WEB APPLICATION With USER ACCOUNTS

Table of Contents

November 2, 2019
Prepared by Andre Masters
Revision 1.0

November 2, 2019
Prepared by Andre Masters
Revision 1.0

Project Setup

This project implements continuous deployment practices using a code repository called Github. The purpose of having a continuous deployment strategy is three fold; one reason is to back up the code and also to provide a revision history of what has been built up to the current date. The most important reason is that it allows for multiple developers to work on the source code while at the same time not step on each other. The following steps will accomplish the task of getting the project ready for prime time.

1.0 Create a Github account using the following link www.github.com

1.1 Create a repository name it anything you like for example **EmployeeWebmanagent**

1.2 Enter the Url of the repository in this case AppMaster/EmployeeWebmanagement

1.3 Install Visual Studio 2019 community edition to your work station.

1.4 Install Asp.net Core 2.2 libraries

1.5 Create a new Asp.Net Core Web Application project, select empty project, and uncheck https.

1.6 Name the project anything you like example **EmployeeManagementWebApplication**

1.6 Run the project and observe the following appears as shown in Figure 1.0

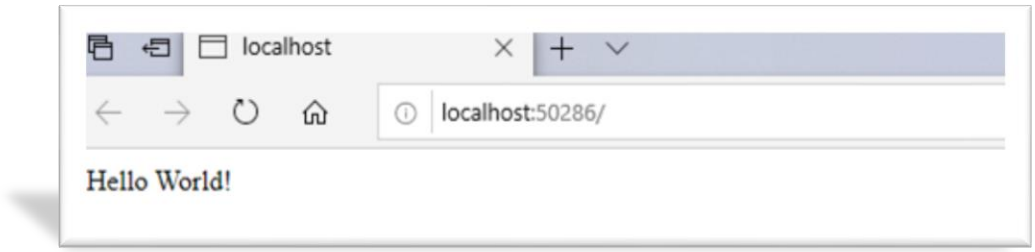


Figure 1.0

1.7 In the Startup.cs file change the output text from hello world to hello your project name.
1.8 Recompile the solution and observe no errors occur.

1.9 Select the project add to Source Control observe locks appear

1.9 Select the solution RMB click select the **Commit** option and observer the following appears as shown in Figure 2.0

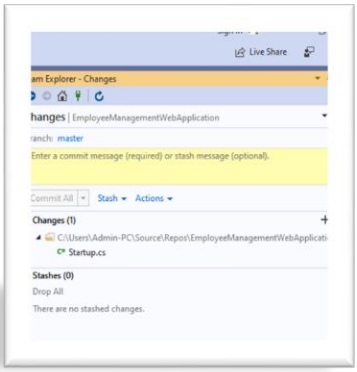


Figure 2.0

November 2, 2019
Prepared by Andre Masters
Revision 1.0

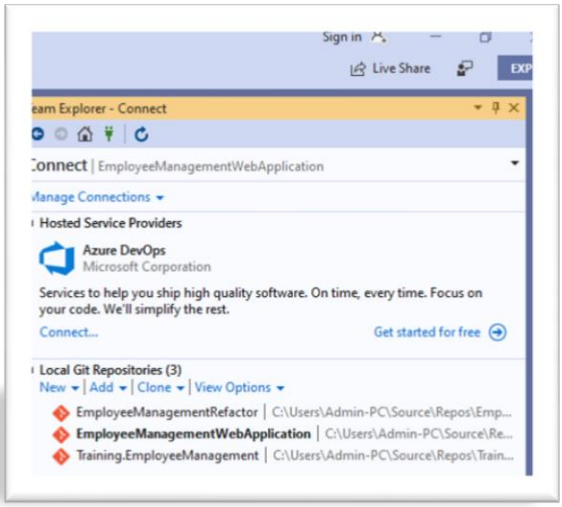


Figure 3.0

1.10 Select **Connections Option** the green plug in Figure 2.0 observe the following appears as shown in Figure 3.0

1.11 Create a new repository by selecting the ADD option in Figure 3.0.

1.12 Give the repository a new name in this case its **EmployeeManagementWebApplication**

1.13 Select the **Clone** option enter the URL address from the Github repository.

NOTE: The url needs to include the full repository path.

https://github.com/apmaster/EmployeeManagermentWeb

1.14 Select the branch that was created in step 1.12 in this case

EmloyeeManagementWebApplication

1.14 Select the **Settings** option in Figure 4, for the current solution. Observe the following appears as shown in Figure 5.0

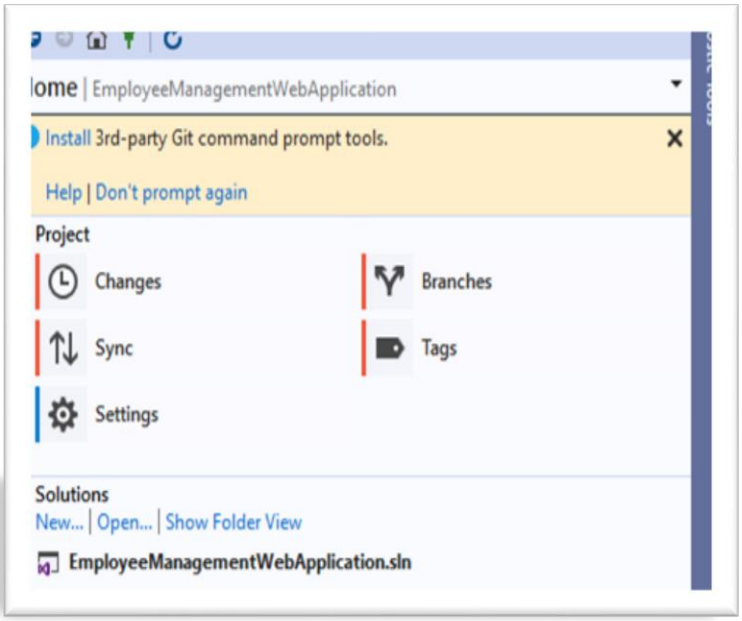


Figure 4.0

November 2, 2019
Prepared by Andre Masters
Revision 1.0

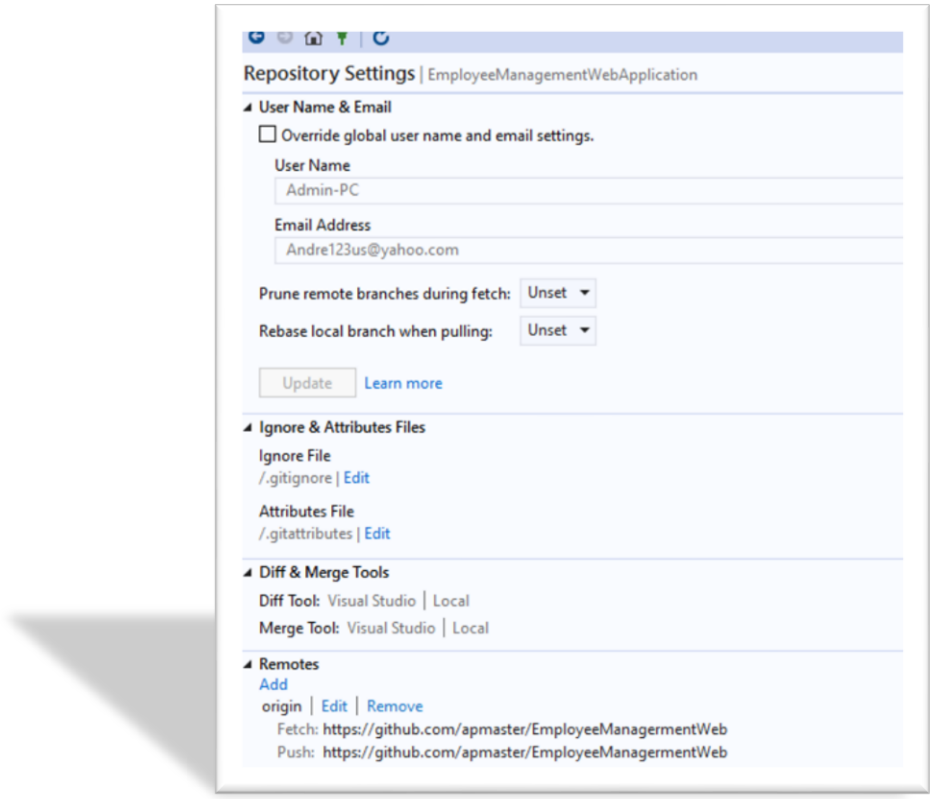


Figure 5.0

1.15 In the **Remotes** section, verify the repository address matches your **Github** repository. This is the location where your code will be placed during the commit operation.

1.16 Make a commit by selecting the solution name, observe the commit screen appears as shown in Figure 3.0. Enter the following comment in the commit screen.

Changed the Output Message from Hello World to Hello project name

1.17 Press the Commit All button and observe the following appears as shown in Figure 6.0

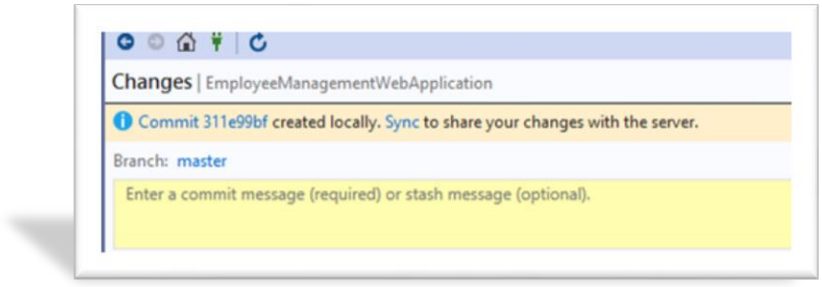


Figure 6.0

1.18 Press the **Sync** link to update the remote repository in the Github account. Observe the following appears as shown in Figure 7.0

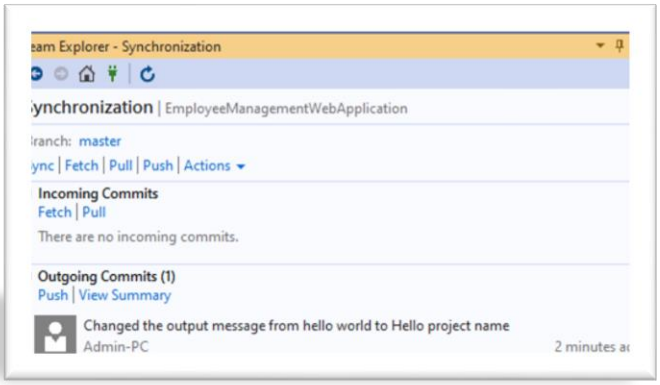


Figure 7.0

1.18 Press the link Push to update the Github repository with the update changes, observe the following appears indicate the changes occurred successfully as shown in Figure 8.0

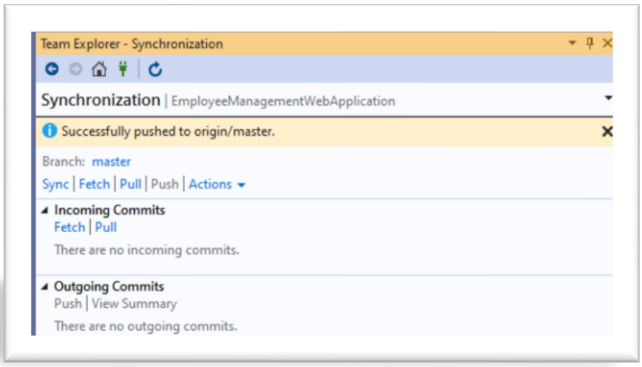


Figure 8.0

1.19 Navigate to the Github account to verify the change took place as shown in Figure 9.0

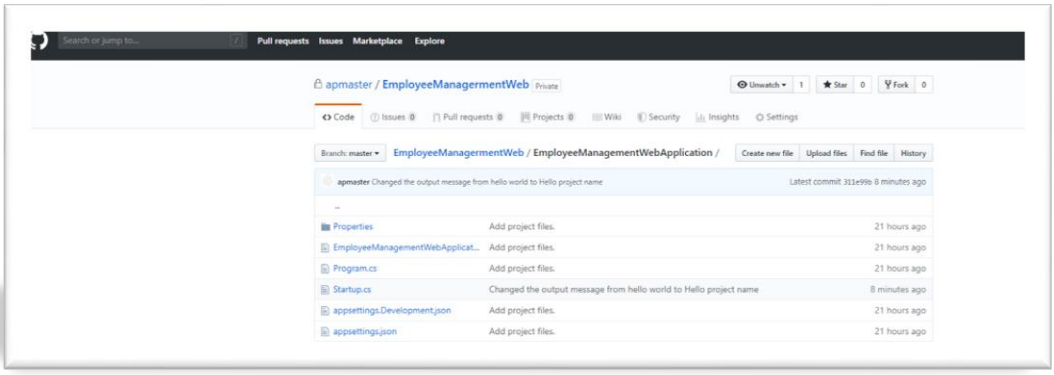


Figure 9.0

November 2, 2019
Prepared by Andre Masters
Revision 1.0

1.20 Select the **Startup.cs** file in the Githup repository to open it and observe the following takes place as shown in Figure 10

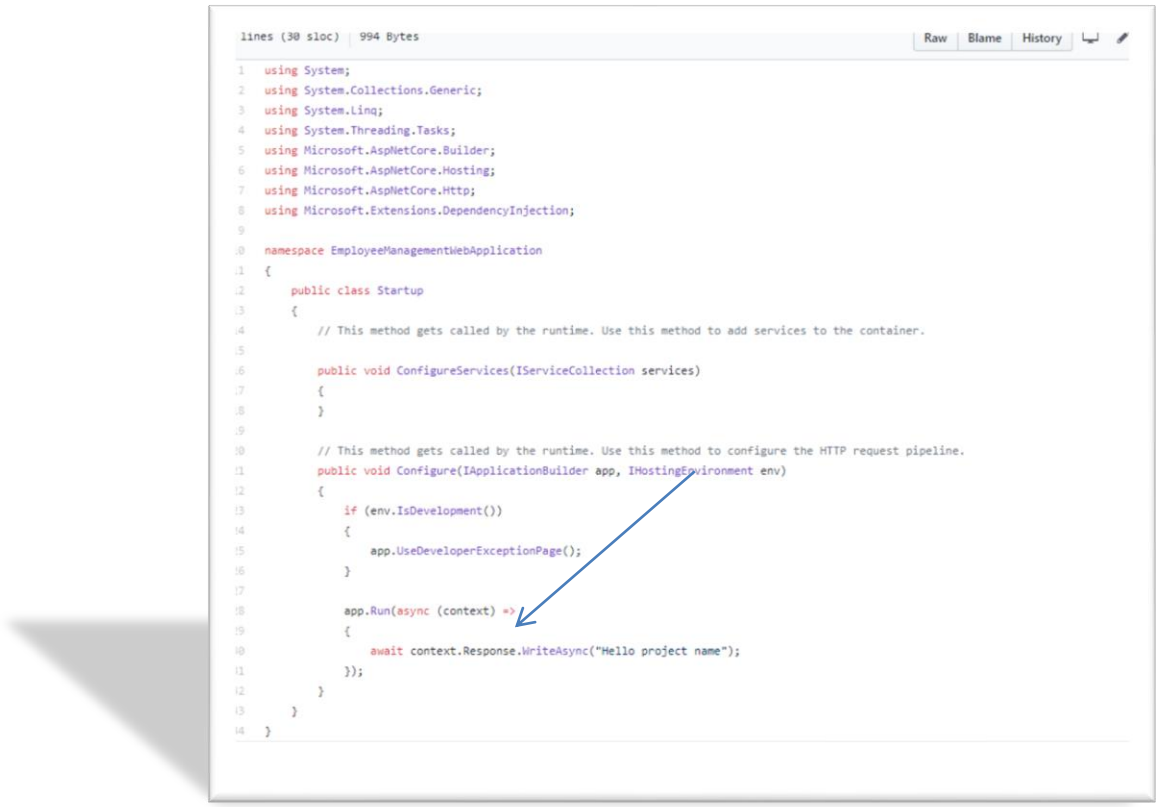


Figure 10.0

Main Application Solution Structure

This project will utilize the MVC design pattern. The aim is to create folders and class templates representing objects for model, view and controller. Pull the **EmployeeManagementWebApplication** in the **EmployeeManagerment respository** from the Github using the process in Project Setup section.

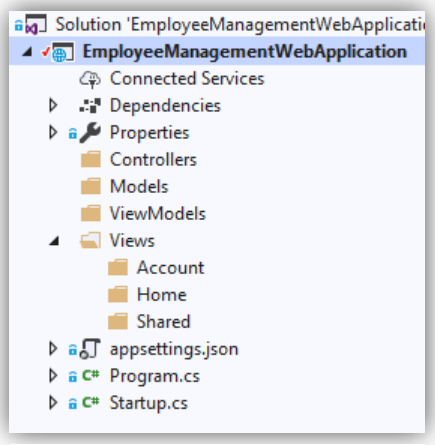


Figure 11

2.0 In the **EmployeeManagementWebApplication** project create the folder structure as displayed in Figure 11.0

2.1 In the Controller Folder Create the following classes as shown in green.

- AccountController
- ErrorController
- HomeController

2.2 In the **Models Folder** create the following classes as shown in blue.

- AppDbContext
- Employee
- IEmployeeRepository
- SQLEmployeeRepository

2.3 In the **ViewModels Folder** create the following classes as shown in red.

- EmployeeCreateViewModel
- EmployeeDeleteViewModel
- EmployeeEditViewModel
- EmployeeDetailsViewModel
- LoginViewModel
- RegisterViewModel

2.4 In the **View -> Account Folder** create the following razor view templates as shown in purple.

- Login
- Register

2.5 In the **View -> Home Folder** create the following razor view templates as shown in purple.

- Create
- Delete
- Details
- Edit
- EmployeeNotFound
- Index

2.6 In the **View -> Shared Folder** create the following razor view templates as shown in purple.

- _Layout
- Error
- NotFound

2.7 In the **View Folder** create the following razor view templates as shown in orange.

- _ViewImports
- _ViewStart

2.8 Compile the project by pressing F5 and observe it compiles successfully by observe the output window in Visual studio contains no errors.

2.9 Commit the project to the Github repository on the main branch using the same sequence of steps contained in the Project Setup Section. Add the following information in the comments section.

November 2, 2019
Prepared by Andre Masters
Revision 1.0

Main Application Setup

Open the application solution on the local machine. In this section, the application controller will be built along with all supporting libraries.

3.0 Create a **wwwroot** folder directory in the main project directory.

3.1 Select Add from the Project Root and navigate to Client-Side Library as shown in Figure 12.

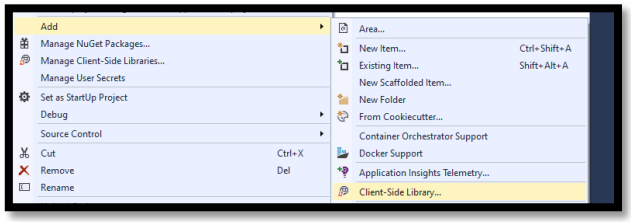


Figure 12

3.2 Add **Libmam** to the project for client side libraries select ALL as shown in Figure 13.

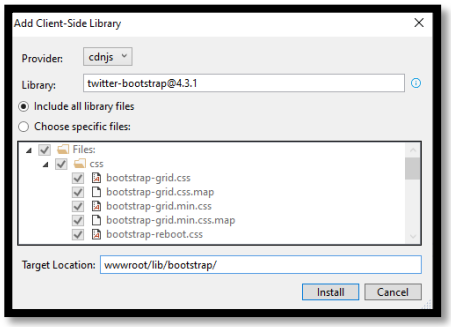


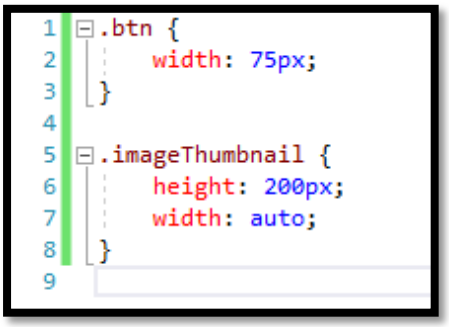
Figure 13

```
{
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "libraries": [
    {
      "library": "twitter-bootstrap@4.3.1",
      "destination": "wwwroot/lib/bootstrap/"
    }
  ]
}
```

Figure 14

3.3 Observe the contents of the **libman.json** file as shown in Figure 14. Pay Close attention to the Bootstrap version it will be referenced in later iterations of this procedure and it will also serve as a key item for configuration management.

3.4 Verify the **site.css** file exists and contains the following content as displayed in Figure 15.



```
.btn {
    width: 75px;
}

.imageThumbnail {
    height: 200px;
    width: auto;
}
```

Figure 15

3.5 In the **Configure.cs** file under the **Configuration Services Method,** replace the existing code with the following code construction.

```
services.AddMvc();
services.AddSingleton<IEmployeeRepository,MockEmployeeRepository>();
```

3.6 Add the following code construction in **Startup.cs** file under the **Configure Method** replace the existing code with the following code construction.

```
app.UseStaticFiles();
app.UseMvcWithDefaultRoute();
```

Home Page Implementation

This section will focus on the code construction of the Home Controller. The purpose of the controller is to interact between the view and the models. ASP.Net core has middleware that will allow for dependency injection between model and view components without the need to instantiate the components for each use.

4.0 Inside the **Home Controller Class** implement a method to handle the dependency activity for the model object.

```
private IEmployeeRepository _employeeRepository;

public HomeController(IEmployeeRepository employeeRepository)
{
    _employeeRepository = employeeRepository;

}
```

4.1 Inside the **Home Controller Class** implement a method to handle passing data from the model to the **Index View**.

```
public ViewResult Index()
{
    var model = _employeeRepository.GetAllEmployees();

    return View(model);
}
```

4.2 Inside the **Models Folder**, in the **Employee Model**, enter the following code whose aim is to serve as a data transfer object.

```
public int Id { get; set;}
public string Name { get; set;}
public string Email { get; set;}
public string Department { get; set;}
```

4.3 Inside the **Models Folder** in the **Mock Employee Repository**, enter the following code.

```
private List<Employee> _employeeList;

public MockEmployeeRepository()
{

    _employeeList = new List<Employee>()
    {
    new Employee() { Id = 1, Name = "Mary", Department = "HR", Email = "mary@pragimtech.com" },
    new Employee() { Id = 2, Name = "John", Department = "IT", Email = "john@pragimtech.com" },
    new Employee() { Id = 3, Name = "Sam", Department = "IT", Email = "sam@pragimtech.com" },
    };
}

public IEnumerable<Employee> GetAllEmployees()
{
    return _employeeList.ToList();
}
```

4.4 Inside the **IEmployeeRepository** interface enter the following code.

```
        Employee GetEmployee(int id);
        IEnumerable<Employee> GetAllEmployees();
```

4.5 Inside the **Index.cshtml** view file enter the following code.

```
@model IEnumerable<Employee>
@{
    ViewBag.Title = "Employee List";
}
<div class="card-deck">
    @foreach (var employee in Model)
    {
        <div class="card m-3" style="min-width:18rem; max-width:30.5%">
            <div class="card-header">
                <h3>@employee.Name</h3>
            </div>
            <div class="card-footer text-center">
                <a asp-controller="home" asp-action="details" asp-route-id="@employee.Id"
                   class="btn btn-primary m-1">View</a>
                <a asp-action="edit" asp-controller="home" asp-route-id="@employee.Id"
                   class="btn btn-primary m-1">Edit</a>
                <a asp-action="delete" asp-controller="home" asp-route-id="@employee.Id"
                   class="btn btn-danger m-1">Delete</a>
            </div>
        </div>
    }
</div>
```

NOTE:

Bootstrap Library Code – The Violet colored text indicates Bootstrap styling. In particular, a bootstrap card is used to format the html content.

NOTE:

Model Binding is used for data transfer of the employee object.

4.6 Compile the project and run it in the browser and observe the following content appears as displayed in Figure 16.

Figure 16

COMMIT THE PROJECT: HOME PAGE IMPLEMENTATION COMPLETE

FULL COMMIT Merge Github Repo Main Page

The left hand pane is the **Remote Copy** and the Right hand pane is the **Local Copy**. The Team Explorer window is an illustration of a successful merge of the local project to the remote project in GitHub.

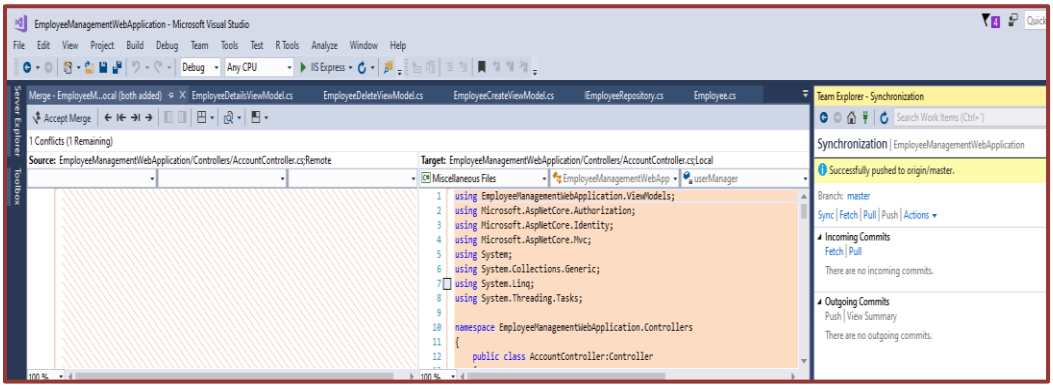


Figure 17

NOTE:

If the project gets pushed from the local workstation to the remote repository, the merge will fail because Github does not know which copy you want to use. To remedy this issue, simply select the merge option and select the local project location since this where the work has been done for the current application. Beware, if the opposite is done the local project will be replaced with the old code and all of the effort thus far will be lost.

November 2, 2019
Prepared by Andre Masters
Revision 1.0

Details Implementation

This section focuses on the creation and construction of the **Details View Page**. In order to make the page function it needs to have a **View Model**, **Model**, and **Controller**. The **View** in this case uses a strongly typed **View Model** and a **view-bag** for the title page. The page styling **uses Boostrap 4.0** and **Razor Helper Tags**.

5.0 Create a Details View Page using the code construction below.

```
@model HomeDetailsViewModel
@{
    ViewBag.Title = "Employee Details";
}
<div class="row justify-content-center m-3">
    <div class="col-sm-8">
        <div class="card">
            <div class="card-header">
                <h1>@Model.Employee.Name</h1>
            </div>
            <div class="card-body text-center">
                <img class="card-img-top" src="~/images/1.JPG" asp-append-version="true" />
                <h4>Employee ID : @Model.Employee.Id</h4>
                <h4>Email : @Model.Employee.Email</h4>
                <h4>Department : @Model.Employee.Department</h4>
            </div>
            <div class="card-footer text-center">
                <a asp-controller="home" asp-action="index" class="btn btn-primary">Back</a>
                <a href="#" class="btn btn-primary">Edit</a>
                <a href="#" class="btn btn-danger">Delete</a>
            </div>
        </div>
    </div>
</div>

@section Scripts {
    <script src="~/js/CustomScript.js"></script>
}
```

5.1 In the **ViewModels** folder, open the **EmployeeDetailsViewModel** file. Enter the following code construction as shown below.

```
using EmployeeManagementWebApplication.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace EmployeeManagementWebApplication.VewModels
{
    public class EmployeeDetailsViewModel
    {
        public Employee Employee { get; set; }
        public string PageTitle { get; set; }
    }
}
```

5.2 In the **HomeController** file, add the following code construction for implementing the Details feature.

```
public ViewResult Details(int? id)
{
    EmployeeDetailsViewModel homeDetailsViewModel = new EmployeeDetailsViewModel()
    {
        Employee = _employeeRepository.GetEmployee(id ?? 1),
        PageTitle = "Employee Details"
    };
    return View(homeDetailsViewModel);
 }
```

5.3 Press F5 to run and compile the project. Select any employee **Employee Detail** in this case Mary. Observe the following appears as illustrated in Figure 18.

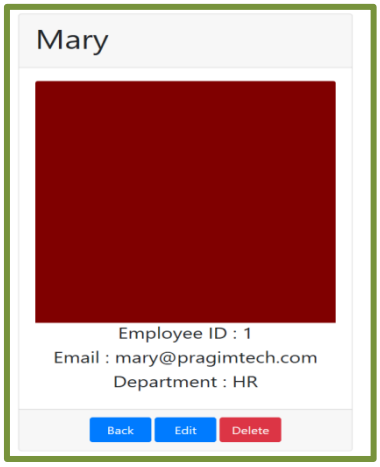


Figure 18

PARTIAL COMMIT Merge Github Repo Details Implementation

Update the Github repository for the **Details View**, **View Models**, and **Home Controller** methods associated with the **Details** Feature of the project. Commit Message = “Implementation of Details Page” As shown in Figure 19.

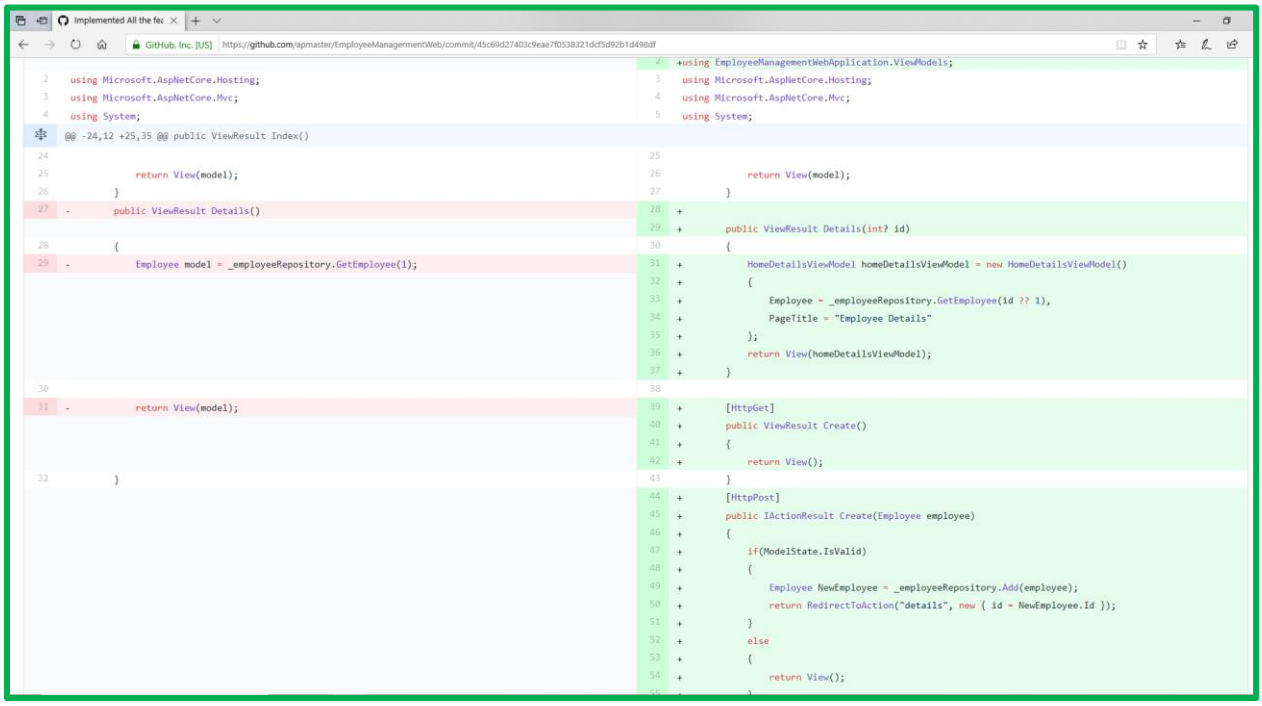


Figure 19

November 2, 2019
Prepared by Andre Masters
Revision 1.0

Create Page Implementation

This section focuses on the creation and construction of the **Create View Page**. In order to make the page function it needs to have a **View**, **Model**, and **Controller**. The **View** in this case uses strongly typed **View Models** and a **View-bag** object for the title page. The page styling uses **Bootstrap** and **Razor Helper** tags.

