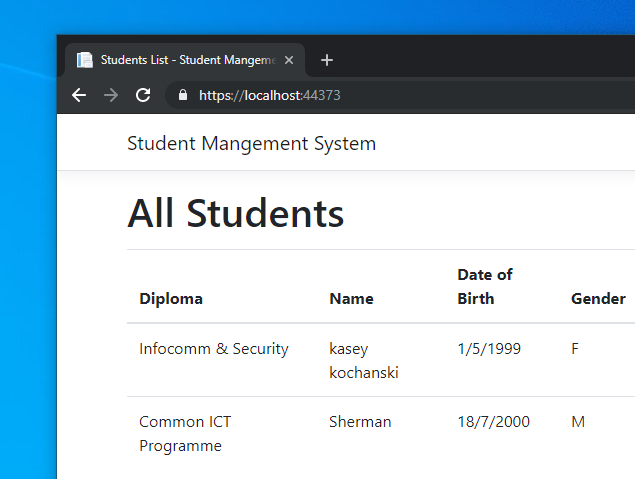
# About this practical

Throughout this practical, we will be building an ASP.NET application that utilises Entity Framework Core for database access.

This practical consist of 2 parts, lab 1 and lab 2. Lab 1 mainly focusing on initial project setup and data model building while lab 2 will focus on interacting with database using our data model.

The web application we are building is a simple student management system that manages student’s personal details, diploma info and course modules.

The source code for the practical demo can be found in Github: <https://github.com/c0j0s/StudentManagementSystem>

All steps can be view separately in corresponding branches, master branch is the complete version.

## Prerequisites:

Latest version of Visual Studio 2019 or NuGet client version 3.6.0.

SQL studio management 18.2

Contents

[About this practical 1](#_Toc16154033)

[Prerequisites: 1](#_Toc16154034)

[5. Setting up the project 3](#_Toc16154035)

[5.1. Create Our Views 3](#_Toc16154036)

[5.2. Configure Our Controller 6](#_Toc16154037)

[6. Create a New Student Record 10](#_Toc16154038)

[6.1. Setup Our Create View 10](#_Toc16154039)

[6.2. Setup Our Create Form Handling Method 13](#_Toc16154040)

[6.3. About Form validation. 15](#_Toc16154041)

[7. Retrieve List of Students 16](#_Toc16154042)

[7.1. Setup Our Index View 16](#_Toc16154043)

[7.2. Retrieve Data from Database Context 18](#_Toc16154044)

[7.3. Student Details View 19](#_Toc16154045)

[8. Make Changes to Student Details 21](#_Toc16154046)

[8.1. Setup Our View and Load Existing Data. 21](#_Toc16154047)

[8.2. Handle Edit Form Submission 23](#_Toc16154048)

[9. Delete Student Record 24](#_Toc16154049)

[9.1. Setup Our View and Confirm Modal 24](#_Toc16154050)

[9.2. Handle Delete Form Submission 26](#_Toc16154051)

[10. Handling Multiple Data using Batch Operation 27](#_Toc16154052)

[10.1. Batch Create Student Modules 27](#_Toc16154053)

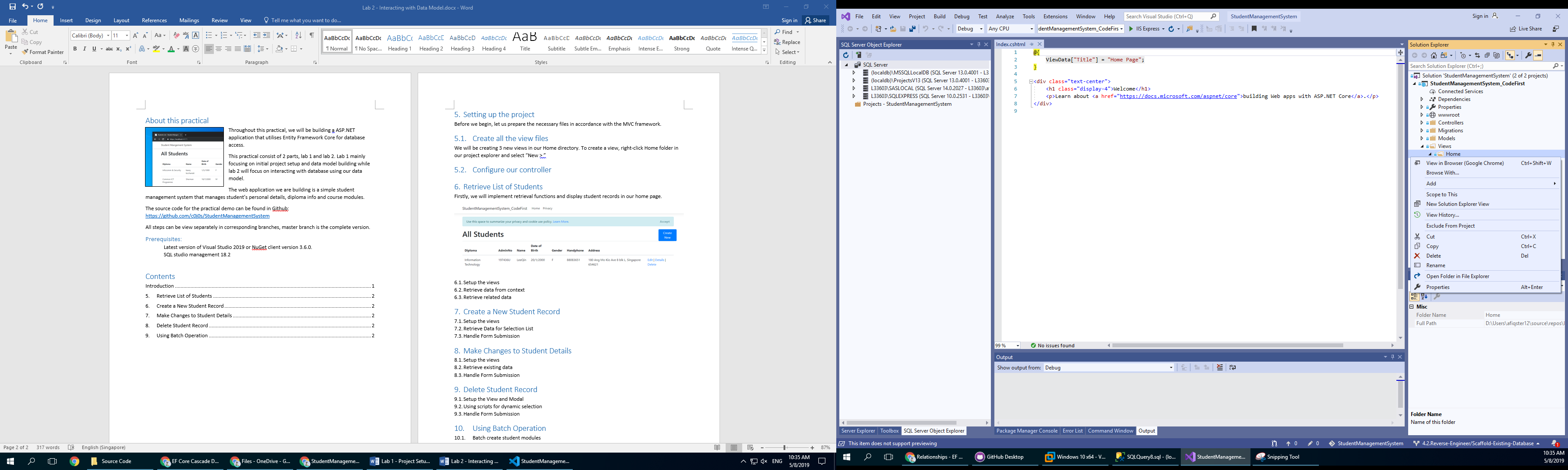
[10.2. Batch Delete Student Modules 31](#_Toc16154054)

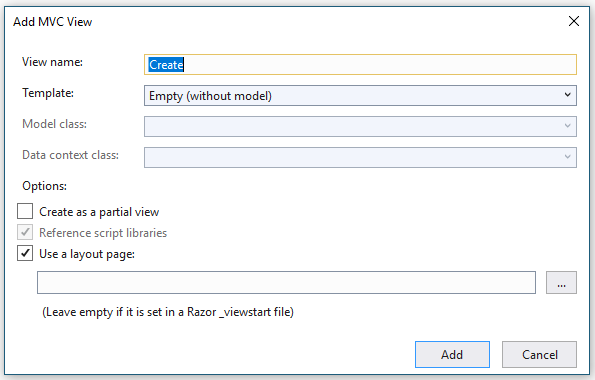
# Setting up the project

Before we begin, let us prepare the necessary files in accordance with the MVC framework.

# Create Our Views

We will be creating 3 new views in our Home directory.

* + 1. To create a view, right-click Home folder in our project explorer and select “**Add > View…**”
    2. We will create a “**Create**” view for inserting student records. Leave other options default and click “Add” to create the view.

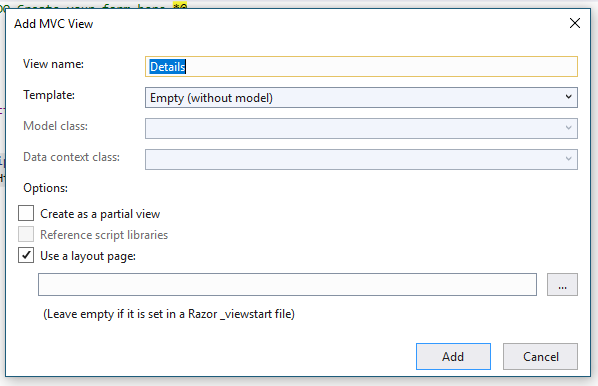
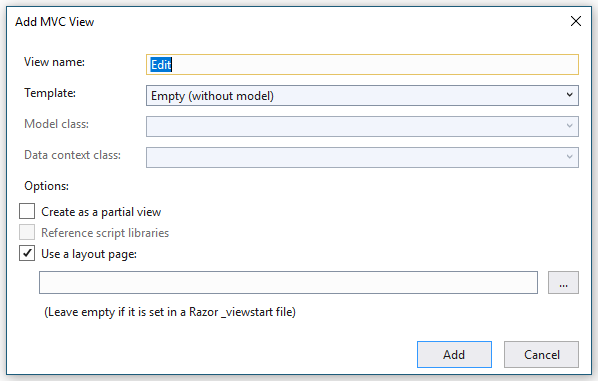


* + 1. Replace the existing content in Create.cshtml to our own template.

