Aug 28, 2016

# ASP.NET Core with PostgreSQL and Dapper – CRUD Operations Example

This article shows how to use PostgreSQL with ASP.NET Core 1.0 using Dapper ORM. We will implement CRUD (Create, Read, Update and Delete) operations in **ASP.NET MVC** step by step.

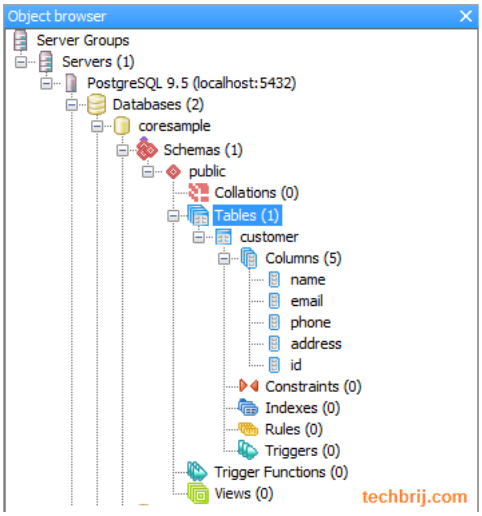

### Environment:

This article uses following software and versions:
.NET core 1.0.0
Postgresql 9.5.4
Npgsql 3.1.7
Dapper 1.50.2
Visual Studio 2015 update 3

### 1. Setup Database:

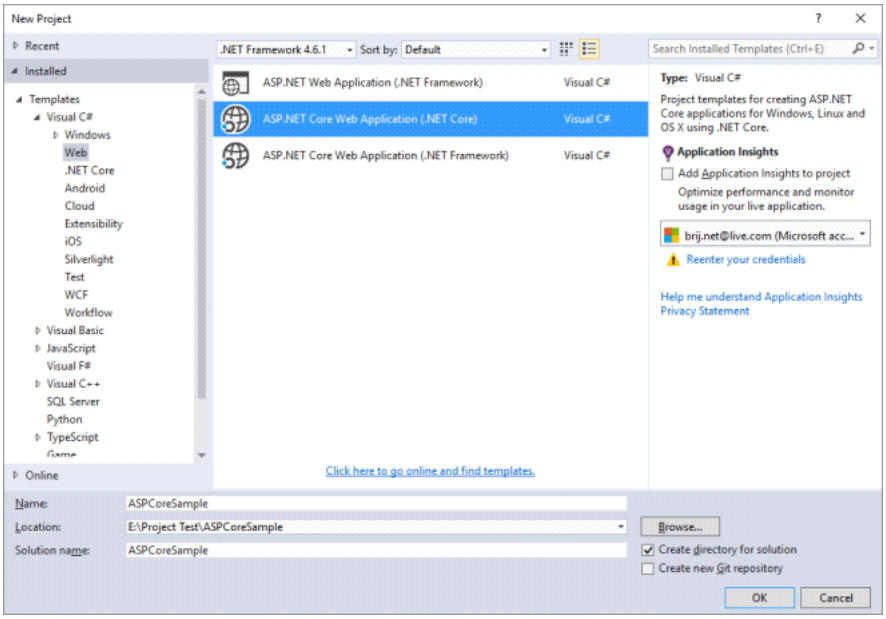
Create a new database "**coresample**" in PostgreSQL and create table using following sql

```
CREATE TABLE public.customer
(
  name text,
  email text,
  phone text,
  address text,
  id serial
)
```