6.0 Create the following code implementation for the **Create View** feature page located in the **Views Home** folder.

```
@model Employee
@{
    ViewBag.Title = "Create Employee";
}
<form asp-controller="home" asp-action="create" method="post" class="mt-3">
    <div class="form-group row">
        <label asp-for="Name" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Name" class="form-control" placeholder="Name">
            <span asp-validation-for="Name" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="Email" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Email" class="form-control" placeholder="Email">
            <span asp-validation-for="Email" class="text-danger"></span>
        </div>
    </div>

    <div class="form-group row">
        <label asp-for="Department" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <select asp-for="Department" class="custom-select mr-sm-2"
                    asp-items="Html.GetEnumSelectList<Department.Dept>()">
                <option value="">Please Select</option>
            </select>
            <span asp-validation-for="Department" class="text-danger"></span>
        </div>
    </div>

    <div class="form-group row">
        <div class="col-sm-10">
            <button type="submit" class="btn btn-primary">Create</button>
        </div>
    </div>
</form>
```

6.1 Verify the **Employee Model** appears as shown in the following code construction

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;
using static EmployeeManagementWebApplication.Models.Department;

namespace EmployeeManagementWebApplication.Models
{
    public class Employee
    {
        public int Id { get; set; }
        [Required]
        [MaxLength(50, ErrorMessage ="Name Cannot Exceed 50 characters")]
        public string Name { get; set; }
        [Required]
        [RegularExpression(@"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$",
                           ErrorMessage = "Invalid email format")]
        public string Email { get; set; }
        [Required]
        public Department.Dept? Department { get; set; }
    }
}
```

6.2 Implemented the **Create Controller** which contains two methods one for **Post** and one for **Get.**

```
[HttpGet]
public ViewResult Create()
{
    return View();
}
[HttpPost]
public IActionResult Create(Employee employee)
{
    if(ModelState.IsValid)
    {
        Employee NewEmployee = _employeeRepository.Add(employee);
        return RedirectToAction("details", new { id = NewEmployee.Id });
    }
    else
    {
        return View();
    }
}
```

6.3 Compile and run the application and observe the following appears as shown below. Observe the following appears as shown in Figure 20.

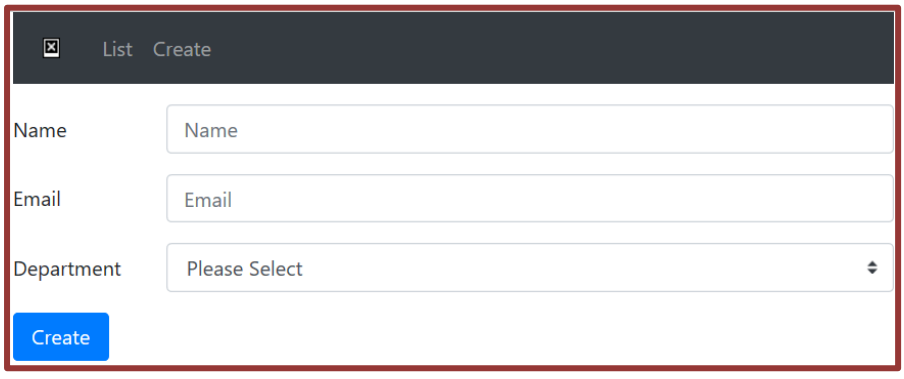


Figure 20

Menu Bar Layout Page Implementation

This section covers the steps required for creating a menu bar which supports page navigation. The user will select an item on the menu bar which in turn will lead to another page. The page will be a **Razor View** page with **Bootstrap** styling and a custom **JavaScript** routine.

7.0 In the **_Layout.cshtl** file enter the following code construction displayed below.

```
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <environment include="Development">
        <link href="~/lib/bootstrap/css/bootstrap.css" rel="stylesheet" />
        <script src="~/lib/jquery/jquery.js"></script>
        <script src="~/lib/bootstrap/js/bootstrap.js"></script>
    </environment>
    <environment exclude="Development">
              <link rel="stylesheet"
                    href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
                    integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
                    crossorigin="anonymous">
    </environment>

    <link href="~/css/site.css" rel="stylesheet"/>
    <title>@ViewBag.Title</title>
</head>
<body>
    <div class="container">
        <nav class="navbar navbar-expand-sm bg-dark navbar-dark">
            <a class="navbar-brand" asp-action="index" asp-controller="home">
                <img src="~/images/people.png" height="30" width="30"/>
            </a>
              <button type="button" class="navbar-toggler" data-toggle="collapse" data- target="#collapibleNavbar">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="collapibleNavbar">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <a asp-action="index" asp-controller="home" class="nav-link">List</a>
                    </li>
                    <li class="nav-item">
                        <a asp-action="create" asp-controller="home" class="nav-link">Create</a>
                    </li>
                </ul>
            </div>
        </nav>
        <div >
            @RenderBody()
        </div>

        @if (IsSectionDefined("Scripts"))
        {
            @RenderSection("Scripts", required: false)
        }
    </div>
 </body>
</html>
```

NOTE:

The CDN should be used for a production instance of project deployment. It may be advisable to have this section referenced as a separate collection of artifacts. Verify Bootstrap and JavaScript libraries are correct, otherwise the menu function will fail to operate as a responsive menu feature.

7.1 Run the application and observe the following appears as shown in Figure 21.

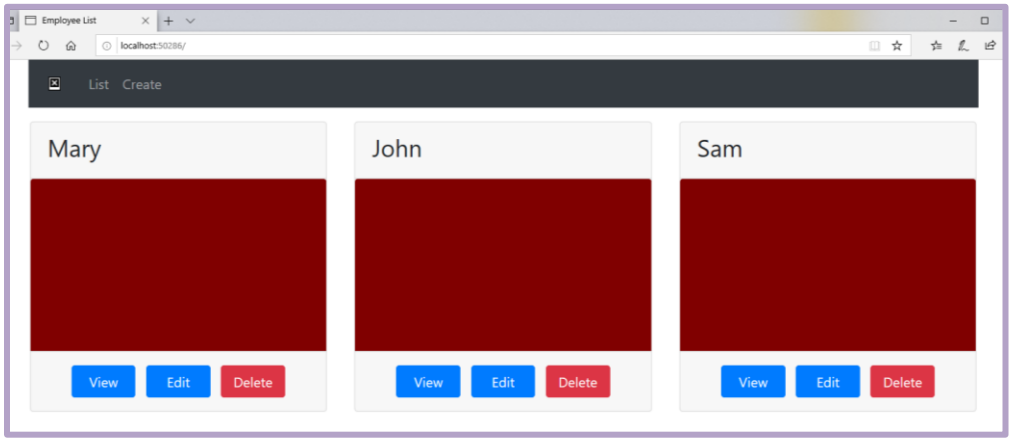


Figure 21

7.2 Collapse the application to verify the menu responds as a responsive object as shown in Figure 22.

Figure 22

Entity Framework Database Implementation

This section involves the creation of a database and table structure using **Entity Framework Core Migrations**.

8.0 Verify the **Configure Services Dependency** injection is pointing to the **Correct Database Repository**.

8.1 Verify **Configure Services** is set to **AddScoped** as shown in the code statement below.

```
services.AddScoped<IEmployeeRepository, SQLEmployeeRepository>();
```

8.2 Verify the Database server matches the connection string details in the **AppSettings.json** file. Use **SQL Object** explorer to verify this as shown in **Figure 23**

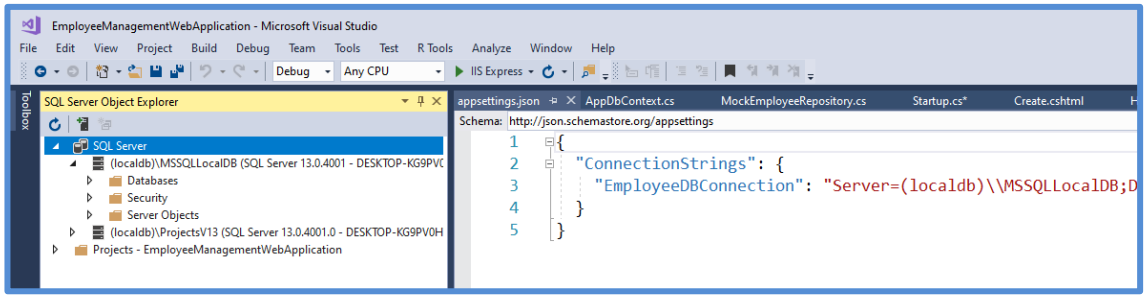


Figure 23

8.3 Open the **Package Manager Console** by selecting **View** followed by **Other Windows**.

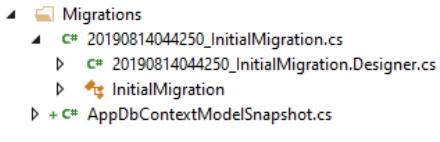
8.4 Enter the following command in the console to add an initial migration.

```
Add-migration
```

8.5 Name the migration **InitialMigration** and observe the following appears as shown in Figure 24.

```
PM> Add-Migration
cmdlet Add-Migration at command pipeline position 1
Supply values for the following parameters:
Name: InitialMigration
Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.2.0-rtm-35687 initialized 'AppDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options:
MaxPoolSize=128
To undo this action, use Remove-Migration.
PM>
```

Figure 24



NOTE:

New Migrations Folder with database snapshot instructions gets added to the project.

8.6 Enter the following command in the **Package Manager Console** to create a new database with a table structure that matches the data models in the project.

Update-Database

8.7 Observe the following appears as shown in Figure 25.

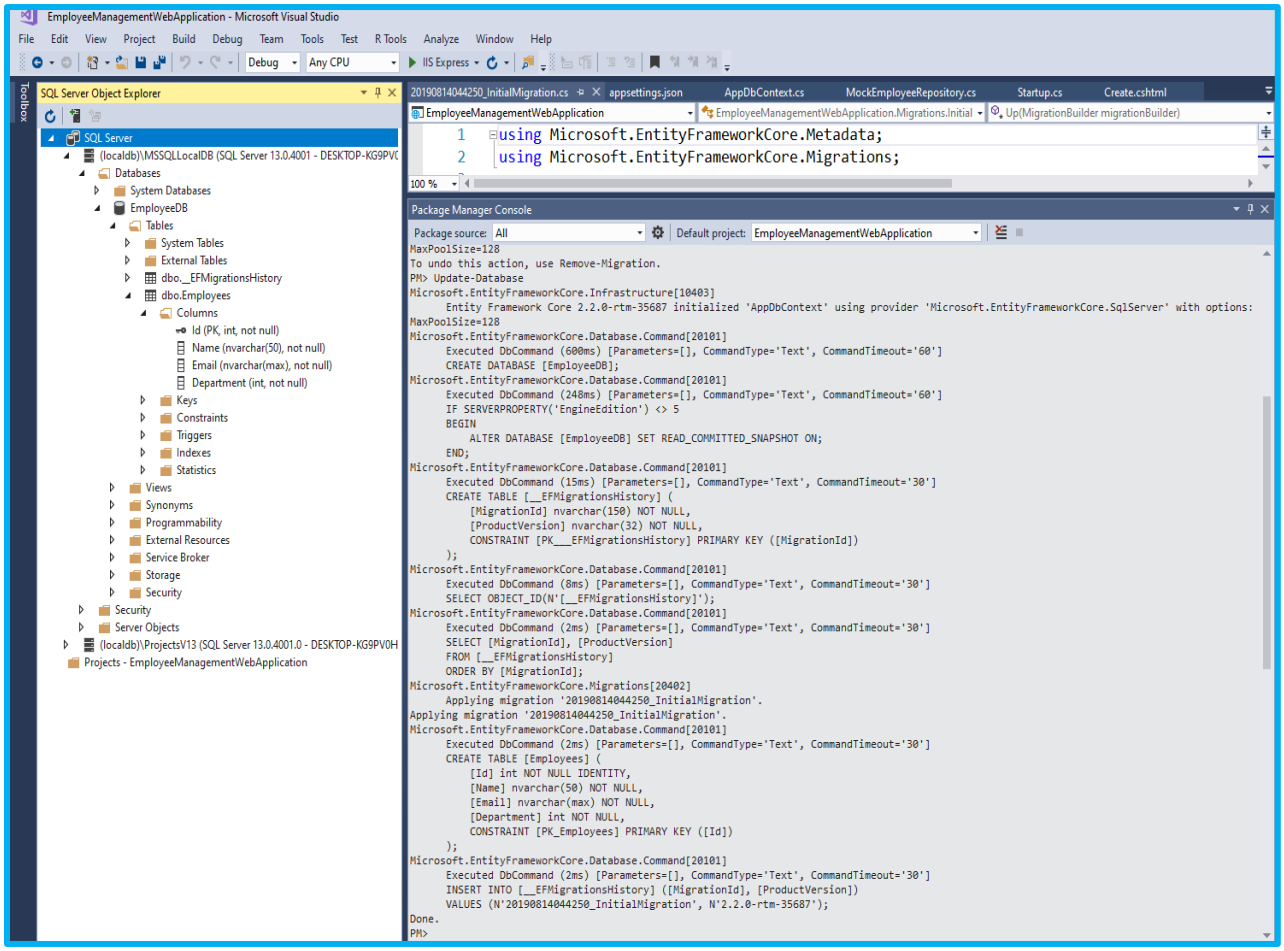


Figure 25

NOTE:

In an enterprise environment you will not have access to production databases or write access to QA databases. In those instances a request ticket will be required using the SQL instructions generated by Entity Framework Core in the package window.
When a migration is not specified the last migration is used in this case the Initial Migration since it's the only one we have.
The newly created database is empty so its expected when the project is run only the Menu will display. If data appears or there is an error check the Configure Services is pointing to the SQL Database. If it is check the Appsettings.json connection string definition is correct and your workstation allows for elevated permissions and single sign on.

November 2, 2019
Prepared by Andre Masters
Revision 1.0

8.8 Press F5 to compile and run the project observe the following appears as shown in Figure 26.

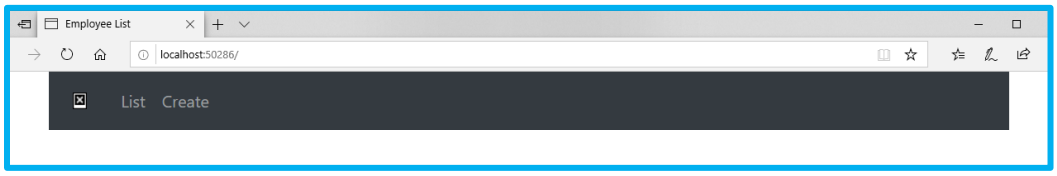


Figure 26

8.9 Select **Create** in the main toolbar in Figure 26 to add a new Employee. Complete the Create Employee Form see step 6.3 as a reference.

8.10 Return to the List option to see the newly created employee as shown in Figure 27.

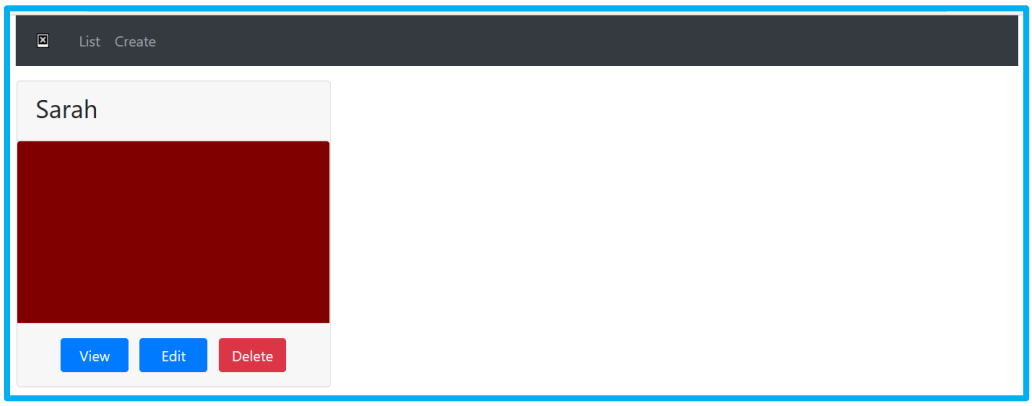


Figure 27

8.11 Verify Sarah's existence in the **Employee Table** of **EmployeeDB** database as shown in Figure 28.

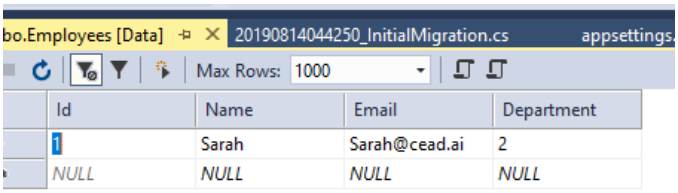


Figure 28

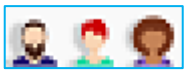
8.12 Terminate application execution and re run it again. Verify the result is the same as displayed in step 8.10. Add another employee called John and repeat step 8.11 and 8.12.

8.13 Commit the project to the **Github Repository**. Add comments to reflect the creation of a database and the SQL Repository Connection setting.

November 2, 2019
Prepared by Andre Masters
Revision 1.0

NOTE:

Update the project photo using the following Media for Group and No-Picture employees. As an exercise try to figure out where the images should be placed.
A: To remove an existing ***Migration*** *enter* ***Remove-Migration*** *if the changes have not been applied to the database.*



B: To remove an existing Migration that involves changes to the database, select the Migration Title from the ***dBo_EFMigration*** *table and enter the following command* ***Update Database*** *Migration Title Name.*

November 2, 2019
Prepared by Andre Masters
Revision 1.0

File Handing

This section covers file handling process of creating and updating a user account. The User should be able to upload a file to the server and remove it as desired. This process will cover the creation of an **Edit** feature and a modification of the **Create** feature.

9.0 Open the Employee **Model Class File** add a new property called **PhotoPath** using the code statement below.

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

9.1 Open the package manager console and enter the following statement to create a new migration

```
Add-migration PhotopathColumn
```

9.2 In the package manager console enter the following command to commit to the database

```
Update-Database
```

9.3 In **Server Explorer** verify the new column was added without removing the existing data in the database.

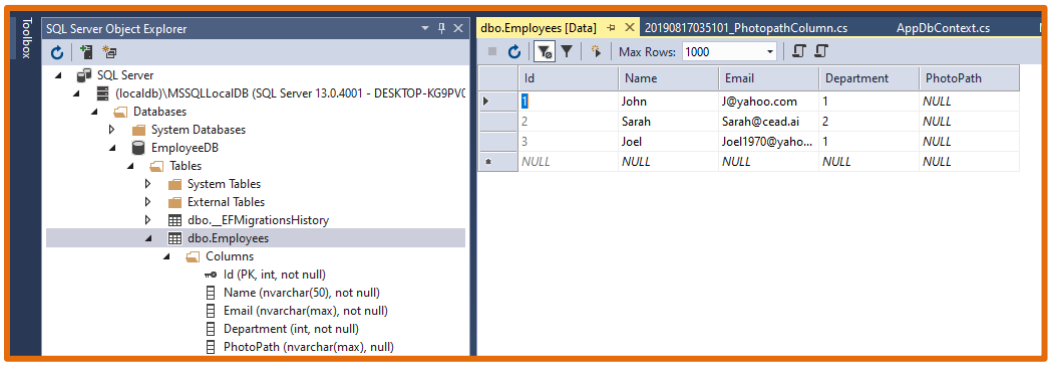


Figure 29

9.4 In the **ViewModels** folder open the file called **EmployeeCreateViewModel**. Enter the following code construction as shown below.

```
public int Id { get; set; }
[Required]
[MaxLength(50, ErrorMessage = "Name Cannot Exceed 50 characters")]
public string Name { get; set; }
[Required]
[RegularExpression(@"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$",
                   ErrorMessage = "Invalid email format")]
public string Email { get; set; }
[Required]
public Department.Dept? Department { get; set; }
public IFormFile Photo { get; set; }
```

9.5 In the **Views**->**Home** folder open the **Create.html** file. Change the **Model** from **Employee** to **EmployeeView Model**.

9.6 Make the following changes to the **Home Controller** file as shown in green.

```
private IEmployeeRepository _employeeRepository;
private readonly IHostingEnvironment hostingEnvironment;

public HomeController(IEmployeeRepository employeeRepository, IHostingEnvironment
                                                              hostingEnvironment)
{
    _employeeRepository = employeeRepository;
    this.hostingEnvironment = hostingEnvironment;
}
```

9.7 Replace the **Create Action Result** method with the following code construction

```
[HttpGet]
public ViewResult Create()
{
    return View();
}
[HttpPost]
public IActionResult Create(EmployeeCreateViewModel model)
{
    if (ModelState.IsValid)
    {
        string sourcePath = null;
        string PhotoName = null;
        string[] DirectoryPath = null;
        string SourceFilePath = null;
        char[] charsToTrim = { '\\'};
        string targetPath = Path.Combine(hostingEnvironment.WebRootPath, "images");

        if (model.Photo != null)
        {
            sourcePath = model.Photo.FileName;
            DirectoryPath = sourcePath.Split("\\");
            PhotoName = DirectoryPath[DirectoryPath.Length - 1];
            DirectoryPath = DirectoryPath.Where(w => w != DirectoryPath.Last()).ToArray();

            foreach(string word in DirectoryPath)
            {
                SourceFilePath = SourceFilePath + word + "\\";
            }

            string sourceFile = System.IO.Path.Combine(SourceFilePath, PhotoName);
            string destFile = System.IO.Path.Combine(targetPath, PhotoName);
            System.IO.File.Copy(sourceFile, destFile, true);
        }
        Employee newEmployee = new Employee
        {
            Name = model.Name,
            Email = model.Email,
            Department = model.Department,
            PhotoPath = "\\images\\" + PhotoName
        };

        _employeeRepository.Add(newEmployee);
        return RedirectToAction("details", new { id = newEmployee.Id });
    }
    return View();
}
```

9.8 Run the application by pressing F5 and creating a new employee and observe the following appears in Figure 30. Select a new image of the employee.

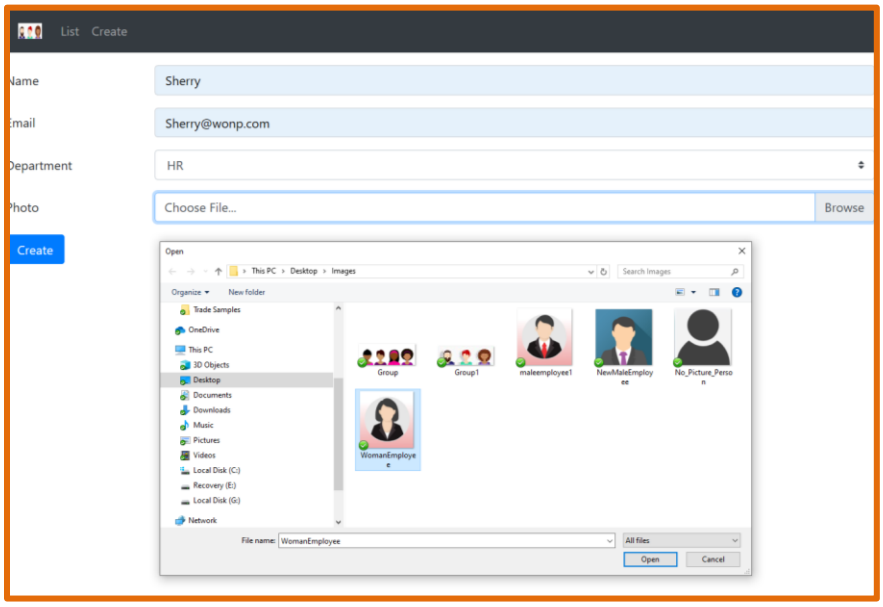


Figure 30

9.9 Once the **Open** button is pressed observe the file name displays on the form as shown in Figure 31.

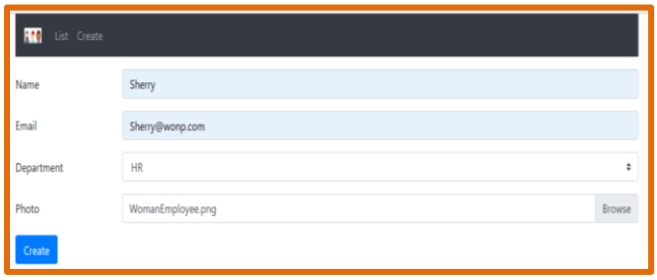


Figure 31

9.10 Press the **Create** button, observe the image appears in Figure 32

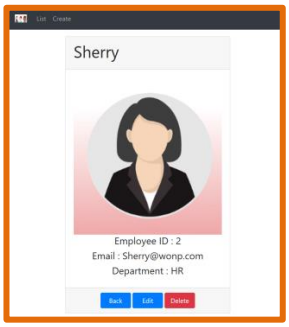


Figure 32

9.11 Select the **List Category** in the main menu bar and observe the following images appear side by side as displayed in Figure 33

Figure 33

Edit Feature

The Edit feature implementation will allow the user to make changes to the Detail record in Employees Table of the EmployeeDB database..

10.0 Open the **Details View** in the **Bootstrap Card** and make the following changes in the red box.

```
@model EmployeeDetailsViewModel
@{
    ViewBag.Title = "Employee Details";
    var photopath = Model.Employee.PhotoPath ?? "images\\No_Picture_Person.png";
}
<div class="row justify-content-center m-3">
    <div class="col-sm-8">
        <div class="card">
            <div class="card-header">
                <h1>@Model.Employee.Name</h1>
            </div>
            <div class="card-body text-center">
                <img class="card-img-top" src="@photopath"/>
                <h4>Employee ID : @Model.Employee.Id</h4>
                <h4>Email : @Model.Employee.Email</h4>
                <h4>Department : @Model.Employee.Department</h4>
            </div>
            <div class="card-footer text-center">
                <a asp-controller="home" asp-action="index" class="btn btn-primary">Back</a>
                <a asp-action="edit"  asp-controller="home" asp-route-id="@Model.Employee.Id"
                   class = "btn btn-primary">Edit</a>
                <a href="#" class="btn btn-danger">Delete</a>
            </div>
        </div>
    </div>
</div>

@section Scripts {
    <script src="~/js/CustomScript.js"></script>
```

10.1 In the **Home Controller** add the following implementation code construction for the **Get Operations** of the current employee as shown below.

```
[HttpGet]
public ViewResult Edit(int id)
{
    Employee employee = _employeeRepository.GetEmployee(id);

    EmployeeEditViewModel employeeEditViewModel = new EmployeeEditViewModel
    {
        Id = employee.Id,
        Name = employee.Name,
        Email = employee.Email,
        Department = employee.Department,
        ExistingPhotoPath = employee.PhotoPath
    };
    return View();
}
```

10.2 In the **Home Controller** add the following implementation details to handle **Post Operations** for an updated employee as shown below.

```
[HttpPost]
public IActionResult Edit(EmployeeEditViewModel model)
{
    if (ModelState.IsValid)
    {
        string sourcePath = null;
        string PhotoName = null;
        string[] DirectoryPath = null;
        string SourceFilePath = null;
        char[] charsToTrim = { '\\' };
        string targetPath = Path.Combine(hostingEnvironment.WebRootPath, "images");

        if (model.Photo != null)
        {
            sourcePath = model.Photo.FileName;
            DirectoryPath = sourcePath.Split("\\");
            PhotoName = DirectoryPath[DirectoryPath.Length - 1];
            DirectoryPath = DirectoryPath.Where(w => w != DirectoryPath.Last()).ToArray();

            foreach (string word in DirectoryPath)
            {
                SourceFilePath = SourceFilePath + word + "\\";
            }

            string sourceFile = System.IO.Path.Combine(SourceFilePath, PhotoName);
            string destFile = System.IO.Path.Combine(targetPath, PhotoName);
            System.IO.File.Copy(sourceFile, destFile, true);
        }
        else
        {
            string[] OldPhoto = null;
            string CandidatePhotoName = "";
            CandidatePhotoName = model.ExistingPhotoPath;
            OldPhoto = CandidatePhotoName.Split("\\");
            PhotoName = OldPhoto[OldPhoto.Length - 1];
        }
        Employee UpdateEmployee = new Employee
        {
            Id = model.Id,
            Name = model.Name,
            Email = model.Email,
            Department = model.Department,
            PhotoPath = "\\images\\" + PhotoName
        };
        _employeeRepository.Update(UpdateEmployee);
        return RedirectToAction("index");
    }
    return View();
}
```

10.3 Open the **Edit View** file add the following code implementation.

