COMP2001 : Information Management & Retrieval

INTRODUCTION

This document talks you through the implementation details for creating an ASP.NET MVC application that derives its data from a Microsoft SQL Server application. This follows the Application Fact Sheet planning document provided in Unit 9 and concentrates only on the first user story of

“As a student I wish to alert the lecturer that I need some help”

The assumption is that you already have the database elements created in the Microsoft SQL Server database. These elements will contain tables, triggers and stored procedures.

There are accompanying podcasts to be found to guide you in developing your prototype application. These can be found on the DLE in the podcast section. Additionally, the code is part of my public GitHub repository ISAD251_2019_ASP.NET.

APPLICATION

Our application is going to be a prototype proof of concept – this means it is NOT a full application but an exploration of how some aspects might work. We will act as if we were working in sprints, short iterative bursts of activity that will drive us to the end goal.

Each of the tasks are described at a reasonably high level. Use the “How to.. “ guides provided on the DLE to help support you with each of the tasks.   The full application details are given in the **Application Fact Sheet**.

IMPLEMENTATION

ASP.NET
Task 1:
Create a new ASP.NET MVC application for the project as you have done in earlier practical sessions. Adjust the layout templates in the views folder as indicated by the storyboard diagrams. See ***How to .. ASP.NET layout files with Bootstrap*** podcast for more information. Refer back to the lecture and practical for unit 6 if you need a reminder of how to incorporate Bootstrap with MVC

What do I need to know?
How to

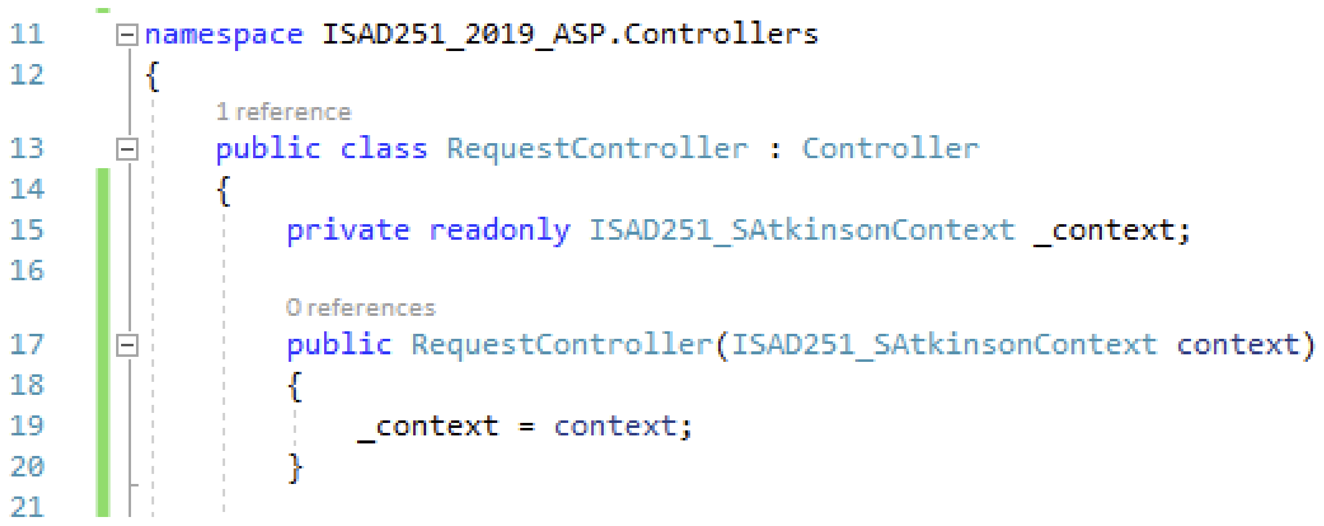
- Combine ASP.NET, Razor and HTML
- Where layout templates are stored in an MVC project

Task 2:
Create your repository classes and unit of work class based on the tables in the database. Use Package Manager and Entity Framework to create those classes. You are now following the Database First workflow which was introduced in Unit 5. See ***How to.. ASP.NET Database First*** podcast for more information.

