

**Tristan Cole McDonald**

**Student Number: 20116247**

**PATHWAY: CLDV6211**

**Lecturer: Isaac Leshaba**

**05 July 2021**

**POE**

# Table of Contents

Table of Figures .....	3
<b>1. Specifications .....</b>	<b>4</b>
1.1. Database Specifications for Domingo Roof Works.....	4
1.2. Web App Specifications for Domingo Roof Works .....	8
1.3. How a user can navigate around and perform the different functions in the Domingo Roof Works Web Application. ....	16
<b>2. Self-Evaluation.....</b>	<b>22</b>
2.1. What were the most and least beneficial aspects of this course? .....	22
2.2. How did each of the learning units contribute to your knowledge of web development? .....	22
2.3. How did the knowledge gained from each of the learning units assist you in completing the tasks in this POE? .....	22
2.4. Do you think you will use the skills in this course in your career? If yes, explain how. ....	23
2.5. If you were to rewind life one semester, what would you do differently in this course and what would you do the same? .....	23
<b>3. Link To Web App (Domingo Roof Works).....</b>	<b>24</b>
References.....	25

## Table of Figures

Figure 1: Domingo Roof Works ERD.....	4
Figure 2: Table creation.....	5
Figure 3: Table creation 2 .....	5
Figure 4: Table creation 3. ....	6
Figure 5: Inserting data into tables.....	6
Figure 6: Inserting data into tables 2 .....	7
Figure 7: Inserting data into tables 3.....	7
Figure 8: Inserting data into tables 4. ....	8
Figure 9: Employee model example. ....	9
Figure 10: EmployeeDAL example. ....	10
Figure 11: EmployeeDAL example 2.....	10
Figure 12: EmployeeDAL example 3.....	11
Figure 13: EmployeeDAL example 4.....	11
Figure 14: Employee controller example. ....	12
Figure 15: Employee controller example 2. ....	13
Figure 16: Employee controller example 3. ....	13
Figure 17: Employee controller example 4. ....	14
Figure 18: Employee controller example 5. ....	14
Figure 19: Employee create view example.....	15
Figure 20: Domingo home page.....	16
Figure 21: Employees page and add button. ....	17
Figure 22: Add new employee page.....	17
Figure 23: Edit button example.....	18
Figure 24: Edit page example.....	18
Figure 25: Details button example. ....	19
Figure 26: Details page example.....	19
Figure 27: Delete button example. ....	20
Figure 28: Delete page example. ....	21

# 1. Specifications

## 1.1. Database Specifications for Domingo Roof Works

The database for Domingo Roof Works was created in Microsoft SQL Server Management Studio using the SQL language.

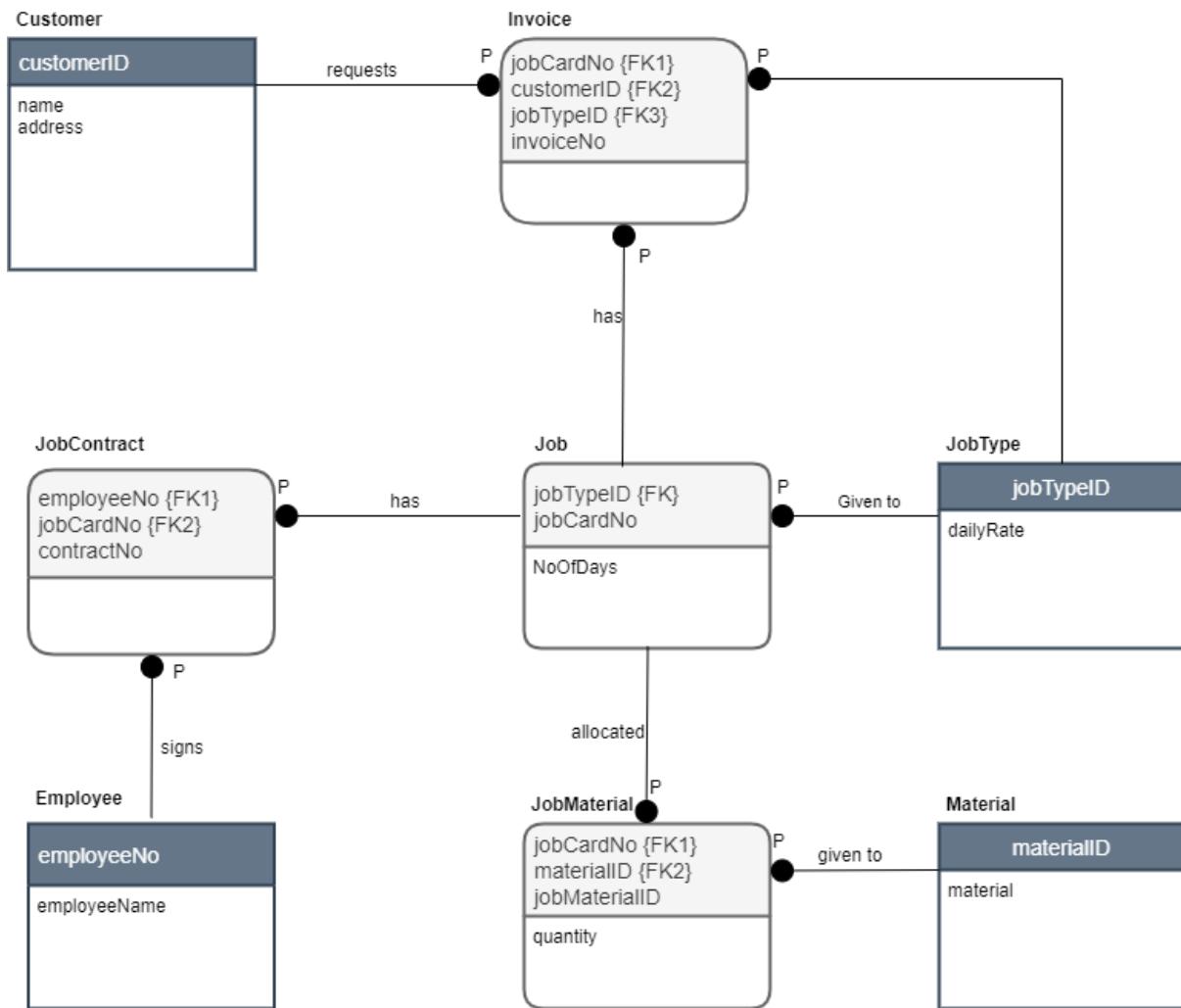


Figure 1: Domingo Roof Works ERD

Figures 2, 3, and 4 below illustrate how the 8 tables(entities) for the database were created, with reference to (Chao, 2014) and (Tik, 2021). Each table has a primary key and the tables (child entities) which contain foreign keys (primary keys from parent entities) are illustrated in figure 1 (above) with rounded corners.

```
7 -- CREATING TABLES (Chao,2014).
8
9
10 CREATE TABLE customer(
11     customerID INT IDENTITY(1,1) NOT NULL,
12     name VARCHAR(50) NOT NULL,
13     address VARCHAR (150) NOT NULL,
14     PRIMARY KEY (customerID)
15 );
16
17 CREATE TABLE employee(
18     employeeNo VARCHAR(6) NOT NULL,
19     employeeName VARCHAR(50) NOT NULL,
20     PRIMARY KEY (employeeNo)
21 );
22
23 CREATE TABLE jobType(
24     jobTypeID INT IDENTITY(1,1) NOT NULL,
25     jobType VARCHAR(20) NOT NULL,
26     dailyRate DECIMAL NOT NULL,
27     PRIMARY KEY (jobTypeID)
28 );
29
```

Figure 2: Table creation.

```
29
30 CREATE TABLE material(
31     materialID INT IDENTITY(1,1) NOT NULL,
32     material VARCHAR(150) NOT NULL,
33     PRIMARY KEY (materialID)
34 );
35
36 CREATE TABLE job(
37     jobCardNo INT NOT NULL,
38     jobTypeID INT NOT NULL,
39     NoOfDays INT NOT NULL,
40     PRIMARY KEY (jobCardNo),
41     FOREIGN KEY (jobTypeID) REFERENCES jobType(jobTypeID)
42 );
43
44 CREATE TABLE jobMaterial(
45     jobMaterialID INT IDENTITY(1,1) NOT NULL,
46     jobCardNo INT NOT NULL,
47     materialID INT NOT NULL,
48     quantity VARCHAR(50) NOT NULL,
49     PRIMARY KEY (jobMaterialID),
50     FOREIGN KEY (jobCardNo) REFERENCES job(jobCardNo)
51 );
```

Figure 3: Table creation 2.

```

52
53  CREATE TABLE jobContract(
54      contractNo INT IDENTITY(1,1) NOT NULL,
55      jobCardNo INT NOT NULL,
56      employeeNo VARCHAR(6) NOT NULL,
57      PRIMARY KEY (contractNo),
58      FOREIGN KEY (jobCardNo) REFERENCES job(jobCardNo),
59      FOREIGN KEY (employeeNo) REFERENCES employee(employeeNo)
60  );
61
62  CREATE TABLE invoice(
63      invoiceNo INT IDENTITY(1,1) NOT NULL,
64      jobCardNo INT NOT NULL,
65      customerID INT NOT NULL,
66      jobTypeID INT NOT NULL,
67      PRIMARY KEY (invoiceNo),
68      FOREIGN KEY (jobCardNo) REFERENCES job(jobCardNo),
69      FOREIGN KEY (customerID) REFERENCES customer(customerID),
70      FOREIGN KEY (jobTypeID) REFERENCES jobType(jobTypeID)
71  );
72
73  -- SHOWING FINISHED TABLES (Chao, 2014).
74

```

Figure 4: Table creation 3.