```
@model EmployeeEditViewModel
@{
    ViewBag.Title = "Edit Employee";
}
<form enctype="multipart/form-data" asp-controller="home" asp-action="edit"
      method="post" class="mt-3">
    <input hidden asp-for="Id" />
    <input hidden asp-for="ExistingPhotoPath" />
    <div class="form-group row">
        <label asp-for="Name" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Name" class="form-control" placeholder="Name">
            <span asp-validation-for="Name" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="Email" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Email" class="form-control" placeholder="Email">
            <span asp-validation-for="Email" class="text-danger"></span>
        </div>
    </div>

    <div class="form-group row">
        <label asp-for="Department" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <select asp-for="Department" class="custom-select mr-sm-2"
                    asp-items="Html.GetEnumSelectList<Department.Dept>()">
                <option value="">Please Select</option>
            </select>
            <span asp-validation-for="Department" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="Photo" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <div class="custom-file">
                <input asp-for="Photo" class="form-control custom-file-input">
                <label class="custom-file-label">Click here to change photo...</label>
            </div>
        </div>
    </div>

    <div asp-validation-summary="All" class="text-danger"></div>

    <div class="form-group row">
        <div class="col-sm-10">
            <button type="submit" class="btn btn-primary">Update</button>
            <a asp-controller="home" asp-action="index"
            class="btn btn-primary">Cancel</a>
        </div>
    </div>
    @section Scripts {
        <script>
            $(document).ready(function () {
                $('.custom-file-input').on("change", function () {
                    var fileName = $(this).val().split("\\").pop();
                    $(this).next('.custom-file-label').html(fileName);
                });
            });
        </script>
    }
</form>
```

10.4 Test the application be pressing F5 and observe the following appears in Figure 34.

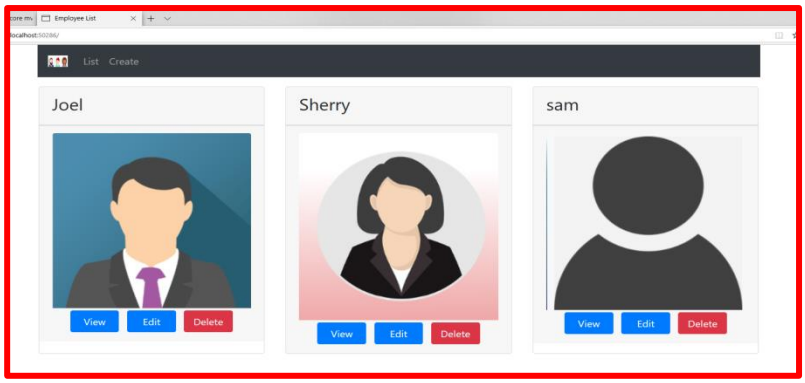


Figure 34

10.5 Press the **Edit** button for any employee in this case Sherry, observe the following appears as shown in Figure 35.

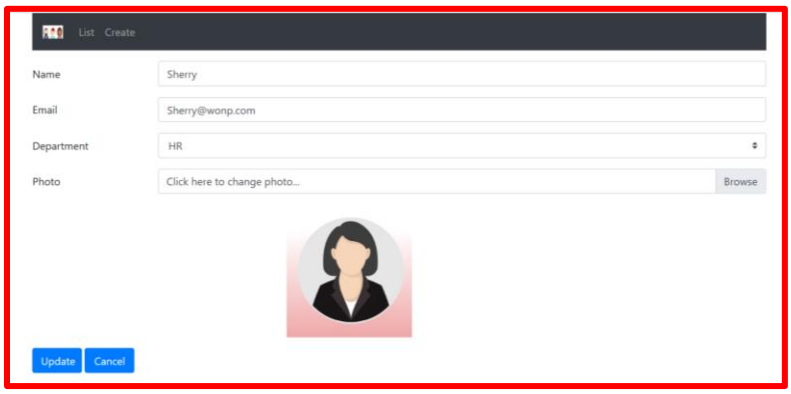


Figure 35

10.6 Press the **Cancel** button, observe the screen appears as shown in step 10.2

10.7 Upload a replacement image using in this case the **No_Person_Image**. Observe the following appears as seen in the Figure 36

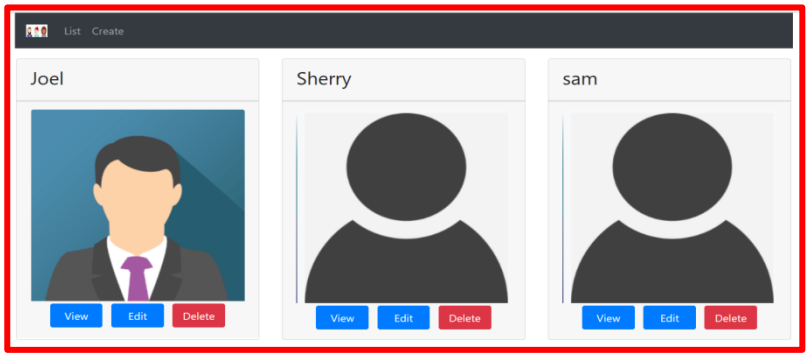


Figure 36

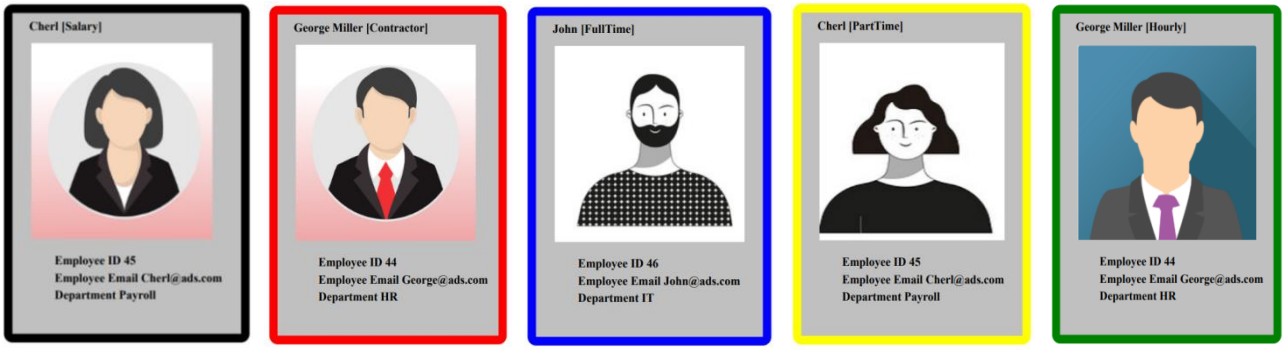
10.8 Repeat the same steps in 10.3 – 10.6 observe Sherry has a women image as seen in step 10.3

November 2, 2019
Prepared by Andre Masters
Revision 1.0

Employee ID Card

The goal for this feature is to allow printing of an ID Card that is based on the content from the **Employee Details Page**. The process will be accomplished using a library called **ItextSharp**. Implementation will consist of a separate **Razor View Page** containing the employee image, ID Number, Name and Department. Once the user clicks the print button, an image file will be generated as a downloadable file.

The ID-Card will be a pdf file containing the following information based on employee classification as shown below: Notice the boundary color representing classification type for each employee.



The user should be able to download the image from the **Employee Details** page. In order to complete the download operation, the user will need to be logged in the system. The logged in user should be the only one who can get their **ID Badge**. If the user is not logged into the system, the user should be redirected to the login screen. If the user is logged in but attempts to get access to download another user's ID card, the user in this case should be redirected to the **Not Authorized** page.

Classification is defined as an employment status such as full time, part time, and or contractor. The following list contains all of the classifications that are applied to the Employee of this site.

- Black Outline = Salary Employee
- Red Outline = Contractor
- Blue Outine = Full Time
- Yellow Outline = Part Time
- Green Ouline = Hourly

November 2, 2019
Prepared by Andre Masters
Revision 1.0

11.0 Modify the Details View page by adding a download button below the Delete button using the code construction below.

Details.cshtml | AccountController.cs | _Layout.cshtml | HomeController.cs | Create.cshtml

```
@model Employee
@{
    ViewBag.Title = "Employee Details";
    var photopath = Model.PhotoPath ?? "images\\No_Picture_Person.png";
}
<div class="row justify-content-center m-3">
    <div class="col-sm-8">
        <div class="card">
                <div class="card-header">
                    <h1>@Model.Name</h1>
                </div>
                <div class="card-body text-center">
                    <img class="card-img-top" src="@photopath" />
                    <h4>Employee ID : @Model.Id</h4>
                    <h4>Email : @Model.Email</h4>
                    <h4>Department : @Model.Department</h4>
                </div>
                <div class="card-footer text-center">
                    <a asp-controller="home" asp-action="index" class="btn btn-primary">Back</a>
                    <a asp-controller="home" asp-action="edit"  asp-route-id="@Model.Id"
                       class="btn btn-primary">Edit</a>
                    <a href="#" class="btn btn-danger">Delete</a>
                    <a asp-controller="home" asp-action="GetEmployeeFile" asp-route-id="@Model.Id"
                       class="btn btn-primary m-1" style="width:auto">Download</a>
                </div>
            </div>
    </div>
</div>

@section Scripts {
    <script src="~/js/CustomScript.js"></script>
```

11.1 Add the iTextSharp nugget package select the version that is suitable for ASP.Net Core as shown In Figure 37. To get access to this screen select Tools from the main menu and then select Manage Nuget packages sub menu.

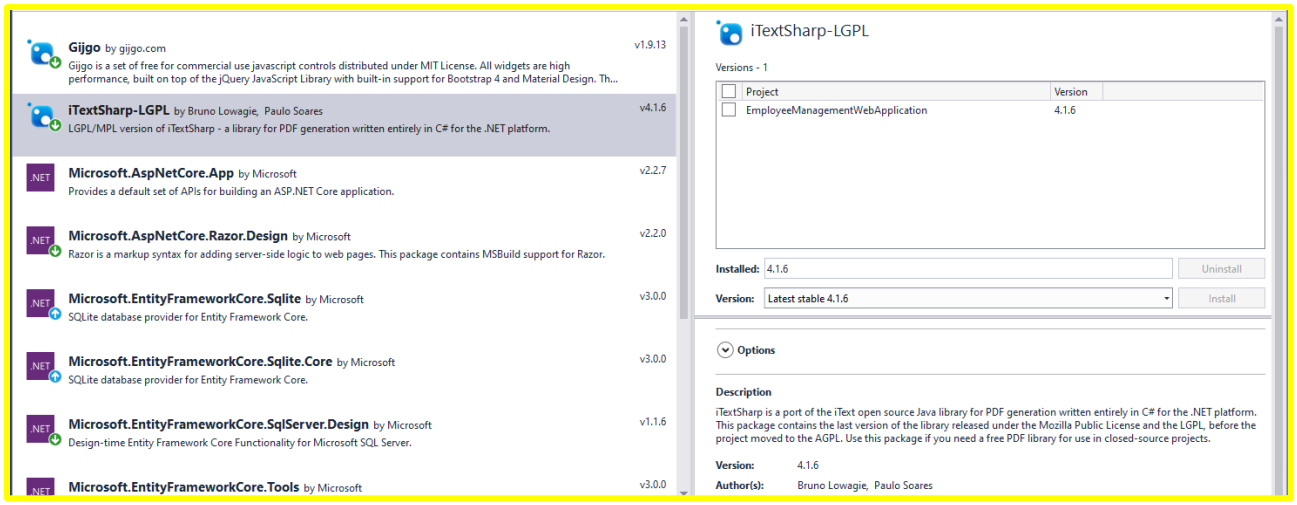


Figure 37

11.2 In the **HomeController** class add the following action method whose aim is to create an employee ID Card. Call this method **GetEmployeeFile** pass in the **Employee Id** as a parameter.

```
[HttpGet("download")]
 public IActionResult GetEmployeeFile(int id)
 {
        return view();
 }
```

11.3 Place a breakpoint at the opening of the **GetEmployeeFile** method and run the application to verify the breakpoint hits.

11.4 The first step in the code implementation, is to get the current **Employee Data** View Model and assign it to the **Employee Object** as shown in the code construction below.

```
Employee employee = _employeeRepository.GetEmployee(id);

EmployeeEditViewModel employeeEditViewModel = new EmployeeEditViewModel
{
    Id = employee.Id,
    Name = employee.Name,
    Email = employee.Email,
    Department = employee.Department,
    ExistingPhoto = employee.PhotoPath,
    Classification = employee.Classification
};
```

11.5 Now, that we have the Employee data, we need to check to see if this data matches the current user data. If the current user is not logged into the system a redirection to the login view is required, otherwise if the user is logged in but their email user name does not match the email which us the user name, that user is redirected to the not authorized view. The following code construction accomplishes this.

```
if(User.Identity.IsAuthenticated)
{
    if(User.Identity.Name == employeeEditViewModel.Email)
       {
            //Main Application Code
       }
    else
       {
            return RedirectToAction("AccessDenied","Account");

       }
}
else
{
    return RedirectToAction("Login","Account");

}
```

11.6 Test this code by trying to be a true authentic user. Observe nothing happens. Try the same process as a fake user.

11.7 Now that we have established access to the account, create an **iTex** document by inserting the following code in the **GetEmployeeFile method** inside the **User.Identity.Name** condition of the **Try Catch Block** under the comment called **Main Application Code**.
Bring in the **iTex** library.

```
Document doc = new Document();
doc.AddTitle("IDcard_" + employeeEditViewModel.Name);
doc.AddCreationDate();
string physicalfilepath = "C:\\Users\\Admin-PC\\
 Source\\Repos\\EmployeeManagementWebApplication\\
 EmployeeManagementWebApplication\\wwwroot\\";
```

11.8 Add a **Try-Catch – Finally** block to process the **iTex** document which will form an **ID Card** for the given employee. Place this code construction under the code construction for document creation in the previous step.

```
try
  {
  }
Catch (Exception ex)
  {
  }
Finally
 {
  }
```

11.9 Since the **ID Card** once created is stored on the server, we need to check if the card exists, if it does delete so a new one can be created. To accomplish this task enter the following code construction inside the try block from step 11.8.

```
string ID_Card_File_Exists = physicalfilepath + FileUploaderPath + "//" +
                             employeeEditViewModel.Name + "_EmployeeIDCard.pdf";

//Step 1 Reconcile Existing ID Cards
if (System.IO.File.Exists(ID_Card_File_Exists))
{
    System.IO.File.Delete(ID_Card_File_Exists);
}
```

11.10 Initialize the **PDF Writer** by entering the following code statements located on the next line after the code construction in step 11.9

```
PdfWriter writer = PdfWriter.GetInstance(doc, new FileStream(physicalfilepath + FileUploaderPath
                                         + "//" +
        employeeEditViewModel.Name + "_EmployeeIDCard.pdf", FileMode.Create));
doc.Open();
```

11.11 Create the **ID Card** outline based on employee classification using the following color scheme.

- FullTime = Blue
- PartTime=Yellow
- Contractor=Red
- Hourly =Green
- Salary =Black

11.12 To Accomplish this task use the following code construction as shown below. Placed as the next the code statement in the **doc.Open** as defined from step 11.10.

```
PdfContentByte cb = writer.DirectContent;
cb.RoundRectangle(90f, 90f, 290f, 390f, 10f);
cb.SaveState();
if (employeeEditViewModel.Classification.ToString() == "FullTime")
{
    cb.SetRGBColorFill(0x00, 0x00, 0xff);
}
if (employeeEditViewModel.Classification.ToString() == "Contractor")
{
    cb.SetRGBColorFill(0xff, 0x00, 0x00);
}
if (employeeEditViewModel.Classification.ToString() == "PartTime")
{
    cb.SetRGBColorFill(0xff, 0xff, 0x00);
}
if (employeeEditViewModel.Classification.ToString() == "Hourly")
{
    cb.SetRGBColorFill(0x00, 0x80, 0x00);
}
if (employeeEditViewModel.Classification.ToString() == "Salary")
{
    cb.SetRGBColorFill(0x00, 0x00, 0x00);
}
cb.Fill();
cb.RestoreState();
```

NOTE:

*The base fill color is silver the **SaveState** and **RestoreState** pair are required for each **Itex** operation. The fill color is silver for any ID Card no matter the employment classification.*

11.13 Create the inner body **Fill Color** using silver as the color scheme by entering the following code construction located after the code in the previous step 11.12.

```
//Inner Border Color Silver
cb.RoundRectangle(100f, 100f, 270f, 370f, 0);
cb.SaveState();
cb.SetRGBColorFill(0xc0, 0xc0, 0xc0);
cb.Fill();
cb.RestoreState();
```

11.14 In this step the aim is to add the **Photo** of the **Current Employee User** to an **iTextSharp** object. To accomplish this task enter the following code construction located in sequence after the code cpnstruction from step 11.13.

```
iTextSharp.text.Image jpg = Image.GetInstance(physicalfilepath +
employeeEditViewModel.ExistingPhoto);
jpg.SetAbsolutePosition(123, 210);
jpg.ScaleToFit(225, 225);
cb.AddImage(jpg);
cb.SaveState();
cb.RestoreState();
```

11.15 In this step the aim is to add the **Employee Name** and **Classification Status** of the **Current Employee User** to the current **iTextSharp object**. Place the following code construction in sequence after the code construction from step 11.14.

```
cb.BeginText();
cb.SetTextMatrix(120, 450);
    BaseFont font = BaseFont.CreateFont(BaseFont.TIMES_BOLD,
    BaseFont.WINANSI, BaseFont.NOT_EMBEDDED);
cb.SetFontAndSize(font, 14);
    cb.ShowText(employeeEditViewModel.Name + "
    [" +  employeeEditViewModel.Classification + "]");
cb.EndText();
cb.SaveState();
cb.RestoreState();
```

NOTE:

*The **BeginText** and **EndText** pair are required to contain a SetMatrix, font definition, font size, and text content in order to remain in balance otherwise an error will result at run time.*

11.16 Enter the following code construction for the remainder of the **ID Card** components **Department**, **ID** number and **Email** in sequence after the code construction in step 11.15.

```
//Step 6 Add Employee ID Number
cb.BeginText();
cb.SetTextMatrix(150, 180);
cb.SetFontAndSize(font, 14);
cb.ShowText("Employee ID " + employeeEditViewModel.Id.ToString());
cb.EndText();
cb.SaveState();
cb.RestoreState();

//Step 7 Add Employee Email
cb.BeginText();
cb.SetTextMatrix(150, 160);
cb.SetFontAndSize(font, 14);
cb.ShowText("Employee Email " + employeeEditViewModel.Email);
cb.EndText();
cb.SaveState();
cb.RestoreState();

//Step 8 Add Department Name
cb.BeginText();
cb.SetTextMatrix(150, 140);
cb.SetFontAndSize(font, 14);
cb.ShowText("Department " + employeeEditViewModel.Department);
cb.EndText();
cb.SaveState();
cb.RestoreState();
```

11.17 In the Finally Block close the document using the code construction as shown below.

```
finally
 {
     doc.Close();
 }
```

11.18 Implement the file download functionality by entering the following code construction outside the finally block of the try catch statement.

```
string link = physicalfilepath + FileUploaderPath + "//" + employeeEditViewModel.Name
+ "_EmployeeIDCard.pdf";
string BadgeFile = employeeEditViewModel.Name + "_EmployeeIDCard.pdf";
var net = new System.Net.WebClient();
var data = net.DownloadData(link);
var content = new System.IO.MemoryStream(data);
var contentType = "APPLICATION/octet-stream";
var fileName = BadgeFile;
return File(content, contentType, fileName);
```

11.19 Run the application make sure testing matches behavior and functionality to the user story.

11.20 Commit the project to the repository using the steps outlined in earlier sections of this procedure.

Reports

This section examines the functionality of the datagrid object in Bootstartp and how it can be applied to the Employee Table. This section is another implementation of the Model, View and Controller design pattern.

12.0 Add the following link in the _Layout view file under create for the report view

```
<li class="nav-item">
      <a asp-action="reports" asp-controller="home" class="nav-link">Reports</a>
</li>
```

12.1 Create a **Reports Controller** class in the **Reports Controller** folder add the following code construction.

```
        private IEmployeeRepository _employeeRepository;

        public ReportsController(IEmployeeRepository employeeRepository)
        {
            _employeeRepository = employeeRepository;
        }
        public IActionResult LaborReport()
        {
            var model = _employeeRepository.GetAllEmployees();
            return View(model);
        }
```

11.2 In the **Labor Reports View** add the following code construction

```
@model IEnumerable<Employee>
@{
    ViewBag.Title = "Employee Reports";
}
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>Labor Report</title>

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css" integrity="sha384-
GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9DkKMp0DgiMDm4iYMj70gZWKYbI706tWS" crossorigin="anonymous">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.6.3/css/all.css" integrity="sha384-
UHRtZLI+pbxtHCWp1t77Bi1L4ZtiqrqD80Kn4Z8NTSRyMA2Fd33n5dQ8lWUE00s/" crossorigin="anonymous">
    <link rel="stylesheet" href="https://unpkg.com/bootstrap-table@1.15.4/dist/bootstrap-table.min.css">
</head>
<body>
    <table data-toggle="table">
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Department</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var employee in Model)
            {
                <tr>
                    <td>@employee.Id.</td>
                    <td>@employee.Name</td>
                    <td>@employee.Department</td>
                </tr>
            }
        </tbody>
    </table>

    <script src="https://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js" integrity="sha384-
wHAiFfRlMFy6i5SRaxvfOCifBUQy1xHdJ/yoi7FRNXMRBu5WHdZYu1hA6ZOblgut" crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js" integrity="sha384-
B0UglyR+jN6CkvvICOB2joaf5I4l3gm9GU6Hc1og6Ls7i6U/mkkaduKaBhlAXv9k" crossorigin="anonymous"></script>
    <script src="https://unpkg.com/bootstrap-table@1.15.4/dist/bootstrap-table.min.js"></script>
</body>
</html>
```

November 2, 2019
Prepared by Andre Masters
Revision 1.0

11.3 Execute the application and observe the following appears for the labor report.

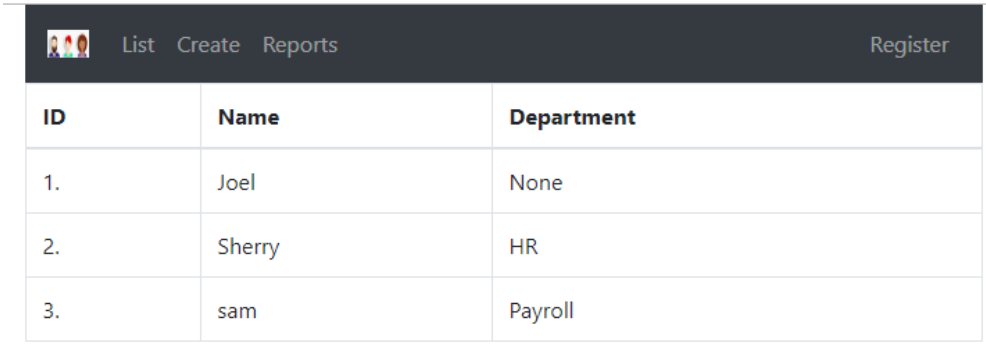


Figure 38

NOTE:

The report in its current form only provides a list of existing employees. The report can be extended to add CRUD operations for existing employee features for a given employee model. If there are additional features that are not currently available for current employees, a new employee object will need to be created or transformed from a concrete type to an abstract base type. As an abstract type, the employee object could be extended using the override key word on a virtual method or type.

Identity Framework

This section covers all the features of the Identity Framework. In this segment the **Login** and **Register** user types will be created using the **SignInManager** and **Usermanager** routines from the **Identity Framework**. All of the items in Figure 39.

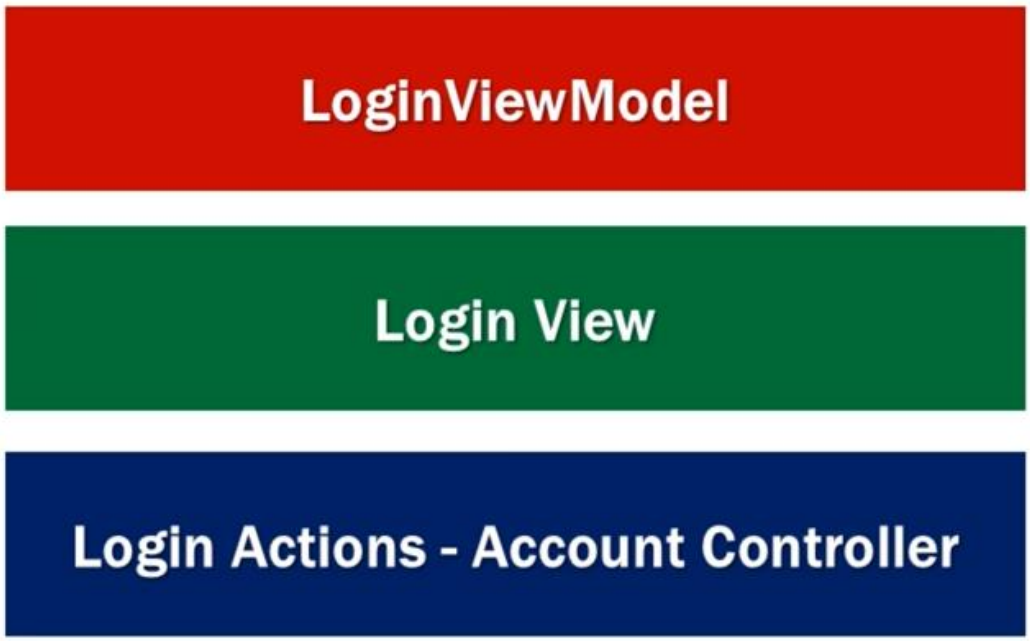


Figure 39

12.0 Change the DB Context to **IdentityDBContext**

```
namespace EmployeeManagementWebApplication.Models
{
    public class AppDbContext:IdentityDbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options): base(options)
        {

        }

        public DbSet<Employee> Employees { get; set; }
    }
}
```

12.1 In the **Startup** class add **Identity service** for roles and users as shown below after SQL server.

```
services.AddIdentity<IdentityUser, IdentityRole>()
        .AddEntityFrameworkStores<AppDbContext>();
```

12.2 In the **Startup** class modify the **configure** method by adding the following code statement for **UserAuthentication**.

```
app.UseAuthentication();
```

12.3 Rebuild and clean the solution.

12.4 Create a **Migration** for the **Identity Table**, this will create a new table in the Identity database in SQL server using the command in Figure 40.0

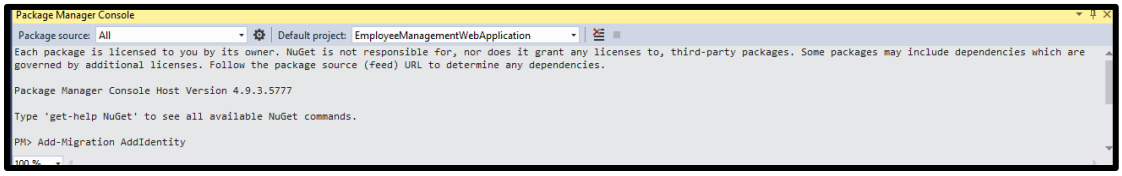


Figure 40

12.5 Observe the following appears as shown below.

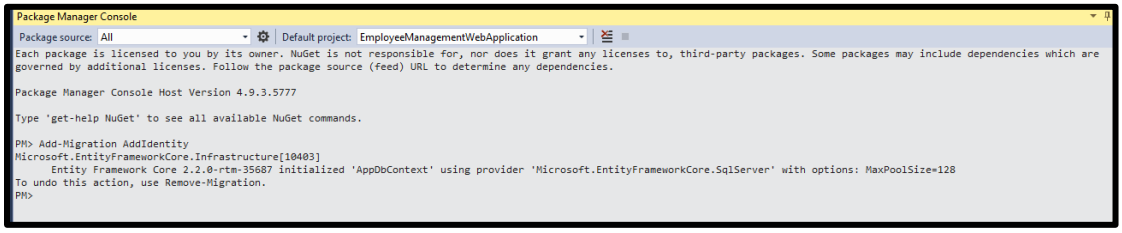


Figure 41

12.6 Observe the Migration Folder for Identity appears as shown in Figure 42.

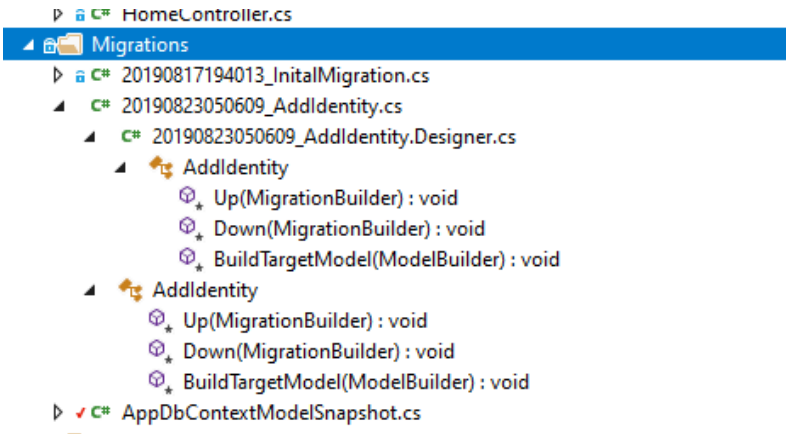


Figure 42

12.7 In the package manager console enter the command **Update-Database** and observe the database gets a table added to it with the Identity details from the Identity service. Examine the schema database design files, this will be used later to create a SQLite database for deployment in a Raspberry pi.

12.8 In the **View Models** folder add a new View Model called **RegisterViewModel** enter the following code statements as shown below.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace EmployeeManagementWebApplication.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm Password")]
        [Compare("Password", ErrorMessage = "Password and confirmation password do not match")]
        public string ConfimredPassword { get; set; }
    }
}
```

12.9 In the Views Accounts folder create a razor view file called Register, enter the following code statements as shown below.

```
@model RegisterViewModel
@{
    ViewBag.Title = "User Registration";
}
<h1>User Registration</h1>
<div class="row">
    <div class="col-md-12">
        <form method="post">
           <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Email"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="ConfimredPassword"></label>
                <input asp-for="ConfimredPassword" class="form-control" />
                <span asp-validation-for="ConfimredPassword" class="text-danger"></span>
            </div>
            <button type="submit" class="btn btn-primary">Register</button>
        </form>
    </div>
</div>
```

12.10 Run the application and observe the following appears as shown in Figure 43

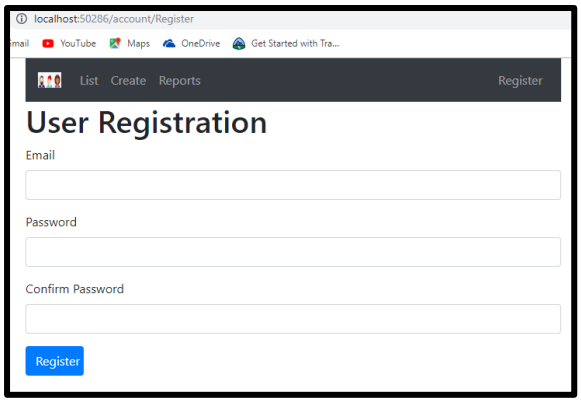


Figure 43

12.11 In the Account Controller modify the existing code to match the following statements which allow for both HTTP Post and HTTP Get operations.

```
using EmployeeManagementWebApplication.ViewModels;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;

using System.Linq;
using System.Threading.Tasks;

namespace EmployeeManagementWebApplication.Controllers
{
    public class AccountController:Controller
    {
        private readonly UserManager<IdentityUser> userManager;
        private readonly SignInManager<IdentityUser> signInManager;

        public AccountController(UserManager<IdentityUser> userManager, SignInManager<IdentityUser> signInManager)
        {
            this.userManager = userManager;
            this.signInManager = signInManager;
        }

        [HttpGet]
        [AllowAnonymous]
        public IActionResult Register()
        {
            return View();
        }

        [HttpPost]
        [AllowAnonymous]
        public async Task<IActionResult> Register(RegisterViewModel model)
        {
            if (ModelState.IsValid)
            {
                var user = new IdentityUser { UserName = model.Email, Email = model.Email };
                var result = await userManager.CreateAsync(user, model.Password);

                if (result.Succeeded)
                {
                    await signInManager.SignInAsync(user, isPersistent: false);
                    return RedirectToAction("Index", "home");
                }

                foreach (var error in result.Errors)
                {
                    ModelState.AddModelError("", error.Description);
                }
            }
            return View(model);
        }
    }
}
```

12.12 Run the Application and select **Register**. Enter an email address as a user name. Create an improper password. Bad password = 123, observe the following appears as in Figure 44

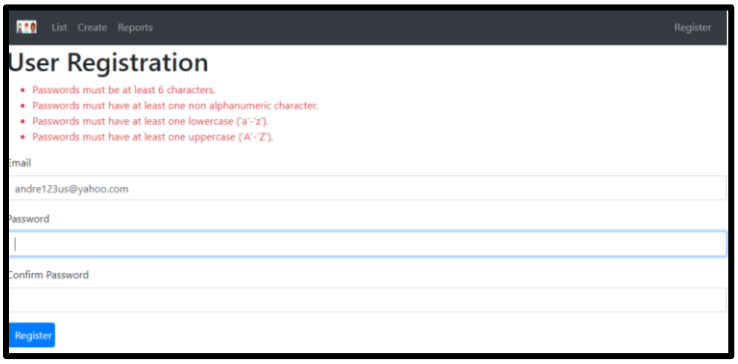


Figure 44

12.13 Change the **Email Address** to make it an improper Email format and observe the following appears as shown in Figure 45.

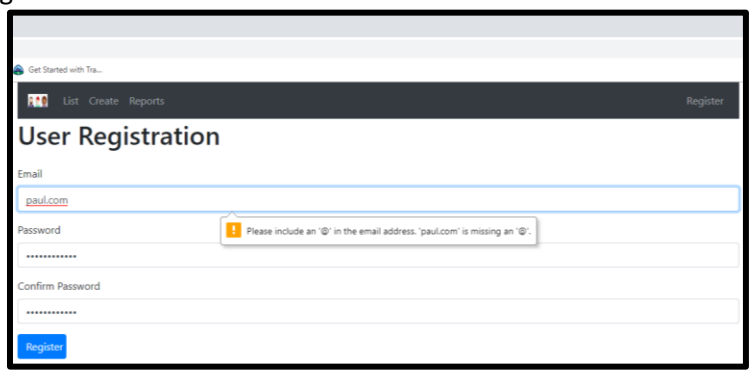


Figure 45

12.14 Correct the Email address and enter two different passwords make sure they do not match. Observe the following appears in Figure 46.