What do I need to know?
How to
- Install Entity Framework into a project
- Scaffold the creation of objects using Package Manager
- Use Entity Framework to create a controller

Task 3:
Link your database to your Request page by carrying out the following actions. See ***How to.. ASP.NET Drop down list in view from DB table*** podcast for information on creating a drop down list in the view based on the values in a database table. Below is the step by step instructions:

1. Inside your RequestController, add a private readonly property that has the type of your projects database context class. In the example below you can see the context class has been automatically named from my database name and Context appended to it.
2. Create a new constructor that takes in a parameter of database context type and assigns it to the private property declared above. See sample code below.



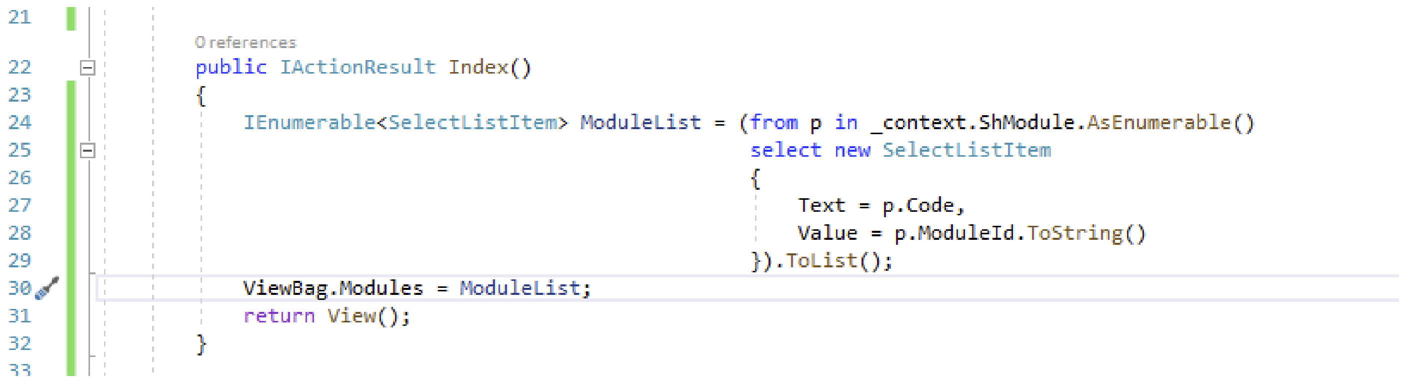
```
namespace ISAD251_2019_ASP.Controllers
{
    1 reference
    public class RequestController : Controller
    {
        private readonly ISAD251_SAtkinsonContext _context;

        0 references
        public RequestController(ISAD251_SAtkinsonContext context)
        {
            _context = context;
        }

```

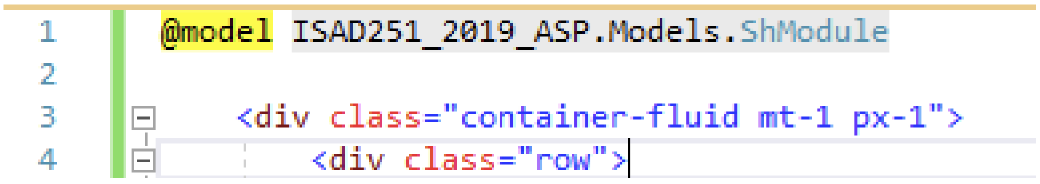
3. Inside the RequestController Index method add the following code so that a new list is created. The list should be the IEnumerable<SelectListItem> type.