**Create.cshtml:**

|  |
| --- |
| @model StudentManagementSystem\_CodeFirst.Models.Student  @{  ViewData["Title"] = "Create";  }  <h1>Create</h1>  <h4>Student</h4>  <hr />  <div class="row">  <div class="col-md-8">  @\*TODO Create your form here.\*@  </div>  </div>  <div>  <**a** **asp-action**="Index">Back to List</**a**>  </div>  @section Scripts {  @{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}  } |

* + 1. Create 2 more view for viewing and editing student details. Name them as Details.cshtml and Edit.cshtml

* + 1. Replace the content with our own template as follow.

**Details.cshtml:**

|  |
| --- |
| @model StudentManagementSystem\_CodeFirst.Models.Student  @{  ViewData["Title"] = "Details";  }  <h1>Student Details</h1>  <hr />  <div class="row">  <div class="col">  <h4>Personal Details</h4>  @\*TODO Display student record here.\*@  </div>  <div class="col">  @\*TODO Display student modules here.\*@  </div>  </div>  <div>  @\*Add navigation link here.\*@  <**a** **asp-action**="Index">Back to List</**a**>  </div> |

**Edit.cshtml:**

|  |
| --- |
| @model StudentManagementSystem\_CodeFirst.Models.Student  @{  ViewData["Title"] = "Edit";  }  <h1>Edit Student [@Model.AdminNo]</h1>  <hr />  <div class="row">  <div class="col-md-8">  @\*TODO Create your form here.\*@  </div>  </div>  <div>  <**a** **asp-action**="Index">Back to List</**a**>  </div>  @section Scripts {  @{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}  } |

* + 1. Of course, don’t forget our index file that list out our student records.

**Index.cshtml:**

|  |
| --- |
| @model IEnumerable<StudentManagementSystem\_CodeFirst.Models.Student>  @{  ViewData["Title"] = "Home Page";  }  <div class="row">  <h1 class="col">All Students</h1>  @\*Add action button here.\*@  </div>  @\*Add student record table here.\*@ |

# Configure Our Controller

Next, we will configure our home controller to handle our navigation and form submission methods.

* + 1. Let us define our database context first for us to use in the controller.

**HomeController.cs:**

|  |
| --- |
| namespace StudentManagementSystem\_CodeFirst.Controllers  {  public class HomeController : Controller  {  #region define context  private readonly StudentManagementSystemContext \_context;  public HomeController(StudentManagementSystemContext context)  {  \_context = context;  }  #endregion  }  } |

* + 1. Add the following navigation methods in our home controller.

**HomeController.cs:**

|  |
| --- |
| namespace StudentManagementSystem\_CodeFirst.Controllers  {  public class HomeController : Controller  {  :  #region Routing Methods  //Error handling page  [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]  public IActionResult Error()  {  return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });  }  public IActionResult Privacy()  {  return View();  }  public async Task<IActionResult> Index()  {  return View();  }  public IActionResult Create()  {  return View();  }  public async Task<IActionResult> Details(string id)  {  return View();  }  public async Task<IActionResult> Edit(string id)  {  return View();  }  #endregion  }  } |

* + 1. Add the following form submission handling methods.

**HomeController.cs:**

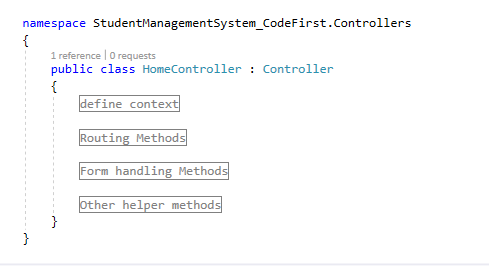
|  |
| --- |
| namespace StudentManagementSystem\_CodeFirst.Controllers  {  public class HomeController : Controller  {  :  #region Form handling Methods  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Create(  [Bind("AdminNo,Name,Dob,Gender,ContactNumber,DiplomaId,Address")] Student student)  {  return View();  }  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Edit(  string id,  [Bind("AdminNo,Name,Dob,Gender,ContactNumber,DiplomaId,Address")] Student student)  {  return View(student);  }  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Delete(string id)  {  return RedirectToAction(nameof(Index));  }  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> AddModules(string id, string[] StudentModules)  {  return RedirectToAction(nameof(Details), new { id = id });  }  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> DeleteModules(string id, string[] StudentModules)  {  return RedirectToAction(nameof(Details), new { id = id });  }  #endregion  }  } |

* + 1. And other helper methods.

**HomeController.cs:**

|  |
| --- |
| namespace StudentManagementSystem\_CodeFirst.Controllers  {  public class HomeController : Controller  {  :  #region Other Helper Methods  private bool StudentExists(string id)  {  return \_context.Students.Any(e => e.AdminNo == id);  }  #endregion  }  } |

* + 1. By now, you should have the following sections in your home controller if you collapse your regions.



There are 2 types of methods we will be focusing primarily in our home controller.

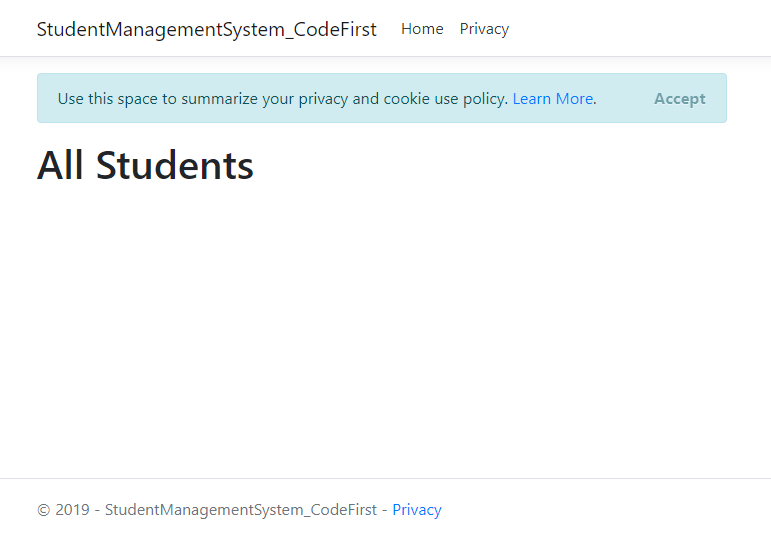
**Routing Methods:**

Handles routing between views, prepares data for the next page.

**Form handling Methods:**

Handles incoming form submission.

* + 1. If you run  the web application now, you should have the following result.



This is the default index page layout.

# Create a New Student Record

Firstly, we will implement our create function to insert student record in the database.

# Setup Our Create View

* + 1. Let us create our “Create Student Record” form following the properties in our data model. The attribute “**asp-for**” in each input specifies which property in Student model to automatically bind to in from submission.