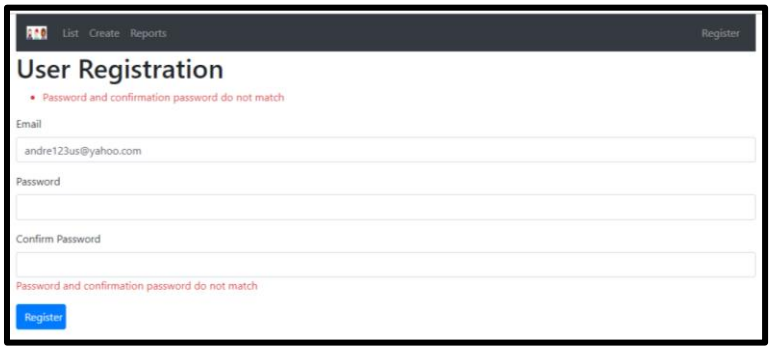


Figure 46

12.15 Enter a proper password and username and observe the following appears in Figure 47.

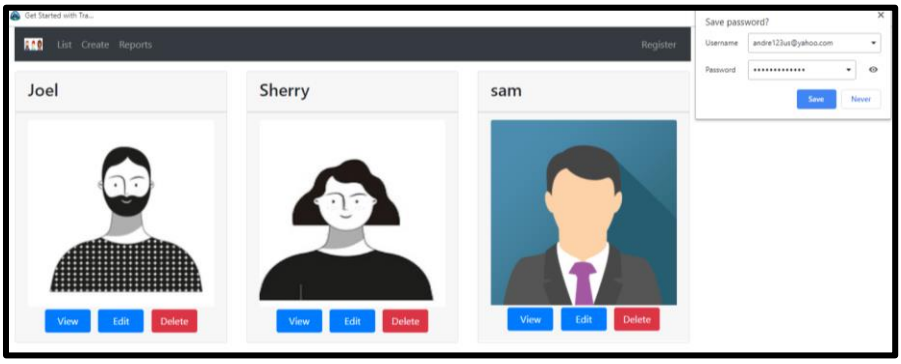


Figure 47

12.16 Update the **Menu Bar** to show a logged in **User**, by making the following changes to the layout view.

```
@inject SignInManager<IdentityUser> signInManager;
```

NOTE: Make sure the _ViewImports view contains the following

```
@using Microsoft.AspNetCore.Identity;
```

12.17 Modify the Register Model to match the following code statements.

```
<ul class="navbar-nav ml-auto">
    @if (signInManager.IsSignedIn(User))
    {
        <li class="nav-item">
            <form method="post" asp-action="logout" asp-controller="account">
                <button type="submit" class="nav-link btn btn-link py-0" style="width:auto">
                    Logout @User.Identity.Name
                </button>
            </form>
        </li>
    }
    else
    {
        <li class="nav-item">
            <a asp-action="Register" asp-controller="account" class="nav-link">Register</a>
        </li>
        <li class="nav-item">
            <a asp-action="login" asp-controller="account" class="nav-link">Login</a>
        </li>
    }
</ul>
```

12.18 Add the following Code statements to the Account Controller Class

```
[HttpPost]
public async Task<IActionResult> Logout()
{
    await signInManager.SignOutAsync();
    return RedirectToAction("index", "home");
}
```

NOTE:

Session cookies are destroyed when the user closes the application whereas persistent cookies remain alive when the user closes the browser. The reason is persistent cookies remain on the users machine where as a session cookie is only alive during the given life cycle of a page request.

12.19 **Register** as a **New User**, and observe the Menu **Bar** contains the newly registered **User Name**.
12.20 Add the following code statements to the **Login View Models** page as shown below.

```
public class LoginViewModel
{    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    public string Password { set; get; }

    [Display(Name = "Remeber Me")]
    public bool RememberMe { get; set; }
}
```

12.21 Add the following statements to the **Login View** file as shown below

```
@model LoginViewModel
@{
    ViewBag.Title = "User Login";
}
<h1>User Login</h1>
<div class="row">
    <div class="col-md-12">
        <form method="post">
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Email"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <button type="submit" class="btn btn-primary">Login</button>
        </form>
    </div>
</div>
```

12.22 In the **Account Controller** implement Login functionality by entering the following code statements as shown below.

```
        [HttpGet]
        [AllowAnonymous]
        public IActionResult Login()
        {
            return View();
        }

        [HttpPost]
        [AllowAnonymous]
        public async Task<IActionResult> Login(LoginViewModel model,  string returnUrl)
        {
            if (ModelState.IsValid)
            {
                var result = await signInManager.PasswordSignInAsync(model.Email, model.Password,
model.RememberMe, false);
                if (result.Succeeded)
                {
                    if(!string.IsNullOrEmpty(returnUrl) && Url.IsLocalUrl(returnUrl))
                    {
                        return Redirect(returnUrl);
                    }
                   else
                    {
                        return RedirectToAction("Index", "home");
                    }
                }
              ModelState.AddModelError(string.Empty,"Invalid Login Appempt")
            }
            return View(model);
         }
```

12.23 Sign in using the same user credentials that were used during the Register User phase in steps 12.24 – 12.25. Observe the system accepts the credentials and the user is able to sign in. Logout out of the system and observe the user is logged out.

Role Based Implementation

At this stage the Registration and Login user features should be fully implemented. The business requirement for this application is to have users who have limited access to certain features such as Accounting and Human resources. There is one user category that will have full access to all the features of the application. The Accounting Feature will be added in a later release of the application

13.0 Create the following if it does not already exist, the following folders and files in the solution.

Controllers Folder -> AdministrationController

Models Folder -> IEmployeeRepository

ViewModels Folder -> UserRoleViewModel

ViewModels Folder -> EditRoleViewModel

ViewModels Folder -> CreateRoleViewModel

View->Administration->CreateRole.cshtml

View->Administration->EditRole.cshtml

View->Administration->EditUser.cshtml

View->Administration->EditUsersInRole.cshtml

View->Administration->ListRoles.cshtml

View->Administration->ListUsers.cshtml

View->Account -> AccessDenied.cshtml

Controller – AdministrationController

13.1 Implement the following code outline structure in the **AdministrationControler**. Each segment contains CRUD operations for Users and Roles.

```
using EmployeeManagementWebApplication.Models;
using EmployeeManagementWebApplication.ViewModels;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace EmployeeManagementWebApplication.Controllers
{

    [Authorize(Roles = "Admin")]
    public class AdministrationController: Controller
    {

        private readonly RoleManager<IdentityRole> roleManager;
        private readonly UserManager<ApplicationUser> userManager;

        public AdministrationController(RoleManager<IdentityRole> roleManager,
                                        UserManager<ApplicationUser> userManager)
        {
            this.roleManager = roleManager;
            this.userManager = userManager;
        }
        [HttpPost]
        public async Task<IActionResult> DeleteUser(string id)
        {
        }
        [HttpGet]
        public IActionResult ListUsers()
        {
        }
        [HttpGet]
        public async Task<IActionResult> EditUser(string id)
        {
        }
        [HttpPost]
        public async Task<IActionResult> EditUser(EditUserViewModel model)
        {
        }
        [HttpGet]
        public IActionResult CreateRole()
        {
            return View();
        }
        [HttpPost]
        public async Task<IActionResult> CreateRole(CreateRoleViewModel model)
        {
        }
        [HttpGet]
        public IActionResult ListRoles()
        {
        }
        [httpPost]
        Public async Task<IactionResult> DeleteRole(string id)
        {
        }
        [HttpGet]
        public async Task<IActionResult> EditRole(string id)
        {
        }
        [HttpPost]
        public async Task<IActionResult> EditRole(EditRoleViewModel model)
        {
        }
        [HttpGet]
        public async Task<IActionResult> EditUsersinRole(string roleId)
        {
        }
        [HttpPost]
        public async Task<IActionResult> EditUsersinRole(List<UserRoleViewModel> model, string roleId)
        {
        }
    }
}
```

November 2, 2019
Prepared by Andre Masters
Revision 1.0

13.2 Implement the following code statements for the **DeleteUser** Routine in the **Administration Controller**. The routine requires the error handling to be displayed in the **ViewBag** object and a redirection to **the NotFound** page if there are no users in the system. Next, check if the model state is valid if it is then execute the **Delete Operation** in the **userManager** routine in the **Identity** framework. If the result is successful, redirect to the **ListUsers** view.

```
public async Task<IActionResult> DeleteUser(string id)
{
    var user = await userManager.FindByIdAsync(id);
    if(user==null)
    {
        ViewBag.ErrorMessage = $"User with Id ={id} cannot be found";
        return View("NotFound");
    }
    else
    {
        var result = await userManager.DeleteAsync(user);

        if(result.Succeeded)
        {
            return RedirectToAction("ListUsers");
        }

        foreach(var error in result.Errors)
        {
            ModelState.AddModelError("", error.Description);
        }

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

13.3 Implement the following code construction for the **ListUsers** routine of the **AdministrationController**.

```
[HttpGet]
public IActionResult ListUsers()
{
    var users = userManager.Users;
    return View(users);
}
```

13.4 Implement the following code construction for the **ListRoles** routine of the **AdministrationController**.

```
[HttpGet]
public IActionResult ListRoles()
{
    var roles = roleManager.Roles;
    return View(roles);
}
```

13.5 Implement the following code construction for the **EditUsers** routine of the **AdministrationController**. This feature uses a HTTPGET verb to retrieve User Data to be edited. The aim of this routine is to change a single user based on their ID number. The ID number is a parameter that is passed in by value to the routine. Based on the user's ID its record in the database will be retrieved for edit processing, alongside its associated role.

NOTE:

It may be advisable to break up the Administration Controller into smaller pieces one controller for Users another for Roles and a third for UsersInRoles. The aim would be to make the code base more maintainable and less fragile.

```
[HttpGet]
public async Task<IActionResult> EditUser(string id)
{
    var user = await userManager.FindByIdAsync(id);

    if(user==null)
    {
        ViewBag.ErrorMessage = $"User with Id ={id} cannot be found";
        return View("NotFound");
    }

    var userRoles = await userManager.GetRolesAsync(user);

    var model = new EditUserViewModel
    {
        Id = user.Id,
        Email = user.Email,
        UserName = user.UserName,
        City = user.City,
        Roles = userRoles
    };

    return View(model);
}
```

13.6 Implement the HTTPPOST feature of the **EditUser** Routine using the code construction below.

```
[HttpPost]
public async Task<IActionResult> EditUser(EditUserViewModel model)
{
    var user = await userManager.FindByIdAsync(model.Id);
    if (user == null)
    {
        ViewBag.ErrorMessage = $"User with Id ={model.Id} cannot be found";
        return View("NotFound");
    }
    else
    {
        user.Email = model.Email;
        user.UserName = model.UserName;
        user.City = model.City;

        var result = await userManager.UpdateAsync(user);
        if(result.Succeeded)
        {
            return RedirectToAction("ListUsers");
        }

        foreach(var error in result.Errors)
        {
            ModelState.AddModelError("", error.Description);
        }
        return View(model);
    }
}
```

13.7 Implement the HTTPGET feature of the **CreateRole** Routine in the **Administration Controller** using the code construction below. This routine returns all the roles in existence prior to creating a new role.

```
[HttpGet]
public IActionResult CreateRole()
{
    return View();
}
```

13.8 Implement the HTTPPOST feature of the **CreateRole** Routine in the **Administration Controller** using the code construction below. A role does not require a user to be in existence and likewise a user does not require a role to exist.

```
[HttpPost]
public async Task<IActionResult> CreateRole(CreateRoleViewModel model)
{
    if(ModelState.IsValid)
    {
        IdentityRole identityRole = new IdentityRole
        {
            Name = model.RoleName
        };

        IdentityResult result = await roleManager.CreateAsync(identityRole);

        if(result.Succeeded)
        {
            return RedirectToAction("ListRoles", "Administration");
        }

        foreach(IdentityError error in result.Errors)
        {
            ModelState.AddModelError("", error.Description);
        }
    }

    return View(model);
}
```

13.9 Implement the HTTPGET feature of the **EditRole** Routine in the **Administration Controller** using the code construction below.

```
[HttpGet]
public async Task<IActionResult> EditRole(string id)
{
    var role = await roleManager.FindByIdAsync(id);
    if(role==null)
    {
        ViewBag.ErrorMessage = $"Role with Id = {id} cannot be found";
        return View("Not Found");
    }
    var model = new EditRoleViewModel
    {
        Id = role.Id,
        RoleName = role.Name
    };

    foreach(var user in userManager.Users)
    {
        if(await userManager.IsInRoleAsync(user, role.Name))
        {
            model.Users.Add(user.UserName);
        }
    }

    return View(model);
}
```

13.10 Implement the HTTPPOST feature of the **EditRole** Routine in the **Administration Controller** using the code construction below.

```
[HttpPost]
public async Task<IActionResult> EditRole(EditRoleViewModel model)
{
    var role = await roleManager.FindByIdAsync(model.Id);
    if (role == null)
    {
        ViewBag.ErrorMessage = $"Role with Id = {model.Id} cannot be found";
        return View("Not Found");
    }
    else
    {
        role.Name = model.RoleName;
        var result = await roleManager.UpdateAsync(role);

        if(result.Succeeded)
        {
            return RedirectToAction("ListRoles");
        }

        foreach(var error in result.Errors)
        {
            ModelState.AddModelError("", error.Description);
        }
        return View(model);
    }
}
```

13.11 Implement the HTTPGET feature of the **EditUserInRole** Routine in the **Administration Controller** using the code construction below. The routine requires the existence of at least one user to exist in a given role. The routine retrieves all the users for a given role.

```
[HttpGet]
public async Task<IActionResult> EditUsersinRole(string roleId)
{
    ViewBag.roleId = roleId;
    var role = await roleManager.FindByIdAsync(roleId);

    if(role == null)
    {ViewBag.errorMessage = $"Role with Id ={roleId} cannot be found";}
    var model = new List<UserRoleViewModel>();
    foreach(var user in userManager.Users)
    {
        var userRoleViewModel = new UserRoleViewModel
        {
            UserId = user.Id,
            UserName = user.UserName
        };
        if(await userManager.IsInRoleAsync(user,role.Name))
        {
            userRoleViewModel.IsSelected = true;
        }
        else
        {
            userRoleViewModel.IsSelected = false;
        }
        model.Add(userRoleViewModel);
    }
    return View(model);
}
```

13.12 Implement the HTTPPOST feature of the **EditUserInRole** Routine in the **Administration Controller** using the code construction below. The routine requires the existence of at least one user to exist in a given role. The routine accepts changes to a User for a given role as opposed to changing the user independent of role assignment.

```
[HttpPost]
public async Task<IActionResult> EditUsersinRole(List<UserRoleViewModel> model, string roleId)
{
    var role = await roleManager.FindByIdAsync(roleId);

    if(role== null)
    {
        ViewBag.ErrorMessage = $"Role with Id ={roleId} cannot be found";
        return View("NotFound");
    }
    for(int i = 0; i< model.Count; i++)
    {
        var user = await userManager.FindByIdAsync(model[i].UserId);

        IdentityResult result = null;

        if (model[i].IsSelected && !await userManager.IsInRoleAsync(user, role.Name))
        {
            result = await userManager.AddToRoleAsync(user, role.Name);
        }
        else if(!model[i].IsSelected && await userManager.IsInRoleAsync(user, role.Name))
        {
            result = await userManager.RemoveFromRoleAsync(user, role.Name);
        }
        else
        {
            continue;
        }

        if(result.Succeeded)
        {
            if (i < (model.Count - 1))
                continue;
            else
                return RedirectToAction("EditRole", new { Id = roleId });
        }
    }
    return RedirectToAction("EditRole", new { Id = roleId });
}
```

13.13 Implement the routine for deleting a Role using HttpPOST in the Administration Controller

```
[HttpPost]
  public async Task<IActionResult> DeleteRole(string id)
  {
      var role = await roleManager.FindByIdAsync(id);

      if (role == null)
      {
          ViewBag.ErrorMessage = $"Role with Id = {id} Cannot be found";
          return View("NotFound");
      }
      else
      {
          var result = await roleManager.DeleteAsync(role);
          if(result.Succeeded)
          {
              return RedirectToAction("ListRoles");
          }
          foreach(var error in result.Errors)
          {
              ModelState.AddModelError("", error.Description);
          }
      }
      return View("ListRoles");
  }
```

13.14 Implement the following code construction in the **IEmployeeRepository** in the Models folder

```
namespace EmployeeManagementWebApplication.Models
{
    public interface IEmployeeRepository
    {
        Employee GetEmployee(int id);
        IEnumerable<Employee> GetAllEmployees();
        Employee Add(Employee employee);
        Employee Update(Employee employeeChanges);
        Employee Delete(int id);
    }
}
```

View Models – User + Role + UserRole

13.15 Open the **UserRoleViewModel** in the **View Models** folder to support the **User** and **Role** relationship.

```
namespace EmployeeManagementWebApplication.ViewModels
{
    public class UserRoleViewModel
    {
        public string UserId { get; set; }
        public string UserName { get; set; }
        public bool IsSelected { get; set; }
    }
}
```

13.16 Open the **EditUserViewModel** in the **View Models** folder to support the **User** and **Role** relationship.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace EmployeeManagementWebApplication.ViewModels
{
    public class EditUserViewModel
    {
        public EditUserViewModel()
        {
            Roles = new List<string>();
        }
        public string Id { get; set; }

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

        [Required]
        [EmailAddress]
        public string Email { get; set; }
        public string City { get; set; }
        public IList<string> Roles { get; set; }
    }
}
```

13.17 Open the **CreateRoleViewModel** in the **View Models** folder to support the **User** and **Role** relationship.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace EmployeeManagementWebApplication.ViewModels
{
    public class CreateRoleViewModel
    {
        [Required]
        public string RoleName { set; get; }
    }
}
```

Views – Administration + Account

13.18 In the **Views -> Administration** folder implement the following code construction in the **CreateRole.cshtml** view file.

```
@model CreateRoleViewModel
@{
    ViewBag.Title = "Create New Role";
}
<h1>Create new Role</h1>
<div class="row">
    <div class="col-md-12">
        <form method="post">
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="RoleName"></label>
                <input asp-for="RoleName" class="form-control" />
                <span asp-validation-for="RoleName" class="text-danger"></span>
            </div>
            <button style="width:auto" type="submit" class="btn btn-primary">Create Role</button>
        </form>
    </div>
</div>
```

13.19 In the **Views -> Administration** folder implement the following code construction in the **EditRole.cshtml** view file.

```
@model EditRoleViewModel

@{
    ViewBag.Title = "Edit Role";
}

<h1>Edit Role</h1>

<form method="post" class="mt-3">
    <div class="form-group row">
        <label asp-for="Id" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Id" disabled class="form-control" />
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="RoleName" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="RoleName" class="form-control" />
            <span asp-validation-for="RoleName" class="text-danger"></span>
        </div>
    </div>
    <div asp-validation-summary="All" class="text-danger"></div>
    <div class="form-group row">
        <div class="col-sm-10">
            <button type="submit" class="btn btn-primary">Update</button>
            <a asp-action="ListRoles" class="btn btn-primary">Cancel</a>
        </div>
    </div>

    <div class="card">
        <div class="card-header">
            <h3>Users in this Role</h3>
        </div>
        <div class="card-body">
            @if(Model.Users.Any())
            {
                foreach(var user in Model.Users)
                {
                    <h5 class="card-title">@user</h5>
                }
            }
            else
            {
                <h5 class="card-title">None at the moment</h5>
            }
        </div>
        <div class="card-footer">
            <a asp-action="EditUsersInRole" asp-controller="Administration"
               asp-route-roleId="@Model.Id"class="btn btn-primary" style="width:auto">Add or Remove Users</a>
        </div>
    </div>
</form>
```

13.20 In the **Views -> Administration** folder implement the following code construction in the **EditUser.cshtml** view file.

```
@model EditUserViewModel

@{
    ViewBag.Title = "Edit User";
}

<h1>Edit User</h1>
<form method="post" class="mt-3">
    <div class="form-group row">
        <label asp-for="Id" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Id" disabled class="form-control" />
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="Email" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Email" class="form-control">
            <span asp-validation-for="Email" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="UserName" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="UserName" class="form-control">
            <span asp-validation-for="UserName" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="City" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="City" class="form-control">
            <span asp-validation-for="City" class="text-danger"></span>
        </div>
    </div>

    <div asp-validation-summary="All" class="text-danger"></div>

    <div class="form-group row">
        <div class="col-sm-10">
            <button type="submit" class="btn btn-primary">Update</button>
            <a asp-controller="Administration" asp-action="ListUsers" class="btn btn-primary">Cancel</a>
        </div>
    </div>

    <div class="card">
        <div class="card-header">
            <h3>User Roles</h3>
        </div>
        <div class="card-body">
            @if(Model.Roles.Any())
            {
                foreach(var role in Model.Roles)
                {
                    <h5 class="card-title">@role</h5>
                }
            }
            else
            {
                <h5 class="card-title">None at the moment</h5>
            }
        </div>
        <div class="card-footer">
            <a href="#" style="width:auto" class="btn btn-primary">Manage Roles</a>
        </div>
    </div>
</form>
```

13.21 In the **Views -> Administration** folder implement the following code construction in the **EditUsersInRole.cshtml** view file.

```
@model List<UserRoleViewModel>

@{
    var roleId = ViewBag.roleId;
}

<form method="post">
    <div class="card">
        <div class="card-header">
            <h2>Add or remove users from this role</h2>
        </div>
        <div class="card-body">
            @for(int i=0; i<Model.Count; i++)
            {
            <div class="form-check m-1">
                <input type="hidden" asp-for="@Model[i].UserId" />
                <input type="hidden" asp-for="@Model[i].UserName" />
                <input asp-for="@Model[i].IsSelected" class="form-check-input" />
                <label class="form-check-label" asp-for="@Model[i].IsSelected">
                    @Model[i].UserName
                </label>
            </div>
            }
        </div>
        <div class="card-footer">
            <input type="submit" value="Update" class="btn btn-primary" style="width:auto"/>
            <a asp-action="EditRole" asp-route-id="@roleId" class="btn btn-primary" style="width:auto">Cancel</a>
        </div>
    </div>
</form>
```

13.22 In the **Views -> Administration** folder implement the following code construction in the **ListRoles.cshtml** view file.

```
@model IEnumerable<IdentityRole>
@{
    ViewBag.Title = "All Roles";
}
<h1>All Roles</h1>
@if (Model.Any())
{
    <a class="btn btn-primary"style="width:auto" asp-action="CreateRole" asp-controller="administration">Add New
Role</a>
    foreach(var role in Model)
    {
        <div class="card mb-3">
            <div class="card-header">
                Role Id: @role.Id
            </div>
            <div class="card-body"style="font-size:large">
                @role.Name
            </div>
            <div class="card-footer">
                 <a class="btn btn-primary" asp-action="EditRole" asp-controller="Administration"
                                                                  asp-route-id="@role.Id">Edit</a>
            </div>
        </div>
    }
}
else
{
    <div class="card">
        <div class="card-header">
            No roles created yet
        </div>
        <div class="card-body">
            <h5 class="card-title"> Use the button below to create a role</h5>
            <a class="btn btn-primary" style="width:auto"
               asp-controller="administration" asp-action="CreateRole">Create Role</a>
        </div>
    </div>
}
```

13.22 In the **Views -> Administration** folder implement the following code construction in the **ListUsers.cshtml** view file.

```
@model IEnumerable<ApplicationUser>

@{
    ViewBag.Title = "All Users";
}

<h1>All Users</h1>

@if (Model.Any())
{
    <a asp-action="Register" asp-controller="Account"
       class="btn btn-primary mb-3" style="width:auto">
        Add new user
    </a>

    foreach (var user in Model)
    {
        <div class="card mb-3">
            <div class="card-header">
                User Id : @user.Id
            </div>
            <div class="card-body">
                <h5 class="card-title">@user.UserName</h5>
            </div>
            <div class="card-footer">
                <form asp-action="DeleteUser" asp-route-id="@user.Id" method="post">
                    <a asp-action="EditUser" asp-controller="Administration"
                       asp-route-id="@user.Id" class="btn btn-primary">Edit</a>
                    <button type="submit" class="btn btn-danger"
                          onclick="return confirm('Are you sure you want to delete user  :
                                   @user.UserName')">
                                  Delete</button>
                </form>
            </div>
        </div>
    }
}
else
{
    <div class="card">
        <div class="card-header">
            No users created yet
        </div>
        <div class="card-body">
            <h5 class="card-title">
                Use the button below to create a user
            </h5>
            <a class="btn btn-primary" style="width:auto"
               asp-controller="Account" asp-action="Register">
                Add new user
            </a>
        </div>
    </div>
}
```

13.23 In the **Views -> Account** folder implement the following code construction in the **AccessDenied.cshtml** view file.

```
<div class="text-center">
    <h1 class="text-danger">Access Denied</h1>
    <h6 class="text-danger">You do not have permision to access this resource</h6>
    <img src="~/images/AccessDenied.png" style="height:400px; width:400px"/>
</div>
```

13.24 Create the database migration call it AddIdentity using the statement below

```
Add-migration AddIdentity
```

13.25 Update the database and verify the following appears as shown in Figure 48

```
PM> update-database
Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.2.0-rtm-35687 initialized 'AppDbContext' using provider 'Micr
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (7ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT OBJECT_ID(N'[__EFMigrationsHistory]');
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT OBJECT_ID(N'[__EFMigrationsHistory]');
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT [MigrationId], [ProductVersion]
      FROM [__EFMigrationsHistory]
      ORDER BY [MigrationId];
Microsoft.EntityFrameworkCore.Migrations[20402]
      Applying migration '20190903204856_ExtendIdentityUser'.
Applying migration '20190903204856_ExtendIdentityUser'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (17ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      ALTER TABLE [Employees] ADD [Classification] int NOT NULL DEFAULT 0;
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      ALTER TABLE [Employees] ADD [EmployeeSalary] float NOT NULL DEFAULT 0.0E0;
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      ALTER TABLE [AspNetUsers] ADD [City] nvarchar(max) NULL;
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
      VALUES (N'20190903204856_ExtendIdentityUser', N'2.2.0-rtm-35687');
Done.
```

Figure 48

UAT Testing

13.26 Create a New Role for Accounting and another Role for HumanResources and a final role for Admin. Observe the following appears as shown in Figure 49, 50, 51, and 52.

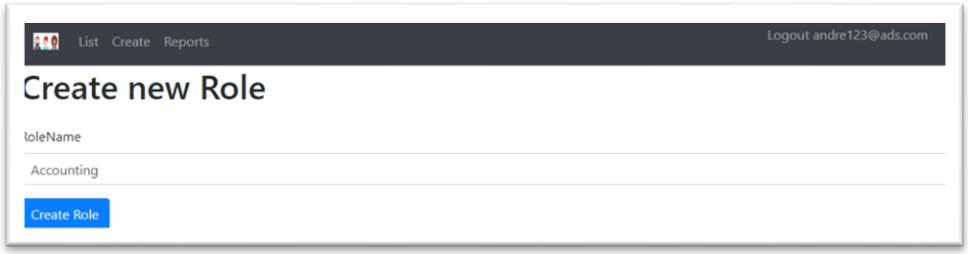


Figure 49

Figure 50

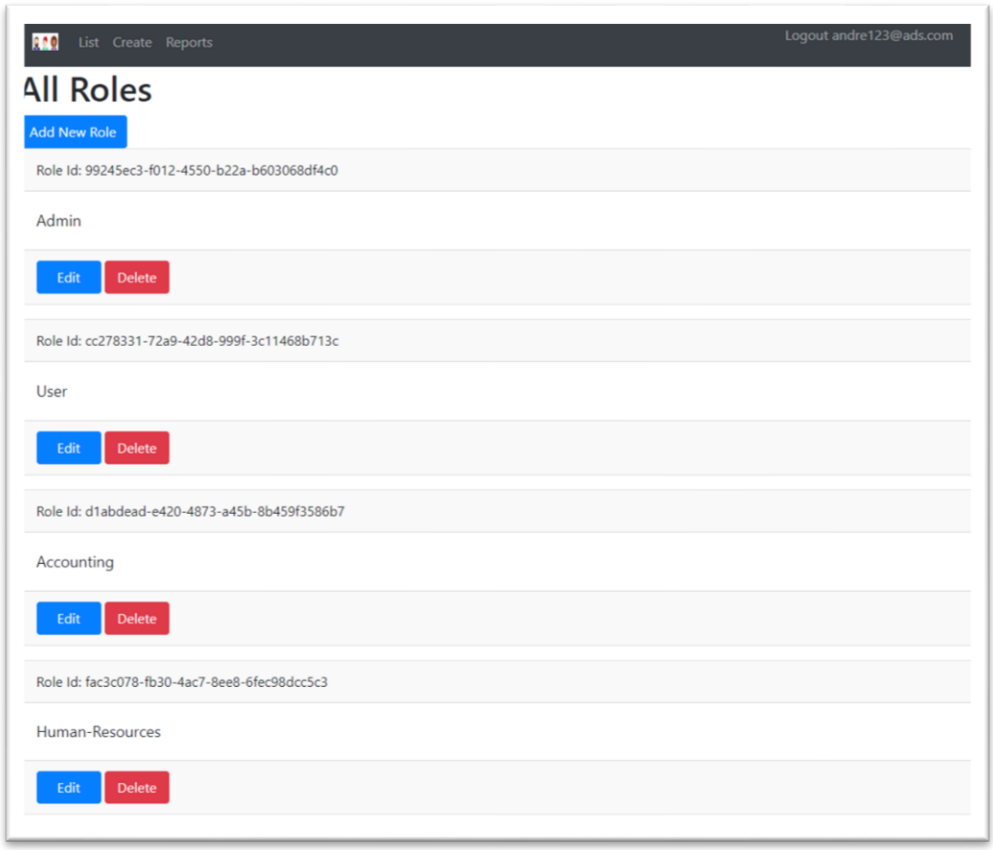


Figure 51

13.27 Go through the same steps to create the **Users. Edit** the **Users** while in **Role** and not in **Role**. Verify the list of **Users** and associated **Roles** appears as expected.

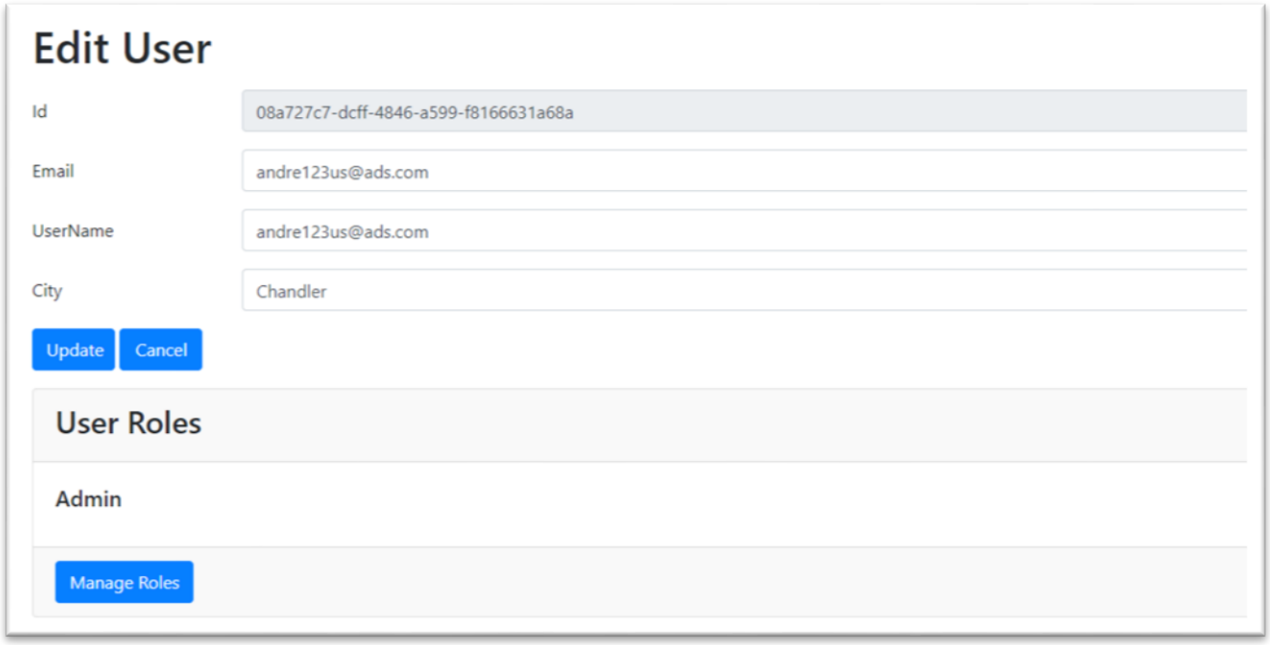


Figure 52

13.28 Test all the form fields for validation errors, if an unexpected validation error occurs make a note of it. The notes that get collected will be used for implementation artifacts in designing the logging section.

NOTE:

The solution is modified using Domain Models to represent each feature as complete product functionality. The Administration Controller as a Domain Object is its own controller folder with individual controllers for each crud operation. The Views have been updated to accommodate this change.

13.29 Implement the changes needed for the **Manage Roles** from the user's persepective as shown inside the red box below.