   a. You will need to add using Microsoft.AspNetCore.Mvc.Rendering; to your using statements.
4. Use the LINQ statement below to extract the items from the shModule table to create a new SelectListItem for each one in the table.
5. Save the result to the ViewBag.



```
        0 references
        public IActionResult Index()
        {
            IEnumerable<SelectListItem> ModuleList = (from p in _context.ShModule.AsEnumerable()
                                                      select new SelectListItem
                                                      {
                                                          Text = p.Code,
                                                          Value = p.ModuleId.ToString()
                                                      }).ToList();
            ViewBag.Modules = ModuleList;
            return View();
        }
```

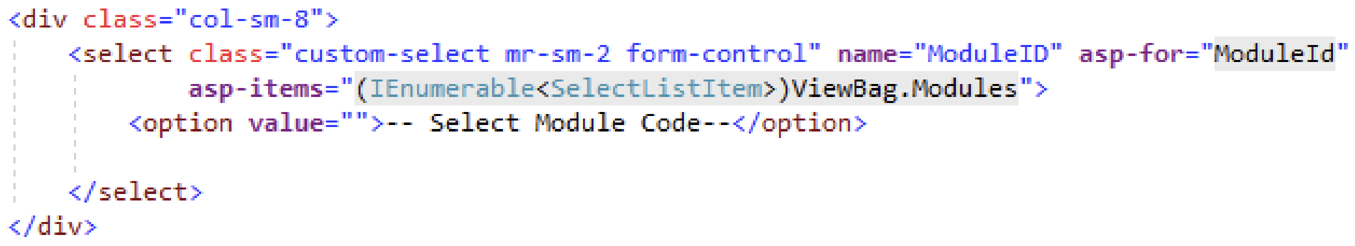
6. Open the Index.cshtml file from the Views-> Request folder. At the top place the Model declaration to link to the model class that relates to the table. See below:



```
@model ISAD251_2019_ASP.Models.ShModule

    <div class="container-fluid mt-1 px-1">
        <div class="row">
```

7. Amend the <select > tag to use the asp-for and asp-items razor tags. Set these as shown belowl



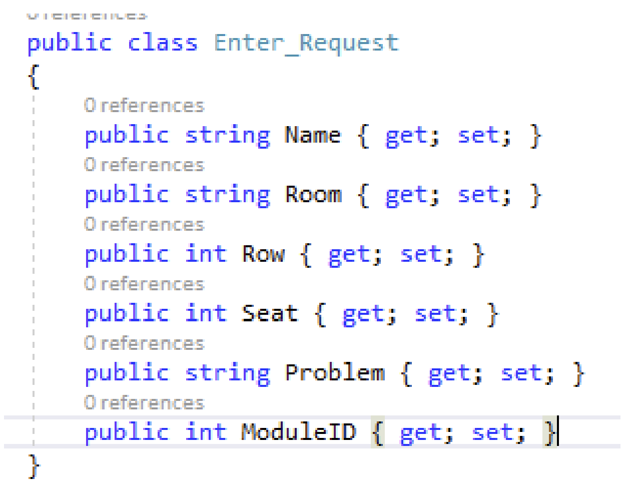
```
<div class="col-sm-8">
    <select class="custom-select mr-sm-2 form-control" name="ModuleID" asp-for="ModuleId"
            asp-items="(IEnumerable<SelectListItem>)ViewBag.Modules">
        <option value="">-- Select Module Code--</option>

    </select>
</div>
```

8. Check for errors and debug where necessary. Build and run. You will need to be on campus or using a VPN for the data to work.

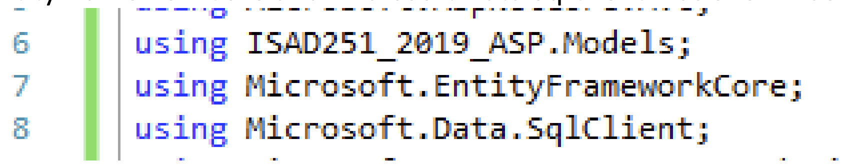
Task 4:
Your next task is to use the stored procedure you created for Enter_Request to take the input from the request form and insert into the database. Refer to podcast ***How to… ASP.NET Calls Stored Proceure for insert*** for further information. The step by step details are provided below:

1. Create a class for the stored procedure and created public properties for each of the input variables. See below for an example. Name the class the same name as the stored procedure.