**Create.cshtml:**

|  |
| --- |
| :  <div class="col-md-8">  @\*TODO Create your form here.\*@  <**form** **asp-action**="Create">  <**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>  <div class="form-group">  <**label** **asp-for**="AdminNo" class="control-label"></**label**>  <**input** **asp-for**="AdminNo" class="form-control" />  <**span** **asp-validation-for**="AdminNo" class="text-danger"></**span**>  </div>  <div class="form-group">  <**label** **asp-for**="Name" class="control-label"></**label**>  <**input** **asp-for**="Name" class="form-control" />  <**span** **asp-validation-for**="Name" class="text-danger"></**span**>  </div>  <div class="form-group">  <**label** **asp-for**="Dob" class="control-label"></**label**>  <**input** **asp-for**="Dob" class="form-control" />  <**span** **asp-validation-for**="Dob" class="text-danger"></**span**>  </div>  <div class="form-group">  <label>Gender:</label>  <div>  <div class="form-check form-check-inline">  <**input** class="form-check-input" **type**="radio" **asp-for**="Gender" id="Gender-M" **value**="M">  <label class="form-check-label" for="Gender-M">Male</label>  </div>  <div class="form-check form-check-inline">  <**input** class="form-check-input" **type**="radio" **asp-for**="Gender" id="Gender-F" **value**="F">  <label class="form-check-label" for="Gender-F">Female</label>  </div>  <**span** **asp-validation-for**="Gender" class="text-danger"></**span**>  </div>  </div>  <div class="form-group">  <**label** **asp-for**="ContactNumber" class="control-label"></**label**>  <**input** **asp-for**="ContactNumber" class="form-control" />  <**span** **asp-validation-for**="ContactNumber" class="text-danger"></**span**>  </div>  <div class="form-group">  <label>Address:</label>  @\*TODO Insert address input here.\*@  </div>  <div class="form-group">  <**label** **asp-for**="DiplomaId" class="control-label"></**label**>    @\*TODO Insert diploma dropdown list here.\*@  </div>  <div class="form-group">  <input type="submit" value="Create" class="btn btn-primary" />  </div>  </**form**>  </div>  : |

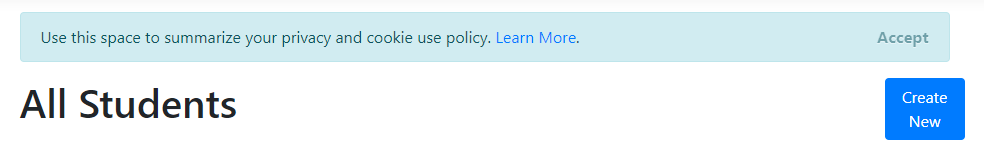
* + 1. Next, create a navigation link in our index page to navigate to our create page.

**Index.cshtml:**

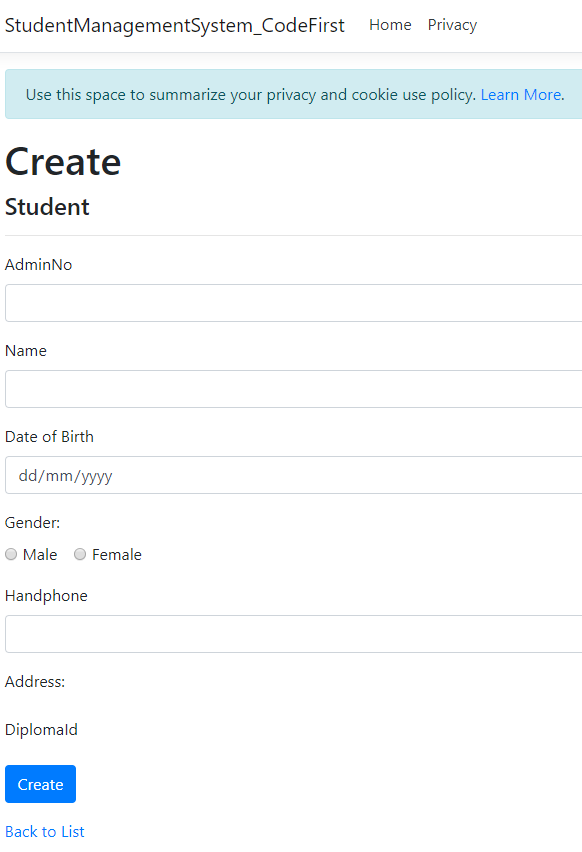
|  |
| --- |
| @\*Add action button here.\*@  <**a** **asp-action**="Create" class="btn btn-primary float-right col-sm-1">Create New</**a**> |

* + 1. If you run  the web application now, you should have the following result.

1. **Index page:**



1. **Create page:**



* + 1. As student address information is defined in a separate entity, we will need to make use of our **Address navigation property** defined in our Student model to refer to Address entity. We will need to implement the following fields.

**Create.cshtml:**

|  |
| --- |
| :  @\*TODO Insert address input here.\*@  <div class="col">  <**label** **asp-for**="Address.StreetName" class="control-label"></**label**>  <**input** **asp-for**="Address.StreetName" class="form-control" />  <**span** **asp-validation-for**="Address.StreetName" class="text-danger"></**span**>  </div>  <div class="col">  <**label** **asp-for**="Address.Details" class="control-label"></**label**>  <**input** **asp-for**="Address.Details" class="form-control" />  <**span** **asp-validation-for**="Address.Details" class="text-danger"></**span**>  </div>  <div class="col">  <**label** **asp-for**="Address.PostalCode" class="control-label"></**label**>  <**input** **asp-for**="Address.PostalCode" class="form-control" />  <**span** **asp-validation-for**="Address.PostalCode" class="text-danger"></**span**>  </div>  : |

* + 1. Since we have data in diploma table, we can load a dropdown selection list for user to select instead of entering the diploma key manually.

**HomeConroller.cs:**

|  |
| --- |
| public IActionResult Create()  {  ViewData["DiplomaSelectionList"] = new SelectList(\_context.Diploma, "DiplomaId", "Name");  return View();  } |

Here, we have create a selection list object that will retrieve data from **\_context.Diploma** and bind **value** and **display** property of a html option tag to **DiplomaId** and **Name** property automatically.

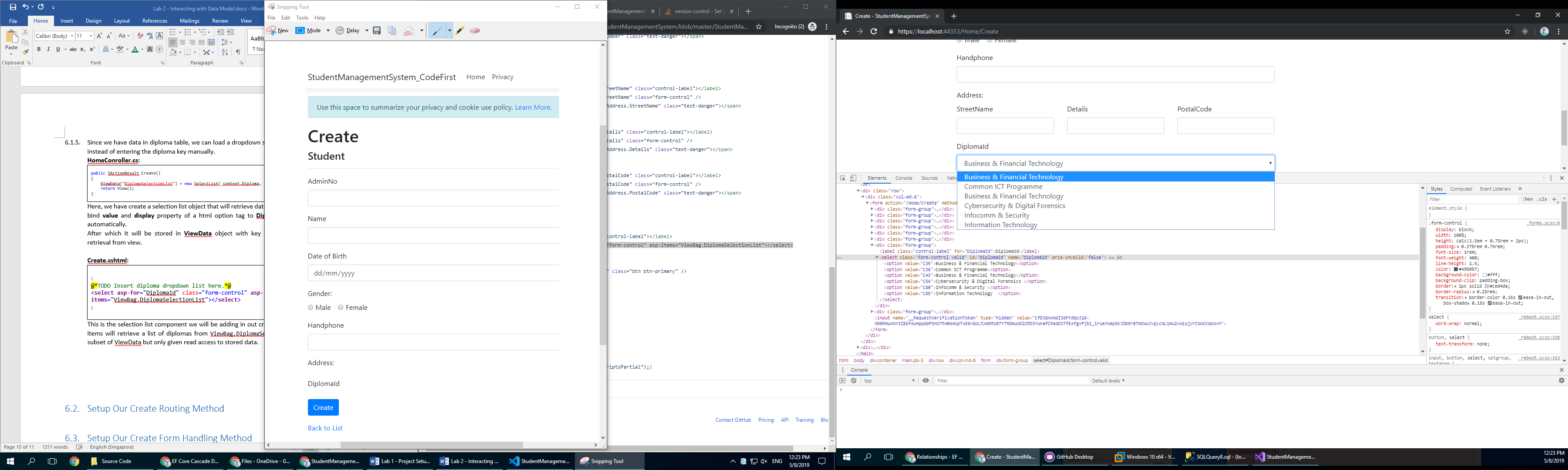
After which it will be stored in **ViewData** object with key “DiplomaSelectionList” for retrieval from view.

**Create.cshtml:**

|  |
| --- |
| :  @\*TODO Insert diploma dropdown list here.\*@  <**select** **asp-for**="DiplomaId" class="form-control" **asp-items**="ViewBag.DiplomaSelectionList"></**select**>  : |

This is the selection list element we will be adding in out create form. The attribute asp-items will retrieve a list of diplomas from ViewBag.DiplomaSelectionList. **ViewBag** is a subset of **ViewData** but only given read access to stored data.