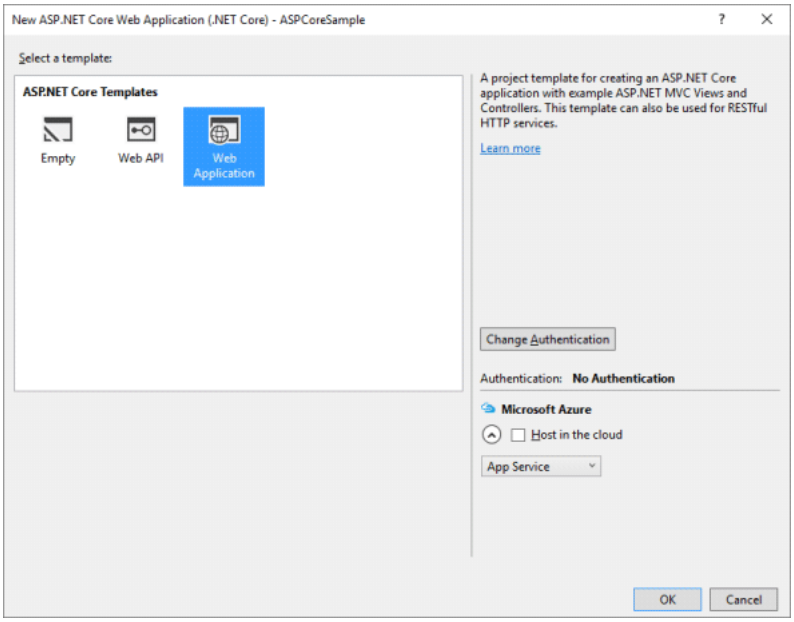



## 2. Setup Project:

Open Visual Studio > File > New Project> Select "**ASP.NET Core Web Application**" > Enter Name "**ASPCoreSample**" & Location > OK




Select "**Web Application**" template > OK




It will create web application project using ASP.NET Core. Now open package manager console and run following command to install **Npgsql** (PostgreSQL driver) and **Dapper**.

```
Install-Package Npgsql
Install-Package Dapper
```

After installation, you will get **Npgsql** and **Dapper** in project.json dependencies.

### 3. ConnectionString:

In earlier version, we define connection string in **web.config** or **app.config** and access it using ConfigurationManager. But **ASP.NET Core** allows to read settings from different sources like XML, JSON and INI files.
Open **appsettings.json** file, you will get Logging settings, we will add connectionstring after it like below:

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  },
  "DBInfo": {
    "Name": "coresample",
    "ConnectionString": "User ID=postgres;Password=xxxxxx;Host=localhost;Port=5432;Database=coresample;Pooling=tr
  }
}
```

Open **Startup.cs**, add following in **ConfigureServices** method to access generic IConfiguration:

```
services.AddSingleton<IConfiguration>(Configuration);
```

### 4. Models:

For simplicity, we are going to add different layers in the same project.
Create Models folder in solution explorer and add "**BaseEntity.cs**" class.

```
namespace ASPCoreSample.Models
{
    public abstract class BaseEntity
    {
    }
}
```

Add class "**Customer.cs**" in **Models** folder

```
using System.ComponentModel.DataAnnotations;

namespace ASPCoreSample.Models
{
    public class Customer : BaseEntity
    {
        [Key]
        public long Id { get; set; }

        [Required]
        public string Name { get; set; }

        [Required]
        public string Email { get; set; }

        [Required]
        public string Phone { get; set; }


        public string Address { get; set; }
    }
}
```

### 5. Repository:

Create **Repository** folder in solution explorer and add interface "**IRepository.cs**"

```
using ASPCoreSample.Models;
using System.Collections.Generic;

namespace ASPCoreSample.Repository
{
    public interface IRepository<T> where T : BaseEntity
    {
        void Add(T item);
        void Remove(int id);
        void Update(T item);
        T FindByID(int id);
        IEnumerable<T> FindAll();
    }
}
```

Now let's create **CustomerRepository** class in **Repository** folder

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.Extensions.Configuration;
using Dapper;
using System.Data;
using Npgsql;
using ASPCoreSample.Models;

namespace ASPCoreSample.Repository
{
    public class CustomerRepository : IRepository<Customer>
    {
        private string connectionString;
        public CustomerRepository(IConfiguration configuration)
        {
            connectionString = configuration.GetValue<string>("DBInfo:ConnectionString");
        }

        internal IDbConnection Connection
        {
            get
            {
                return new NpgsqlConnection(connectionString);
            }
        }

        public void Add(Customer item)
        {
            using (IDbConnection dbConnection = Connection)
            {
                dbConnection.Open();
                dbConnection.Execute("INSERT INTO customer (name,phone,email,address) VALUES(@Name,@Phone,@Email,
            }
```

```
        }

        public IEnumerable<Customer> FindAll()
        {
            using (IDbConnection dbConnection = Connection)
            {
                dbConnection.Open();
                return dbConnection.Query<Customer>("SELECT * FROM customer");
            }
        }

        public Customer FindByID(int id)
        {
            using (IDbConnection dbConnection = Connection)
            {
                dbConnection.Open();
                return dbConnection.Query<Customer>("SELECT * FROM customer WHERE id = @Id", new { Id = id }).Fir
            }
        }

        public void Remove(int id)
        {
            using (IDbConnection dbConnection = Connection)
            {
                dbConnection.Open();
                dbConnection.Execute("DELETE FROM customer WHERE Id=@Id", new { Id = id });
            }
        }

        public void Update(Customer item)
        {
            using (IDbConnection dbConnection = Connection)
            {
                dbConnection.Open();
                dbConnection.Query("UPDATE customer SET name = @Name,  phone  = @Phone, email= @Email, address= @
            }
        }
    }
}
```

In above code, we implemented **IRepository** methods and used **Dapper** to perform operations. To get connectionstring, the following command is used.