```
@model EditUserViewModel

@{
    ViewBag.Title = "Edit User";
}

<h1>Edit User</h1>
<form method="post" class="mt-3">
    <div class="form-group row">
        <label asp-for="Id" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Id" disabled class="form-control" />
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="Email" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Email" class="form-control">
            <span asp-validation-for="Email" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="UserName" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="UserName" class="form-control">
            <span asp-validation-for="UserName" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="City" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="City" class="form-control">
            <span asp-validation-for="City" class="text-danger"></span>
        </div>
    </div>

    <div asp-validation-summary="All" class="text-danger"></div>

    <div class="form-group row">
        <div class="col-sm-10">
            <button type="submit" class="btn btn-primary">Update</button>
            <a asp-controller="Administration" asp-action="ListUsers" class="btn btn-primary">Cancel</a>
        </div>
    </div>

    <div class="card">
        <div class="card-header">
            <h3>User Roles</h3>
        </div>
        <div class="card-body">
            @if(Model.Roles.Any())
            {
                foreach(var role in Model.Roles)
                {
                    <h5 class="card-title">@role</h5>
                }
            }
            else
            {
                <h5 class="card-title">None at the moment</h5>
            }
        </div>
        <div class="card-footer">
            <a asp-action="ManageUserRoles" asp-route-userId="@Model.Id"
               style="width:auto" class="btn btn-primary">Manage Roles</a>
        </div>
    </div>
</form>
```

13.30 Create a New View Model called UserRolesViewModel in the View Models folder. Enter the following code construction below.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace EmployeeManagementWebApplication.ViewModels
{
    public class UserRolesViewModel
    {
        public string RoleId { get; set; }
        public string RoleName { get; set; }
        public bool IsSelected { get; set; }
    }
}
```

13.31 In the **Administrator Controller** add the following routine name it **ManageUserRoles** use the code construction below. This routine performs a get request for users that are assigned to specific roles.

```
[HttpGet]
public async Task<IActionResult> ManageUserRoles (string userId)
{
    ViewBag.userId = userId;

    var user = await userManager.FindByIdAsync(userId);
    if(user==null)
    {
        ViewBag.ErrorMessage = $"User with id = {userId} cannot be found";
        return View("NotFound");
    }
    var model = new List<UserRolesViewModel>();

    foreach(var role in roleManager.Roles)
    {
        var userRolesViewModel = new UserRolesViewModel
        {
            RoleId = role.Id,
            RoleName = role.Name
        };

        if(await userManager.IsInRoleAsync(user, role.Name))
        {
            userRolesViewModel.IsSelected = true;
        }
        else
        {
            userRolesViewModel.IsSelected = false;
        }

        model.Add(userRolesViewModel);
    }

    return View(model);
}
```

13.32 In the **Administrator Controller** add the following routine name it **ManageUserRoles** use the code construction below. This routine performs a Post request for users that are assigned to specific roles.

```
[HttpPost]
public async Task<IActionResult> ManageUserRoles(List<UserRolesViewModel>model, string userId)
{
    var user = await userManager.FindByIdAsync(userId);
    if(user==null)
    {
        ViewBag.ErrorMessage = $"User with Id = {userId} cannot be found";
        return View("NotFound");
    }

    var roles = await userManager.GetRolesAsync(user);
    var result = await userManager.RemoveFromRolesAsync(user, roles);

    if(!result.Succeeded)
    {
        ModelState.AddModelError("", "cannot remove user from existing roles");
        return View(model);
    }

    result = await userManager.AddToRolesAsync(user, model.Where(x => x.IsSelected).Select(y => y.RoleName));

    if(!result.Succeeded)
    {
        ModelState.AddModelError("", "cannot add selected roles to user");
        return View(model);
    }

    return RedirectToAction("EditUser", new { Id = userId });
}
```

13.33 In the **Administration** sub folder within **Views Folder**, add a new view called **ManageUserRoles**. Use the following code construction for this view as shown below.

```
@model List<UserRolesViewModel>
@{
    var userId = ViewBag.userid;
}
<form method="post">
    <div class="card">
        <div class="card-header">
            <h2>Manage User Roles</h2>
        </div>
        <div class="card-body">
            @for(int i=0;i<Model.Count; i++)
            {
                <div class="form-check m-1">
                    <input type="hidden" asp-for="@Model[i].RoleId"/>
                    <input type="hidden" asp-for="@Model[i].RoleName"/>
                    <input asp-for="@Model[i].IsSelected" class="form-check-input" />
                    <label class="form-check-label" asp-for="@Model[i].IsSelected">@Model[i].RoleName</label>
                </div>
            }
            <div asp-validation-summary="All" class="text-danger"></div>
        </div>
        <div class="card-footer">
            <input type="submit" value="Update" class="btn btn-primary" style="width:auto" />
            <a asp-action="EditUser" asp-route-id="@userId" class="btn btn-primary" style="width:auto">Cancel</a>
        </div>
    </div>
</form>
```

UAT Testing

13.34 Complete the following steps to add and remove Users for a given role from the User's perspective.

13.35 Run the application by pressing F5 and login as the Admin User, observe the following appears as shown in Figure 53.

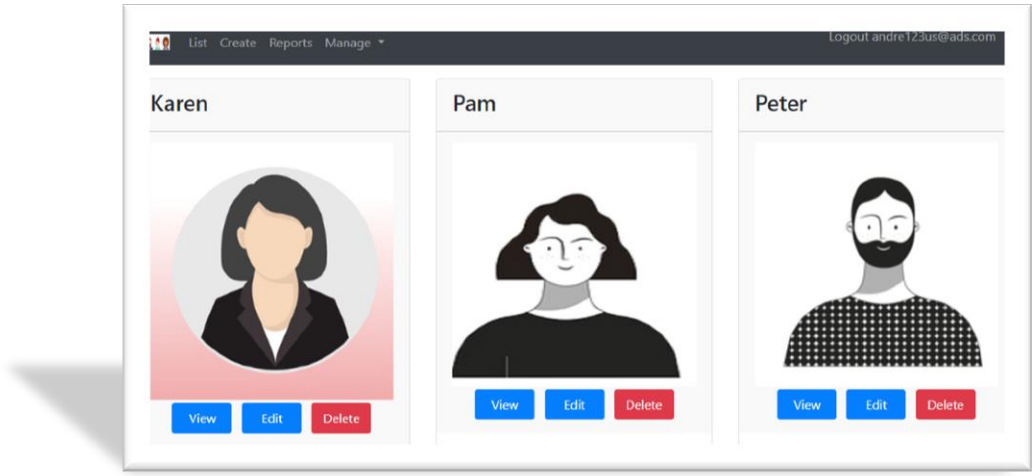


Figure 53

13.36 Under the **Manage Menu Option** select Users and observe a list of Users appears as shown in Figure 54.

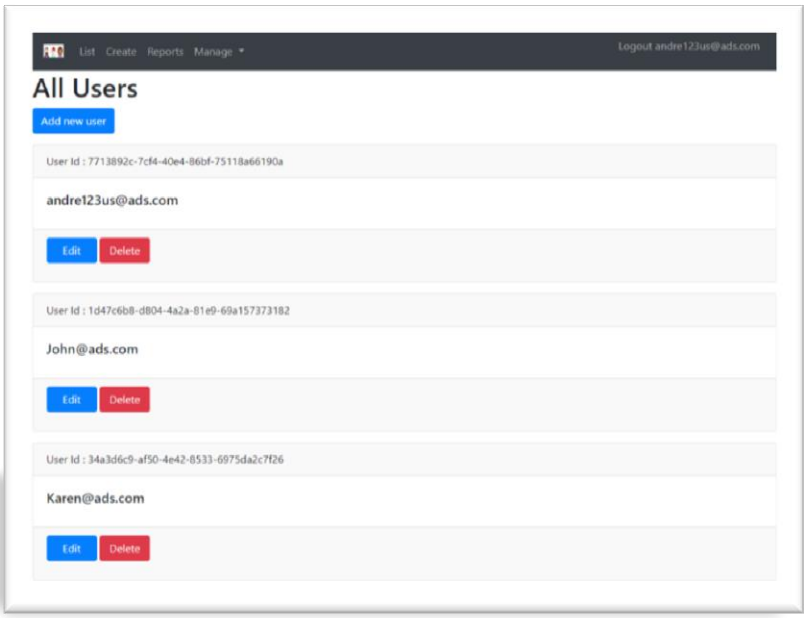


Figure 54

13.37 Select a User in this case it will be Karen@ads.com. Do this by selecting the Edit button for Karen. Observe the following appears as shown in Figure 55.

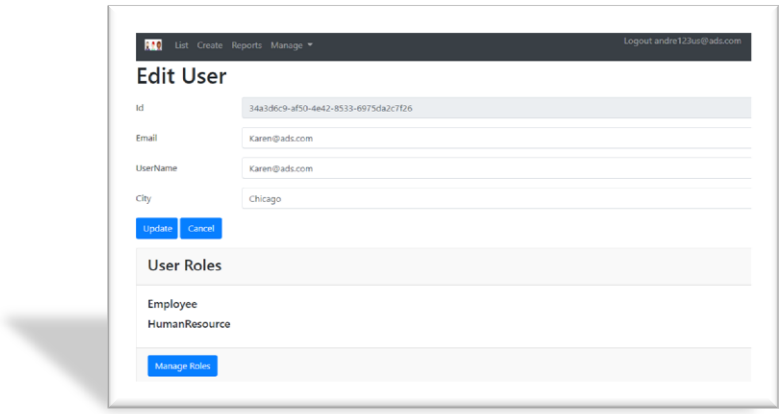


Figure 55

13.38 Press the **Manage Roles** button and observe the following screen appears as shown in Figure 56.

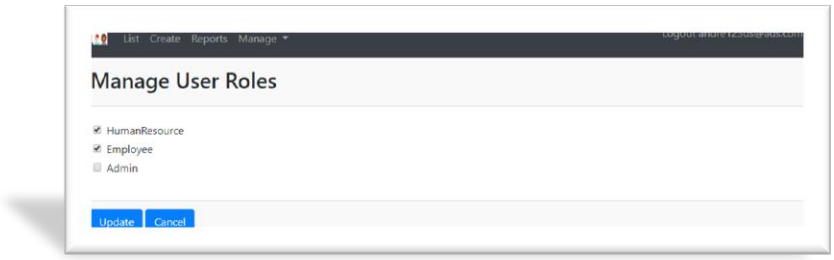


Figure 56

13.39 Remove the **Role HumanResource** by unchecking the checkbox for **HumanResource** and press **Update Button**. Observe the following takes place as shown in Figure 57.

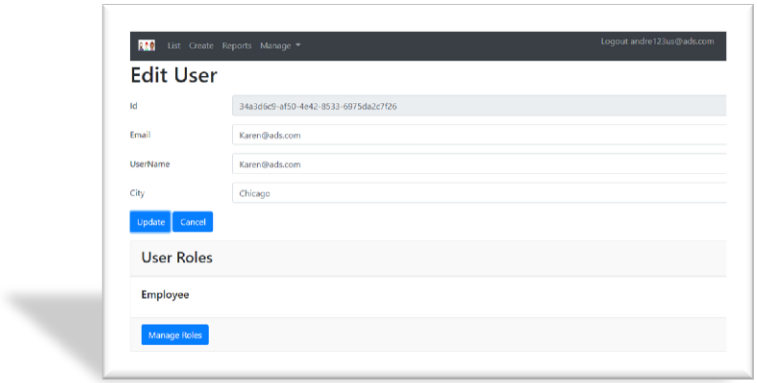


Figure 57

13.40 Press the **Update Button** and observe the following appears as shown in Figure 58.

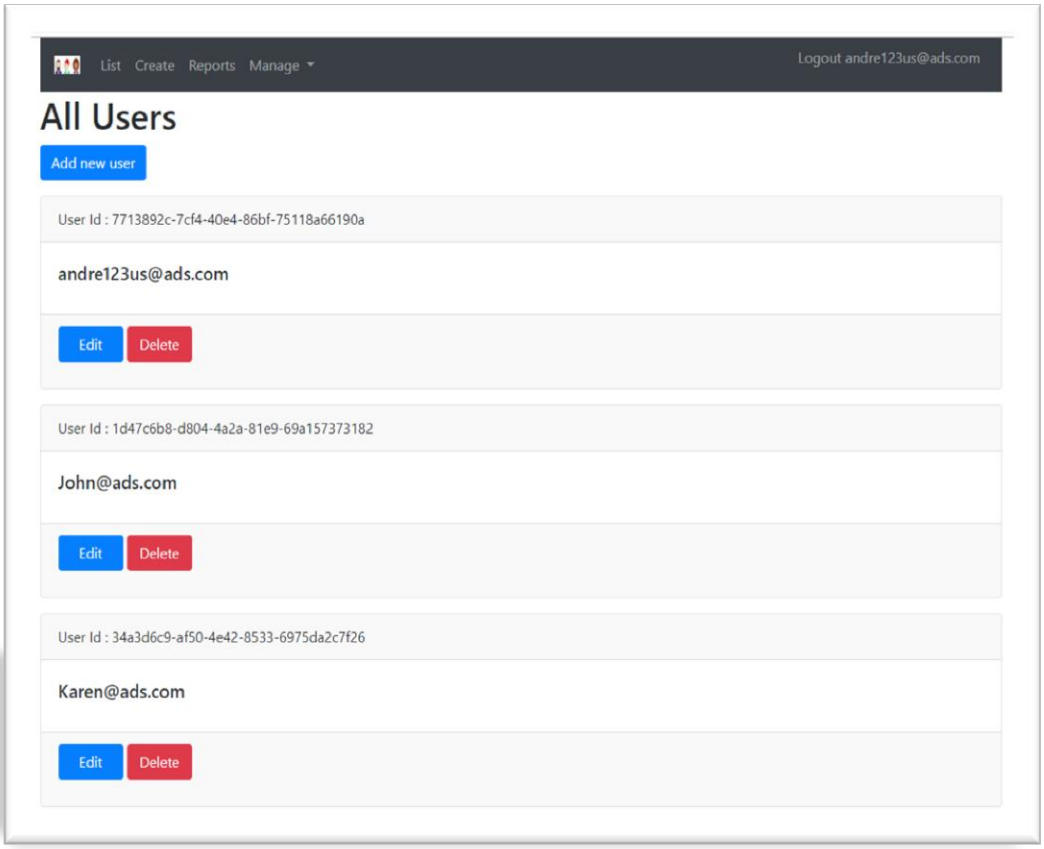
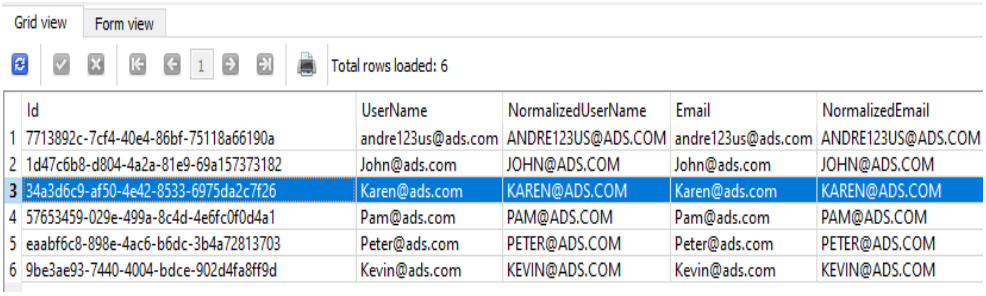


Figure 58

13.41 Verify the change has taken place in the database by entering the following SQL command in **SQLite database below**. First we need to find Karen's User ID number from the User Table

select * from AspNetUsers



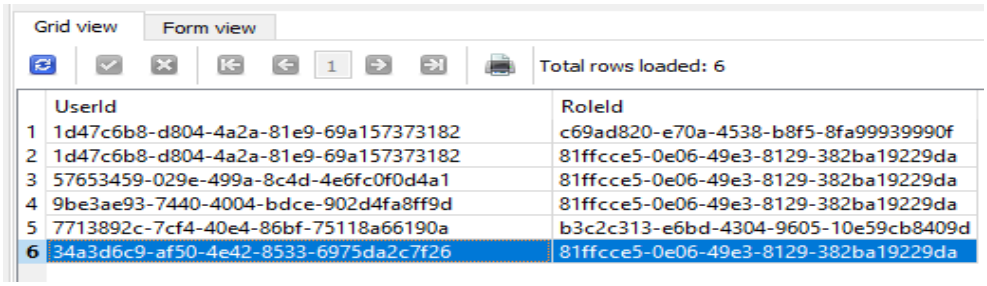
Grid view | Form view

Total rows loaded: 6

	Id	UserName	NormalizedUserName	Email	NormalizedEmail
1	7713892c-7cf4-40e4-86bf-75118a66190a	andre123us@ads.com	ANDRE123US@ADS.COM	andre123us@ads.com	ANDRE123US@ADS.COM
2	1d47c6b8-d804-4a2a-81e9-69a157373182	John@ads.com	JOHN@ADS.COM	John@ads.com	JOHN@ADS.COM
3	34a3d6c9-af50-4e42-8533-6975da2c7f26	Karen@ads.com	KAREN@ADS.COM	Karen@ads.com	KAREN@ADS.COM
4	57653459-029e-499a-8c4d-4e6fc0f0d4a1	Pam@ads.com	PAM@ADS.COM	Pam@ads.com	PAM@ADS.COM
5	eaabf6c8-898e-4ac6-b6dc-3b4a72813703	Peter@ads.com	PETER@ADS.COM	Peter@ads.com	PETER@ADS.COM
6	9be3ae93-7440-4004-bdce-902d4fa8ff9d	Kevin@ads.com	KEVIN@ADS.COM	Kevin@ads.com	KEVIN@ADS.COM

13.42 Select the Users in Roles table for Karen's Id number its location should be in the Employee Role only.

select * from AspNetUserRoles



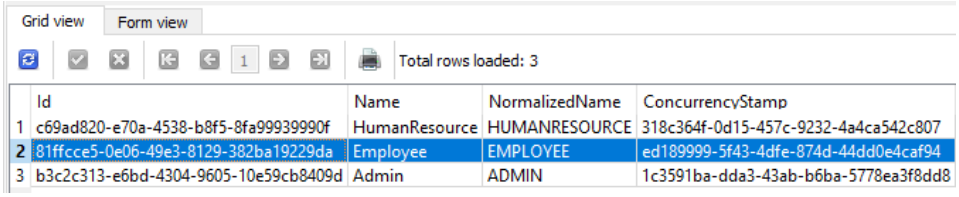
Grid view | Form view

Total rows loaded: 6

	UserId	RoleId
1	1d47c6b8-d804-4a2a-81e9-69a157373182	c69ad820-e70a-4538-b8f5-8fa99939990f
2	1d47c6b8-d804-4a2a-81e9-69a157373182	81ffcce5-0e06-49e3-8129-382ba19229da
3	57653459-029e-499a-8c4d-4e6fc0f0d4a1	81ffcce5-0e06-49e3-8129-382ba19229da
4	9be3ae93-7440-4004-bdce-902d4fa8ff9d	81ffcce5-0e06-49e3-8129-382ba19229da
5	7713892c-7cf4-40e4-86bf-75118a66190a	b3c2c313-e6bd-4304-9605-10e59cb8409d
6	34a3d6c9-af50-4e42-8533-6975da2c7f26	81ffcce5-0e06-49e3-8129-382ba19229da

13.43 Find the **RoleId** in the **Roles** table that matched the role Karen's **Employee ID** relates too by executing the following SQL statement below.

select * from AspNetRoles



Grid view | Form view

Total rows loaded: 3

	Id	Name	NormalizedName	ConcurrencyStamp
1	c69ad820-e70a-4538-b8f5-8fa99939990f	HumanResource	HUMANRESOURCE	318c364f-0d15-457c-9232-4a4ca542c807
2	81ffcce5-0e06-49e3-8129-382ba19229da	Employee	EMPLOYEE	ed189999-5f43-4dfe-874d-44dd0e4caf94
3	b3c2c313-e6bd-4304-9605-10e59cb8409d	Admin	ADMIN	1c3591ba-dda3-43ab-b6ba-5778ea3f8dd8

13.44 Add **HumanResource** to User Karen as a role by completing the same process as defined in previous test steps. The only difference is that instead of removing a role the existing HumanResource role gets added by checking the box for HumanResources in the **Manage User Roles** dashboard.

Claims Policy Implementation

The purpose of a claim is to allow membership in a given role to have acces to a claim within the assigned role. The claim can allow the user within the role to perform any action but is typically given in the form of a given CRUD operation.

14.0 The first step is to make the following additions to the EditUserViewModel as shown below in yellow.

```
public class EditUserViewModel
{
    public EditUserViewModel()
    {
        Roles = new List<string>();
        Claims = new List<string>();
    }

    public string Id { get; set; }

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


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

    public string City { get; set; }

    public IList<string> Roles { get; set; }
    public IList<string> Claims { get; set; }

}
```

14.1 Add a **Boostrap Card** to the **EditUserView** in order to access the User Claims view from the **EditUserView**. Place the following code construction just before the closing form tag in the **EditUserView**.

```
<div class="card">
    <div class="card-header">
        <h3>User Claims</h3>
    </div>
    <div class="card-body">
        @if(Model.Claims.Any())
        {
            foreach(var claim in Model.Claims)
            {
                <h5 class="card-title">@claim</h5>
            }
        }
        else
        {
            <h5 class="card-title">None at the moment</h5>
        }
    </div>
    <div class="card-footer">
        <a asp-action="ManageUserClaims" asp-route-userId="@Model.Id"
                    style="width:auto" class="btn btn-primary"> Manage Claims</a>
    </div>
</div>
```

NOTE:

In order for there to be a claim there must be a role and a policy associated with the role.

14.2 Create an object model for **ClaimsStore** using the code construction below.

```
public class ClaimsStore
{
    public static List<Claim> AllClaims = new List<Claim>()
    {
        new Claim("Create Role","true"),
        new Claim("Edit Role","true"),
        new Claim("Delete Role","true")
    };

}
```

NOTE:

The claims type and the claim value are the same. It's advisable to have a claim value set to true or false for a single selection. In the case where there are multiple options lets say a user can be a member of any number of approved groups in a list, then a comma separated list of group names should be used. If the user is a member any group in the list, the user's claim value is valid permitting the user to have access.

The value part of the given claim is case sensitive where as the claim name is case insensitive.

14.3 Create another object model for **UserClaim** using the code construction below.

```
public class UserClaim
{
    public string ClaimType { get; set; }
    public bool IsSelected { get; set; }
}
```

14.4 Create a view model object named **UserClaimsViewModel** using the code construction below.

```
public class UserClaimsViewModel
{
    public UserClaimsViewModel()
    {
        Claims = new List<UserClaim>();
    }

    public string userId { get; set; }
    public List<UserClaim> Claims { get; set; }
}
```

14.5 In the **AdministratorController** set the controller access authorization to **AdministrationRolePolicy** by changing the attribute at the controller level.

```
[Authorize(Policy = "AdminRolePolicy")]
```

14.6 Create a routine called **ManageUserClaims** in the **AdministrationController** class using the code construction below. This routine gets the existing claims for a given user

```
[HttpGet]
public async Task<IActionResult> ManageUserClaims(string userid)
{
    var user = await userManager.FindByIdAsync(userid);

    if(user==null)
    {
        ViewBag.ErrorMessage = $"User with Id= {userid} cannot be found";
        return View("NotFound");
    }

    var existingUserClaims = await userManager.GetClaimsAsync(user);
    var model = new UserClaimsViewModel
    {
        userId = userid
    };

    foreach(Claim claim in ClaimsStore.AllClaims)
    {
        UserClaim userClaim = new UserClaim
        {
            ClaimType = claim.Type
        };

        if(existingUserClaims.Any(c=>c.Type == claim.Type))
        {
            userClaim.IsSelected = true;
        }

        model.Claims.Add(userClaim);
    }

    return View(model);
}
```

14.7 Create a new View Called **ManageUserClaims** using the following code construction below.

```
@model UserClaimsViewModel
@{
    ViewBag.Title = "Manage User Claims";
}
<form method="post">
    <div class="card">
       <div class="card-header">
           <h2>Manage User Claims</h2>
       </div>
        <div class="card-body">
            @for(int i = 0; i < Model.Claims.Count; i++ )
            {
                <div class="form-check m-1">
                    <input type="hidden" asp-for="@Model.Claims[i].ClaimType" />
                    <input asp-for="@Model.Claims[i].IsSelected" class="form-check-input" />
                    <label class="form-check-label"asp-for="@Model.Claims[i].IsSelected">
                        @Model.Claims[i].ClaimType
                    </label>
                </div>
            }
            <div asp-validation-summary="All" class="text-danger"></div>
        </div>
        <div class="card-footer">
            <input type="submit" value="Update" class="btn btn-primary" style="width:auto"/>
            <a asp-action="EditUser" asp-route-id="@Model.userId" class="btn btn-primary" style="width:auto">Cancel</a>
        </div>
    </div>
</form>
```

14.8 Create a routine called **ManageUserClaims** in the **AdministrationController** class using the code construction below to POST a claim state to the database for a given user using the code construction below.

```
[HttpPost]
public async Task<IActionResult> ManageUserClaims(UserClaimsViewModel model)
{
    var user = await userManager.FindByIdAsync(model.userId);

    if (user == null)
    {
        ViewBag.ErrorMessage = $"User with Id= {model.userId} cannot be found";
        return View("NotFound");
    }

    var claims = await userManager.GetClaimsAsync(user);
    var result = await userManager.RemoveClaimsAsync(user, claims);

    if(!result.Succeeded)
    {
        ModelState.AddModelError("", "Cannot remove user from existing claims");
        return View(model);
    }

    result = await userManager.AddClaimsAsync(user, model.Claims.Where(c => c.IsSelected).Select(c => new

                                                                    Claim(c.ClaimType, c.ClaimType)));

    if(!result.Succeeded)
    {
        ModelState.AddModelError("", "cannot add selected claims to user");
        return View(model);
    }
    return RedirectToAction("EditUser", new { Id = model.userId });
}
```

14.9 Make the following changes to the EditUser routine as indicated in yellow highlight. The goal for this task is to allow the Edit User dashboard to contain the details for the user's existing claims status rights.

```
[HttpGet]
public async Task<IActionResult> EditUser(string id)
{
    var user = await userManager.FindByIdAsync(id);

    if(user==null)
    {
        ViewBag.ErrorMessage = $"User with Id ={id} cannot be found";
        return View("NotFound");
    }

    var userRoles = await userManager.GetRolesAsync(user);

    var userClaims = await userManager.GetClaimsAsync(user);

    var model = new EditUserViewModel
    {
        Id = user.Id,
        Email = user.Email,
        UserName = user.UserName,
        City = user.City,
        Roles = userRoles,
        Claims = (from Claim in userClaims
                  where Claim.Value == "true"
                  select Claim.Type).ToList()
    };

    return View(model);
}
```

14.10 In the ConfigureServices section add a policy authorization by implementing the following code construction highlighted in yellow.

```
public void ConfigureServices(IServiceCollection services)
{
    //Section I Database Configurations
    var DevelopmentDatabase = _config.GetConnectionString("EmployeeDBConnection");
    var ProductionDatabase = "Filename=employees.db";

    services.AddDbContextPool<AppDbContext>(
             options => options.UseSqlite(DevelopmentDatabase));

    //Section II XML Configuration
    services.AddMvc().AddXmlSerializerFormatters();

    //Section III Repository Section
    services.AddScoped<IEmployeeRepository, SQLEmployeeRepository>();
    services.AddScoped<IHumanResourcesRepository, SQLHumanResourceRepository>();

    //Section V Security Section

    services.AddIdentity<ApplicationUser, IdentityRole>()
            .AddEntityFrameworkStores<AppDbContext>();
    services.AddMvc(options =>
    {
        var policy = new AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();
        options.Filters.Add(new AuthorizeFilter(policy));
    });

    services.AddAuthorization(option =>
        {
            option.AddPolicy("AdminRolePolicy", policy => policy.RequireClaim("Delete Role")
                                                                .RequireClaim("Create Role"));
        });
}
```

14.11 In the **AdministrationController** add the following attribute to the **CreateRole** and **DeleteRole** POST action routines.

```
[Authorize(Policy ="RolePolicy")]
```

NOTE:

This stage of the program design, in order to create or delete a role; the user must be in the Admin role and be granted the permission to delete or create a role in the claims manager.

Chaining more than one claim to a given policy creates an AND condition requiring all members of the chain to be true in order to permit the policy claim to be successful.

RequireAssertion claims policy permits an OR condition that does not require all members of the policy to be true in order for the claim policy to execute successfully. This policy type requires only one condition be met in order to execute the policy successfully.

UAT Testing

The test objective is to verify the Role Manager and Claims Manager function as designed. Incidentily the Claims feature and the Role feature are both policy based from ASP.Net Core perspective. Since Admin is the only role that has access to the Claims Manager, a user with this role will be used for UAT testing purposes.

14.12 Run the application by loging into to the system using an **Admin** role user **Account**. In the **Manage Menu Option** select **User** as shown in Figure 59.

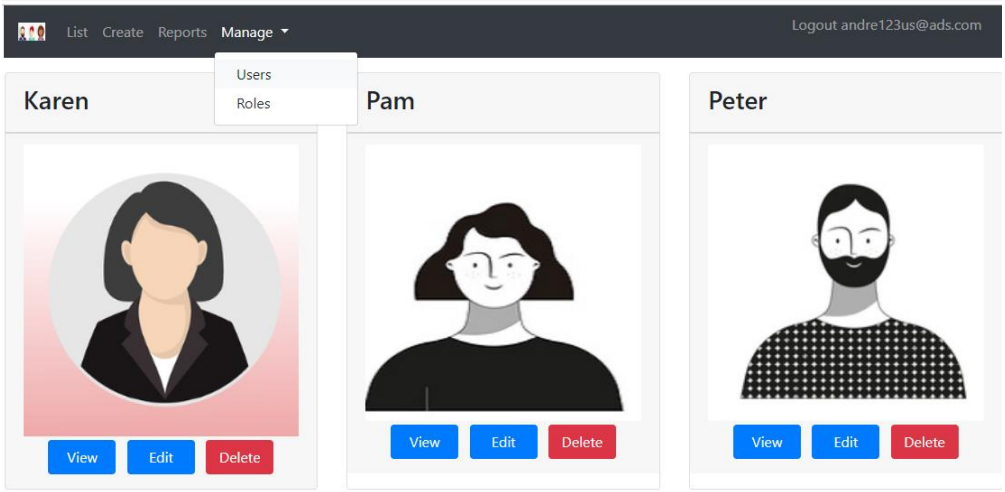


Figure 59

NOTE:

The current user may need to login and logout in order for changes to take a affect.