Figures 5 to 8 below illustrate how the predetermined data (given by Domingo Roof Works) was inserted into the database tables that were created above.

```

100  -- POPULATING DATA FOR ALL JOB CARDS AND DETAILS OF THOSE JOBS (Chao,2014).
101
102  INSERT INTO jobType (jobType, dailyRate)
103      VALUES ('Full Conversion', 1200.00),
104      ('Semi Conversion', 1080.00),
105      ('Floor Boarding', 900.00);
106
107  INSERT INTO job (jobCardNo, jobTypeID, NoOfDays)
108      VALUES (11000, 1, 7),
109      (10478, 2, 2),
110      (14253, 3, 2),
111      (11258, 1, 8),
112      (12058, 2, 3),
113      (13697, 1, 7),
114      (10211, 1, 7),
115      (10471, 2, 2),
116      (13521, 2, 3),
117      (10102, 3, 2);
118
119  INSERT INTO customer(name, address)
120      VALUES('Tendai Ndoro', '3 Leos Place 457 Church Str PRETORIA, 0002'),
121      ('Donald Puttingh', '408 Oubos 368 Prinsloo Street PRETORIA, 0001'),
122      ('Tracy Samson', '206 Albertros 269 Stead Avenue PRETORIA, 0186');

```

Figure 5: Inserting data into tables.

DomingoDB.sql - domingodb.database.windows.net master (Tristan (85)) - Microsoft SQL Server Management Studio

```
177 -- POPULATING DATA ON MATERIALS THAT ARE USED ON A JOB CARD (Chao, 2014).
178
179 INSERT INTO material(material)
180 VALUES('Standard floorboards'),
181 ('Power points'),
182 ('Standard electrical wiring'),
183 ('Standard stairs pack');
184
185 INSERT INTO jobMaterial(jobCardNo, materialID, quantity)
186 VALUES
187 -- JOB 11000.
188 (11000, 1, '90'),
189 (11000, 2, '3'),
190 (11000, 3, '20 metres'),
191 (11000, 4, '1'),
192
193 -- JOB 10478.
194 (10478, 1, '50'),
195 (10478, 2, '1'),
196 (10478, 3, '10 metres'),
197
198 -- JOB 14253.
199 (14253, 1, '40'),
```

Figure 6: Inserting data into tables 2

DomingoDB.sql - domingodb.database.windows.net master (Tristan (85)) - Microsoft SQL Server Management Studio

```
148 -- POPULATING DATA FOR EMPLOYEES (Chao, 2014).
149
150
151 INSERT INTO employee(employeeNo, employeeName)
152 VALUES('EMP100', 'Albert Malose'),
153 ('EMP920', 'Chris Byne'),
154 ('EMP010', 'John Hendriks'),
155 ('EMP771', 'Smallboy Modipa'),
156 ('EMP681', 'Stanley Jacobs');
157
158 -- ASSIGNING EMPLOYEES TO JOB CARDS (Chao, 2014).
159
160 INSERT INTO jobContract(jobCardNo, employeeNo)
161 VALUES(11000, 'EMP100'),
162 (11000, 'EMP920'),
163 (11000, 'EMP010'),
164 (10478, 'EMP920'),
165 (14253, 'EMP771'),
166 (11258, 'EMP681'),
167 (11258, 'EMP010'),
168 (11258, 'EMP771'),
169 (12058, 'EMP681');
```

Figure 7: Inserting data into tables 3.

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar reads "DomingoDB.sql - domingodb.database.windows.net master (Tristan (85)) - Microsoft SQL Server Management Studio". The main area displays a query window titled "DomingoDB.sql - do...ster (Tristan (85))". The code in the query window is as follows:

```

121 ('Donald Puttingh', '408 Oubos 368 Prinsloo Street PRETORIA, 0001'),
122 ('Tracy Samson', '206 Albertros 269 Stead Avenue PRETORIA, 0186'),
123 ('Jacob Smith', 'A201 Overton 269 Debouvlrde Str PRETORIA, 0002'),
124 ('Thato Molopo', '11 Lutting Court 289 MALTZAN Str PRETORIA, 0001'),
125 ('Dakolo Mudau', '1182 CEBINIA Str PRETORIA, 0082'),
126 ('Sifiso Myeni', '503 Hamilton Gardens 337 Visagie Str PRETORIA, 0001'),
127 ('Ricardo Keyla', '10 Silville 614 Jasmyn Str PRETORIA, 0184'),
128 ('Smallboy Mtshali', '307 FEORA East PRETORIA-WEST, 0183'),
129 ('Wilson Jansen', '701 Monticchico Flat251 Jacob Mare Str PRETORIA, 0002');
130
131 INSERT INTO invoice(jobCardNo, customerID, jobTypeID)
132 VALUES(11000, 1, 1),
133 (10478, 2, 2),
134 (14253, 3, 3),
135 (11258, 4, 1),
136 (12058, 5, 2),
137 (13697, 6, 1),
138 (10211, 7, 1),
139 (10471, 8, 2),
140 (13521, 9, 2),
141 (10102, 10, 3);
142
143 -- LISTING OF THE JOB TABLE (Chao, 2014).

```

The status bar at the bottom indicates "Connected (1/1)" and "Ln 159 Col 1 Ch 1 INS". The taskbar below shows various open applications like File Explorer, Mail, and Microsoft Edge.

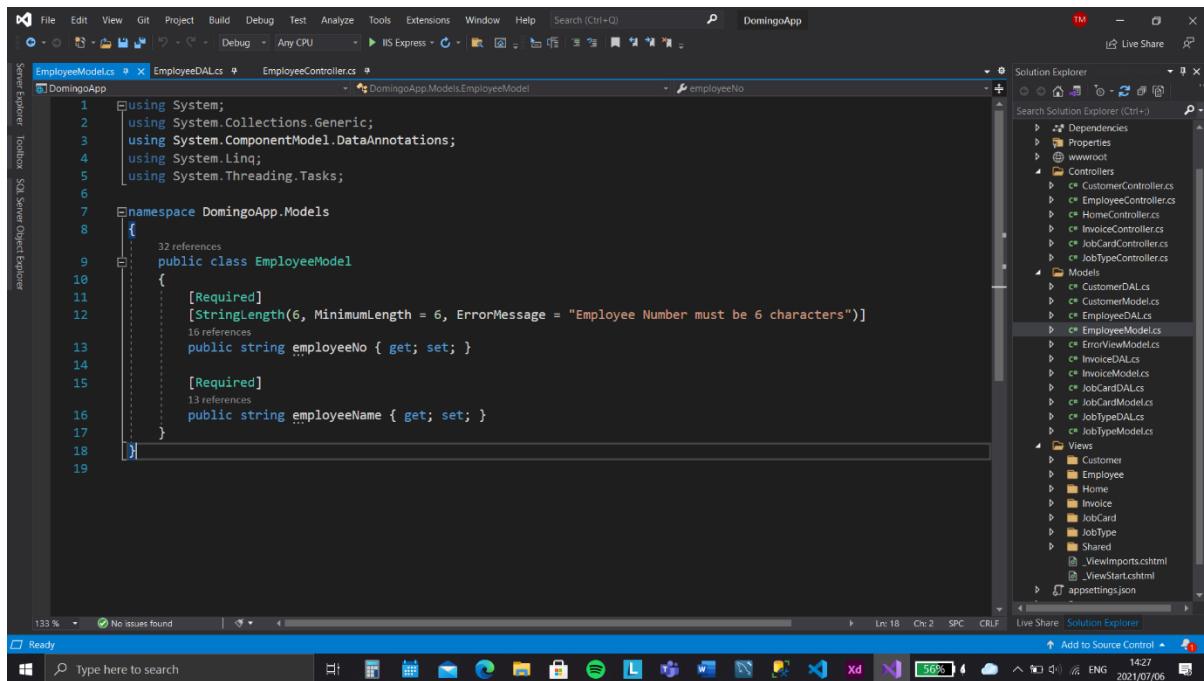
Figure 8: Inserting data into tables 4.