If we run the web application and inspect the element, we will see that our select element is rendered according to what we have configured.



# Setup Our Create Form Handling Method

In order to insert our form data, we will have to configure our form handling method in home controller.

* + 1. Add the following code to handle the create form.

**HomeController.cs:**

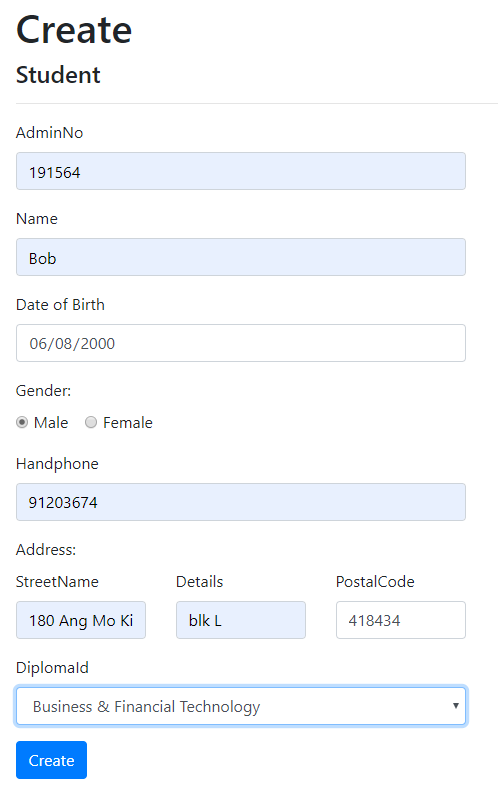
|  |
| --- |
| [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Create(  [Bind("AdminNo,Name,Dob,Gender,ContactNumber,DiplomaId,Address")] Student student)  {  if (ModelState.IsValid)  {  \_context.Add(student);  await \_context.SaveChangesAsync();  return RedirectToAction(nameof(Index));  }  else  {  //return to create form if model invalid  ViewData["DiplomaSelectionList"] = new SelectList(\_context.Diploma, "DiplomaId", "DiplomaId", student.DiplomaId);  return View(student);  }  } |

Here is a breakdown of process flow for this method:

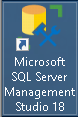
1. Create form request triggers create method with matching parameters.

* The input parameter will bind form values to the **Student** object automatically, address value will be bind to the navigation reference properties contained in **Student** object as well.

1. Create method check if from value is valid.
   1. If yes, add object to database context, entity framework core keeps track of the new Student object.
   2. If no, load the diploma selection list object again and return back to create page.
2. Save changes method triggered and entity framework core perform the pending action to database. It will add related data such as address as well.
   * 1. Run the web application and create a student record.

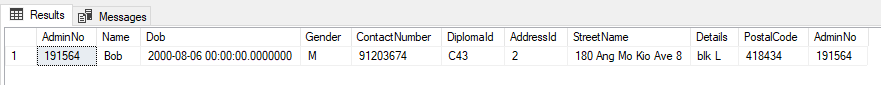


* + 1. You should now be able to view the data created in SQL Server Management Studio. You can use the following SQL statement to view student data.

Click on  and enter and  the following SQL statement in the blank query page.

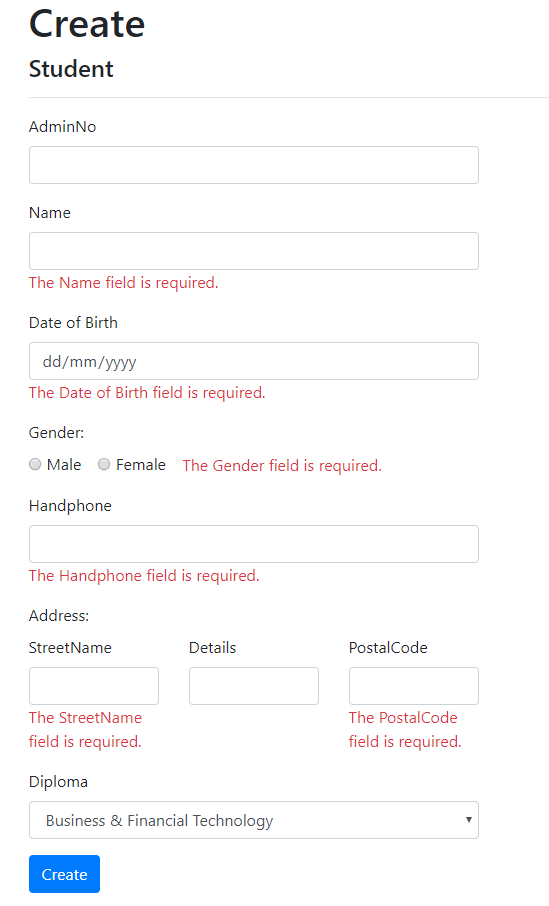
|  |
| --- |
| SELECT \*  FROM [StudentManagementSystem].[dbo].[Students] a  JOIN [StudentManagementSystem].[dbo].[Addresses] b on a.AdminNo = b.AdminNo; |

The output should look something like this.



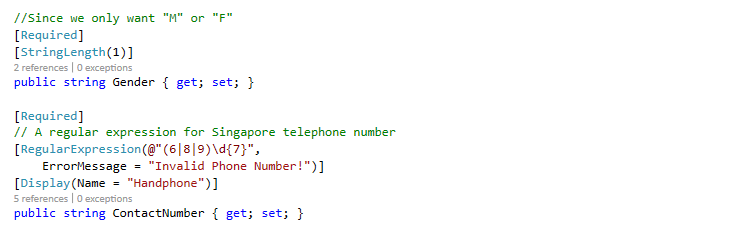
# About Form validation.

**Create.cshtml:**



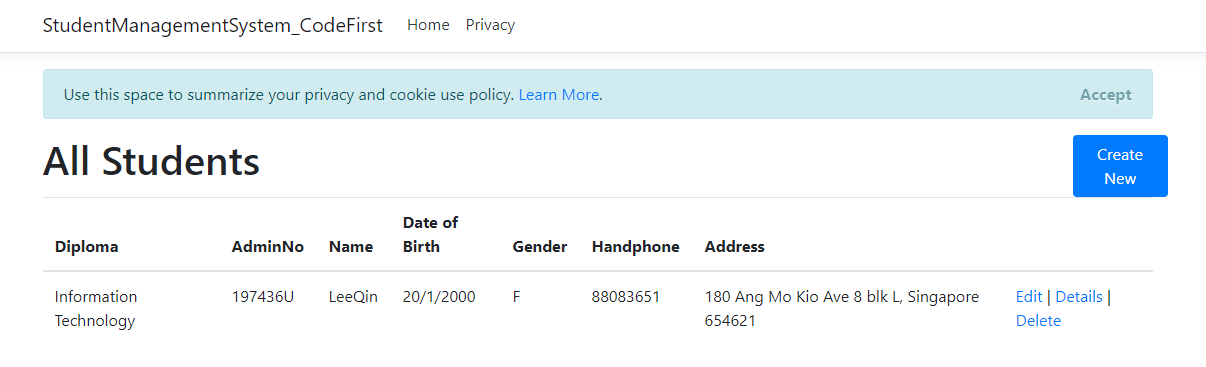
Submit empty will trigger form validation. As you can see, the form validation is already been setup for us as we have define all the necessary constrains in our data model.

**Student.cs:**



# Retrieve List of Students

Next, we will implement retrieval functions and display student records in our home page.



# Setup Our Index View

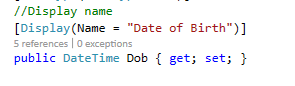
We will be create a table element in our index page to list out all our student records.

