CO550 – WEB APPLICATIONS

ASP.NET CORE TECHNICAL “HOW TO” GUIDE

BUILDING AN ASP.NET CORE WEB APP

For this module, the Web Apps CW2 assignment requires you to complete at least some coding within ASP.NET.

This presentation outlines the fundamental functions and features you are expected to be able to implement. These have been divided into the following sections:

- **Fundamentals** (ideally I'd like to see you achieving all of these)
- **Advanced** (some more advanced techniques you can try to improve your grade)

The guidance on this presentation assumes you are tackling a **Razor Pages** project (not MVC)

bucks new university

THE FUNDAMENTALS

bucks new university

FUNDAMENTALS

1. **Code First Model Creation** - C# classes in a “Models” folder
2. **Scaffolding Your Model** - for CRUD functionality
3. **Database Migrations** – to manage database creation and changes
4. **Data Annotations** – on your Models, for implementing validation, for example
5. **Seeding Data**

bucks new university

1. CODE FIRST MODEL CREATION

Refer to the “Get Started” section of the Microsoft Razor Pages tutorial: https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/intro?view=aspnetcore-2.2&tabs=visual-studio

Create your models as classes. Example below:

C#

```
using System;
using System.Collections.Generic;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        public DateTime EnrollmentDate { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

bucks new university

2. SCAFFOLDING YOUR MODEL

Again, refer to the “Get Started” section of the Microsoft Razor Pages tutorial: https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/intro?view=aspnetcore-2.2&tabs=visual-studio#scaffold-the-student-model

Your aim: to generate the CRUD pages (Create, Read, Update, Delete) within the “Pages” folder of your Razor Pages project

bucks new university

3. DATABASE MIGRATIONS

For detailed information, refer to the relevant section on the Microsoft tutorial: https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/migrations?view=aspnetcore-2.2&tabs=visual-studio

The underlying principles, though, are fairly simple:

- Database Migrations are effectively version control for your database
- Whenever you make a change to any of your models, you need to do two things:
  1. Add a Migration ("Add-Migration <MigrationName>" command)
  2. Update the database ("Update-Database" command)
- You run the above commands in the "Package Manager Console" in Visual Studio
- If you get a "database cannot be opened" error when trying to run your web app, you likely need to run "Update-Database" to create the database.
- Each migration file is stored in the "Migrations" folder and is a basic C# class with an Up() method and a Down() method

bucks new university

4. DATA ANNOTATIONS

- Use data annotations to make your models more comprehensive and useful.
- You can use data annotations to specify the front-end label for an attribute, whether it is a required field or not, other validation and formatting rules, etc…
- Data annotations are included in square brackets and you must include the relevant “using” statements at the top of the class:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

Reference: https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/complex-data-model?view=aspnetcore-2.2&tabs=visual-studio

bucks new university

4. DATA ANNOTATIONS

Example Data Annotations…

```
namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        [Required]
        [StringLength(50)]
        [Display(Name = "Last Name")]
        public string LastName { get; set; }
        [Required]
        [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]
        [Column("FirstName")]
        [Display(Name = "First Name")]
        public string FirstMidName { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Enrollment Date")]
        public DateTime EnrollmentDate { get; set; }
```

5. SEEDING DATA

“Seeding data” refers to the process of prepopulating the database with data hard-coded into the project code (a “database initialiser”).

The purpose of this: prepopulate your database with some sample records to demonstrate how your application will function with data. Leaving your database empty will not help in testing the functionality.

See final part of “Getting Started” part of Microsoft tutorial: https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/intro?view=aspnetcore-2.2&tabs=visual-studio#add-code-to-initialize-the-db-with-test-data

5. SEEDING DATA

```
namespace ContosoUniversity.Models
{
    public static class DbInitializer
    {
        public static void Initialize(SchoolContext context)
        {
            // context.Database.EnsureCreated();