With reference to (Chao, 2014), (Tik, 2021), and (w3schools.com, 2021), various Stored Procedures were created for the Domingo Roof Works Web Application to Insert, read data, update data, and delete data to and from the Domingo Roof Works database. The scripts for these Stored Procedures are included in the POE folder and the document from task 2 (included in the Task 2 Revised folder) provides an in-depth illustration of how this database was created and manipulated.

## 1.2. Web App Specifications for Domingo Roof Works

The Web Application for Domingo Roof Works was created in Visual Studio Using the ASP.NET Core Model View Controller (MVC) Framework. This Web App is hosted in the Microsoft Azure cloud and has been connected to the database which is also hosted in the Microsoft Azure cloud.

With reference to (Andrew Troelsen & Philip Japikse, 2017), (IAmTimCorey, 2018), and (Tik, 2021), a Model for each required table(entity) in the database has been created which serves as a template for the data transferred between the database tables and the Web Application. An example of a model created for an employee is in Figure 9 shown below. Note that all other models follow this or a similar structure.



The screenshot shows the Microsoft Visual Studio IDE interface. The left pane displays the code for `EmployeeModel.cs` in the `DomingoApp` project. The code defines a class `EmployeeModel` with two properties: `employeeNo` and `employeeName`, both annotated with `[Required]` and `[StringLength(6, MinimumLength = 6, ErrorMessage = "Employee Number must be 6 characters")]`. The right pane shows the Solution Explorer with various files like `CustomerController.cs`, `EmployeeController.cs`, `InvoiceController.cs`, etc., and the `Models` folder containing `EmployeeModel.cs`. The bottom status bar shows the date as 2021/07/06 and the time as 14:27.

```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations;
4  using System.Linq;
5  using System.Threading.Tasks;
6
7  namespace DomingoApp.Models
8  {
9      public class EmployeeModel
10     {
11         [Required]
12         [StringLength(6, MinimumLength = 6, ErrorMessage = "Employee Number must be 6 characters")]
13         public string employeeNo { get; set; }
14
15         [Required]
16         public string employeeName { get; set; }
17     }
18 }
19

```

Figure 9: Employee model example.

Utilizing information from (Tik, 2021) and (Andrew Troelsen & Philip Japikse, 2017), a Data Access Layer (DAL) for each table(Model) has also been created which allows the Web Application to Insert, Read, Update, or Delete data to or from the Domingo database. For the application to perform the Create, Read, Update, and Delete (CRUD) functions with the database each function required in the POE has a stored procedure created for it in the database (See Task 2 Revision Folder for the Stored Procedures). An example of the DAL for an employee is provided in Figures 10 to 13 below. Note that all other DAL's follow this or a similar structure.

Figure 10: EmployeeDAL example.

Figure 11: EmployeeDAL example 2.

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the `EmployeeDAL.cs` file under the `DomingoApp` project. The code implements methods for adding, updating, and deleting employees using stored procedures. The `AddEmployee` method takes an `EmployeeModel` object and inserts it into the database. The `UpdateEmployee` method takes an `EmployeeModel` object and updates the corresponding row in the database. The `Delete` method takes an employee number and deletes the record from the database. The `Solution Explorer` pane on the right shows the project structure with various models, controllers, and views.

```
60 //Get Employee By ID
61
62     public EmployeeModel GetEmployeeByEmployeeNo(string employeeNo)
63     {
64         EmployeeModel emp = new EmployeeModel();
65         using (SqlConnection con = new SqlConnection(connectionStringPROD))
66         {
67             SqlCommand cmd = new SqlCommand("SP_GetEmployeeByEmployeeNo", con);
68             cmd.CommandType = CommandType.StoredProcedure;
69
70             cmd.Parameters.AddWithValue("@employeeNo", employeeNo);
71             con.Open();
72             SqlDataReader dr = cmd.ExecuteReader();
73             while (dr.Read())
74             {
75                 emp.employeeNo = dr["employeeNo"].ToString();
76                 emp.employeeName = dr["employeeName"].ToString();
77             }
78             con.Close();
79         }
80         return emp;
81     }
82
83     //Update Employee
84
85     public void UpdateEmployee(EmployeeModel emp)
86     {
87         using (SqlConnection con = new SqlConnection(connectionStringPROD))
88         {
89             SqlCommand cmd = new SqlCommand("SP_UpdateEmployee", con);
90             cmd.CommandType = CommandType.StoredProcedure;
91
92             cmd.Parameters.AddWithValue("@employeeNo", emp.employeeNo);
93             cmd.Parameters.AddWithValue("@employeeName", emp.employeeName);
94
95             con.Open();
96             cmd.ExecuteNonQuery();
97             con.Close();
98         }
99
100    //Delete Employee
101
102    public void Delete(string employeeNo)
103    {
104        using (SqlConnection con = new SqlConnection(connectionStringPROD))
105        {
106            SqlCommand cmd = new SqlCommand("SP_DeleteEmployee", con);
107            cmd.CommandType = CommandType.StoredProcedure;
108
109            cmd.Parameters.AddWithValue("@employeeNo", employeeNo);
110
111            con.Open();
112            cmd.ExecuteNonQuery();
113            con.Close();
114        }
115    }
116 }
```

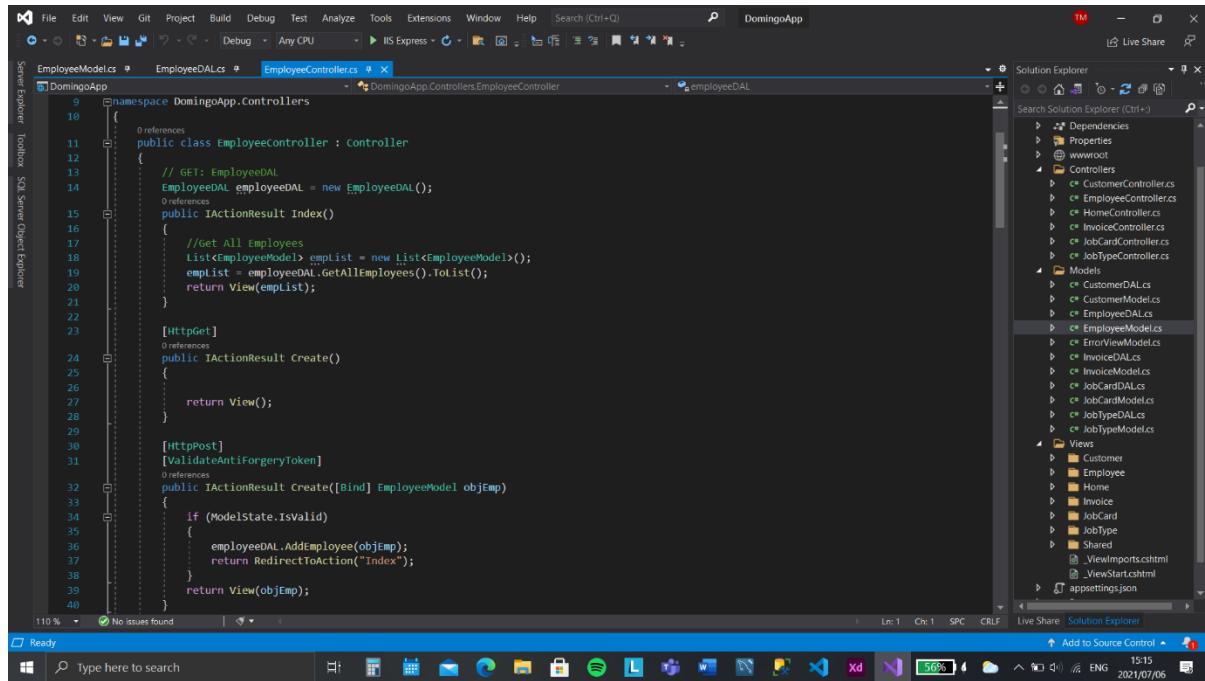
Figure 12: EmployeeDAL example 3.