**Index.cshtml:**

|  |
| --- |
| @\*Add student record table here.\*@  <table class="table">  <thead>  <tr>  <th>  @Html.DisplayNameFor(model => model.Diploma)  </th>  <th>  @Html.DisplayNameFor(model => model.AdminNo)  </th>  <th>  @Html.DisplayNameFor(model => model.Name)  </th>  <th>  @Html.DisplayNameFor(model => model.Dob)  </th>  <th>  @Html.DisplayNameFor(model => model.Gender)  </th>  <th>  @Html.DisplayNameFor(model => model.ContactNumber)  </th>  <th>  @Html.DisplayNameFor(model => model.Address)  </th>  <th></th>  </tr>  </thead>  <tbody>  @foreach (var item in Model)  {  <tr>  <td>  @Html.DisplayFor(modelItem => item.Diploma.Name)  </td>  <td>  @Html.DisplayFor(modelItem => item.AdminNo)  </td>  <td>  @Html.DisplayFor(modelItem => item.Name)  </td>  <td>  @Html.DisplayFor(modelItem => item.Dob)  </td>  <td>  @Html.DisplayFor(modelItem => item.Gender)  </td>  <td>  @Html.DisplayFor(modelItem => item.ContactNumber)  </td>  <td>  @Html.DisplayFor(modelItem => item.Address.FullAddress)  </td>  <td>  @\*Add row action button here.\*@  </td>  </tr>  }  </tbody>  </table> |

**Table header:**

We will use display name for method to retrieve display name annotation for each properties in Student Model. If there isn’t any value set, the property will be use as default.



As you can see, header for Dob is Data of Birth instead of Dob.

**Table row:**

Since we will receive a list of students, we will use a loop code block to loop our list and create our table rows.



And each table data will display corresponding model properties.

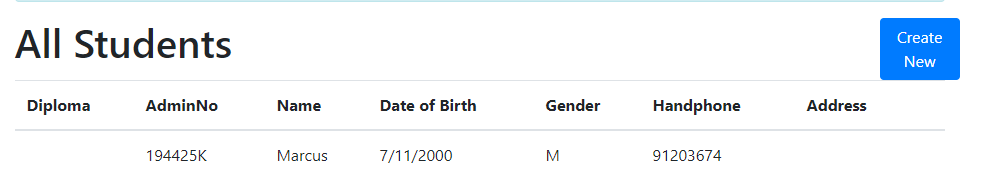
# Retrieve Data from Database Context

* + 1. Load Student data and return a list to the view.

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  :  public async Task<IActionResult> Index()  {  var studentContext = \_context.Students;  return View(await studentContext.ToListAsync());  }  : |

Add the above codes in index routing method in home controller.

If you run the web application, you will be able to see the student records.

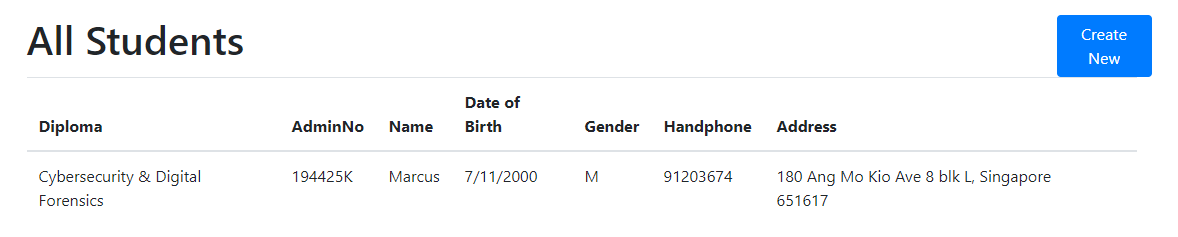


However the table did not show student’s diploma and address data. This is because we only loaded student data from database context, we will have to use another method to load related data.

* + 1. To load **related data**, there should be a navigation property in our data model. Since we already have Diploma and Address, we can use the **include** method provided by the entity framework core to load related data. **Include** is equivalent to SQL(INNER JOIN)

|  |
| --- |
| public async Task<IActionResult> Index()  {  var studentContext = \_context.Students.Include(s => s.Diploma)  .Include(s => s.Address);  return View(await studentContext.ToListAsync());  } |

If you run the web application, you will be able to see the student records with related data.



# Student Details View

Same as our index page, we will load student details and display them in a view, the only difference is that we will be displaying one model only.

* + 1. Setup Our Views

**Index.cshtml:**

|  |
| --- |
| @\*Add row action button here.\*@  <**a** **asp-action**="Details" **asp-route-id**="@item.AdminNo">Details</**a**> |

We will add a navigational link to details page in each data row. **Asp-action** here refers to the page we are directing and **asp-route-id** is the parameter we are passing to the details page.

**Details.cshtml:**

|  |
| --- |
| @\*TODO Display student record here.\*@  <dl class="row">  <dt class="col-sm-4">  @Html.DisplayNameFor(model => model.AdminNo)  </dt>  <dd class="col-sm-8">  @Html.DisplayFor(model => model.AdminNo)  </dd>  <dt class="col-sm-4">  @Html.DisplayNameFor(model => model.Name)  </dt>  <dd class="col-sm-8">  @Html.DisplayFor(model => model.Name)  </dd>  <dt class="col-sm-4">  @Html.DisplayNameFor(model => model.Dob)  </dt>  <dd class="col-sm-8">  @Html.DisplayFor(model => model.Dob)  </dd>  <dt class="col-sm-4">  @Html.DisplayNameFor(model => model.Gender)  </dt>  <dd class="col-sm-8">  @Html.DisplayFor(model => model.Gender)  </dd>  <dt class="col-sm-4">  @Html.DisplayNameFor(model => model.ContactNumber)  </dt>  <dd class="col-sm-8">  @Html.DisplayFor(model => model.ContactNumber)  </dd>  <dt class="col-sm-4">  @Html.DisplayNameFor(model => model.Address)  </dt>  <dd class="col-sm-8">  @Html.DisplayFor(model => model.Address.FullAddress)  </dd>  <dt class="col-sm-4">  @Html.DisplayNameFor(model => model.Diploma)  </dt>  <dd class="col-sm-8">  @Html.DisplayFor(model => model.Diploma.Name)  </dd>  </dl> |

We will put a table in our details view to display our model data.

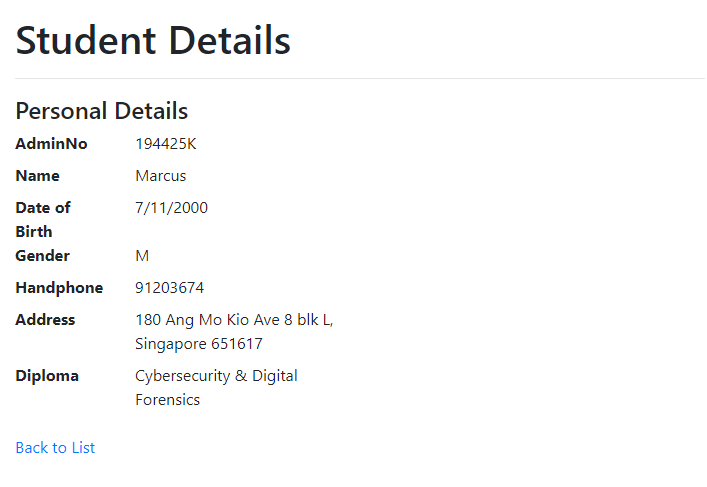
* + 1. Load Student Data in Controller

**Controller.cs:**

|  |
| --- |
| public async Task<IActionResult> Details(string id)  {  if (id == null)  {  return NotFound();  }  var student = await \_context.Students  .Include(s => s.Diploma)  .Include(s => s.Address)  .FirstOrDefaultAsync(m => m.AdminNo == id);  if (student == null)  {  return NotFound();  }  return View(student);  } |

Add the above code in our details page routing method. Similar to index routing method, we will use include to load all related data and use FirstOrDefaultAsync to load only one matching student record.

If you run the web application and click on the details link, you will be able to see the student records in details page as follow.