            // Look for any students.
            if (context.Student.Any())
            {
                return;   // DB has been seeded
            }

            var students = new Student[]
            {
            new Student{FirstMidName="Carson",LastName="Alexander",EnrollmentDate=DateTime.Parse
            new Student{FirstMidName="Meredith",LastName="Alonso",EnrollmentDate=DateTime.Parse(
            new Student{FirstMidName="Arturo",LastName="Anand",EnrollmentDate=DateTime.Parse("20
            new Student{FirstMidName="Gytis",LastName="Barzdukas",EnrollmentDate=DateTime.Parse(
            new Student{FirstMidName="Yan",LastName="Li",EnrollmentDate=DateTime.Parse("2002-09-
            new Student{FirstMidName="Peggy",LastName="Justice",EnrollmentDate=DateTime.Parse("2
            new Student{FirstMidName="Laura",LastName="Norman",EnrollmentDate=DateTime.Parse("20
            new Student{FirstMidName="Nino",LastName="Olivetto",EnrollmentDate=DateTime.Parse("2
            };
```

ADVANCED FUNCTIONALITY

bucks new university

ADVANCED FEATURES AND FUNCTIONALITY

If you achieve the fundamentals, move on to trying to implement some of these…

1. **Displaying Related Data**
2. **Customised Login and Registration** – using ASP.NET Identity
3. Sorting, Filtering, Paging
4. Saving / Updating Related Data
5. Uploading Files or Images
6. Sending emails

bucks new university

1. DISPLAYING RELATED DATA

- An example: we may wish to display a Student's enrollments on their Details page. A student has (or is linked to) multiple enrollments.
- We need to build a query which fetches this related data and makes it available to the page we are working on.

See "Add related data" part of section 2 (of 8) of the Microsoft tutorial: https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/crud?view=aspnetcore-2.2#add-related-data

Also see the "Read Related Data" section (6 of 8) of the Microsoft tutorial for a **more advanced implementation**: https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/read-related-data?view=aspnetcore-2.2&tabs=visual-studio

bucks new university

1. DISPLAYING RELATED DATA - EXAMPLE

Option 1: on the OnGetAsync() method of a page, we can customize the underlying data query to pull in related data:

```
Student = await _context.Student
                        .Include(s => s.Enrollments)
                            .ThenInclude(e => e.Course)
                        .AsNoTracking()
                        .FirstOrDefaultAsync(m => m.ID == id);
```

bucks new university

1. DISPLAYING RELATED DATA - EXAMPLE

Option 2: on the OnGetAsync() method of a page, we can build a completely separate LINQ query and assign the results to an IEnumerable object:

```
IEnumerable<PurchaseOrderLines> purchaseOrderLinesIQ = from s in _context.PurchaseOrderLines
                                                       where s.DeliveryID == id
                                                       select s;