This screenshot shows the same Microsoft Visual Studio environment as Figure 12, but with different code in the `EmployeeDAL.cs` file. The `UpdateEmployee` method now includes a `cmd.Parameters.AddWithValue("@employeeName", emp.employeeName);` line. The `Delete` method has been modified to use a non-query command (`cmd.ExecuteNonQuery();`) instead of a delete command. The rest of the code remains the same, including the stored procedure names and connection logic.

```
83 //Update Employee
84
85     public void UpdateEmployee(EmployeeModel emp)
86     {
87         using (SqlConnection con = new SqlConnection(connectionStringPROD))
88         {
89             SqlCommand cmd = new SqlCommand("SP_UpdateEmployee", con);
90             cmd.CommandType = CommandType.StoredProcedure;
91
92             cmd.Parameters.AddWithValue("@employeeNo", emp.employeeNo);
93             cmd.Parameters.AddWithValue("@employeeName", emp.employeeName);
94
95             con.Open();
96             cmd.ExecuteNonQuery();
97             con.Close();
98         }
99
100    //Delete Employee
101
102    public void Delete(string employeeNo)
103    {
104        using (SqlConnection con = new SqlConnection(connectionStringPROD))
105        {
106            SqlCommand cmd = new SqlCommand("SP_DeleteEmployee", con);
107            cmd.CommandType = CommandType.StoredProcedure;
108
109            cmd.Parameters.AddWithValue("@employeeNo", employeeNo);
110
111            con.Open();
112            cmd.ExecuteNonQuery();
113            con.Close();
114        }
115    }
116 }
```

Figure 13: EmployeeDAL example 4.

Using information provided by (Tik, 2021) and (Andrew Troelsen & Philip Japikse, 2017), a controller for each table(Model) has been created which handles the requests sent by the user to the web application. With reference to (tutorialspoint.com, 2021), the controllers for each of the functions perform various duties requested by the user such as returning a details page, displaying error pages when necessary, and returning other pages(views) associated with the CRUD operations when requested. The controllers also work in conjunction with the DAL's to submit or receive data to or from the database and display the outcomes. An example of the controller for an employee is provided in Figures 14 to 18 below. Note that all other controller's follow this or a similar structure.



The screenshot shows the Microsoft Visual Studio interface with the following details:

- Code Editor:** Displays the `EmployeeController.cs` file under the namespace `DomingoApp.Controllers`. The code implements the `IController` interface and contains methods for listing employees and creating new employees.
- Solution Explorer:** Shows the project structure for `DomingoApp`, including the `Controllers`, `Models`, and `Views` folders, along with their respective sub-files.
- Status Bar:** Shows the current zoom level (110%), issue count (No issues found), and system status (Ready).
- Taskbar:** Shows the Windows taskbar with various pinned icons.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using DomingoApp.Models;
using DomingoApp.DAL;

namespace DomingoApp.Controllers
{
    public class EmployeeController : Controller
    {
        // GET: Employee
        EmployeeDAL employeeDAL = new EmployeeDAL();

        public ActionResult Index()
        {
            //Get All Employees
            List<EmployeeModel> empList = new List<EmployeeModel>();
            empList = employeeDAL.GetAllEmployees().ToList();
            return View(empList);
        }

        [HttpGet]
        public ActionResult Create()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create([Bind] EmployeeModel objEmp)
        {
            if (ModelState.IsValid)
            {
                employeeDAL.AddEmployee(objEmp);
                return RedirectToAction("Index");
            }
            return View(objEmp);
        }
    }
}

```

Figure 14: Employee controller example.

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `EmployeeController.cs`. The code contains two action methods: `Details` and `Edit`. Both methods use `[HttpGet]` attribute. The `Details` method retrieves an employee by `employeeNo` from the `employeeDAL` and returns a view. The `Edit` method retrieves an employee by `employeeNo` from the `employeeDAL`, checks if it's null, and then returns a view. The `Solution Explorer` pane on the right shows the project structure with files like `CustomerController.cs`, `EmployeeController.cs`, `InvoiceController.cs`, etc.

```
43     //Get Employee Details
44     [Route("Employee/Details/{employeeNo}")]
45     [HttpGet]
46     public ActionResult Details(string employeeNo)
47     {
48         if (employeeNo == null)
49         {
50             return NotFound();
51         }
52
53         EmployeeModel emp = employeeDAL.GetEmployeeByEmployeeNo(employeeNo);
54
55         if (emp == null)
56         {
57             return NotFound();
58         }
59
60         return View(emp);
61     }
62
63     //Edit Employees
64     [Route("Employee/Edit/{employeeNo}")]
65     [HttpGet]
66     public ActionResult Edit(string employeeNo)
67     {
68         if (employeeNo == null)
69         {
70             return NotFound();
71         }
72
73         EmployeeModel emp = employeeDAL.GetEmployeeByEmployeeNo(employeeNo);
74
75         if (emp == null)
76         {
77             return NotFound();
78         }
79
80         return View(emp);
81     }
```

Figure 15: Employee controller example 2.

This screenshot shows the same `EmployeeController.cs` file as Figure 15, but with additional code added. The `Edit` method now includes validation logic using `[HttpPost]` and `[ValidateAntiForgeryToken]`. It checks if the `employeeNo` is null and if the `ModelState` is valid. If both conditions are met, it updates the employee in the `employeeDAL` and redirects to the index action. The rest of the code remains the same as in Figure 15.

```
63     //Edit Employees
64     [Route("Employee/Edit/{employeeNo}")]
65     [HttpGet]
66     public ActionResult Edit(string employeeNo)
67     {
68         if (employeeNo == null)
69         {
70             return NotFound();
71         }
72
73         EmployeeModel emp = employeeDAL.GetEmployeeByEmployeeNo(employeeNo);
74
75         if (emp == null)
76         {
77             return NotFound();
78         }
79
80         return View(emp);
81     }
82
83     //Update Employee
84     [Route("Employee/Edit/{employeeNo}")]
85     [HttpPost]
86     [ValidateAntiForgeryToken]
87     public ActionResult Edit(string employeeNo, [Bind] EmployeeModel objEmp)
88     {
89         if (employeeNo == null)
90         {
91             return NotFound();
92         }
93
94         if (ModelState.IsValid)
95         {
96             employeeDAL.UpdateEmployee(objEmp);
97             return RedirectToAction("Index");
98         }
99     }
```

Figure 16: Employee controller example 3.

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for the `EmployeeController.cs` file. The code implements methods for updating and deleting employees. It uses `[HttpPost]` and `[ValidateAntiForgeryToken]` attributes. It also includes logic to check if the employee exists and to validate the model state. The Solution Explorer on the right shows the project structure with various controllers, models, and views.

```
82     //Update Employee
83     [Route("Employee/Edit/{employeeNo}")]
84     [HttpPost]
85     [ValidateAntiForgeryToken]
86     public IActionResult Edit(string employeeNo, [Bind] EmployeeModel objEmp)
87     {
88         if (employeeNo == null)
89         {
90             return NotFound();
91         }
92
93         if (ModelState.IsValid)
94         {
95             employeeDAL.UpdateEmployee(objEmp);
96             return RedirectToAction("Index");
97         }
98         return View(employeeDAL);
99     }
100
101    //Get Delete View
102    [Route("Employee/Delete/{employeeNo}")]
103    public IActionResult Delete(string employeeNo)
104    {
105        if (employeeNo == null)
106        {
107            return NotFound();
108        }
109        EmployeeModel emp = employeeDAL.GetEmployeeByEmployeeNo(employeeNo);
110
111        if (emp == null)
112        {
113            return NotFound();
114        }
115        return View(emp);
116    }
```

Figure 17: Employee controller example 4.

This screenshot shows the same Visual Studio environment as Figure 17, but with more code in the `EmployeeController.cs` file. It includes a new method for performing a delete operation via POST. This method uses `[HttpPost, ActionName("Delete")]` and `[ValidateAntiForgeryToken]`. It calls the `DeleteEmployee` method from the `employeeDAL` and then redirects to the index action.