# Make Changes to Student Details

Next, we will create our edit page to edit student details.

# Setup Our View and Load Existing Data.

* + 1. Create a navigation link to edit page.

**Index.cshtml:**

|  |
| --- |
| @\*Add row action button here.\*@  <**a** **asp-action**="Details" **asp-route-id**="@item.AdminNo">Details</**a**>  <**a** **asp-action**="Edit" **asp-route-id**="@item.AdminNo">Edit</**a**> |

* + 1. Setup up the edit form.

**Edit.cshtml:**

|  |
| --- |
| @\*TODO Create your form here.\*@  <**form** **asp-action**="Edit">  <**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>  <**input** **type**="hidden" **asp-for**="AdminNo" />  <div class="form-group">  <**label** **asp-for**="Name" class="control-label"></**label**>  <**input** **asp-for**="Name" class="form-control" />  <**span** **asp-validation-for**="Name" class="text-danger"></**span**>  </div>  <div class="form-group">  <**label** **asp-for**="Dob" class="control-label"></**label**>  <**input** **asp-for**="Dob" class="form-control" />  <**span** **asp-validation-for**="Dob" class="text-danger"></**span**>  </div>  <div class="form-group">  <label>Gender:</label>  <div>  <div class="form-check form-check-inline">  <**input** class="form-check-input" **type**="radio" **asp-for**="Gender" id="Gender-M" **value**="M">  <label class="form-check-label" for="Gender-M">Male</label>  </div>  <div class="form-check form-check-inline">  <**input** class="form-check-input" **type**="radio" **asp-for**="Gender" id="Gender-F" **value**="F">  <label class="form-check-label" for="Gender-F">Female</label>  </div>  <**span** **asp-validation-for**="Gender" class="text-danger"></**span**>  </div>  </div>  <div class="form-group">  <**label** **asp-for**="ContactNumber" class="control-label"></**label**>  <**input** **asp-for**="ContactNumber" class="form-control" />  <**span** **asp-validation-for**="ContactNumber" class="text-danger"></**span**>  </div>  <div class="form-group">  <label>Address:</label>  <**input** **type**="hidden" **asp-for**="Address.AddressId" />  <**input** **type**="hidden" **asp-for**="Address.AdminNo" />  <div class="row">  <div class="col">  <**label** **asp-for**="Address.StreetName" class="control-label"></**label**>  <**input** **asp-for**="Address.StreetName" class="form-control" />  <**span** **asp-validation-for**="Address.StreetName" class="text-danger"></**span**>  </div>  <div class="col">  <**label** **asp-for**="Address.Details" class="control-label"></**label**>  <**input** **asp-for**="Address.Details" class="form-control" />  <**span** **asp-validation-for**="Address.Details" class="text-danger"></**span**>  </div>  <div class="col">  <**label** **asp-for**="Address.PostalCode" class="control-label"></**label**>  <**input** **asp-for**="Address.PostalCode" class="form-control" />  <**span** **asp-validation-for**="Address.PostalCode" class="text-danger"></**span**>  </div>  </div>  </div>  <div class="form-group">  <**label** **asp-for**="DiplomaId" class="control-label"></**label**>  <**select** **asp-for**="DiplomaId" class="form-control" **asp-items**="ViewBag.DiplomaSelectionList"></**select**>  <**span** **asp-validation-for**="DiplomaId" class="text-danger"></**span**>  </div>  <div class="form-group">  <input type="submit" value="Save" class="btn btn-primary" />  </div>  </**form**> |

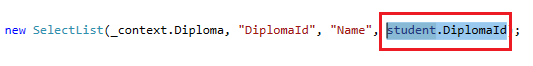
As you can see, the edit form is not much different from our create form.

* + 1. Preload existing data into edit form.

**HomeController.cs:**

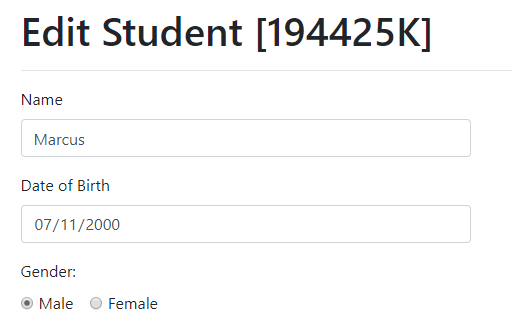
|  |
| --- |
| public async Task<IActionResult> Edit(string id)  {  if (id == null)  {  return NotFound();  }  var student = await \_context.Students  .Include(s => s.Address)  .SingleOrDefaultAsync(s => s.AdminNo == id);  if (student == null)  {  return NotFound();  }  ViewData["DiplomaSelectionList"] = new SelectList(\_context.Diploma, "DiplomaId", "Name", student.DiplomaId);  return View(student);  } |

Edit details routing method is similar to our index routing method. The only difference is we will be setting the **default value** for diploma selection list as our student record already have the information.



The **third** parameter here sets the default value.

If you run the web application and click on the edit link, you will be able to see the edit form in edit page as follow.



# Handle Edit Form Submission

**HomeController.cs**

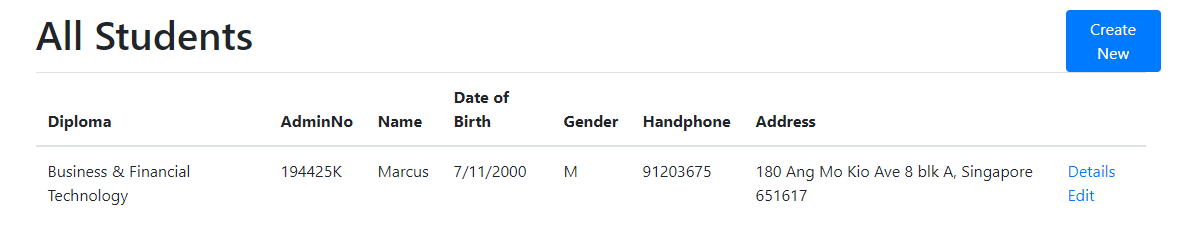
|  |
| --- |
| [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Edit(  string id,  [Bind("AdminNo,Name,Dob,Gender,ContactNumber,DiplomaId,Address")] Student student)  {  if (id != student.AdminNo)  {  return NotFound();  }  if (ModelState.IsValid)  {  try  {  \_context.Update(student);  await \_context.SaveChangesAsync();  }  catch (DbUpdateConcurrencyException)  {  if (!StudentExists(student.AdminNo))  {  return NotFound();  }  else  {  throw;  }  }  return RedirectToAction(nameof(Index));  }  else  {  ViewData["DiplomaSelectionList"] = new SelectList(\_context.Diploma, "DiplomaId", "DiplomaId", student.DiplomaId);  return View(student);  }  } |

Similar to create form, the form data is automatically bind to student object in method arguments.

After adding the student object to context through update method, database context will automatically keep track of changes made to the data and add pending update to entity framework.

Entity framework then execute the pending updates to database on SaveChangesAsync() called.

If you run the web application and click on the edit link, you should able to make changes to the student data and it will be reflected in our index page once the edit form is submitted.



# Delete Student Record

Next, we will add delete method to remove student record from database.

# Setup Our View and Confirm Modal

Unlike previous methods, we will not be creating a view to handle delete function, instead, we will be implementing the delete function in index page with the help of javascript.

* + 1. Add confirm delete modal

**Index.cshtml:**

|  |
| --- |
| <div class="modal fade" id="deleteStudentConfirmModal" tabindex="-1" role="dialog" aria-hidden="true">  <div class="modal-dialog" role="document">  <**form** **asp-action**="Delete">  <div class="modal-content">  <div class="modal-header">  <h5 class="modal-title">Delete Student Detail</h5>  <button type="button" class="close" data-dismiss="modal" aria-label="Close">  <span aria-hidden="true">&times;</span>  </button>  </div>  <div class="modal-body">  <label>Are you sure?!</label>  </div>  <div class="modal-footer">  <button type="button" class="btn btn-secondary" data-dismiss="modal">Cancel</button>  <button type="submit" class="btn btn-primary">Yes</button>  </div>  </div>  </**form**>  </div>  </div> |

Add the confirm modal directly below index student record table. This is a standard bootstrap modal.