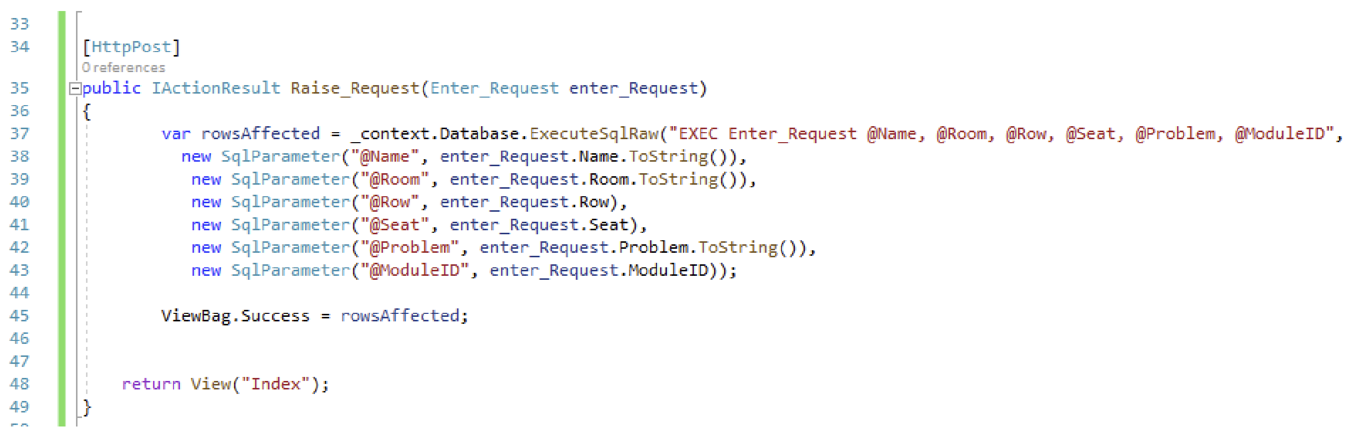
```
public class Enter_Request
{
    0 references
    public string Name { get; set; }
    0 references
    public string Room { get; set; }
    0 references
    public int Row { get; set; }
    0 references
    public int Seat { get; set; }
    0 references
    public string Problem { get; set; }
    0 references
    public int ModuleID { get; set; }
}
```

2. Inside the RequestController, create a new method called Raise_Request that takes in a parameter of the class you have just created. Ensure the method is decorated with [HttpPost] to tie in the http method that is in use. (See image below point 6 for code)
3. Place the using statements of the project models folder, Microsoft EntityFrameworkCore and Microsoft Data SqlClient – as shown below.



```
using ISAD251_2019_ASP.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.Data.SqlClient;
```

4. Create a variable called rowsaffected and fill that variable using the ExecuteSqlRaw method from the context.Database property.
    a. Create the string to call the stored procedure using named variables.
    b. Add named parameters.
5. Save the rows affected to a ViewBag.Success property.
6. Call the index view.