```
99
100
101    //Get Delete View
102    [Route("Employee/Delete/{employeeNo}")]
103    public IActionResult Delete(string employeeNo)
104    {
105        if (employeeNo == null)
106        {
107            return NotFound();
108        }
109        EmployeeModel emp = employeeDAL.GetEmployeeByEmployeeNo(employeeNo);
110
111        if (emp == null)
112        {
113            return NotFound();
114        }
115        return View(emp);
116    }
117
118    //Perform Delete
119    [Route("Employee/Delete/{employeeNo}")]
120    [HttpPost, ActionName("Delete")]
121    [ValidateAntiForgeryToken]
122
123    public IActionResult DeleteEmployee(string employeeNo)
124    {
125        employeeDAL.Delete(employeeNo);
126        return RedirectToAction("Index");
127    }
128
129
130 }
```

Figure 18: Employee controller example 5.

Finally, with reference to (Tik, 2021) and (IAmTimCorey, 2018), a few views were created for each of the CRUD operations and error pages required in each of the tables(Models). The views utilize a mixture of HTML, CSS, JavaScript, and C# which is called Razor pages (IAmTimCorey, 2018). The front-end styling/presentation was created using these Razor page views, and each page provided a functionality for the user. An example of 1 of the 4 views created for the employee table(Model) is shown in Figure 19 below. Note that all other views follow this or a similar structure.

The screenshot shows the Microsoft Visual Studio interface. The left pane displays the code for 'Create.cshtml':