PurchaseOrder.POLines = purchaseOrderLinesIQ;
```

- Notice the LINQ syntax to build the query
- Note that “id” has been passed in via a URL parameter on the page (?id=10 for example)

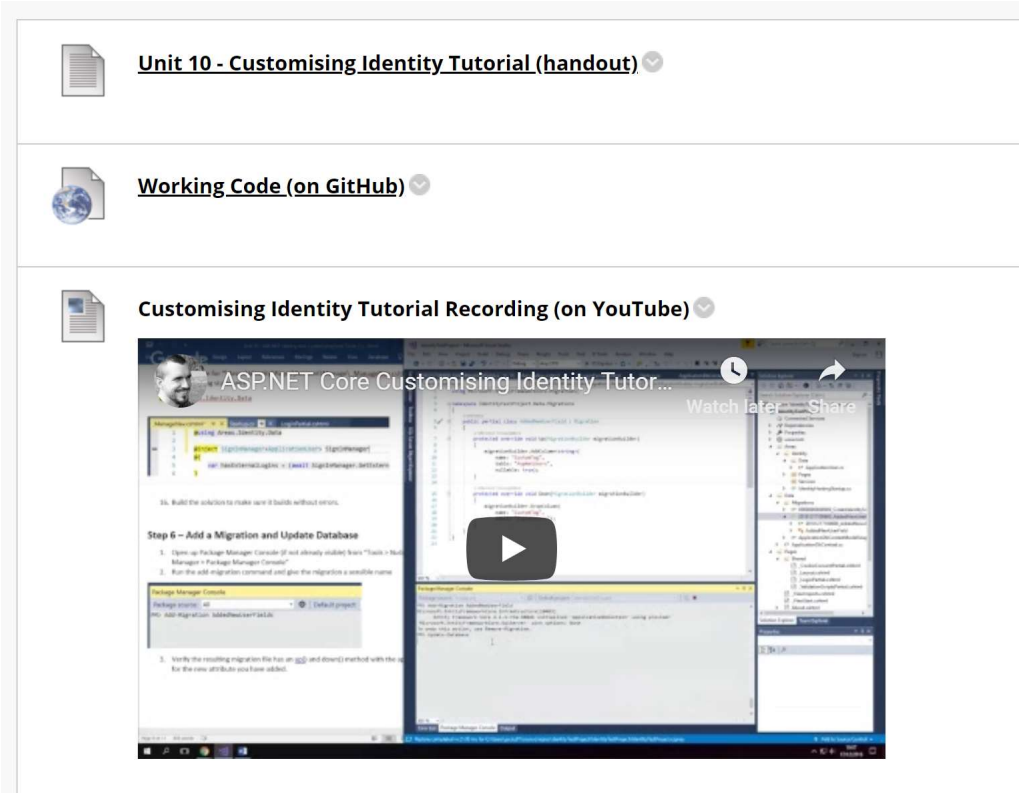
```
public async Task<IActionResult> OnGetAsync(int? id)
{
```

- You can then use a foreach loop in the Razor Page to loop over the data held in the “POLines” property, in this example

bucks new university

2. CUSTOMISED LOGIN AND REGISTRATION

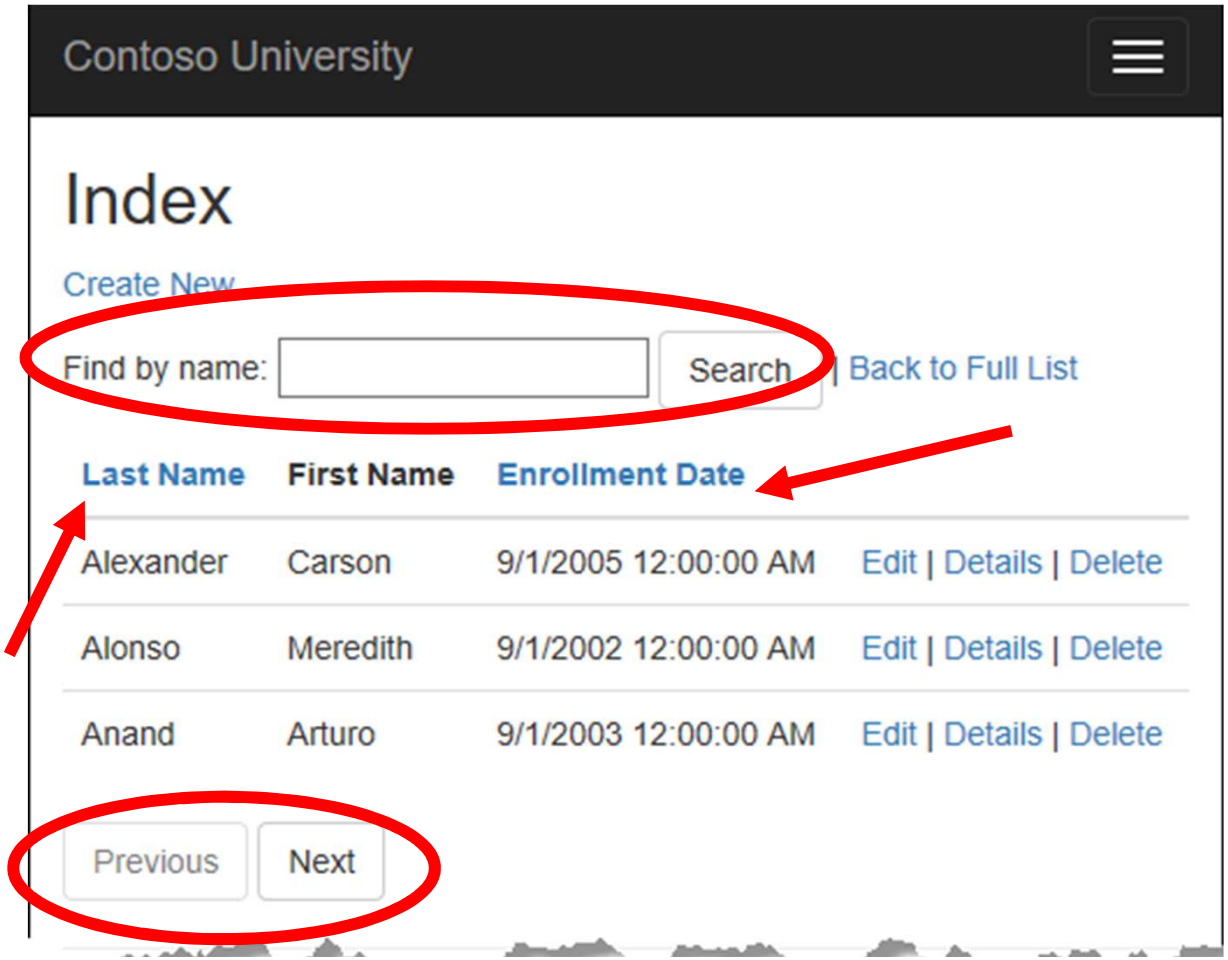
- Refer to Unit 10 materials on Blackboard and the working code also supplied on GitHub: https://github.com/iamjonjackson/IdentityCustomisationTest/releases/tag/0.1.0 (use this as the basis of your project)
- You can also add Identity to an existing project but it requires some extra changes to get it working effectively in the same database context



bucks new university