* + 1. Add delete action link

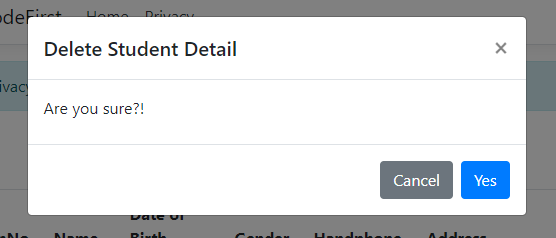
**Index.cshtml:**

|  |
| --- |
| @\*Add row action button here.\*@  <**a** **asp-action**="Details" **asp-route-id**="@item.AdminNo">Details</**a**>  <**a** **asp-action**="Edit" **asp-route-id**="@item.AdminNo">Edit</**a**>  <**a** **asp-action**="Delete" data-toggle="modal" data-target="#deleteStudentConfirmModal" data-adminno="@item.AdminNo">Delete</**a**> |

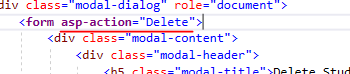
Add the delete action link in the table rows.

As we are using the build in bootstrap library, we can simply define the target with our confirm delete modal id and bootstrap will automatically handle the toggle interaction for us.

If you run the web application and click on the delete link, you should able to see our confirm modal.



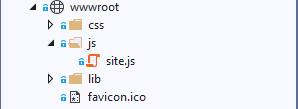
* + 1. Configure Our Delete Form Dynamically



If you look at our modal carefully, you will notice that it has a delete form. However, there isn’t a data attribute to indicate to the form which student record to be deleted.

This is because we are using a generic modal that can be use by all student row delete action. In order to delete the correct student record, we will have to add some java scripts to help us configure the delete form during client runtime in browser. We will be using jQuery library to simplify our implementation here.

In order to use jQuery library, our javascript code should be placed in site.js under our wwwroot/js folder.



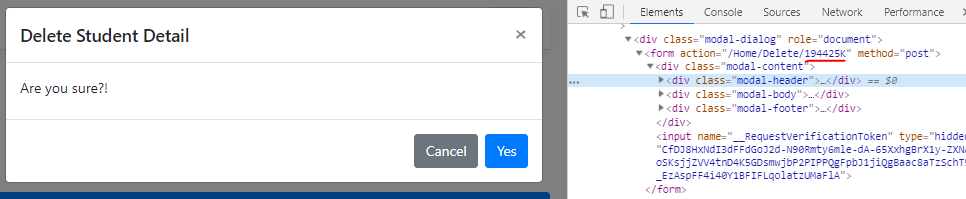
**Site.js:**

|  |
| --- |
| // Write your JavaScript code.  $('#deleteStudentConfirmModal').on('show.bs.modal', function (event) {  var button = $(event.relatedTarget);  var adminNo = button.data('adminno');  var modal = $(this);  modal.find('.modal-dialog form').attr('action', '/Home/Delete/' + adminNo);  }); |

Add the above javascript to site.js

By using jQuery selector, client will listen for modal opening event and append the value of data-“**adminno**” attribute we set in our table row delete action to our delete form. Now the delete form know the id of student to be submitted to our controllers.

If you run the web application and click on the delete link now and inspect our form element, you should be able to see the admin number of student that we would like to delete.



# Handle Delete Form Submission

Now the front end implementation is completed, we will implement our delete method in our backend to delete the student record.

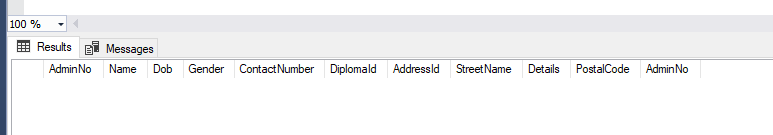
**HomeController.cs:**

|  |
| --- |
| [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Delete(string id)  {  var student = await \_context.Students  .FirstOrDefaultAsync(s => s.AdminNo == id);  \_context.Students.Remove(student);  await \_context.SaveChangesAsync();  return RedirectToAction(nameof(Index));  } |

Add the above code to our delete method in our controller.

We will search the database for our student object by the admin number we received from our delete form. We then inform the database context to delete the student data and apply the pending changes using SaveChangesAsync() .

If you run the web application and delete a student record, it will be remove from database. You can view the database to verify the delete.



# Handling Multiple Data using Batch Operation

Batch operation is a new feature of Entity Framework Core, It provides better performance when executing many SQL statements by batching up multiple INSERT/UPDATE/DELETE commands into a single roundtrip to the database.

# Batch Create Student Modules

* + 1. Setup our details view to list out student modules.

**Details.cshtml:**

|  |
| --- |
| @\*TODO Display student modules here.\*@  <div class="row col">  <h4 class="col">Modules</h4>  @if (Enumerable.Count(ViewBag.ModuleSelectionList) > 0)  {  <button type="button" class="btn btn-primary float-right col-md-3" data-toggle="modal" data-target="#addModuleModal">  Add Module  </button>  }  @\*TODO Add Delete Button\*@  </div>  <table class="table">  <tbody>  @if (Model.StudentModules.Count > 0)  {  @foreach (var item in Model.StudentModules)  {  <tr>  @\*TODO Add CheckBox Here\*@  <td>  @Html.DisplayFor(modelItem => item.Module.ModuleId)  </td>  <td>  @Html.DisplayFor(modelItem => item.Module.Name)  </td>  </tr>  }  }  else  {  <tr>  <td>  No modules assigned.  </td>  </tr>  }  </tbody>  </table> |

Add the above table in the student module section, this table will display a list of modules students assigned. We have added a create button to add modules to students, this button will open the following modal box.

**Details.cshtml:**

|  |
| --- |
| <div class="modal fade" id="addModuleModal" tabindex="-1" role="dialog" aria-hidden="true">  <div class="modal-dialog" role="document">  <**form** **asp-action**="AddModules" **asp-route-id**="@Model.AdminNo">  <div class="modal-content">  <div class="modal-header">  <h5 class="modal-title">Add Modules</h5>  <button type="button" class="close" data-dismiss="modal" aria-label="Close">  <span aria-hidden="true">&times;</span>  </button>  </div>  <div class="modal-body">  <div class="form-group">  <**label** **asp-for**="StudentModules" class="control-label"></**label**>  <**select** **asp-for**="StudentModules" class="form-control" **asp-items**="ViewBag.ModuleSelectionList"></**select**>  </div>  </div>  <div class="modal-footer">  <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>  <button type="submit" class="btn btn-primary">Save changes</button>  </div>  </div>  </**form**>  </div>  </div> |

Insert the above code in the bottom of details cshtml. This modal contains a selection list of modules for user to select to assign to students.

* + 1. Load Modules Data for Selection List

Since we have provided a module selection list in our add modules modal/pop up, we will have to supply it with data. Similar to diploma selection list, we will load our module data in our controller.

**HomeController.cs**

|  |
| --- |
| public async Task<IActionResult> Details(string id)  {  if (id == null)  {  return NotFound();  }  var student = await \_context.Students  .Include(s => s.Diploma)  .Include(s => s.StudentModules)  .ThenInclude(sm => sm.Module)  .Include(s => s.Address)  .FirstOrDefaultAsync(m => m.AdminNo == id);  if (student == null)  {  return NotFound();  }  var studentModuleIds = student.StudentModules.Select(c => c.ModuleId).ToList();  var modulesAvailable = \_context.Modules.Where(sm => !studentModuleIds.Contains(sm.ModuleId));  ViewData["ModuleSelectionList"] = new SelectList(modulesAvailable, "ModuleId", "Name");  return View(student);  } |

There are 2 changes here compared to the previous details routing method.