```

1 @model DomingoApp.Models.EmployeeModel
2
3 ViewData["Title"] = "Create";
4 Layout = "~/views/Shared/_Layout.cshtml";
5
6
7
8 <div class="container">
9     <div class="row">
10         <div class="col-md-12">
11             <h2 class="text-center">Insert New Employee</h2>
12             <div class="row">
13                 <div class="col-md-4"></div>
14                 <div class="col-md-4">
15                     <form asp-action="Create">
16                         <div asp-validation-summary="ModelOnly" class="text-danger"></div>
17
18                         <div class="form-group">
19                             <label asp-for="employeeNo" class="control-label"></label>
20                             <input asp-for="employeeNo" class="form-control" />
21                             <span asp-validation-for="employeeNo" class="text-danger"></span>
22                         </div>
23                         <div class="form-group">
24                             <label asp-for="employeeName" class="control-label"></label>
25                             <input asp-for="employeeName" class="form-control" />
26                             <span asp-validation-for="employeeName" class="text-danger"></span>
27                         </div>
28                         <div class="form-group">
29                             <input type="submit" value="Create" class="btn btn-success" />
30                             <a asp-action="Index" class="btn btn-warning">Cancel</a>
31                         </div>
32                     </form>
33
34
35
36             </div>
37         </div>
38     </div>
39 </div>

```

The right pane shows the 'Solution Explorer' with the project structure:

- Models
  - CustomerDAL.cs
  - CustomerModel.cs
  - EmployeeDAL.cs
  - EmployeeModel.cs
  - ErrorViewModel.cs
  - InvoiceDAL.cs
  - InvoiceModel.cs
  - JobCardDAL.cs
  - JobCardModel.cs
  - JobTypeDAL.cs
  - JobTypeModel.cs
- Views
  - Customer
  - Employee
    - Create.cshtml
    - Delete.cshtml
    - Details.cshtml
    - Edit.cshtml
    - Index.cshtml
  - Home
  - Invoice
  - JobCard
  - JobType
  - Shared
    - \_ViewImports.cshtml
    - \_ViewStart.cshtml
- Properties
  - appsettings.json
  - Program.cs
  - Startup.cs

Figure 19: Employee create view example.

### 1.3. How a user can navigate around and perform the different functions in the Domingo Roof Works Web Application.

Upon arrival to the Domingo Roof Works website the user will be greeted with this home page in Figure 20 below.

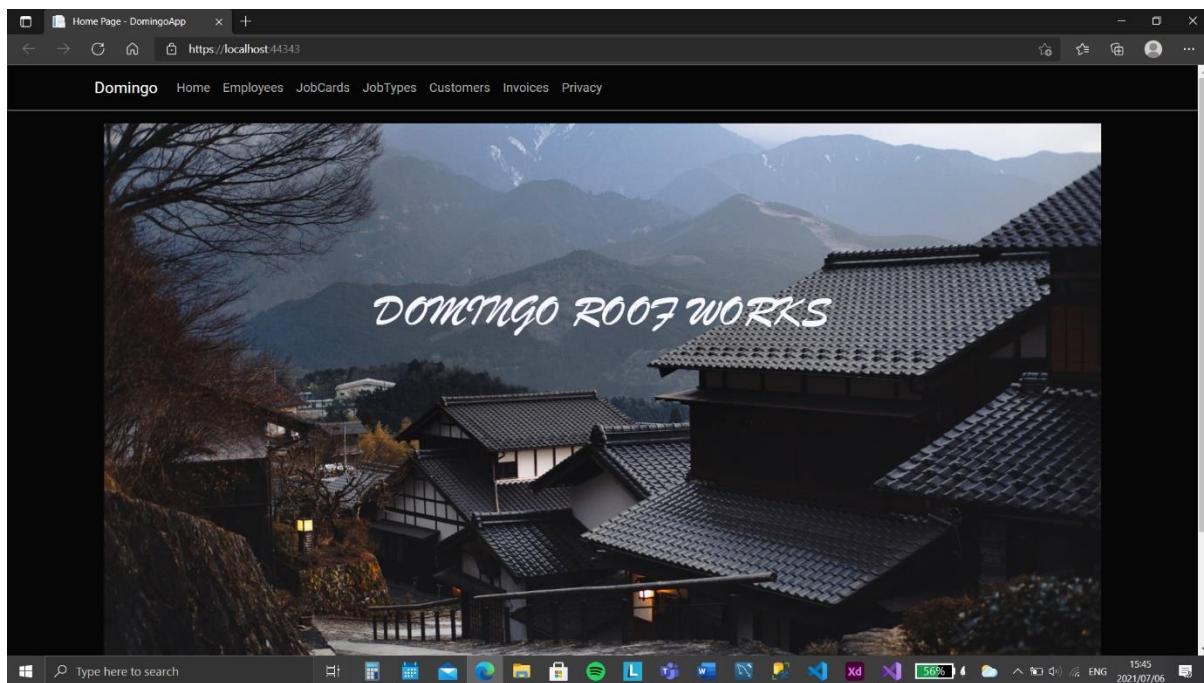


Figure 20: Domingo home page.

Referencing (Tik, 2021) and (IAmTimCorey, 2018), the user can click on any of the titles displayed in the top navigation bar to navigate to the various pages(functions) that they would like to access. For this example, the employees title(page) in the navigation bar will be utilized. See figure 21 below for the displayed page after the employees title(page) in the navigation bar was clicked.

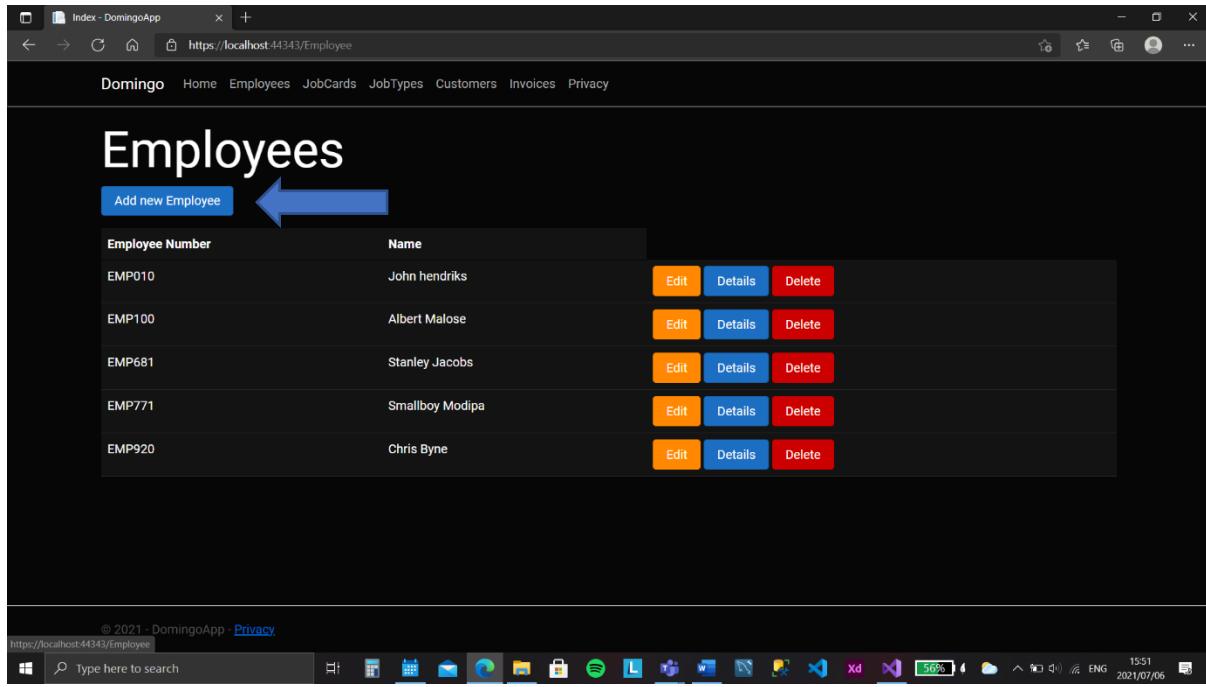


Figure 21: Employees page and add button.

Once the page is displayed the user is provided the necessary information for the operations that they would like to perform within the specific page such as Inserting, Reading, Updating, and Deleting data. In this example with the employees a new employee will be created by clicking the Add New Employee button. The user is then prompted to enter the employee's details as shown in Figure 22 below. For this example, an employee with an employeeNo of EMP123 and a name of Jeff Smith will be created.

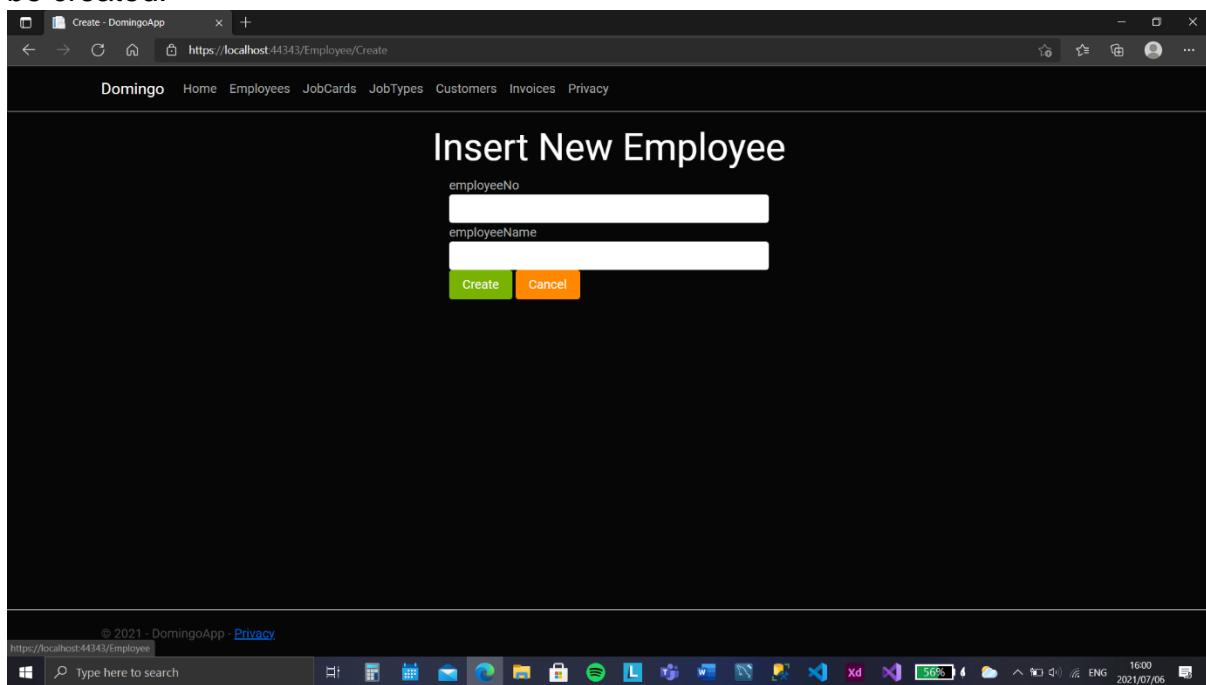
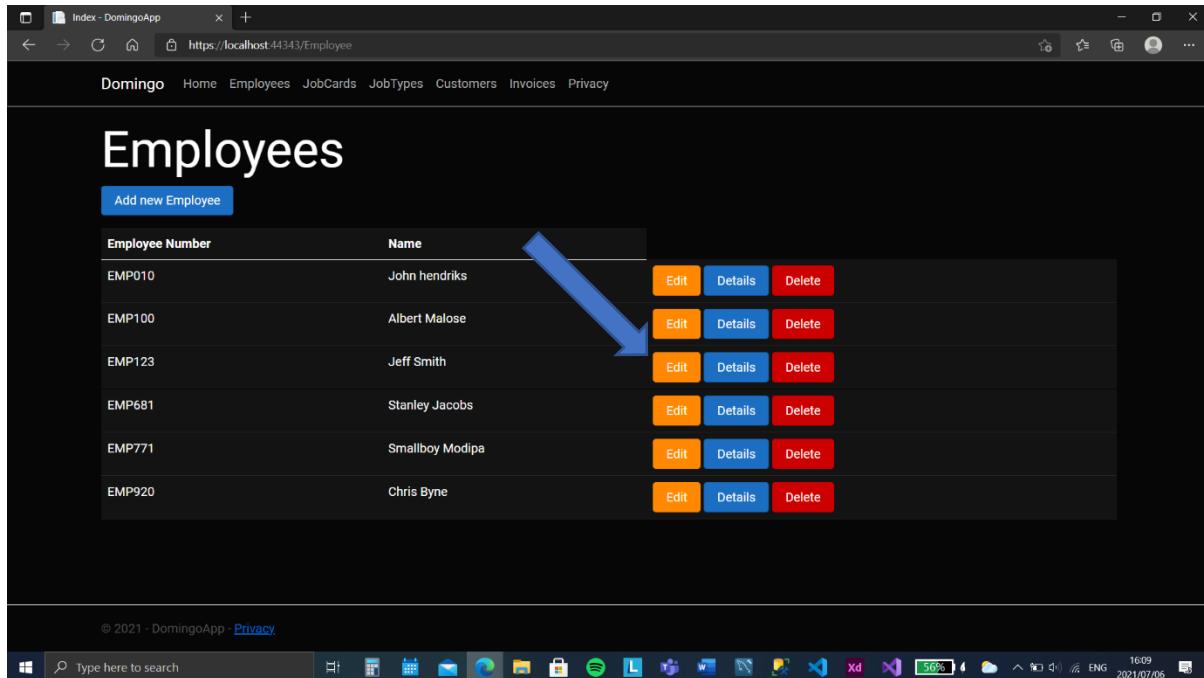


Figure 22: Add new employee page.

The user can also edit the details of an item by clicking the Edit button indicated on the right-hand side of the item's row of information (see Figure 23 below). The user will then be provided with the necessary page for updating the item's information, as displayed in Figure 24 below. For this example, the employee's name will be changed to Jeff Jackson.

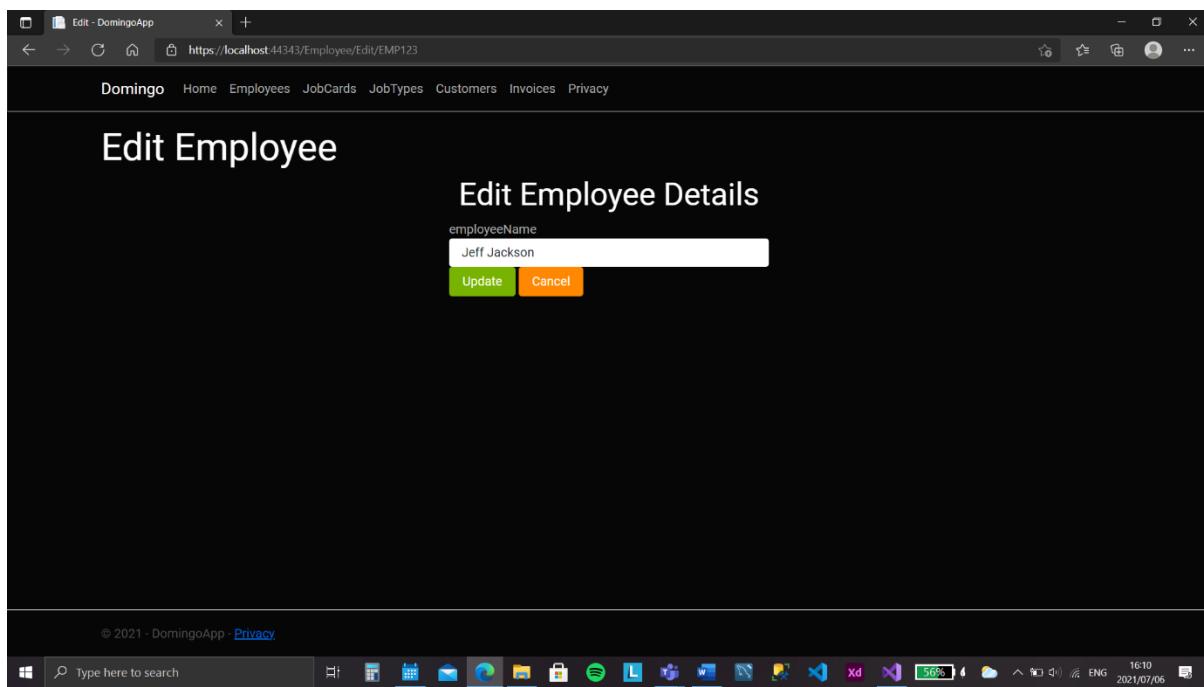


A screenshot of a web browser displaying the 'Employees' page of the DomingoApp. The page has a dark theme. At the top, there is a navigation bar with links for Home, Employees, JobCards, JobTypes, Customers, Invoices, and Privacy. Below the navigation bar is a large heading 'Employees'. A blue button labeled 'Add new Employee' is visible. The main content is a table with columns for 'Employee Number' and 'Name'. Each row contains an 'Edit' button, a 'Details' button, and a 'Delete' button. A large blue arrow points from the text in the preceding paragraph to the 'Edit' button in the third row, which corresponds to the employee 'Jeff Smith'. The table rows are as follows:

Employee Number	Name	Edit	Details	Delete
EMP010	John hendriks	<span>Edit</span>	<span>Details</span>	<span>Delete</span>
EMP100	Albert Malose	<span>Edit</span>	<span>Details</span>	<span>Delete</span>