14 .13 In the User's list select a **User** to examine their profile by pressing the **Edit** button. In this case select John's profile as shown in Figure 60.

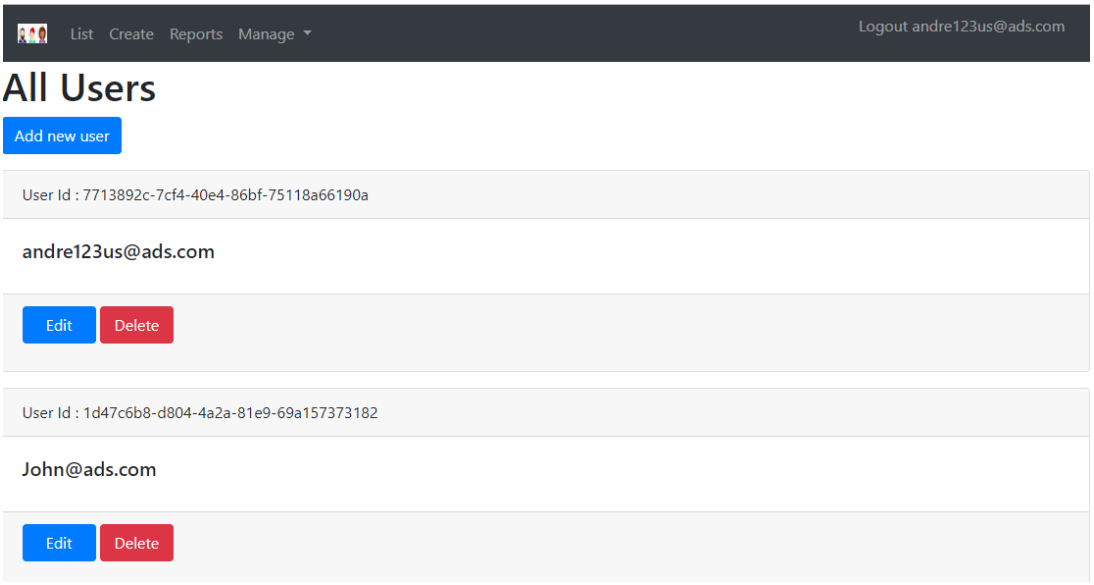


Figure 60

14.14 Observe that John has two **Roles** and has **Delete** and **Create Claims** in his profile as shown in Figure 61.

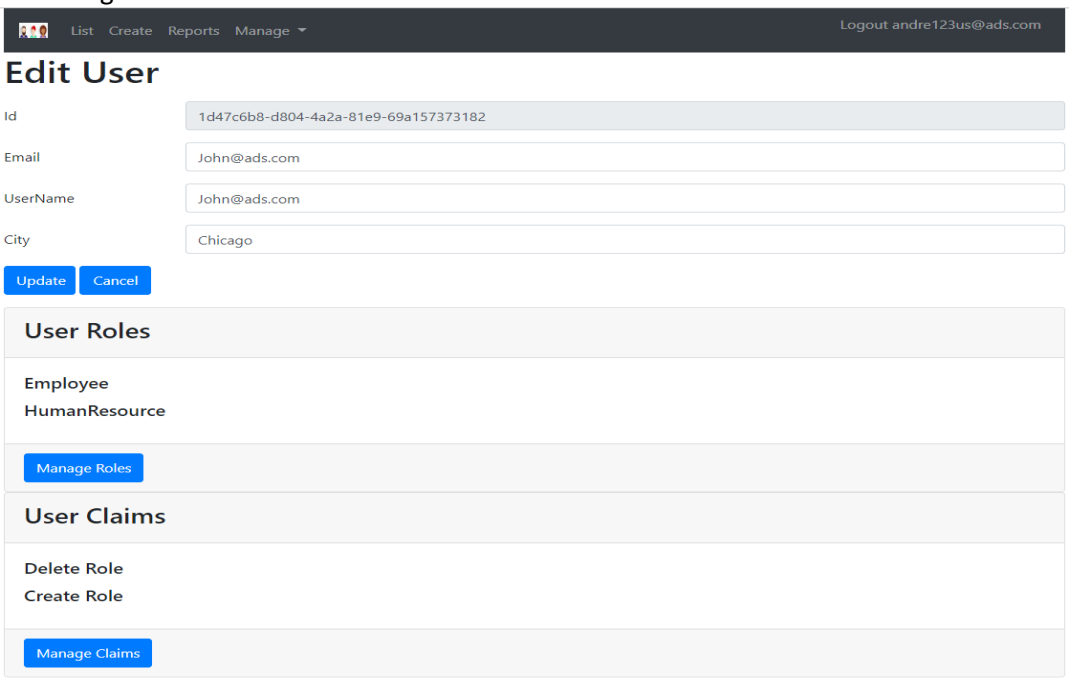


Figure 61

14.15 Press the **Manage Claims** button to access the options available for claims as shown in Figure 62.

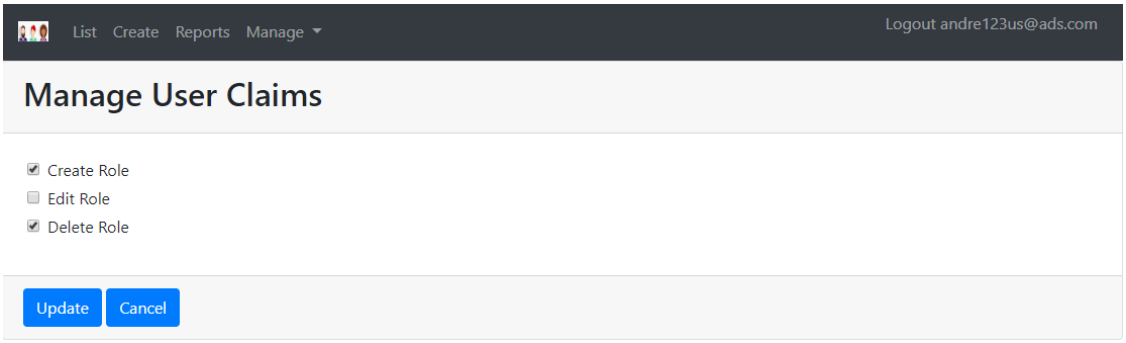


Figure 62

14.16 Observe that **Edit Role** is not checked. Check the **Edit Role** for John and Press the **Update button**. John is not able to make changes to any claims action items because John is not in the **admin role**. Observe Figure 62 and 63 to make the changes on the UI.

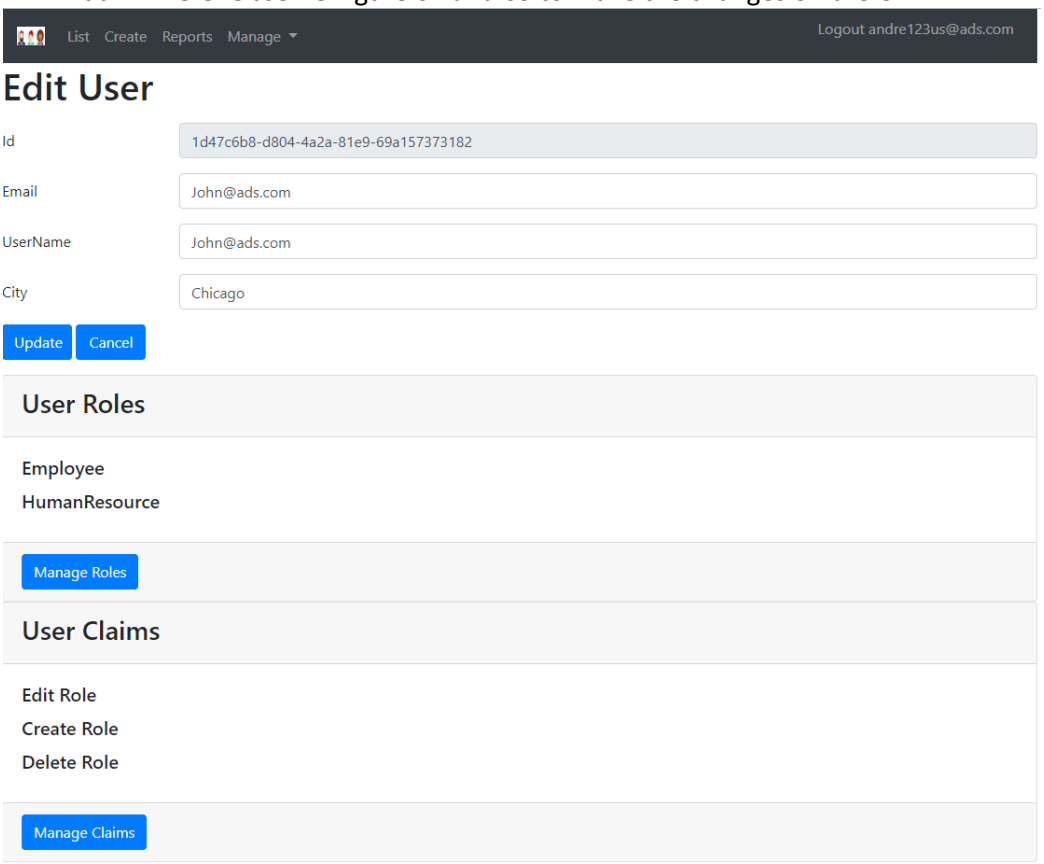
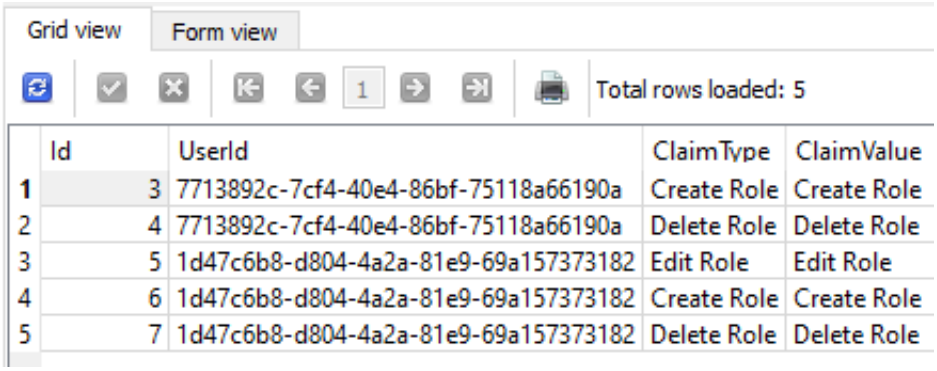


Figure 63

14.17 Verify John has access to all the features in the **Claims** section by executing the following SQL query below. The database in this case is SQLITE the procedure for migrating to SQLITE is covered in the section on Windows IOT for Raspberry Pi.

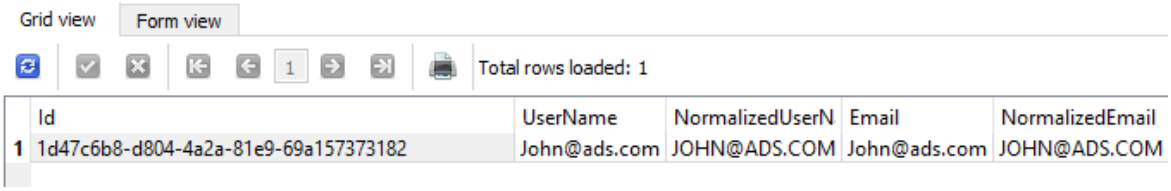
```
select * from AspNetUserClaims
```



	Id	UserId	ClaimType	ClaimValue
1	3	7713892c-7cf4-40e4-86bf-75118a66190a	Create Role	Create Role
2	4	7713892c-7cf4-40e4-86bf-75118a66190a	Delete Role	Delete Role
3	5	1d47c6b8-d804-4a2a-81e9-69a157373182	Edit Role	Edit Role
4	6	1d47c6b8-d804-4a2a-81e9-69a157373182	Create Role	Create Role
5	7	1d47c6b8-d804-4a2a-81e9-69a157373182	Delete Role	Delete Role

14.18 Verify John's UserId is **1d47c6b8-d804-4a2a-81e9-69a157373182** by executing the following query.

```
select * from AspNetUsers
where Id = '1d47c6b8-d804-4a2a-81e9-69a157373182'
```

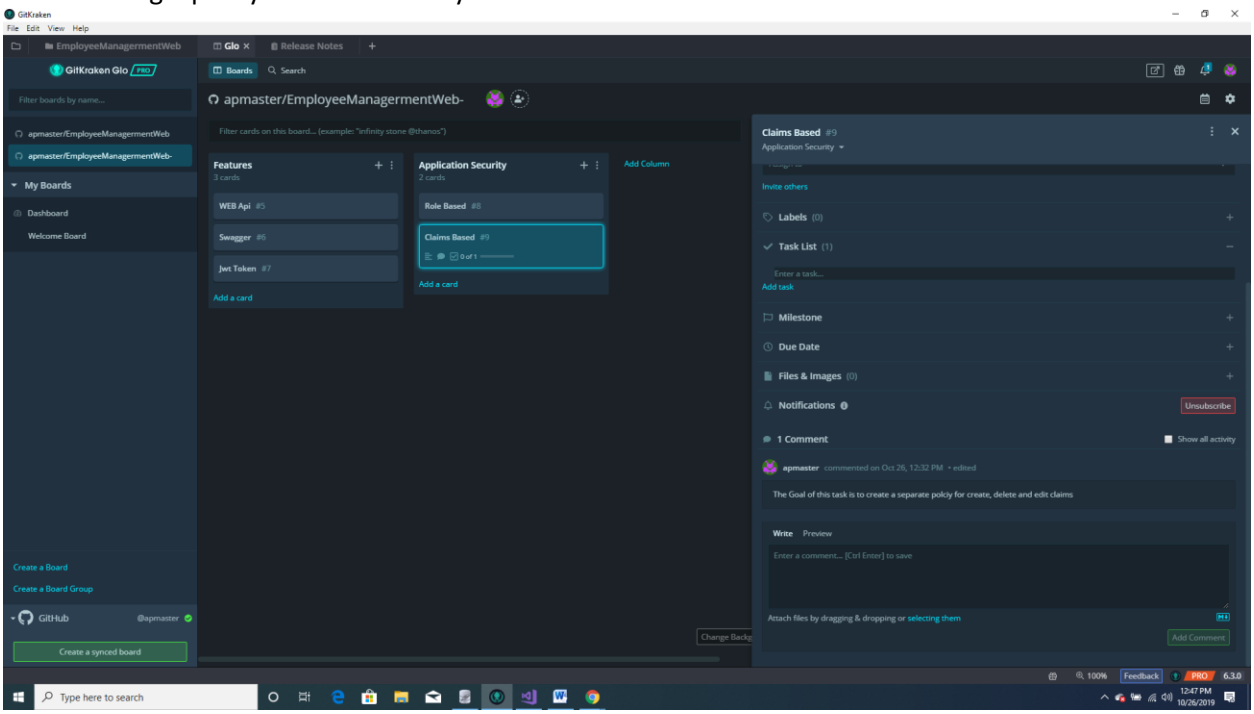


	Id	UserName	NormalizedUserN	Email	NormalizedEmail
1	1d47c6b8-d804-4a2a-81e9-69a157373182	John@ads.com	JOHN@ADS.COM	John@ads.com	JOHN@ADS.COM

14.19 Since John's UserId contains all the features for **Claims Policy** as shown in the **Edit User** dashboard for John's account this test is deemed a successs. Remove all the **User Claims** for John redo all the steps defined in this UAT test process. Observe the claims have access for John is removed in the **Employee Edit** profile dashboard and in the database. Repeat this process for each employee in the organization. Next, grant some Users with Admin role access and redo the process using the newly granted admin rights.

Breakup Role Policy to Individual Claim Policies

15.0 Based on the story in the **GitKracken** the product owner wants the application to allow User's to have a separate claim policy for Create, Delete and Edit claim. The current design has this feature as a single policy called RolePolicy.



15.1 In the StartUp.cs class in the Configure Services routine make the following changes as shown below.

EXISTING ROLE POLICY SERVICE

```
services.AddAuthorization(option =>
    {
        option.AddPolicy("RolePolicy", policy => policy.RequireClaim("Delete Role")
                                                       .RequireClaim("Create Role")
                                                       .RequireClaim("Edit Role"));
         });
```

NEW CLAIMS POLICY SERVICE

```
services.AddAuthorization(option =>
    {
        option.AddPolicy("DeleteRolePolicy", policy => policy.RequireClaim("Delete Role",”true”));
        option.AddPolicy("EditRolePolicy", policy => policy.RequireClaim("Edit Role",”true”));
        option.AddPolicy("CreateRolePolicy", policy => policy.RequireClaim("Create Role",”true”));
        option.AddPolicy("AdminRolePolicy", policy => policy.RequireClaim("Admin"));
    });
```

November 2, 2019
Prepared by Andre Masters
Revision 1.0

15.2 In the **AdministratorController** remove **RolePolicy** attribute on all controller actions methods. This is being done because the policy no longer exists in the configure serverices section in the **Startup.cs** file.

15.3 In the **AdministratorController** add the following attribute to the **EditRole Action Method** for both **Http:Get** and **Http:Post** methods.

```
[Authorize(Policy = "EditRolePolicy")]
```

15.4 In the **AdministratorController** add the following attribute to the **DeleteRole Action Method** under the **Http:Post** attribute.

```
[Authorize(Policy = "DeleteRolePolicy")]
```

15.5 In the **AdministratorController** add the following attribute to the **CreateRole Action Method** for both **Http:Get** and **Http:Post** methods.

```
[Authorize(Policy = "CreateRolePolicy")]
```

15.6 In the **ListRoles View** make the following changes in cyan highlighted text in order to support **Create**, **Update** and **Delete** operations for a given role.

```
@model IEnumerable<IdentityRole>
@using Microsoft.AspNetCore.Authorization
@inject IAuthorizationService authorizationservice

@{
    ViewBag.Title = "All Roles";
}
<h1>All Roles</h1>
@if (Model.Any())
 {
    <a class="btn btn-primary" style="width:auto" asp-action="CreateRole" asp-controller="administration">Add New Role</a>

 @foreach (var role in Model)
 {
    <div class="card mb-3">
        <div class="card-header">
            Role Id: @role.Id
        </div>
        <div class="card-body" style="font-size:large">
            @role.Name
        </div>
        <div class="card-footer">

            <form asp-action="DeleteRole" asp-route-id="@role.Id" method="post">
                @if ((await authorizationservice.AuthorizeAsync(User, "CreateRolePolicy")).Succeeded)
                {
                    <a class="btn btn-primary" asp-action="CreateRole"
                       asp-controller="Administration" asp-route-id="@role.Id">
                        Create
                    </a>
                }
                @if ((await authorizationservice.AuthorizeAsync(User, "EditRolePolicy")).Succeeded)
                {
                    <a class="btn btn-primary" asp-action="EditRole"
                       asp-controller="Administration" asp-route-id="@role.Id">
                        Edit
                    </a>
                }
                @if ((await authorizationservice.AuthorizeAsync(User, "DeleteRolePolicy")).Succeeded)
                {
                    <span id="confirmDeleteSpan_@role.Id" style="display:none">
                        <span>Are you sure you want to delete?</span>
                        <button type="submit" class="btn btn-danger">Yes</button>
                        <a href="#" class="btn btn-primary"
                           onclick="confirmDelete('@role.Id', false)">No</a>
                    </span>

                    <span id="deleteSpan_@role.Id">
                        <a href="#" class="btn btn-danger"
                           onclick="confirmDelete('@role.Id', true)">Delete</a>
                    </span>
                }
            </form>
        </div>
```

UAT Testing Multiple Policy Claims

The purpose for this test is to validate the functionality of **Create**, **Edit**, and **Update Claims** for a given role. In order for a given role to be modified the user must be in the admin role since this role is a required attribute at the **Controller Level**.

15.7 Run the application and login as an Admin user whose has **Update**, **Create**, **Edit Role Authorization**.
In the Manage option menu, select roles and observe the following appears as shown in Figure 64

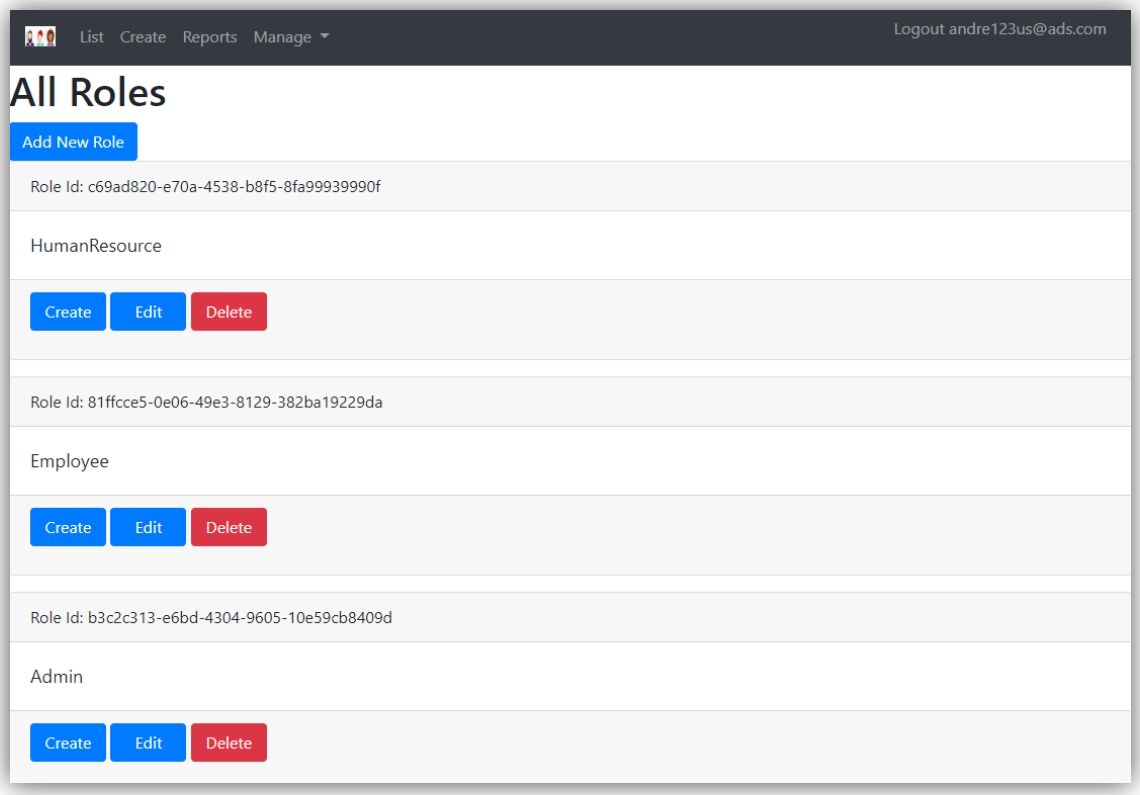


Figure 64

NOTE:

Since the Create button functionality is duplicated in the AddNew Role, it does not make sense to add a separate button for each user unless we want to remove the Add Role Button functionality as a global feature of the application.

If the Create feature is a local event related to a single user only then it's advisable to remove the global Create Role button and retain the Create role as a local per user relationship a single button for each user.

November 2, 2019
Prepared by Andre Masters
Revision 1.0

If an Admin desires to update multiple users as single batch, then a datagrid approach similar to the HR section may be more productive feature since the Admin could make changes to multiple users at once.

15.8 Under the Manage menu option select the Users option and observe the following appears as shown in Figure 65.

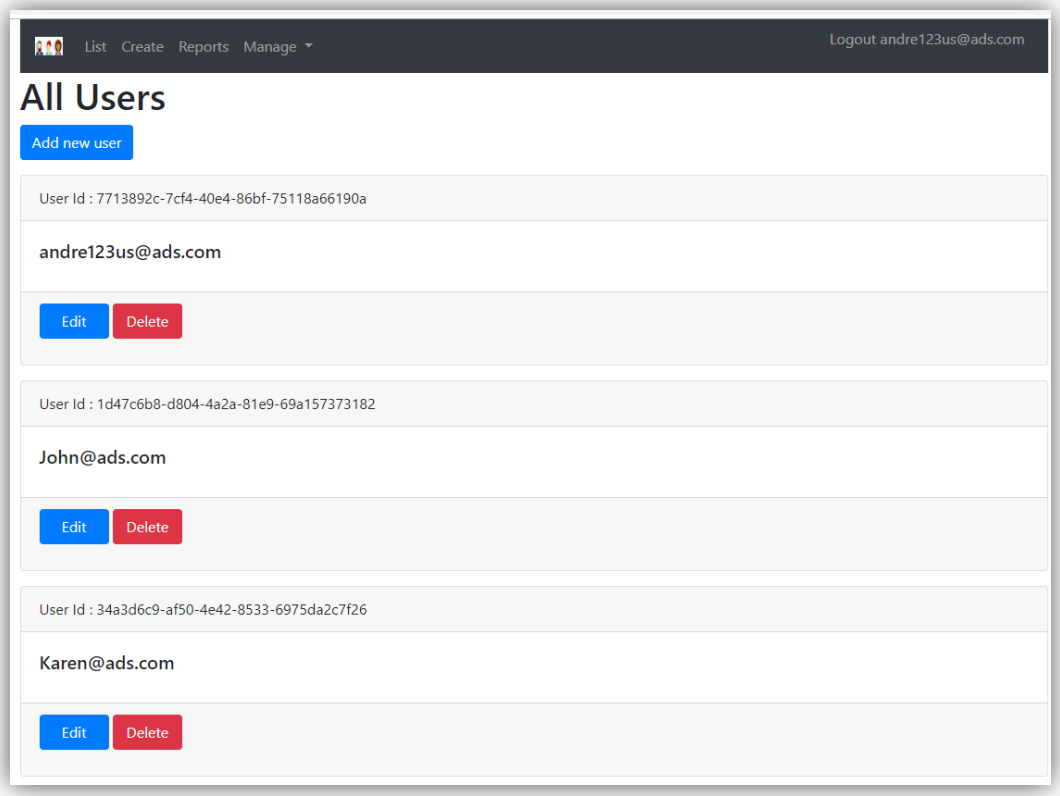


Figure 65

15.9 Query the list to see which user has **Create** but not **Edit** or **Delete** role option in their **Account Profile**. The User must have access to the **Admin Role** in this case John meets all of the requirements for this test case as shown in Figure 66

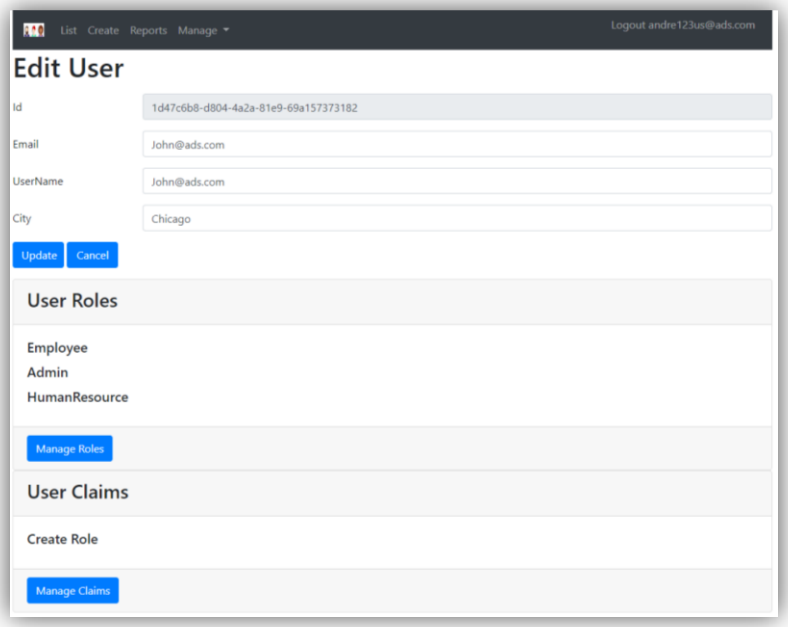


Figure 66

15.10 Logout of the application and log back in using John's credentials and navigate to the roles option and observe the following appears as shown in Figure 67

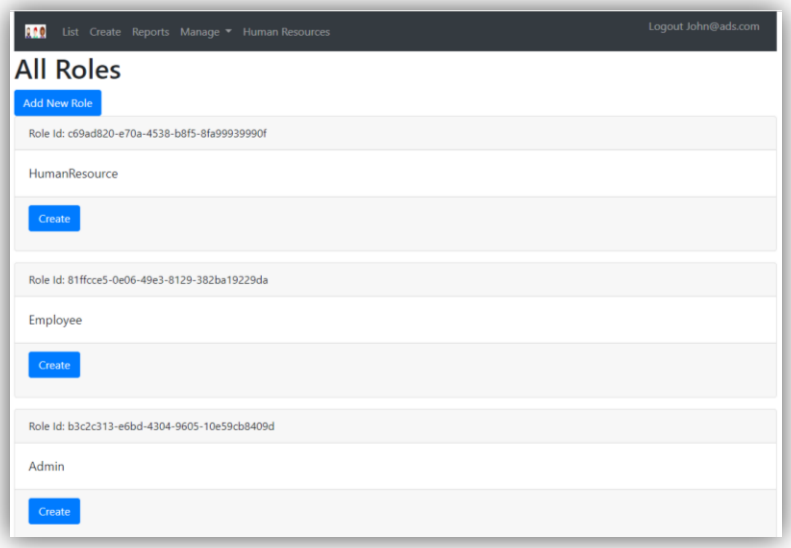


Figure 67

15.11 Try to navigate to the EditRole feature in the url and observe the following screen appears as shown in Figure 68.

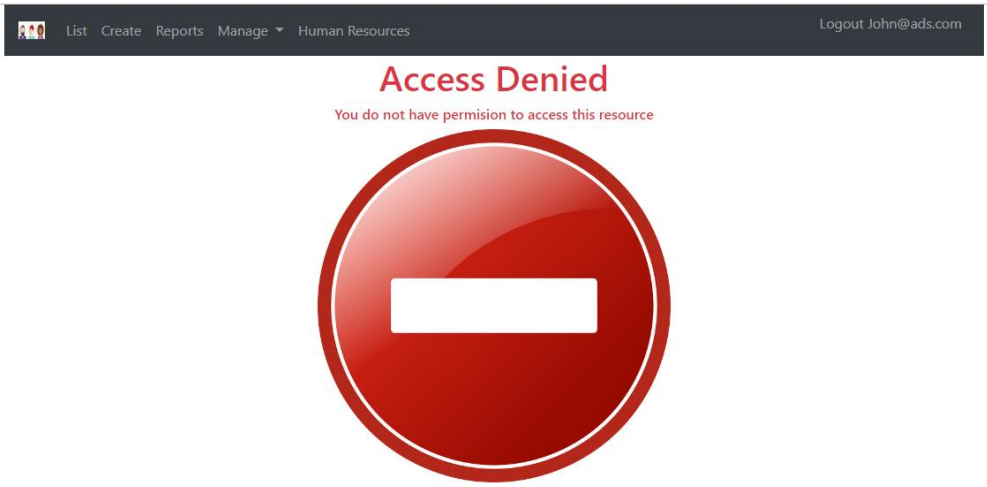


Figure 68

NOTE:

The purpose of having a create button was for illustrative purposes. To keep the application coherent remove the Create Claims policy the associated UI controls from the List Roles view.