**Loading Student Modules:**

**ThenInclude** is used to link to entities that are indirectly related to Student entity. As you know, we have a **many to many** relationship in our data base schema, **ThenInclude** is equivalent to joining multiple database.

**Preparing Module Selection List Data:**

Compare to diploma selection list, module selection list should be pre-process before passing to view for rendering. Pre-processing is needed because we want to filter out modules that student have already been assigned to from the selection list.

Firstly we get a list of module ids from the context.



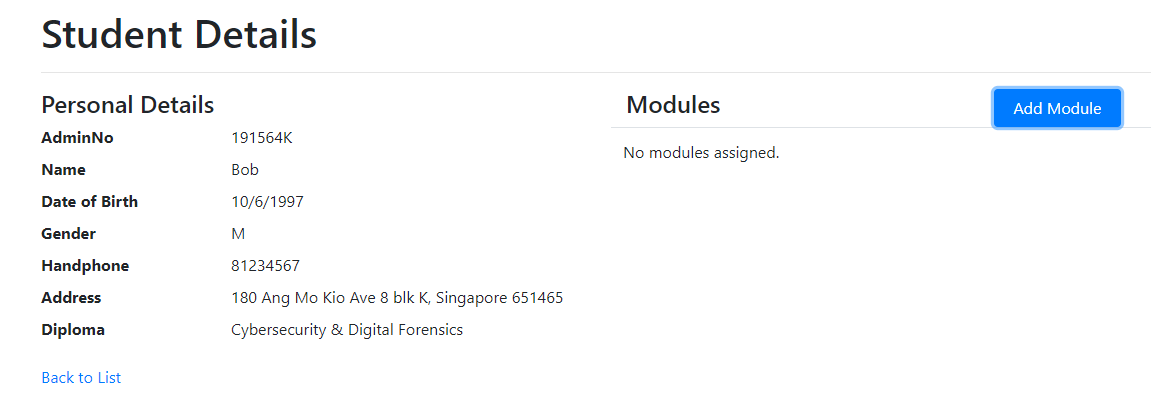
Next we use a where function to filter out modules that are no in the list of assigned student modules.



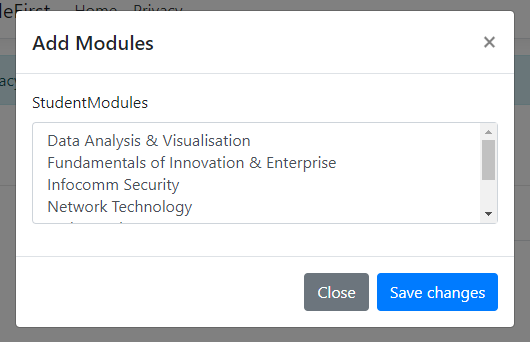
Then finally we store It in our ViewData container and pass it to details view to render.



If you run the web application and view a student detail, you should see the following content.



And if you click on “**Add Modules”**, you should see a modal with a multiple selection list option.



You can select multiple item by “**Drag-Select**” or hold “**Ctrl**” key on keyboard and click on modules you wish to assign.

* + 1. Configure Batching Size

We can configure the batching size settings for our database context.

**Startup.cs**

|  |
| --- |
| public void ConfigureServices(IServiceCollection services)  {  :  services.AddDbContext<StudentManagementSystemContext>  (options => options.UseSqlServer(connectionString, o => o.MaxBatchSize(100)));  } |

This controls the number of statement entity framework executes for each database operation, for demo we will set to 100. To disable batching, simply set the batching size to 1.

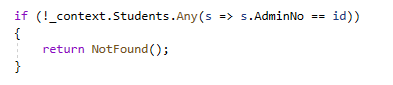
* + 1. Handle Add Module Form Submission

**HomeController.cs**

|  |
| --- |
| [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> AddModules(string id, string[] StudentModules)  {  if (!\_context.Students.Any(s => s.AdminNo == id))  {  return NotFound();  }  if (StudentModules.Length > 0)  {  foreach (var moduleId in StudentModules)  {  \_context.StudentModules.Add(new StudentModules { AdminNo = id, ModuleId = moduleId });  }  await \_context.SaveChangesAsync();  }  return RedirectToAction(nameof(Details), new { id = id });  } |

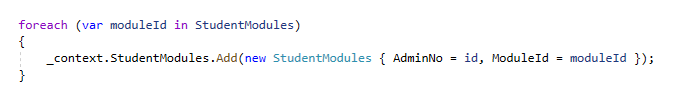
Add the above code in home controller. When the add module form is submitted, the selection will be bind to StudentModules string array.

1. Firstly we will ensure that the student exists using **Any** method.



1. Next, we loop through our array and add the Modules to student modules context. At this point, no SQL is been executed, all INSERT will be batch together, this is known as batching commands.

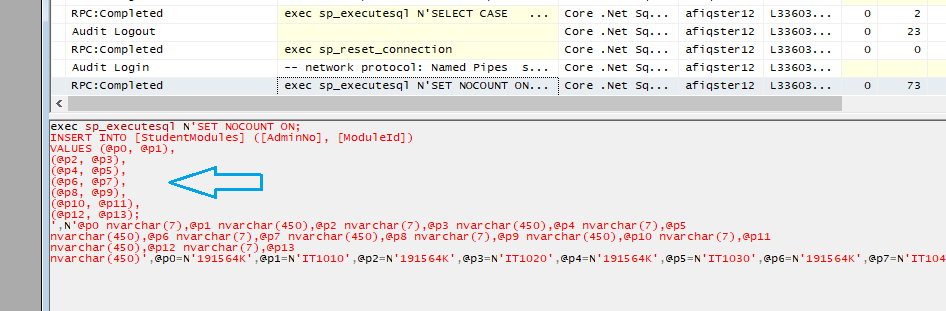
**Batching Commands:**



1. The actual INSERT will be triggered by saving command.



And if you examine the SQL statement generated by Entity Framework, you will see that there is only a single line of INSERT statement, resulting in a more efficient database access.



# Batch Delete Student Modules

* + 1. Encapsulate the table with a delete form

**Details.cshtml**

|  |
| --- |
| @\*TODO Display student modules here.\*@  <**form** **asp-action**="DeleteModules" **asp-route-id**="@Model.AdminNo">  <div class="row col">  <h4 class="col">Modules</h4>  @if (Enumerable.Count(ViewBag.ModuleSelectionList) > 0)  {  <button type="button" class="btn btn-primary float-right col-md-3" data-toggle="modal" data-target="#addModuleModal">  Add Module  </button>  }  @\*TODO Add Delete Button\*@  @if (Model.StudentModules.Count > 0)  {  <button type="submit" class="btn btn-danger float-right col-md-4">  Remove Modules  </button>  }  </div>  <table class="table">  <tbody>  @if (Model.StudentModules.Count > 0)  {  @foreach (var item in Model.StudentModules)  {  <tr>  @\*TODO Add CheckBox Here\*@  <td>  <input type="checkbox" class="form-check-input" name="StudentModules" value="@item.ModuleId">  </td>  <td>  @Html.DisplayFor(modelItem => item.Module.ModuleId)  </td>  <td>  @Html.DisplayFor(modelItem => item.Module.Name)  </td>  </tr>  }  }  else  {  <tr>  <td>  No modules assigned.  </td>  </tr>  }  </tbody>  </table>  </**form**> |

We have added a delete button and checkboxes for each table row in the above code.

* + 1. Handle Delete Module Form

**HomeController.cs**

|  |
| --- |
| [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> DeleteModules(string id, string[] StudentModules)  {  if (!\_context.Students.Any(s => s.AdminNo == id))  {  return NotFound();  }  if (StudentModules.Length > 0)  {  foreach (var moduleId in StudentModules)  {  \_context.StudentModules.Remove(new StudentModules { AdminNo = id, ModuleId = moduleId });  }  await \_context.SaveChangesAsync();  }  return RedirectToAction(nameof(Details), new { id = id });  } |

If you run the web application, you should be able to delete selected module in student details page.