EMP123	Jeff Smith	<span>Edit</span>	<span>Details</span>	<span>Delete</span>
EMP681	Stanley Jacobs	<span>Edit</span>	<span>Details</span>	<span>Delete</span>
EMP771	Smallboy Modipa	<span>Edit</span>	<span>Details</span>	<span>Delete</span>
EMP920	Chris Byne	<span>Edit</span>	<span>Details</span>	<span>Delete</span>

At the bottom of the page, there is a copyright notice: '© 2021 - DomingoApp - [Privacy](#)'. The browser's taskbar at the bottom shows various open applications like Spotify, Microsoft Word, and Microsoft Excel.

Figure 23: Edit button example.



A screenshot of a web browser displaying the 'Edit Employee Details' page of the DomingoApp. The page has a dark theme. At the top, there is a navigation bar with links for Home, Employees, JobCards, JobTypes, Customers, Invoices, and Privacy. Below the navigation bar is a heading 'Edit Employee'. The main content is a form titled 'Edit Employee Details' with a single input field labeled 'employeeName' containing the value 'Jeff Jackson'. Below the input field are two buttons: 'Update' (green) and 'Cancel' (orange). The browser's taskbar at the bottom shows various open applications like Spotify, Microsoft Word, and Microsoft Excel.

Figure 24: Edit page example.

Viewing details for a specific item is also possible by clicking on the blue Details button as indicated in Figure 25 below. In this example we will view the details of Jeff Jackson.

The screenshot shows a web browser window for 'Index - DomingoApp' at <https://localhost:44343/Employee>. The page title is 'Employees'. A blue arrow points down to the 'Details' button for the employee with Employee Number 'EMP123' and Name 'Jeff Jackson'. The table has columns 'Employee Number' and 'Name', and each row contains 'Edit', 'Details', and 'Delete' buttons.

Employee Number	Name	Edit	Details	Delete
EMP010	John hendriks	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>
EMP100	Albert Malose	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>
EMP123	Jeff Jackson	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>
EMP681	Stanley Jacobs	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>
EMP771	Smallboy Modipa	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>
EMP920	Chris Byne	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>

Figure 25: Details button example.

The user is presented with a new page displaying all the necessary details(see figure 26 below).

The screenshot shows a web browser window for 'Details - DomingoApp' at <https://localhost:44343/Employee/Details/EMP123>. The page title is 'Employee Details'. It displays the employee's number and name, and includes 'Edit' and 'Back' buttons.

Employee Details

- Employee Number: EMP123
- Employee Name: Jeff Jackson

Edit Back

Figure 26: Details page example.

Finally, a specific item can be deleted from the database by clicking on the red Delete button indicated in Figure 27 below. For this example, we will be deleting Jeff Jackson from the database as he is no longer an employee.

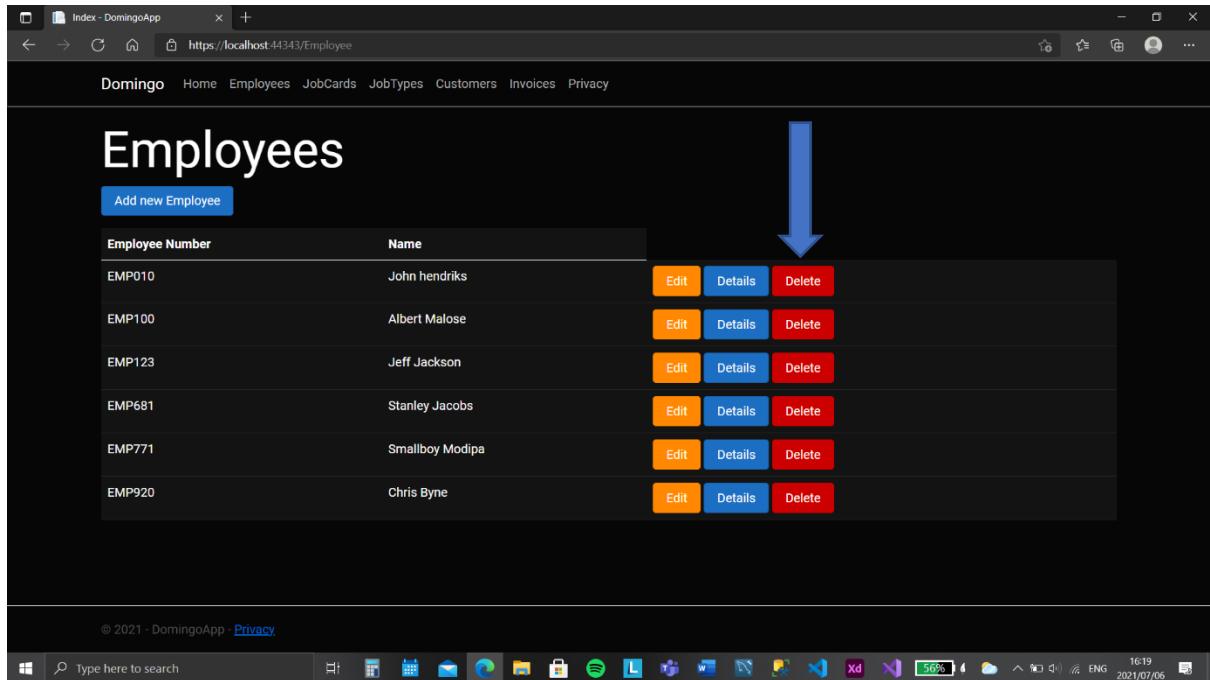


Figure 27: Delete button example.

Once the delete button is clicked the user will be redirected to the delete page which displays the information to be deleted and allows the user to cancel the delete operation if necessary(see Figure 28 below).

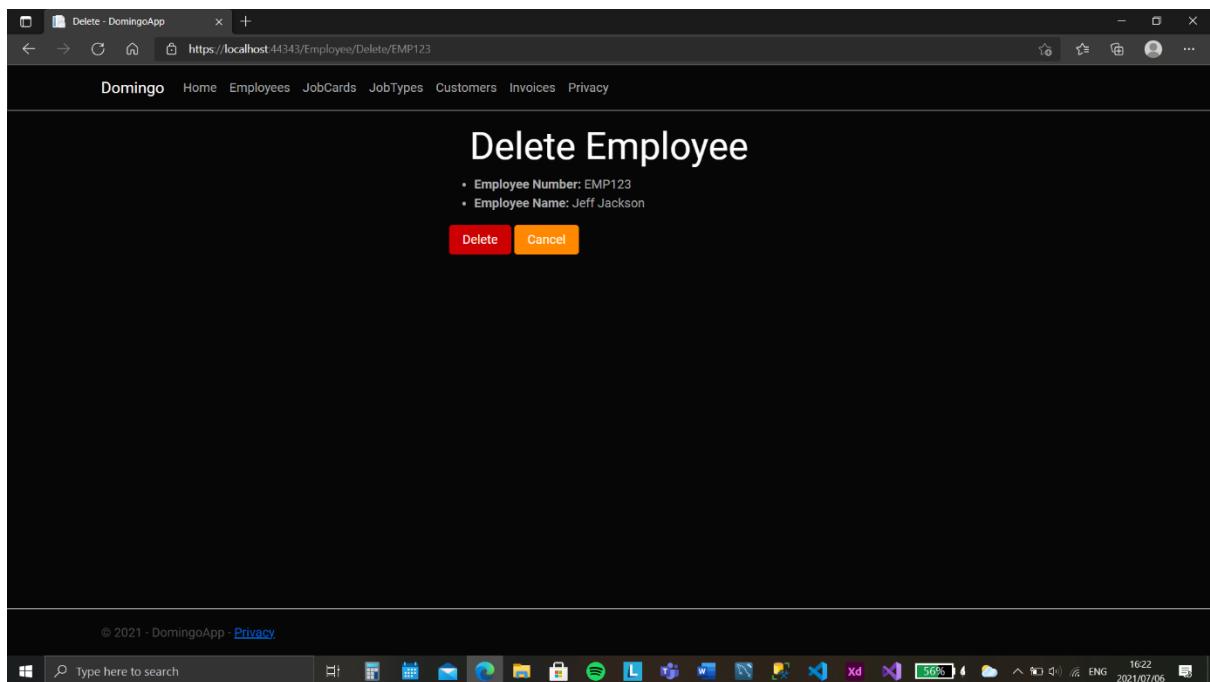


Figure 28: Delete page example.

## **2. Self-Evaluation**

### **2.1. What were the most and least beneficial aspects of this course?**

The most beneficial aspect of this course would be the knowledge gained on architectures, such as MVC and relational databases, as well as how various databases, and websites are designed and created. I would say there were no least beneficial aspects as all content was important and contributed to a deeper understanding of web development and cloud development.

### **2.2. How did each of the learning units contribute to your knowledge of web development?**

Learning units 1 to 5 (Chao, 2014), in the cloud development and databases modules, taught me the database design, database implementation, and migration aspects of web development and the learning units provided in the programming module (Andrew Troelsen & Philip Japikse, 2017), enabled me to physically code the web application using C#, HTML, and CSS.

### **2.3. How did the knowledge gained from each of the learning units assist you in completing the tasks in this POE?**

Learning unit 1 (Chao, 2014), helped me understand the fundamentals of what a database is and how it works as well as how a database interacts, or works, with the cloud.

Learning unit 2 (Chao, 2014), provided me the knowledge of the design levels of a database (conceptual design, logical design, and physical design) which enabled me to successfully design the database for Domingo Roof Works.