3. SORTING, FILTERING, PAGING

- Refer to this section of the Microsoft Tutorial: https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/sort-filter-page?view=aspnetcore-2.2



bucks new university

4. SAVING / UPDATING RELATED DATA

See the “Update Related Data” section (7 of 8) of the Microsoft tutorial: https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/update-related-data?view=aspnetcore-2.2

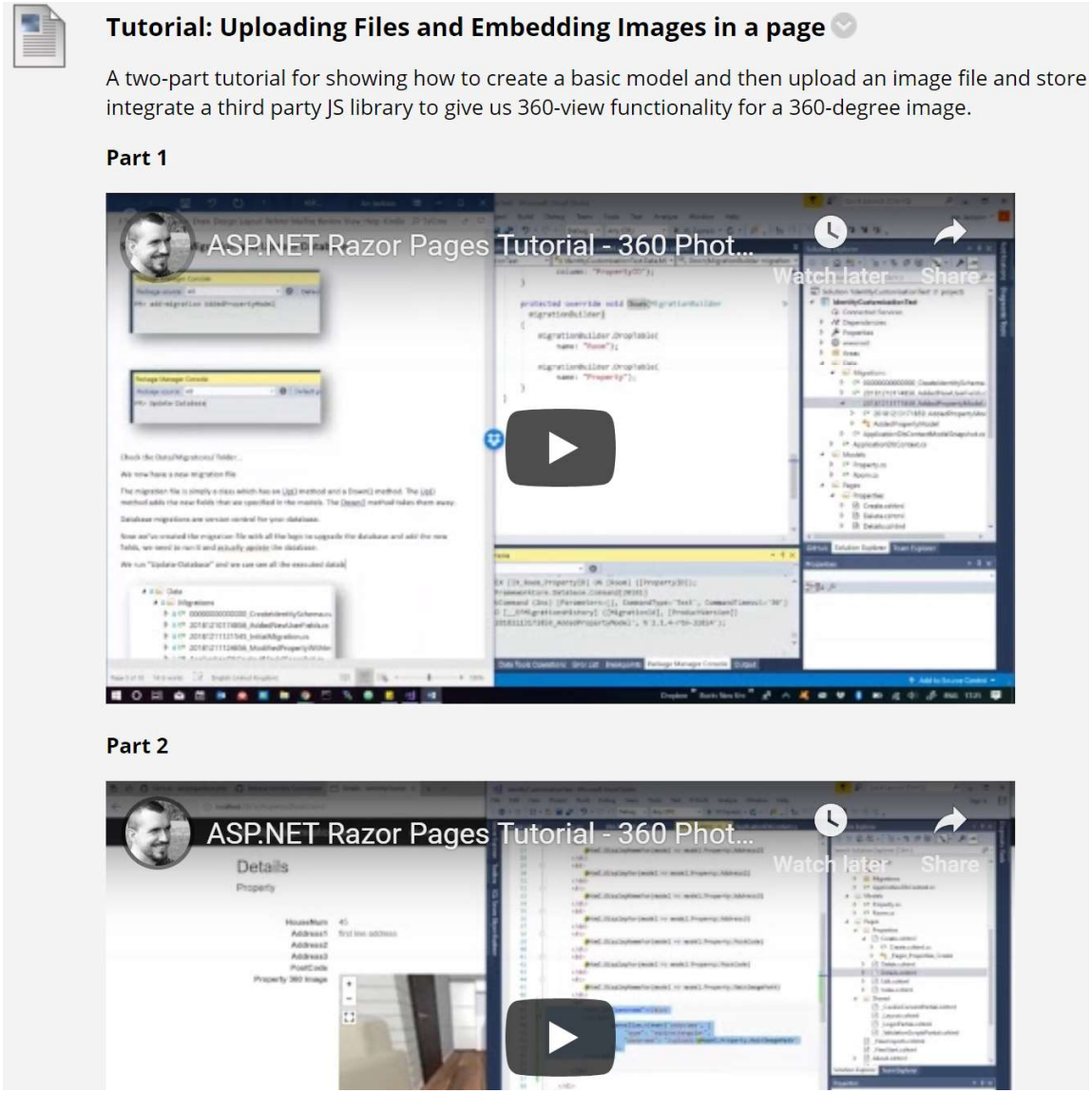
bucks new university

5. UPLOADING FILES OR IMAGES

Tutorials

- https://docs.microsoft.com/en-us/aspnet/core/razor-pages/upload-files?view=aspnetcore-2.1
- https://www.learnrazorpages.com/razor-pages/forms/file-upload

Video Walkthrough: see Unit 10 on Blackboard for the recorded YouTube demos which cover this.



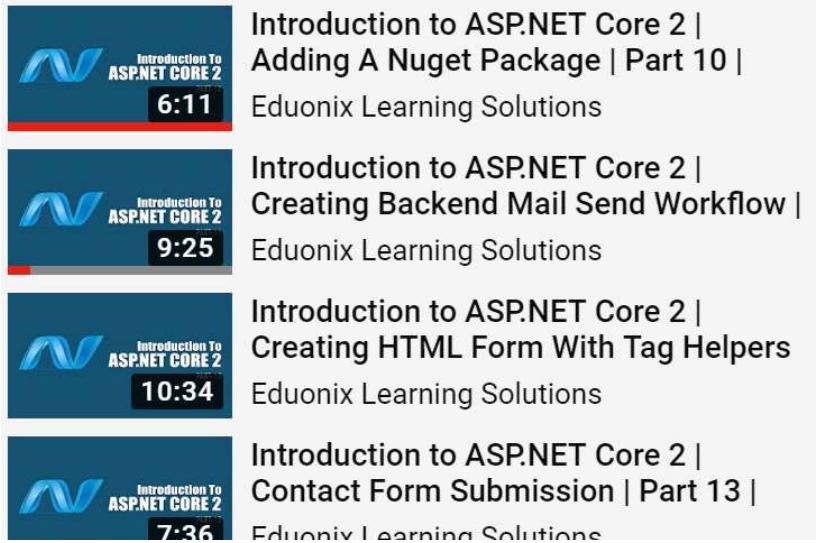
6. SENDING EMAILS

Your system may have a requirement for sending email notifications when some user action is carried out. Some examples:

- Confirmation of a new booking having been made
- Confirmation of a new game tournament having been entered
- An email alert to the admin user when stock levels of a product are running low

https://www.youtube.com/watch?v=E5SNMd8MO04&index=9&list=PLDmvslp_VR0x2CmC6c4AZhZfYX7G2nBIo

Watch videos 10-13 of this YouTube series for a full walkthrough to send email from your web app via Gmail



AND REMEMBER...

DOCUMENT YOUR CODING

- Remember to explain and showcase your coding efforts in your technical report, part of your CW2 submission.
- Make it easy for the marker to see the types of functionality you have implemented, how you've done it, and whether or not you understand it!
- Don't be afraid to go into detail
- Include screenshots of your code where applicable (but make sure they are readable)!