**configuration.GetValue("DBInfo:ConnectionString")**

As defined in appsettings.json.

### 6. Controllers:

Add new **CustomerController.cs** in **Controllers** folder

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using ASPCoreSample.Models;
using ASPCoreSample.Repository;

namespace ASPCoreSample.Controllers
{
    public class CustomerController : Controller
    {
        private readonly CustomerRepository customerRepository;

        public CustomerController(IConfiguration configuration)
        {
            customerRepository = new CustomerRepository(configuration);
        }


        public IActionResult Index()
        {
            return View(customerRepository.FindAll());
        }

        public IActionResult Create()
        {
            return View();
        }

        // POST: Customer/Create
        [HttpPost]
        public IActionResult Create(Customer cust)
        {
            if (ModelState.IsValid)
```

```
            {
                customerRepository.Add(cust);
                return RedirectToAction("Index");
            }
            return View(cust);

        }

        // GET: /Customer/Edit/1
        public IActionResult Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            Customer obj = customerRepository.FindByID(id.Value);
            if (obj == null)
            {
                return NotFound();
            }
            return View(obj);

        }

        // POST: /Customer/Edit
        [HttpPost]
        public IActionResult Edit(Customer obj)
        {

            if (ModelState.IsValid)
            {
                customerRepository.Update(obj);
                return RedirectToAction("Index");
            }
            return View(obj);
        }

        // GET:/Customer/Delete/1
        public IActionResult Delete(int? id)
        {

            if (id == null)
            {
                return NotFound();
            }
            customerRepository.Remove(id.Value);
            return RedirectToAction("Index");
        }
    }
}
```

Repository object is created in constructor.

### 7. Views:

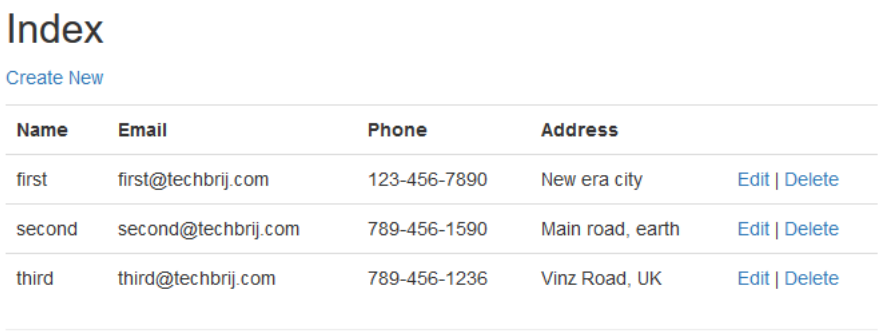
Add **Customer** folder in **Views** folder and add Index, Create and Edit views.

**Index.cshtml:**

```
@model IEnumerable<ASPCoreSample.Models.Customer>

@{
    ViewData["Title"] = "Index";
}

<h2>Index</h2>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Name)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Email)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Phone)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Address)
        </th>
        <th></th>
```

```
    </tr>

    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Email)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Phone)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Address)
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
                <a asp-action="Delete" asp-route-id="@item.Id" onclick="return confirm('Are sure wants to delete
            </td>
        </tr>
    }
</table>
```

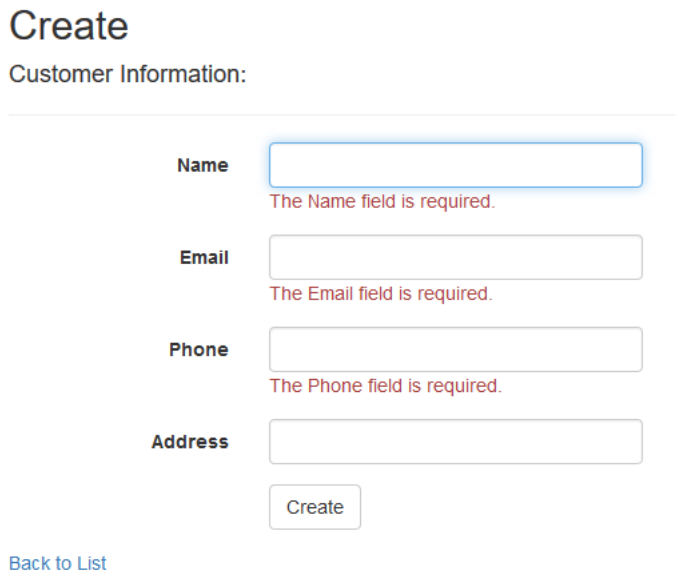
It will show Grid of Customer data.




**Create.cshtml:**