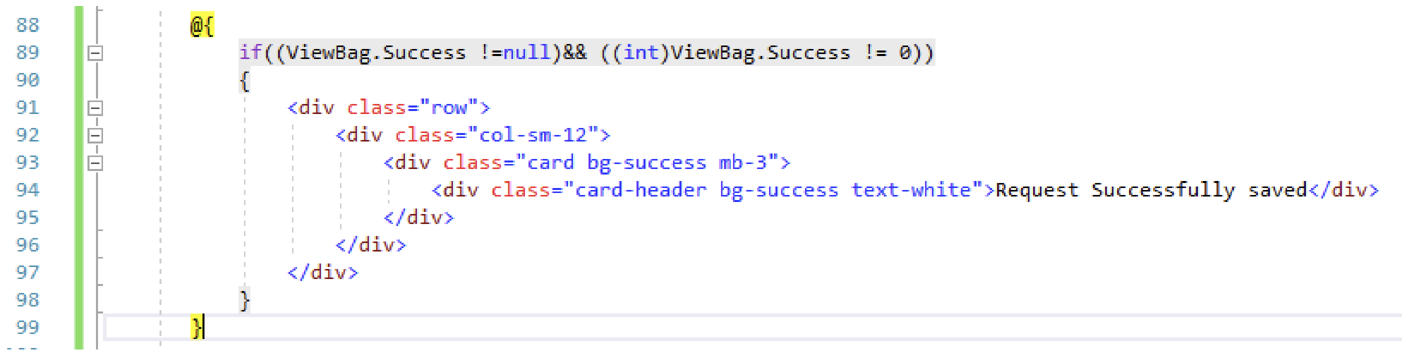
```

[HttpPost]
0 references
public IActionResult Raise_Request(Enter_Request enter_Request)
{
        var rowsAffected = _context.Database.ExecuteSqlRaw("EXEC Enter_Request @Name, @Room, @Row, @Seat, @Problem, @ModuleID",
          new SqlParameter("@Name", enter_Request.Name.ToString()),
           new SqlParameter("@Room", enter_Request.Room.ToString()),
           new SqlParameter("@Row", enter_Request.Row),
           new SqlParameter("@Seat", enter_Request.Seat),
           new SqlParameter("@Problem", enter_Request.Problem.ToString()),
           new SqlParameter("@ModuleID", enter_Request.ModuleID));

        ViewBag.Success = rowsAffected;


    return View("Index");
}
```

7. Open the index.cshtml page for the Request view. Check that the input box names exactly match the property names for the enter_request object. Check that the form helper tags of asp-controller and asp-action map to the Request controller and the method name you have created above.
8. At the bottom of the page use the code block to check if the ViewBag.Success has a value that is not zero. If this is successful, create a green div to indicate success. Use the code below.



```
@{
    if((ViewBag.Success !=null)&& ((int)ViewBag.Success != 0))
    {
        <div class="row">
            <div class="col-sm-12">
                <div class="card bg-success mb-3">
                    <div class="card-header bg-success text-white">Request Successfully saved</div>
                </div>
            </div>
        </div>
    }
}
```

9. Build and debug if necessary. Run your application and test the Request page.

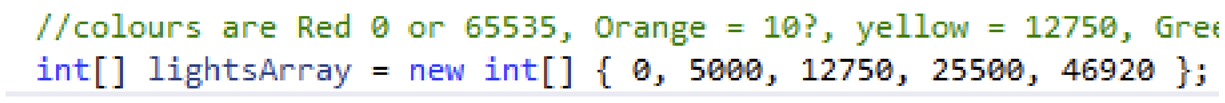
Task 5:
The next part of the application is to call out to the Hue lights RESTful API to turn the light on. Inside the Request Controller create a new method that does not return anything called Alert. Pass in as a parameter the enter_request object.

Within this method you need to do the following:

1. Set the URI – copy the code below (this one is not an image). This URI is the IP address of the Hue lights in SMB109 and provides the developer key within the URI.

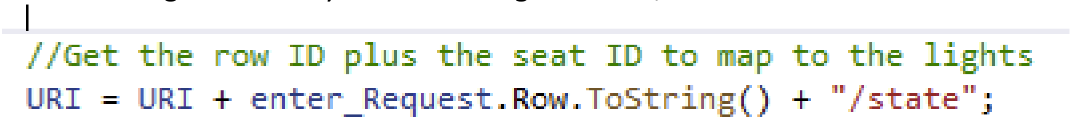
string URI = "http://192.168.0.50/api/stlaB2I6VZ8O80Qepc-1xfmLrHgyTFvB9IGupaQz/lights";