15.12 Modify the ListView Form control to contain Delete and Edit features as shown below.

EXISTING LISTROLES VIEW FORM

```
<form asp-action="DeleteRole" asp-route-id="@role.Id" method="post">
    @if ((await authorizationservice.AuthorizeAsync(User, "CreateRolePolicy")).Succeeded)
    {
        <a class="btn btn-primary" asp-action="CreateRole"
           asp-controller="Administration" asp-route-id="@role.Id">
            Create
        </a>
    }
    @if ((await authorizationservice.AuthorizeAsync(User, "EditRolePolicy")).Succeeded)
    {
        <a class="btn btn-primary" asp-action="EditRole"
           asp-controller="Administration" asp-route-id="@role.Id">
            Edit
        </a>
    }
    @if ((await authorizationservice.AuthorizeAsync(User, "DeleteRolePolicy")).Succeeded)
    {
        <span id="confirmDeleteSpan_@role.Id" style="display:none">
            <span>Are you sure you want to delete?</span>
            <button type="submit" class="btn btn-danger">Yes</button>
            <a href="#" class="btn btn-primary"
               onclick="confirmDelete('@role.Id', false)">No</a>
        </span>

        <span id="deleteSpan_@role.Id">
            <a href="#" class="btn btn-danger"
               onclick="confirmDelete('@role.Id', true)">Delete</a>
        </span>
    }
</form>
```

NEW LISTROLES VIEW FORM

```
<form asp-action="DeleteRole" asp-route-id="@role.Id" method="post">
    @if ((await authorizationservice.AuthorizeAsync(User, "EditRolePolicy")).Succeeded)
    {
        <a class="btn btn-primary" asp-action="EditRole"
           asp-controller="Administration" asp-route-id="@role.Id">
            Edit
        </a>
    }
    @if ((await authorizationservice.AuthorizeAsync(User, "DeleteRolePolicy")).Succeeded)
    {
        <span id="confirmDeleteSpan_@role.Id" style="display:none">
            <span>Are you sure you want to delete?</span>
            <button type="submit" class="btn btn-danger">Yes</button>
            <a href="#" class="btn btn-primary"
               onclick="confirmDelete('@role.Id', false)">No</a>
        </span>

        <span id="deleteSpan_@role.Id">
            <a href="#" class="btn btn-danger"
               onclick="confirmDelete('@role.Id', true)">Delete</a>
        </span>
    }
</form>
```

15.13 In the **AdministratorController** remove the following attribute to the **CreateRole Action Method** for both **Http:Get** and **Http:Post** methods.

```
[Authorize(Policy = "CreateRolePolicy")]
```

15.14 In the **StartUp.cs** file remove the following service for create policy claim as shown below.

```
services.AddAuthorization(option =>
    {
        option.AddPolicy("DeleteRolePolicy", policy => policy.RequireClaim("Delete Role",”true”));
        option.AddPolicy("EditRolePolicy", policy => policy.RequireClaim("Edit Role",”true”));
        option.AddPolicy("CreateRolePolicy", policy => policy.RequireClaim("Create Role",”true”));
        option.AddPolicy("AdminRolePolicy", policy => policy.RequireClaim("Admin"));
    });
```

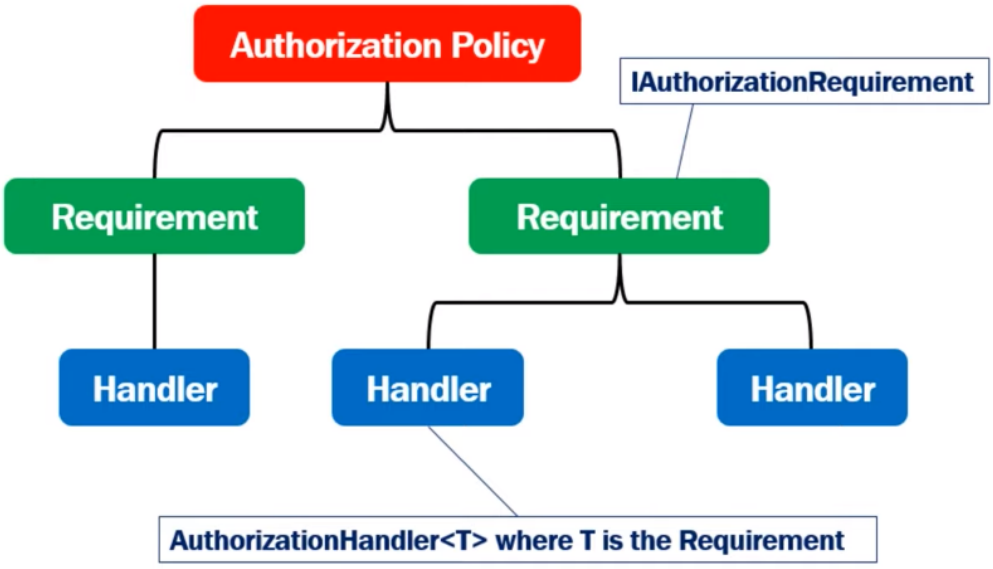
15.15 Redo the UAT tests to verify the Create button is removed in the Users list and that all the same functionality remains with no errors or issues.

Custom Security Policy

In this section an authorization and access policy will be created to limit the power of the Admin role. The goal is to not allow a given Admin to make changes to their own account but require another admin to make changes instead. The diagram below shows the builtin authorization work flow process.

An Admin user can manage other Admin user roles and claims but not their own claims and roles

Custom authorization requirement



NOTE:

The following steps show how to implement a Custom Security Policy in ASP.Net Core 2.2. The steps list below show the implementation for what is needed to develop a coherent code design which illustrates clearly the Authorization Policy design from a procedural perspective.

Use a microservice implementation instead of a procedural approach. A microserivce offers a cleaner architecture that is suitable for a message based systems which may utilize a service bus for communication between entities. Also, a microservice allows for better granualarity for cross custting concerns. This has the benfit of allowing the system to operate without compilation of the entire system during periods of enhancements and updates.

November 2, 2019
Prepared by Andre Masters
Revision 1.0

Step - 1 : Create the Custom Requirement

```
public class ManageAdminRolesAndClaimsRequirement : IAuthorizationRequirement
{ }
```

Step - 2 : Create the Custom Handler

```
public class CanEditOnlyOtherAdminRolesAndClaimsHandler :
AuthorizationHandler<ManageAdminRolesAndClaimsRequirement>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
        ManageAdminRolesAndClaimsRequirement requirement)
    {
        var authFilterContext = context.Resource as AuthorizationFilterContext;
        if (authFilterContext == null)
            return Task.CompletedTask;

        string loggedInAdminId =
            context.User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;

        string adminIdBeingEdited = authFilterContext.HttpContext.Request.Query["userId"];

        if (context.User.IsInRole("Admin") &&
            context.User.HasClaim(claim => claim.Type == "Edit Role" && claim.Value == "true") &&
            adminIdBeingEdited.ToLower() != loggedInAdminId.ToLower())
        {
            context.Succeed(requirement);
        }

        return Task.CompletedTask;
    }
}
```

Step - 3 : Register the Custom Handler

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthorization(options =>
    {
        options.AddPolicy("EditRolePolicy", policy =>
            policy.AddRequirements(new ManageAdminRolesAndClaimsRequirement()));
    });

    services.AddSingleton<IAuthorizationHandler,
        CanEditOnlyOtherAdminRolesAndClaimsHandler>();
}
```

Step - 4 : Use the Custom Policy

```
[HttpGet]
[Authorize(Policy = "EditRolePolicy")]
public async Task<IActionResult> ManageUserRoles(string userId)
{
    // Code
    return View();
}
```

16.0 Create a new folder named **Security** in the main project structure. In this folder create a new class called **ManageAdminRolesAndClaimsRequirements.**

16.1 Implement the interface IAuthorizationRequirement to the **ManageAdminRolesAndClaimsRequirements** class.

16.2 Implement the ManageAdminRolesAndClaimsRequirements class using the following code construction.

```
public class CanEditOnlyOtherAdminRolesAndClaimsHandler:AuthorizationHandler<ManageAdminRolesAndClaimsRequirements>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                        ManageAdminRolesAndClaimsRequirements requirement)
    {
        var authFilterContext = context.Resource as AuthorizationFilterContext;

        if(authFilterContext == null)
        {
            return Task.CompletedTask;
        }

        string loggedInAdminId = context.User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
        string adminIdBeingEdited = authFilterContext.HttpContext.Request.Query["userId"];

        if(context.User.IsInRole("Admin") && context.User.HasClaim(claim => claim.Type == "Edit Role"
                                        && claim.Value =="true")
                                        && adminIdBeingEdited.ToLower() != loggedInAdminId.ToLower())
        {
            context.Succeed(requirement);
        }

        return Task.CompletedTask;
    }
}
```

16.3 Register the service in the Startp Configuration Services section of the StartUp Class as shown in yellow high-lighted text.

```
        //Section V Security Section

        services.AddIdentity<ApplicationUser, IdentityRole>()
                .AddEntityFrameworkStores<AppDbContext>();
        services.AddMvc(options =>
        {
            var policy = new AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();
            options.Filters.Add(new AuthorizeFilter(policy));
        });

        services.AddAuthorization(option =>
            {
                option.AddPolicy("DeleteRolePolicy", policy => policy.RequireClaim("Delete Role","true"));
                option.AddPolicy("EditRolePolicy", policy => policy.AddRequirements(new
                                                        ManageAdminRolesAndClaimsRequirements()));
                option.AddPolicy("AdminRolePolicy", policy => policy.RequireRole("Admin"));
            });

        services.AddSingleton<IAuthorizationHandler, CanEditOnlyOtherAdminRolesAndClaimsHandler>();
```

16.4 Remove the **EditRolePolicy** from **EditUsers Action** in the **Administrator Controller** and place it in the **ManageUserRoles** controller action methods for both Post and Get.

```
[Authorize(Policy = "EditRolePolicy")]
```

UAT Testing Admin can't change their own Role

16.5 **Login** as **Admin** and navigate to the **List All Users** section of the **Manage Sub Menu** located in the Main menu bar as shown in Figure 69.

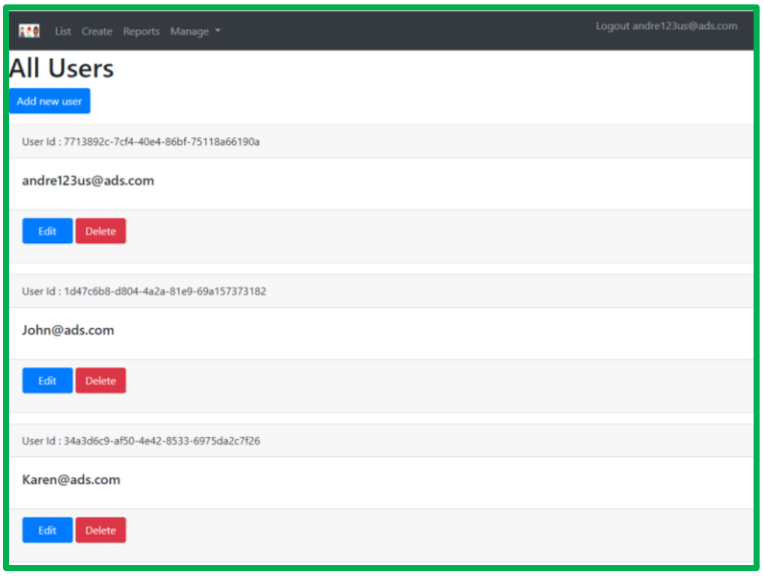


Figure 69

16.6 Select the **Edit Option** for the current logged in **User**. Observe the following appears as shown in Figure 70

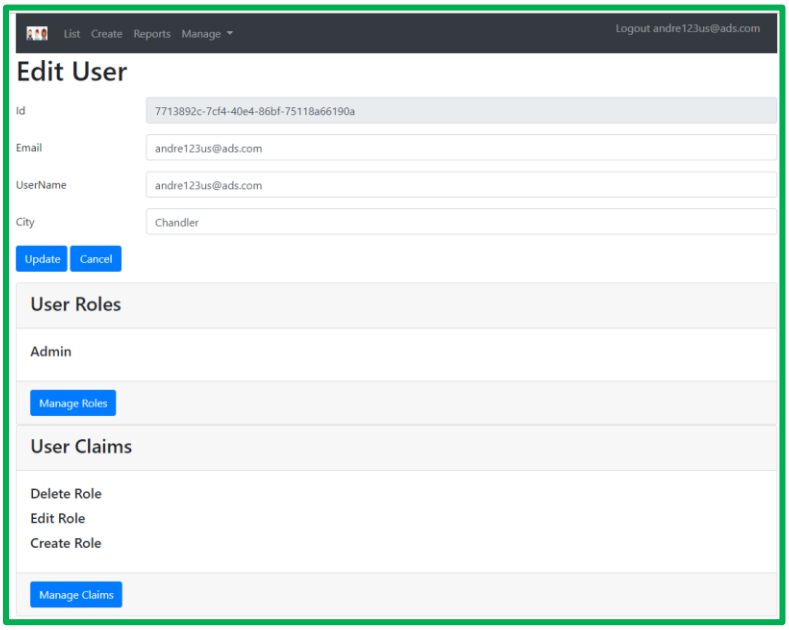


Figure 70

16.7 Select **Manage Roles** for this User who is already logged in and observe the following screen appears in Figure 71.

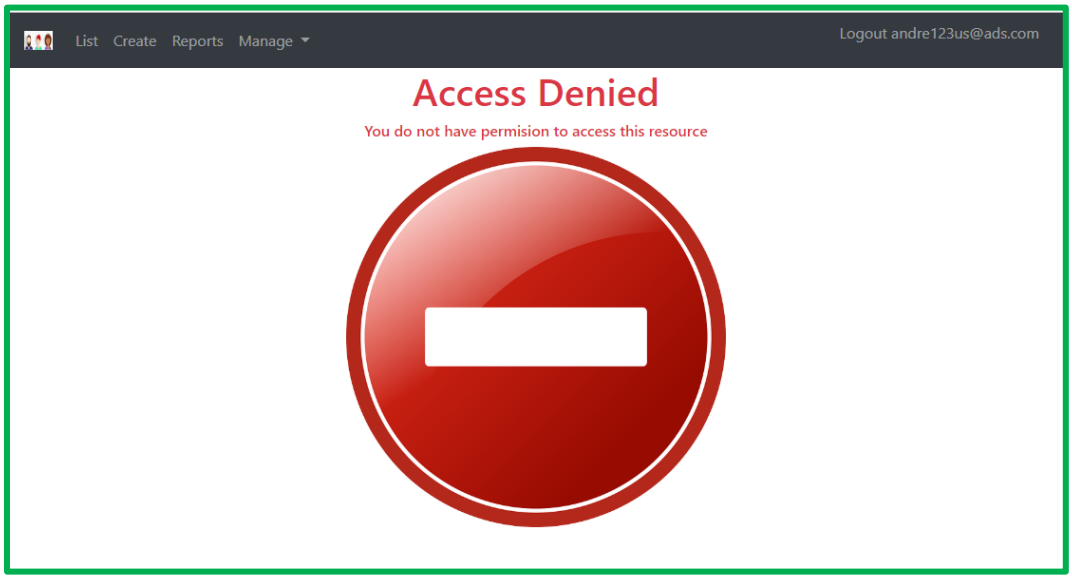


Figure 71

NOTE:

The current design of the application only blocks a user from modifying their own role but does not restrict them from modifying the claims that is assoictaed with this user account. The next UAT test will check for the resitcion on claims and roles restrictions. There will be a final UAT test to verify the Admin user is able to change other user accounts including Admin role based account types.

16.8 Make the following changes to the **Security Section** of the **Configure Services** routine in the **StartUp** class. The changes are in yellow highlisghted text.

```
services.AddAuthorization(option =>
 {
   option.AddPolicy("DeleteRolePolicy", policy => policy.AddRequirements(new ManageAdminRolesAndClaimsRequirements()));
   option.AddPolicy("EditRolePolicy", policy => policy.AddRequirements(new ManageAdminRolesAndClaimsRequirements()));
   option.AddPolicy("EditClaimPolicy", policy => policy.AddRequirements(new ManageAdminRolesAndClaimsRequirements()));
   option.AddPolicy("AdminRolePolicy", policy => policy.RequireRole("Admin"));
 });
```

16.9 Make the following changes to the CanEditOnlyOtherAdminRolesAndClaimsHandler routine located in the Security folder.

```
public class CanEditOnlyOtherAdminRolesAndClaimsHandler:AuthorizationHandler<ManageAdminRolesAndClaimsRequirements>
{
   protected override Task HandleRequirementAsync(AuthorizationHandlerContext context, ManageAdminRolesAndClaimsRequirements requirement)
       {
           var authFilterContext = context.Resource as AuthorizationFilterContext;
           if(authFilterContext == null)
           {
               return Task.CompletedTask;
           }
           string loggedInAdminId = context.User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
           string adminIdBeingEdited = authFilterContext.HttpContext.Request.Query["userId"];
           if(context.User.IsInRole("Admin") && context.User.HasClaim(claim => claim.Type == "Edit Role" && claim.Value =="true") &&
                                               adminIdBeingEdited.ToLower() != loggedInAdminId.ToLower())
           {
               context.Succeed(requirement);
           }
           if (context.User.IsInRole("Admin") && context.User.HasClaim(claim => claim.Type == "Edit Claim" && claim.Value == "true") &&
                adminIdBeingEdited.ToLower() != loggedInAdminId.ToLower())
           {
               context.Succeed(requirement);
           }
           return Task.CompletedTask;
       }
   }
```

16.10 Replace the **Authorization Policy** on **ManageUserRoles** routine with the following authorization policy as shown below. Do this for both Http Post and Http Get verbs.

```
[Authorize(Policy = "EditRolePolicy")]
```

16.11 Replace the **Authorization Policy** on **DeleteRole** routine with the following authorization policy as shown below. Do this for both Http Post and Http Get verbs.

```
[Authorize(Policy = "DeleteRolePolicy")]
```

16.12 Replace the **Authorization Policy** on **ManageUserClaims** routine with the following authorization policy as shown below. Do this for both Http Post and Http Get verbs.