Learning unit 3 (Chao, 2014), taught the structure of database tables, normalization of tables, as well as the steps required to convert a database model into a relational database. Learning unit 4 (Chao, 2014), equipped me with the various SQL statements and keywords necessary to physically create the database, manipulate it, and manage it. Learning unit 3 and 4 enabled me to successfully create the database for Domingo Roof Works.

Finally learning unit 5 (Chao, 2014) as well as the tutorial provided by (Tik, 2021), provided knowledge on database migration between cloud and local servers which enabled me to migrate my local database and web app to the Azure cloud.

#### 2.4. Do you think you will use the skills in this course in your career? If yes, explain how.

Yes. Within this career path (software engineering/IT) I see many jobs on sites like LinkedIn that require most, if not all, of the concepts(skills) provided in this course. Even jobs that do not use the exact frameworks used in this course will have a fundamentally similar layout/structure. The skills learned, such as the SQL language, and frameworks, such as MySQL and Microsoft SQL Server Management Studio, will allow me to create and manipulate databases for my employer. Architecture such as the MVC architecture, used in creating the web app, has equipped me with the necessary skills for web development with many frameworks which use this architecture.

#### 2.5. If you were to rewind life one semester, what would you do differently in this course and what would you do the same?

I would have begun the POE Task 1 sooner as I feel I could have done better for that task, and I would have studied the theory sooner as it would have made the processes of coding and design more efficient for me as less mistakes would have been made. The process of learning the MVC architecture for the web app is something that I would have done the same as very little time was given to learn this framework and I think that I have successfully implemented it.

### **3. Link To Web App (Domingo Roof Works)**

My Domingo Roof Works Web App URL:

<https://domingoapplication.azurewebsites.net>

## References

- Andrew Troelsen & Philip Japikse, 2017. *Pro C#7*. 8th ed. New York: Apress.
- app.termly.io, 2021. *app.termly.io*. [Online]  
Available at: <https://app.termly.io/dashboard/website/c042eab6-af24-410e-8142-ee4bea06d9e5/privacy-policy#infocollect>  
[Accessed 01 July 2021].
- bootswatch.com, 2021. *bootswatch.com*. [Online]  
Available at: <https://bootswatch.com/>  
[Accessed 23 June 2021].
- Chao, L., 2014. *Cloud Database Development & Management*. Boca Raton, FL: CRC Press.
- dotnettutorials.net, 2021. *dotnettutorials.net*. [Online]  
Available at: <https://dotnettutorials.net/lesson/variables-and-query-strings-in-routing/>  
[Accessed 19 June 2021].
- IAmTimCorey, 2018. *youtube.com*. [Online]  
Available at: [https://www.youtube.com/watch?v=bliEv\\_QNxw&t=2950s](https://www.youtube.com/watch?v=bliEv_QNxw&t=2950s)  
[Accessed 17 June 2021].
- Palmer, J., 2020. *liquidweb.com*. [Online]  
Available at: <https://www.liquidweb.com/kb/troubleshooting-microsoft-sql-server-error-18456-login-failed-user/>  
[Accessed 04 July 2021].
- Rijn, B. v., 2020. *stackoverflow.com*. [Online]  
Available at: <https://stackoverflow.com/questions/58476819/entity-framework-error-invalidcastexception-the-field-of-type-system-int32-m>  
[Accessed 21 June 2021].
- techfunda.com, 2021. *techfunda.com*. [Online]  
Available at: <https://techfunda.com/howto/958/model-attributes>  
[Accessed 20 June 2021].
- Terevinto, C., 2018. *stackoverflow.com*. [Online]  
Available at: <https://stackoverflow.com/questions/51388735/the-model-item-passed->

into-the-viewdatadictionary-is-of-type-x-but-this-viewda

[Accessed 22 June 2021].

Tik, 2021. *youtube.com*. [Online]

Available at: <https://www.youtube.com/watch?v=3ltcTmkjLsQ>

[Accessed 12 June 2021].

Tik, 2021. *youtube.com*. [Online]

Available at: <https://www.youtube.com/watch?v=OH8RXp8aEcU>

[Accessed 18 June 2021].

Tik, 2021. *youtube.com*. [Online]

Available at: <https://www.youtube.com/watch?v=kmtq7SUK718>

[Accessed 23 June 2021].

tutorialbrain.com, 2020. *tutorialbrain.com*. [Online]

Available at: [https://www.tutorialbrain.com/css\\_tutorial/css\\_font\\_family\\_list/](https://www.tutorialbrain.com/css_tutorial/css_font_family_list/)

[Accessed 01 July 2020].

tutorialspoint.com, 2021. *tutorialspoint.com*. [Online]

Available at:

[https://www.tutorialspoint.com/mvc\\_framework/mvc\\_framework\\_controllers.htm](https://www.tutorialspoint.com/mvc_framework/mvc_framework_controllers.htm)

[Accessed 06 July 2021].

w3schools.com, 2021. *w3schools.com*. [Online]

Available at: [https://www.w3schools.com/SQL/sql\\_stored\\_procedures.asp](https://www.w3schools.com/SQL/sql_stored_procedures.asp)

[Accessed 06 July 2021].

wdexplorer.com, 2021. *wdexplorer.com*. [Online]

Available at: <https://wdexplorer.com/20-examples-beautiful-css-typography-design/>

[Accessed 22 June 2021].