2. Create an array of integers to hold the Hue values. Colours are as follows
    a. Red = 0 or 65535,
    b. Orange = 5000
    c. Yellow = 12750
    d. Green = 25500
    e. Blue = 46920



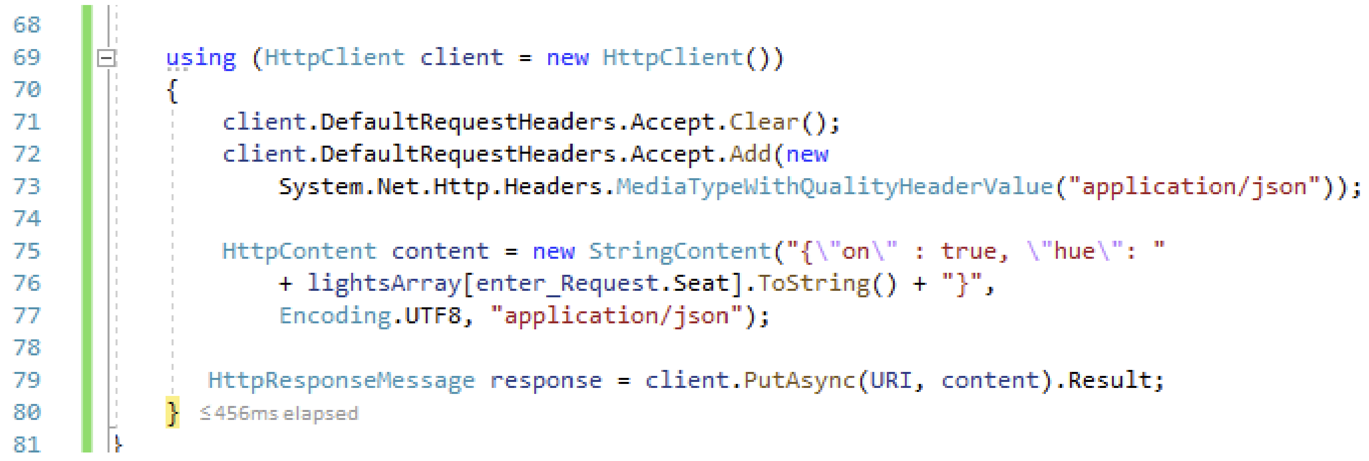
```
//colours are Red 0 or 65535, Orange = 10?, yellow = 12750, Gree
int[] lightsArray = new int[] { 0, 5000, 12750, 25500, 46920 };
```

3. Use the Row property of the incoming enter_request object to append to the URI for the lights so that you have the light ID and /state afterwards.



```
//Get the row ID plus the seat ID to map to the lights
URI = URI + enter_Request.Row.ToString() + "/state";
```

4. Create and use an HttpClient object. Clear all the headers and add in an “application/json” header.
5. Create an HttpContent object and pass in as a parameter the string below to set the light on and the hue to be the colour of the seat location.
6. Use the client object to call the method PutAsync passing in the URI parameter and the content.



```

    using (HttpClient client = new HttpClient())
    {
        client.DefaultRequestHeaders.Accept.Clear();
        client.DefaultRequestHeaders.Accept.Add(new
            System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"));

        HttpContent content = new StringContent("{\"on\" : true, \"hue\": "
            + lightsArray[enter_Request.Seat].ToString() + "}",
            Encoding.UTF8, "application/json");

       HttpResponseMessage response = client.PutAsync(URI, content).Result;
    } ≤456ms elapsed
}
```

Test your application. Check the page in the W3 validator and run the page in two separate browsers.

What do I need to know?
How to
- Use the W3 validator
- Check accessibility

WRAP UP

The whole of this exercise has shown you how to create an ASP.NET MVC application from start to finish. It shows you how to link your interface with your database layer using the classes in the middle. It finally showed you how to use a RESTful API and the HTTP verb PUT.