```
@model ASPCoreSample.Models.Customer

@{
    ViewData["Title"] = "Create";
}

<h2>Create</h2>

<form asp-action="Create">
    <div class="form-horizontal">
        <h4>Customer Information:</h4>
        <hr />
        <div asp-validation-summary="ModelOnly" class="text-danger"></div>
        <div class="form-group">
            <label asp-for="Name" class="col-md-2 control-label"></label>
            <div class="col-md-10">
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger" />
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Email" class="col-md-2 control-label"></label>
            <div class="col-md-10">
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger" />
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Phone" class="col-md-2 control-label"></label>
            <div class="col-md-10">
                <input asp-for="Phone" class="form-control" />
                <span asp-validation-for="Phone" class="text-danger" />
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Address" class="col-md-2 control-label"></label>
            <div class="col-md-10">
                <input asp-for="Address" class="form-control" />
                <span asp-validation-for="Address" class="text-danger" />
            </div>
```

```
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
</form>

<div>
    <a asp-action="Index">Back to List</a>
</div>

@section Scripts {
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
}
```

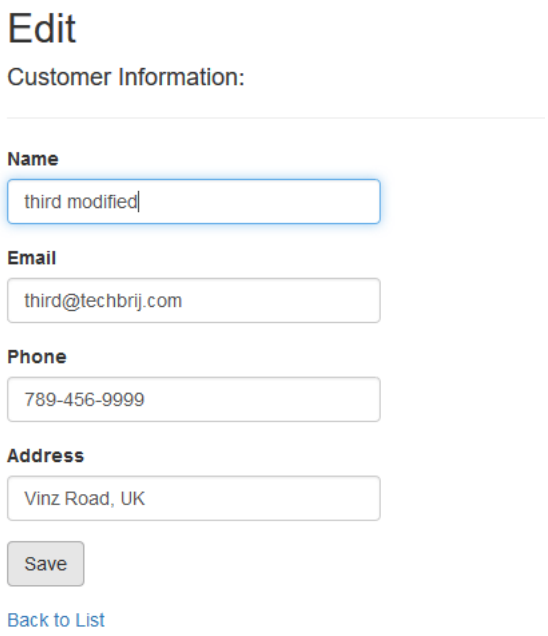



**Edit.cshtml:**

```
@model ASPCoreSample.Models.Customer

@{
    ViewData["Title"] = "Create";
}

<h2>Edit</h2>

<form asp-action="Edit">
    <div class="form-horizontal">
        <h4>Customer Information:</h4>
        <hr />
        <div asp-validation-summary="ModelOnly" class="text-danger"></div>
        <div class="form-group">
            <label asp-for="Name" class="col-md-2 control-label"></label>
            <div class="col-md-10">
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger" />
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Email" class="col-md-2 control-label"></label>
            <div class="col-md-10">
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger" />
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Phone" class="col-md-2 control-label"></label>
            <div class="col-md-10">
                <input asp-for="Phone" class="form-control" />
                <span asp-validation-for="Phone" class="text-danger" />
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Address" class="col-md-2 control-label"></label>
            <div class="col-md-10">
```

```
                <input asp-for="Address" class="form-control" />
                <span asp-validation-for="Address" class="text-danger" />
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </div>
    </div>
</form>

<div>
    <a asp-action="Index">Back to List</a>
</div>

@section Scripts {
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
}
```

The logic is same but Tag helpers are provided for easy and quick development.




To add **Customer** option in menu bar, open **_Layout.cshtml** in **Views\Shared** folder and add following line after Contact menu

```
<li><a asp-area="" asp-controller="Customer" asp-action="Index">Customer</a></li>
```

Run the application and start adding/editing/deleting Customers.

### Source Code:

**Download Code**

### Conclusion:

In this post, we implemented CRUD operations in **ASP.NET Core** with **PostgreSQL** database using **Dapper** ORM. Basically it covers many things like how to configure connectionstring in .NET core, how to connect PostgreSQL database using Npgsql, how Dapper is used, new tag helpers in Razor views...etc.

Hope, It helps. Enjoy **ASP.NET Core** !!