItems method with this:

public IEnumerable<ToDoItem> GetTodoItems()

{

return DbConnection(conn =>

conn.Query<ToDoItem>("dbo.GetItems", commandType: CommandType.StoredProcedure).ToList());

}

1. Let’s understand what is happening here. Notice the .Query comes from the Dapper namespace. You can do a Query with or without a Type, this is using Type ToDoItem. In the Query, we are passing in two parameters which are the name of the stored procedure, and then we specify and tell Dapper that it is a stored procedure.
2. Let’s replace the GetToDoItemById method next with this code:

public ToDoItem GetTodoItemById(int id)

{

return DbConnection(conn =>

conn.QuerySingle<ToDoItem>("dbo.GetItemById", new { id }, commandType: CommandType.StoredProcedure));

}

Note above this code is different. We are using QuerySingle to get one item out, instead of multiple. Query is multiple. QuerySingle is for one. We are also passing in 3 parameters, the middle highlighted parameter is the id that our stored procedure requires to find the right ToDoItem by ID. Note we have to create a new {} object, because it required a named parameter in the stored procedure. Without the new {}, this would fail and throw an error. Also note, this new { id } is equivalent to saying new { id = id } but that is redundant since we named our parameter the same name as what the stored proc requires. If in our C# method we had called it toDoId for example, the highlighted portion would need to be new { id = toDoId }.

1. Let’s replace InsertToDoItem now with:

public void InsertTodoItem(ToDoItem toDoItem)

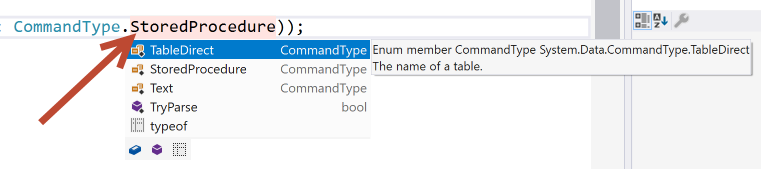
{

DbConnection(conn =>

conn.Execute("dbo.InsertItem", toDoItem, commandType: CommandType.StoredProcedure));

}

Note: We are using the Query for GET methods. Now we use the Execute method from Dapper for PUT, POST, and DELETE. We use the same format where we pass in the name of the stored proc, parameter that stored proc required, and the type (which is stored proc). Try seeing what other CommandTypes can be entered by putting your text/typing cursor to the immediate right of the period and hitting CTRL + Space.



1. Try to guess how to replace EditDescription now based on what you have learned so far. You may need to go to your SSMS and check the name of your Stored Procedure. If you are unsure or want to check your answer, it is below but the text color has been changed to white. You can either change the text color or just copy paste it into your ToDoListRepo.cs page:

public void EditDescription(ToDoItem toDoItem)

{

DbConnection(conn =>

conn.Execute("dbo.UpdateItem", toDoItem, commandType: CommandType.StoredProcedure));

}

1. And try to guess how to replace DeleteById. If you are unsure or want to check your answer, it is below but the text color has been changed to white. You can either change the text color or just copy paste it into your ToDoListRepo.cs page:

public void DeleteById(int id)

{

DbConnection(conn =>

conn.Execute("dbo.DeleteItemById", new { id }, commandType: CommandType.StoredProcedure));

}

1. You are all done editing your Repo page.
2. Run your solution and
3. Test out each of the API methods one at a time by running through the following steps: (you should know how to do this from Lab1 on Swagger, you can reference this lab if you need to for the instructions and screenshots)
   1. Get all
   2. Insert a new ToDoItem with an unused ID
   3. Get your new ToDoItem by ID
   4. Edit your ToDoItem by ID
   5. Get all
   6. Delete one by ID
   7. Get all
4. If it all works, check your SSMS and make sure everything matches up. Congrats you are done!
5. If something is not working, then check your ToDoListRepo.cs against what it should look like on the next page:

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Web;

using ToDoListDataAPI.Models;

using System.Configuration;

using System.Data;

using Dapper;

namespace ToDoListDataAPI.Repo

{

public class ToDoListRepo

{

public T DbConnection<T>(Func<IDbConnection, T> getData)

{

using (SqlConnection conn = new SqlConnection())

{

conn.ConnectionString = ConfigurationManager.ConnectionStrings["todoItems"].ConnectionString;

conn.Open();

return getData(conn);

}

}

public IEnumerable<ToDoItem> GetTodoItems()

{

return DbConnection(conn =>

conn.Query<ToDoItem>("dbo.GetItems", commandType: CommandType.StoredProcedure).ToList());

}

public void EditDescription(ToDoItem toDoItem)

{

DbConnection(conn =>

conn.Execute("dbo.UpdateItem", toDoItem, commandType: CommandType.StoredProcedure));

}

public void DeleteById(int id)

{

DbConnection(conn =>

conn.Execute("dbo.DeleteItemById", new { id }, commandType: CommandType.StoredProcedure));

}

public ToDoItem GetTodoItemById(int id)

{

return DbConnection(conn =>

conn.QuerySingle<ToDoItem>("dbo.GetItemById", new { id }, commandType: CommandType.StoredProcedure));

}

public void InsertTodoItem(ToDoItem toDoItem)

{

DbConnection(conn =>

conn.Execute("dbo.InsertItem", toDoItem, commandType: CommandType.StoredProcedure));

}

}

}