```
[Authorize(Policy = "EditClaimPolicy")]
```

UAT Testing Admin can't change their own Claim options

16.13 Login as Admin and navigate to the list users section of the Manage Option located in the Main menu bar as shown in Figure 72.

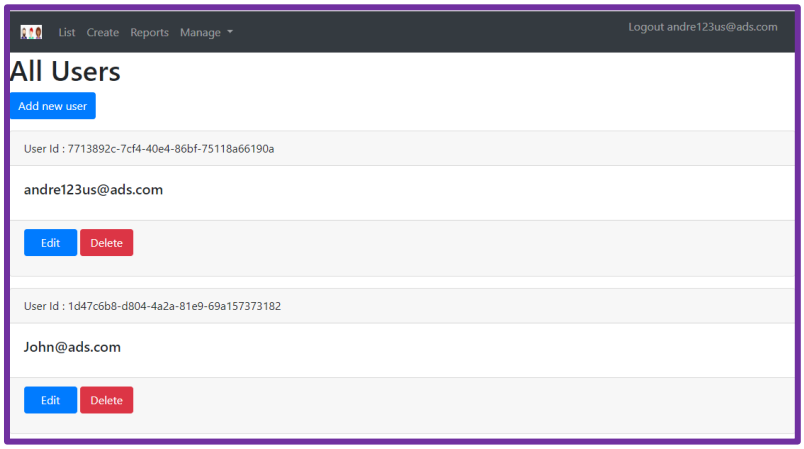


Figure 72

16.14 Select the **Edit Option** for the current logged in **User** and observe the following appears as shown in Figure 73.

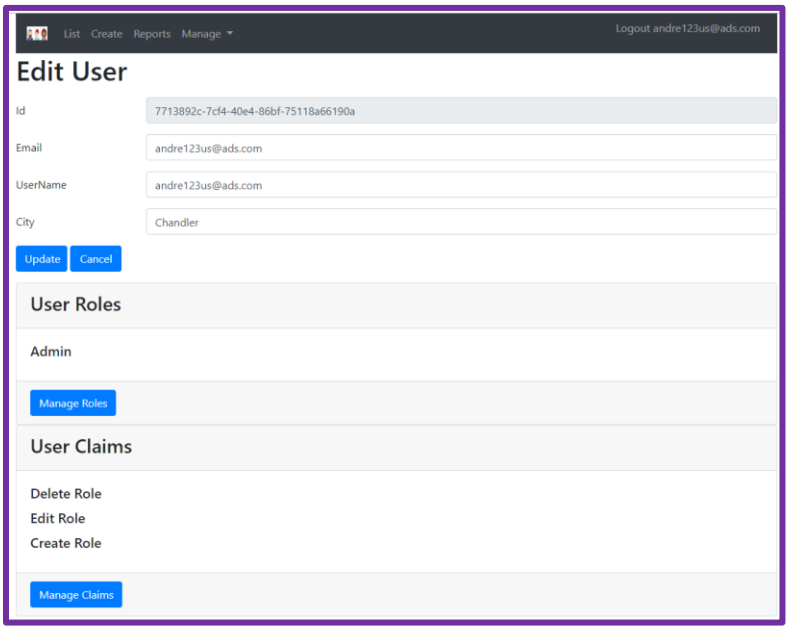


Figure 73

16.15 Select Manage Claime for this user who is already logged in and observe the following screen appears as displayed in Figure 74.

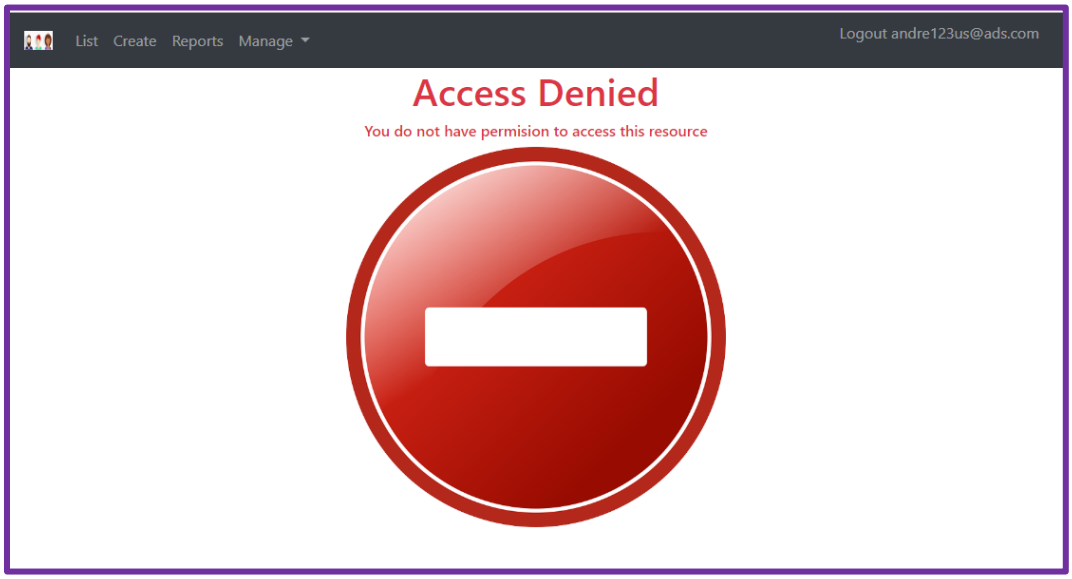


Figure 74

Git Kracken

As the project grows in complexity, it becomes important to setup a process for more than one developer to work on a given project. Git Kracken is a great tool for collaboration with a team of developers working on a given project. Git Kracken contains a dashboard for stories and tasks that a developer can work on and a checkin/checkout process for keeping track of who worked on what and when as it pertains to the developer development and implementation of features.

NOTE:

Git Kracken is free for public Github repos only. There is a small fee for a one year membership that allows for private repos in additional to public repos.

17.0 To begin, go to the GitKracken website to download the windows version of the client tool as shown in Figure 75.

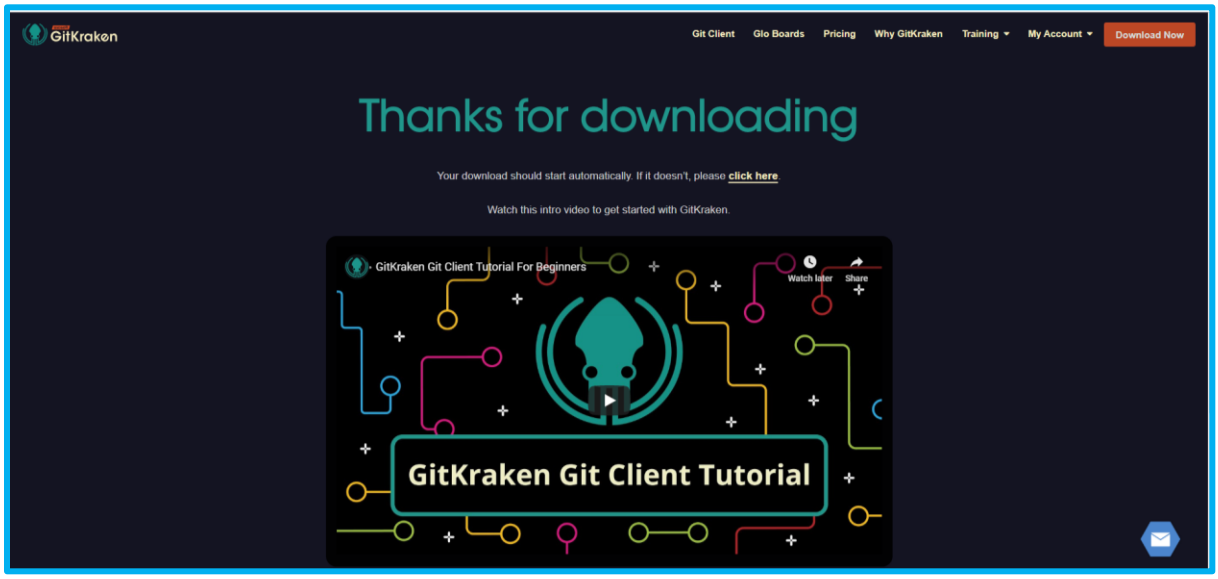


Figure 75

17.1 Run the installer and connect to the **Github** account following the onscreen instructions. This process will make Git Kracken an application User for the **Github Account** associated with the repository connected to the EmployeeManagement solution.

17.2 Once Git Kracken is accepted as an **Application User**, the repository in which the Employee Managerment project resides needs to also be linked to **Git Kracken**. Complete the same process in the prior step to achieve this. Observe the following screen appears in Figure 76.

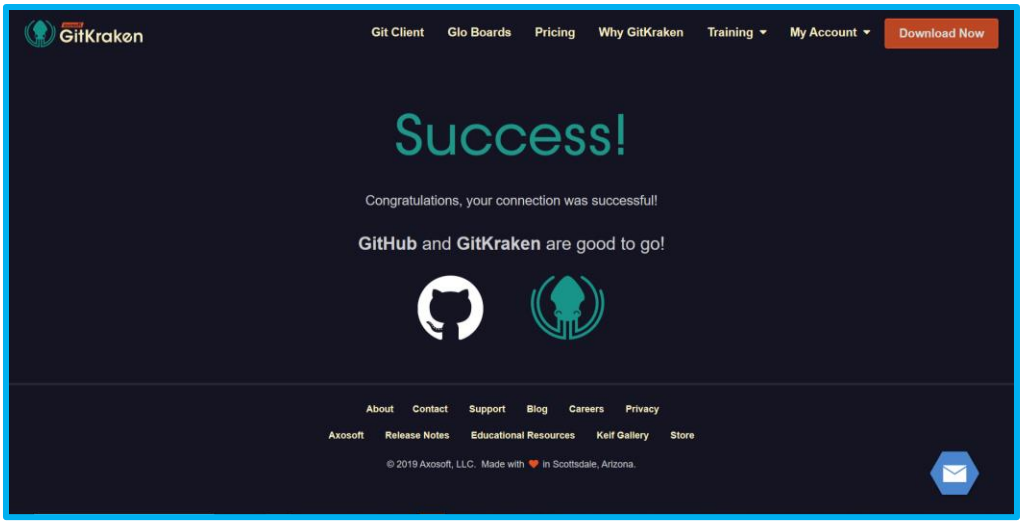


Figure 76

17.3 In the Start Menu, open GitKracken by clicking on the Squid Icon and clone the Github account for the **Employee Management Repository** as shown below. Click the Clone Repo button, this will create a local copy that will be used for development on the local workstation as see Figure 77.

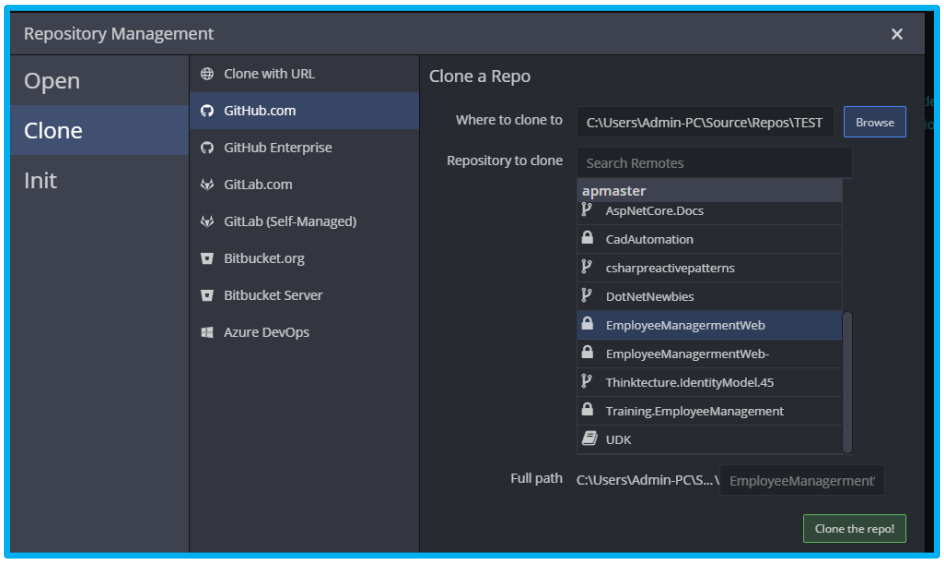


Figure 77

17.4 Observe the following screen appears with the banner indicating a successful repo clone as displayed in Figure 78.

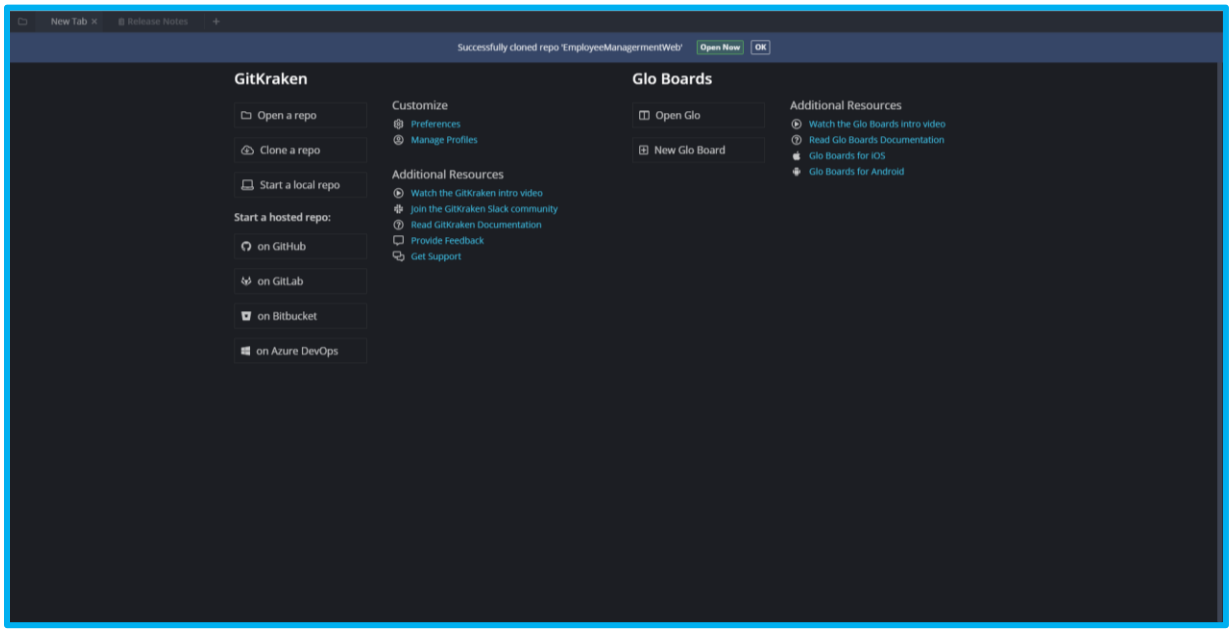


Figure 78

17.5 Click the Open Now button to access the solution. Observe the following screen appears as shown In Figure 79.

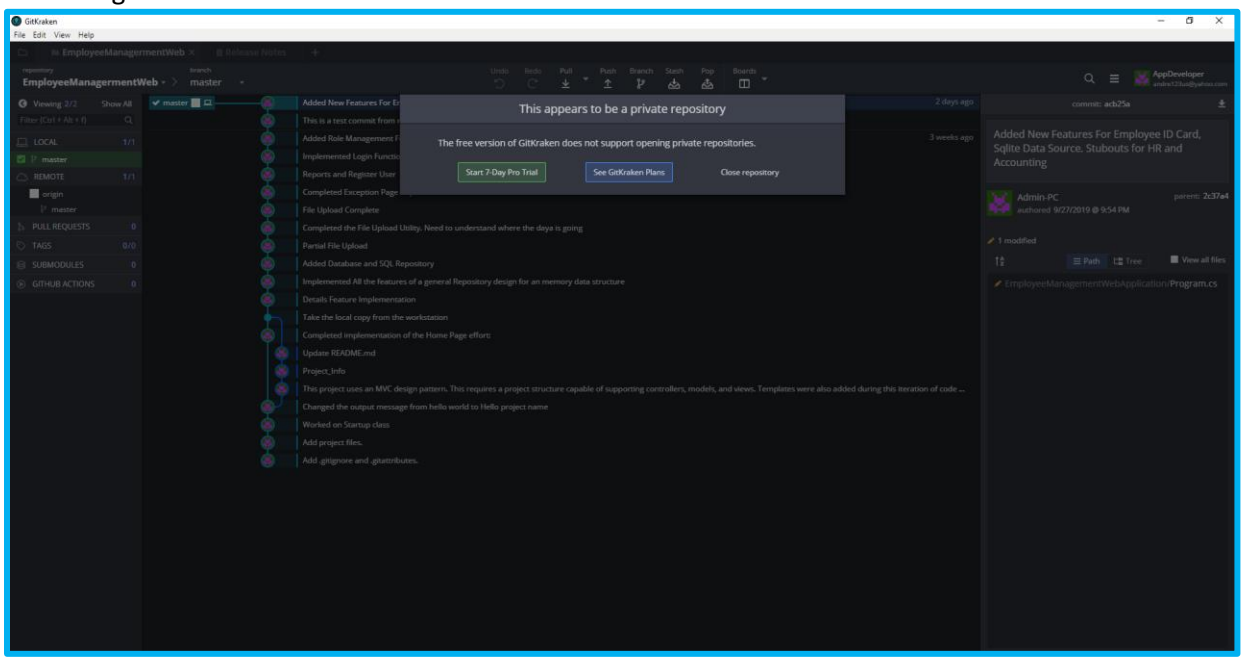


Figure 79

17.6 Since the current Repo is in the private repository category, this dialog box pops up. There are two options either move to the public repo in Github or select the 7 day trial version and pay the small fee to have access to all the features in the tool. Observe the following screen appears as shown in Figure 80.

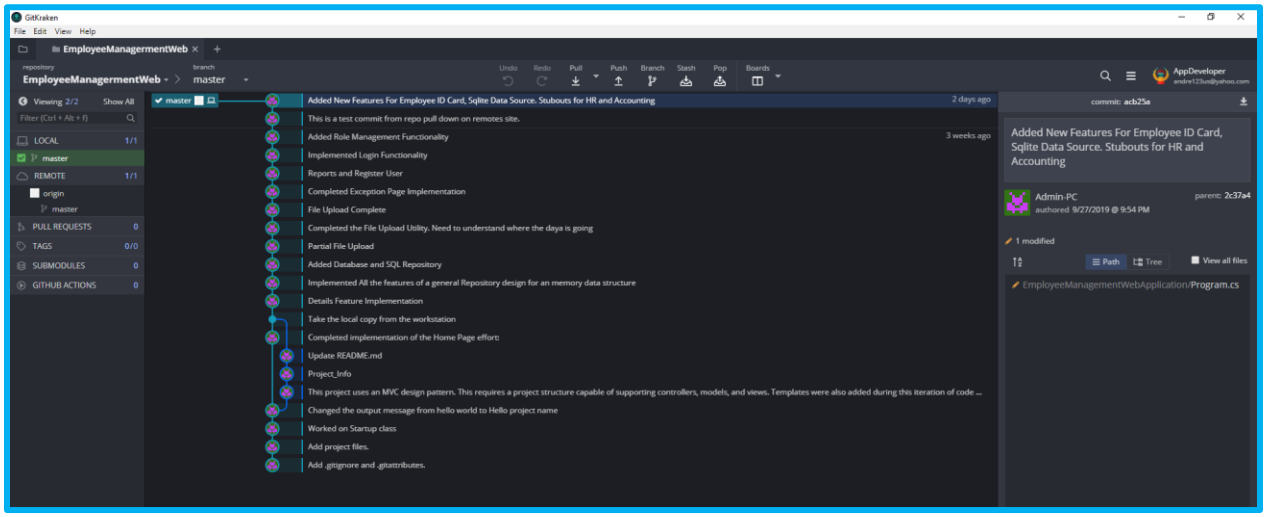


Figure 80

The workflow process for GitKracken consists of the following operations

- Work on the Code in Visual Studio
- Stage the File in GitKracken
- Commit the Action in GitKracken
- Push the Code to the Remote Repository in Github

NOTE:

Each Developer assigned to the project is represented using an Icon and the user's local workstation machine name.

When the Github remote repository and the local workstation are synced to gather they will appear on the same line under the master category on the left hand pane of the Repo Dashboard.

Since the project's inception all prior work is tracked within the master branch, a parallel branch occurs when there is a merge between branches.

If a change is made in github it will show up on top of the local workstation icon because the Github repo is ahead of the workstation local clone repo.

Likewise if a change is made within Gitkracken the local workstation will show ahead of the Github repo.

The changes mentioned in the last two examples bypass the Stage, Commit and push process. Following this workflow should keep the Remote and local repos in sync.

Now that GitKracken code repository is working lets next focus on getting the ticketing system working by installing the Glo client. Install Glo client from the GitKracken website.

In the main dashboard create a new Glo Dashboard provide the same permission to the remote github repository as was done in previous section. Add an Issue and observe the following appears as shown in Figure 81. The Story name can be called anything, for brevity a feature will included as part of the story. In this example lets builds a new enhancement called Human **Resource View Model** as shown in Figure 81 & 82.

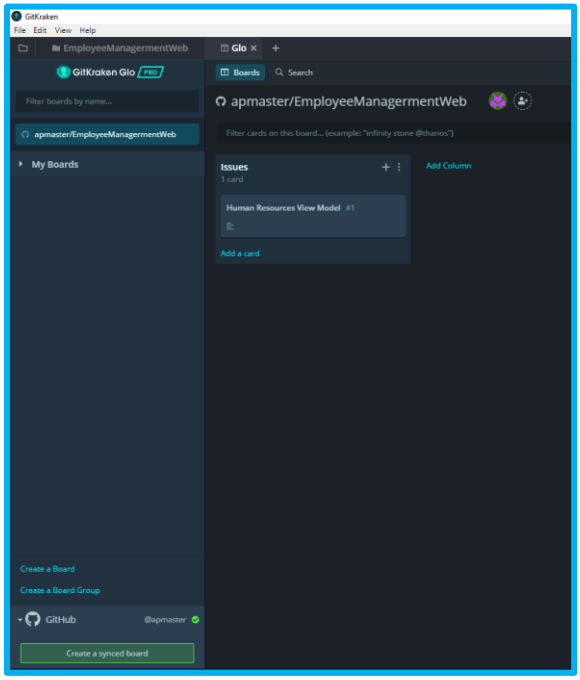


Figure 80

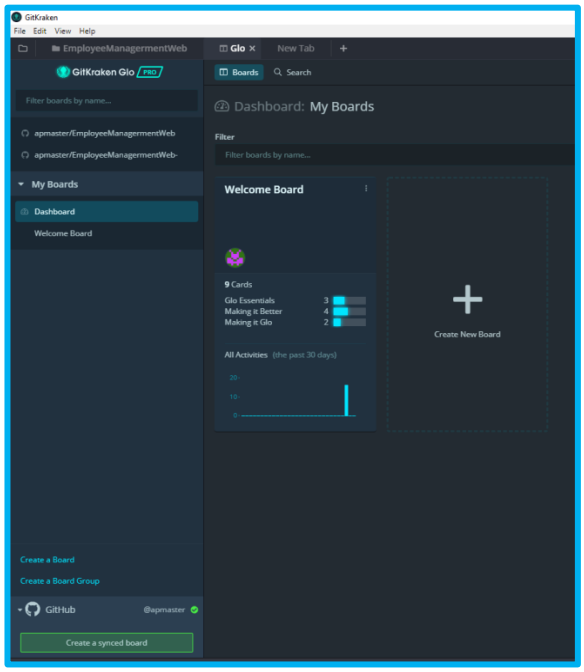


Figure 81

NOTE:

Each Story can be broken down into tasks with details defining the functionality of the feature enhancement or issue. Additional Columns can be added for either more categories or be construed as workflow process.

Each Board can be synced to a given repo by clicking the Create a synced board

There can be multiple boards for a given repo or one board per repo as shown below.

Employee Section

The goal for this feature is to restrict the activity of the **Employee User** from being able to make changes to **HR Controller** aspects of **Employee Edit** routine. For instance, it is not desirable to allow the Employee to make changes to the **Salary**, **Department**, and **Classification** attributes. However, it is desirable for the **Employee** to be able to make changes to their own **Name**, **Email** and **Photo**. Consequently it is also, advisable to allow the **Employee** to print their own **ID Card** so they may have physical access the facility.

18.0 Begin the process by writing a Story in on the Glo Board in Gitkracken as shown in Figure 82. Consider all that is required in making this work. Is there existing code in the project that will be affected by this feature? If so it needs to be included as a task item in the Story. Observe the workflow in Figure 82.

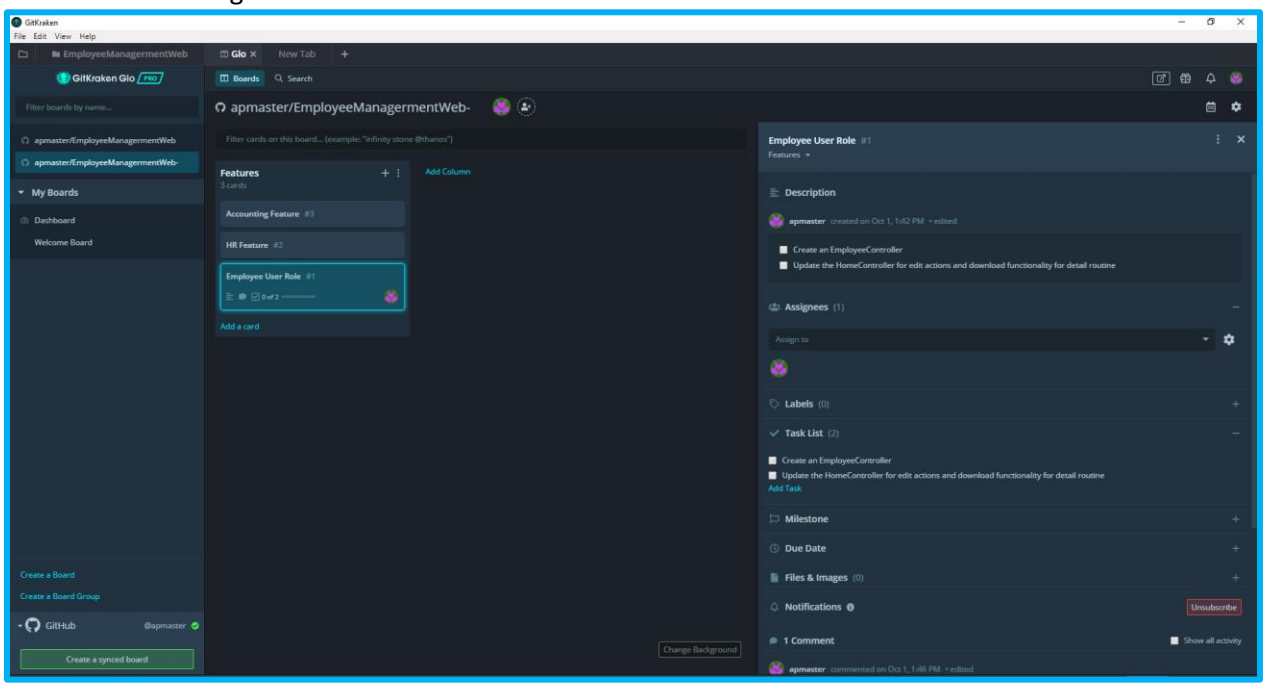


Figure 82

NOTE:

Continue Adding Tasks in the process it should become clear that refactoring is needed in order to make the application scalable. How to do that will be covered in a revised document where this same application will be developed using DDD and CQRS using ASP.Net Core 3.0.

18.1 Sign in the Employee Management system as Admin and create a role called Employee. Add an Employee named Peter@ads.com to the group.

18.2 In the **HomeController** under the **Edit** routine add the following code construction as shown below for the **Employee Role**.

NOTE:

We need to create Stub outs for Employee and Human Resource roles. And then fill in code functionality so both roles can be implemented since they are closely connected.

```
if (User.IsInRole("Employee"))
  {
   }
if (User.IsInRole("HumanResources"))
  {
  }
```

18.3 Since it is desirable to restrict **Edit** access to a given User or Human Resource role we need to place the following authorization restriction on the **Edit** routines for both **Post** and **Get** operations.

```
[Authorize(Roles = "Employee,HumanResources")]
```

18.4 Change the **Edit** routine containing **HttpPost** attribute signature; to **async** and its return type to **Task<IactionResult>** .

18.5 In the **Models Folder** add a new file called **HumanResource** add the following code construction as shown below.

```
public static class HumanResource
{
    public static string EmployeeName;
    public static string EmployeeEmail;
    public static Department.Dept? Department;
    public static Classification.type? Classification;
    public static double EmployeeSalary;
}
```

NOTE:

The reason for creating a static class with these objects is to allow for staging the original view of data for a given employee profile during the get operation of the Edit View. The purpose of having staged data is to prevent null data entering the database by the employee user during the Edit post operation. Null data that would be passed in the Edit Post routine would be derived from those features the Employee role is not allowed to change. The Employee role does not have access to making changes to the Department, Classification, and Employee Salary.

18.6 In the **HomeController** in the under the get version of the Edit routine add the following code construction to incorporate the Human **Resouces Model**.

```
Employee employee = _employeeRepository.GetEmployee(id);
HumanResource.EmployeeName = employee.Name;
HumanResource.EmployeeEmail = employee.Email;
HumanResource.Classification = employee.Classification;
HumanResource.Department = employee.Department;
HumanResource.EmployeeSalary = employee.EmployeeSalary;
```

18.7 In the **HomeController** observe the Edit Routine for get operations appears as shown below.

```
[HttpGet]
 [Authorize(Roles = "Employee,HumanResources")]
 public ViewResult Edit(int id)
 {
     Employee employee = _employeeRepository.GetEmployee(id);
     HumanResource.EmployeeName = employee.Name;
     HumanResource.EmployeeEmail = employee.Email;
     HumanResource.Classification = employee.Classification;
     HumanResource.Department = employee.Department;
     HumanResource.EmployeeSalary = employee.EmployeeSalary;

     EmployeeEditViewModel employeeEditViewModel = new EmployeeEditViewModel
     {
         Id = employee.Id,
         Name = employee.Name,
         Email = employee.Email,
         Classification = employee.Classification,
         Department = employee.Department,
         EmployeeSalary = employee.EmployeeSalary,
         ExistingPhoto = employee.PhotoPath
     };
     return View(employeeEditViewModel);
 }
```

NOTE:

Since the Name and Email properties of the Employee Model can change this needs to be reflected in the Employee User role so a Controller needs to be implemented in order to support this change.

18.8 In **GitKracken** add a new story card called **EmployeeController** keep the same assignment as in previous **Glo Dashboard** cards involving this project. Fill in the Details as shown below.

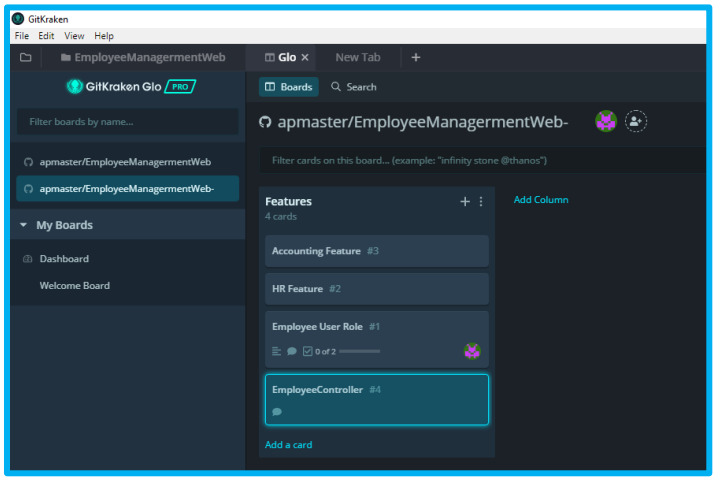


Figure 83

18.9 In the **Controller Folder** create a Controller Class called **EmployeeController** and complete the Class using the following code construction below.

```
namespace EmployeeManagementWebApplication.Controllers
{
    [Authorize(Roles ="Employee")]
    public class EmployeeController:Controller
    {

        private readonly RoleManager<IdentityRole> roleManager;
        private readonly UserManager<ApplicationUser> userManager;

        public EmployeeController(RoleManager<IdentityRole> roleManager,
                                  UserManager<ApplicationUser> userManager)
        {
            this.roleManager = roleManager;
            this.userManager = userManager;
        }
     }
}
```

18.10 In the **EmployeeController** add a **DeleteUser** routine as shown below.

```
public async Task<IActionResult> DeleteUser(string id)
{
    var user = await userManager.FindByIdAsync(id);
    if (user == null)
    {
        ViewBag.ErrorMessage = $"User with Id ={id} cannot be found";
        return View("NotFound");
    }
    else
    {
        var result = await userManager.DeleteAsync(user);
        if (result.Succeeded)
        {
            return RedirectToAction("ListUsers");
        }
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError("", error.Description);
        }

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

18.11 In the **EmployeeController** add a **ListUsers** routine using the code construction below.

```
[HttpGet]
public IActionResult ListUsers()
{
    var users = userManager.Users;
    return View(users);
}
```

18.12 In the **EmployeeController** add an **EditUser** routine for **HttpGet** using the code below.

```
[HttpGet]
public async Task<IActionResult> EditUser(string id)
{
    var user = await userManager.FindByIdAsync(id);

    if (user == null)
    {
        ViewBag.ErrorMessage = $"User with Id ={id} cannot be found";
        return View("NotFound");
    }

    var userRoles = await userManager.GetRolesAsync(user);

    var model = new EditUserViewModel
    {
        Id = user.Id,
        Email = user.Email,
        UserName = user.UserName,
        City = user.City,
        Roles = userRoles
    };

    return View(model);
}
```

18.13 In the **EmployeeController** add an **EditUser** routine for **HttpPost** using the code construction below.

```
[HttpPost]
    public async Task<IActionResult> EditUser(EditUserViewModel model)
    {
        var user = await userManager.FindByIdAsync(model.Id);

        if (user == null)
        {
            ViewBag.ErrorMessage = $"User with Id ={model.Id} cannot be found";
            return View("NotFound");
        }
        else
        {
            user.Email = model.Email;
            user.UserName = model.UserName;
            user.City = model.City;

            var result = await userManager.UpdateAsync(user);

            if (result.Succeeded)
            {
                return RedirectToAction("ListUsers");
            }

            foreach (var error in result.Errors)
            {
                ModelState.AddModelError("", error.Description);
            }

            return View(model);
        }
```

18.14 In the **EmployeeController** add a **CreateRole** routine using the code construction below.

```
    [HttpGet]
    public IActionResult CreateRole()
    {
        return View();
    }
    [HttpPost]
    public async Task<IActionResult> CreateRole(CreateRoleViewModel model)
    {
        if (ModelState.IsValid)
        {
            IdentityRole identityRole = new IdentityRole
            {
                Name = model.RoleName
            };
            IdentityResult result = await roleManager.CreateAsync(identityRole);
            if (result.Succeeded)
            {
                return RedirectToAction("ListRoles", "Administration");
            }
            foreach (IdentityError error in result.Errors)
            {
                ModelState.AddModelError("", error.Description);
            }
        }
        return View(model);
    }
```

18.15 In the **EmployeeController** add a **ListRoles** routine as shown below.

```
[HttpGet]
public IActionResult ListRoles()
{
    var roles = roleManager.Roles;
    return View(roles);
}
```

NOTE:

The Role based management system is coupled to the User and Controller for the Employee Object. That is why there are two sets of crud operations one for the Employee Controller and another for Employee Role. This is because there are two database tables in the Employee.db file.

18.16 In the **EmployeeController** add an **EditRole** routine with an **HttpGet** attribute.

```
[HttpGet]
public async Task<IActionResult> EditRole(string id)
{
    var role = await roleManager.FindByIdAsync(id);
    if (role == null)
    {
        ViewBag.ErrorMessage = $"Role with Id = {id} cannot be found";
        return View("Not Found");
    }
    var model = new EditRoleViewModel
    {
        Id = role.Id,
        RoleName = role.Name
    };

    foreach (var user in userManager.Users)
    {
        if (await userManager.IsInRoleAsync(user, role.Name))
        {
            model.Users.Add(user.UserName);
        }
    }

    return View(model);
}
```

18.17 In the **EmployeeController** add an **EditRole** routine with an **HttpPost** attribute.

```
[HttpPost]
public async Task<IActionResult> EditRole(EditRoleViewModel model)
{
    var role = await roleManager.FindByIdAsync(model.Id);
    if (role == null)
    {
        ViewBag.ErrorMessage = $"Role with Id = {model.Id} cannot be found";
        return View("Not Found");
    }
    else
    {
        role.Name = model.RoleName;
        var result = await roleManager.UpdateAsync(role);
        if (result.Succeeded)
        {
            return RedirectToAction("ListRoles");
        }
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError("", error.Description);
        }
        return View(model);
    }
}
```

18.18 In the **EmployeeController** add an **EditUserinRole** routine. This routine contains an **HttpGet** attribute.

```
[HttpGet]
public async Task<IActionResult> EditUsersinRole(string roleId)
{
    ViewBag.roleId = roleId;
    var role = await roleManager.FindByIdAsync(roleId);
    if (role == null)
    {
        ViewBag.errorMessage = $"Role with Id ={roleId} cannot be found";
    }
    var model = new List<UserRoleViewModel>();
    foreach (var user in userManager.Users)
    {
        var userRoleViewModel = new UserRoleViewModel
        {
            UserId = user.Id,
            UserName = user.UserName
        };
        if (await userManager.IsInRoleAsync(user, role.Name))
        {
            userRoleViewModel.IsSelected = true;
        }
        else
        {
            userRoleViewModel.IsSelected = false;
        }
        model.Add(userRoleViewModel);
    }
    return View(model);
}
```

18.19 In the **EmployeeController** add an **EditUserinRole** routine. This routine contains an **HttpPost** attribute.

```
[HttpPost]
public async Task<IActionResult> EditUsersinRole(List<UserRoleViewModel> model, string roleId)
{
    var role = await roleManager.FindByIdAsync(roleId);
    if (role == null)
    {
        ViewBag.ErrorMessage = $"Role with Id ={roleId} cannot be found";
        return View("NotFound");
    }
    for (int i = 0; i < model.Count; i++)
    {
        var user = await userManager.FindByIdAsync(model[i].UserId);
        IdentityResult result = null;

        if (model[i].IsSelected && !await userManager.IsInRoleAsync(user, role.Name))
        {
            result = await userManager.AddToRoleAsync(user, role.Name);
        }
        else if (!model[i].IsSelected && await userManager.IsInRoleAsync(user, role.Name))
        {
            result = await userManager.RemoveFromRoleAsync(user, role.Name);
        }
        else
        {continue;}
        if (result.Succeeded)
        {
            if (i < (model.Count - 1))
                continue;
            else
                return RedirectToAction("EditRole", new { Id = roleId });
        }
    }
    return RedirectToAction("EditRole", new { Id = roleId });
}
```

18.20 In the **HomeControler** in the http Post version of the Edit routine inside the **Is.Role("Employee")** statement condition enter the following code construction.

```
var user = await   userManager.FindByEmailAsync(HumanResource.EmployeeEmail);
user.Email = model.Email;
user.UserName = model.Name + "@ads.com";
var result = await userManager.UpdateAsync(user);

ConvertedViewModel.Id = model.Id;
ConvertedViewModel.Name = model.Name;
ConvertedViewModel.Email = model.Email;

ConvertedViewModel.Classification = HumanResource.Classification;
ConvertedViewModel.Department = HumanResource.Department;
ConvertedViewModel.EmployeeSalary = HumanResource.EmployeeSalary;
ConvertedViewModel.Photo = model.Photo;
```

18.21 In the **HomeControler** file in the http Post version of the Edit routine inside the **Is.Role("Employee")** check model to be sure its valid

```
if (ModelState.IsValid)
{}
```

18.22 In the **HomeControler** file in the http Post version of the Edit routine inside the **ModelState.IsValid** condition enter the following code construction.

```
char[] charsToTrim = { '\\' };
string targetPath = Path.Combine(FileUploaderPath, "home");
Task<bool> FileUploadSuccess;
FileUploadSuccess = UploadFile(ConvertedViewModel);

if (ConvertedViewModel.Photo != null)
{
    Employee newEmployee = new Employee
    {
        Id = ConvertedViewModel.Id,
        Name = ConvertedViewModel.Name,
        Email = ConvertedViewModel.Email,
        Department = ConvertedViewModel.Department,
        Classification = ConvertedViewModel.Classification,
        EmployeeSalary = ConvertedViewModel.EmployeeSalary,
        PhotoPath = FileUploaderPath + ConvertedViewModel.Photo.FileName
    };
    _employeeRepository.Update(newEmployee);
    return RedirectToAction("details", new { id = newEmployee.Id });
}
else
{
    Employee newEmployee = new Employee
    {
        Id = ConvertedViewModel.Id,
        Name = ConvertedViewModel.Name,
        Email = ConvertedViewModel.Email,
        Department = ConvertedViewModel.Department,
        Classification = ConvertedViewModel.Classification,
        EmployeeSalary = ConvertedViewModel.EmployeeSalary,
        PhotoPath = FilePathNoImage + "No_Picture_Person.png"
    };
    _employeeRepository.Update(newEmployee);
    return RedirectToAction("details", new { id = newEmployee.Id });

}
```

18.23 In the AppDbContext class add the following line for the Employee as a database set.

```
public DbSet<Employee> Employees { get; set; }
```

UAT Testing

18.24 Begin Testing by running the Application by pressing F5 observe the following appears as shown In Figure 84

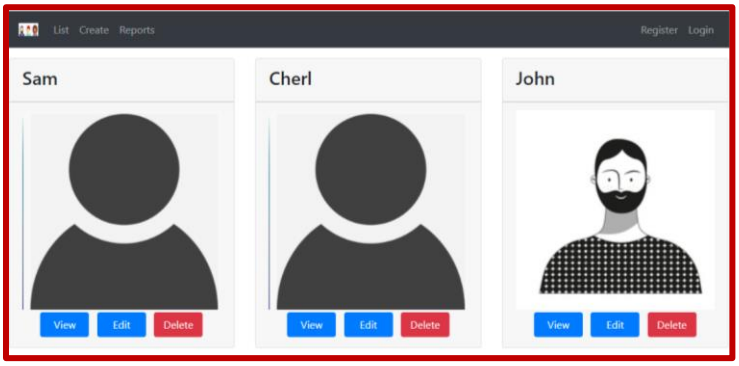


Figure 84

18.25 **Login** as **Administrator** and select **Role Management** to add a **User** to the **Employee Role**. Make note of the Employee that was added. This **Employee** will serve as our **Test User** for the **Employee Role**. Observe the Login User is displayed in Figure 85.

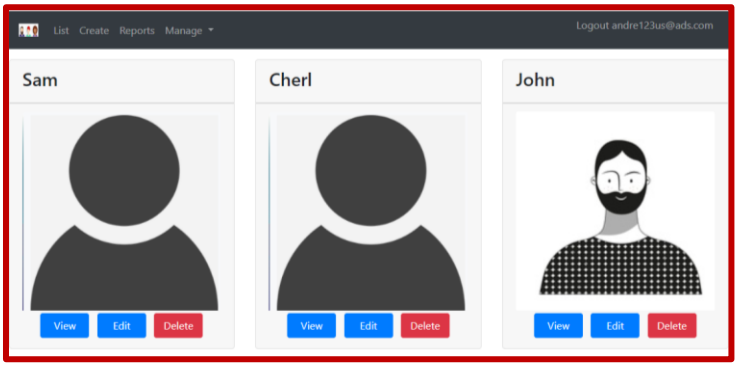


Figure 85

18.26 **Logout** as **Administrator** and **Login** using our **Test User** who is a member of the **Employee Role**. Observe the following appears as shown in Figure 86.

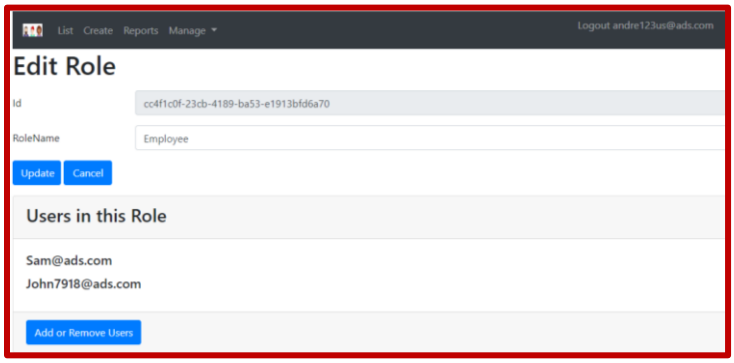


Figure 86

18.27 In the **List View** navigate to the **Test User** profile and click Edit. In this case lets select Sam. Logout as administrator and signin as Sam observe the following appears as shown in Figure 87.

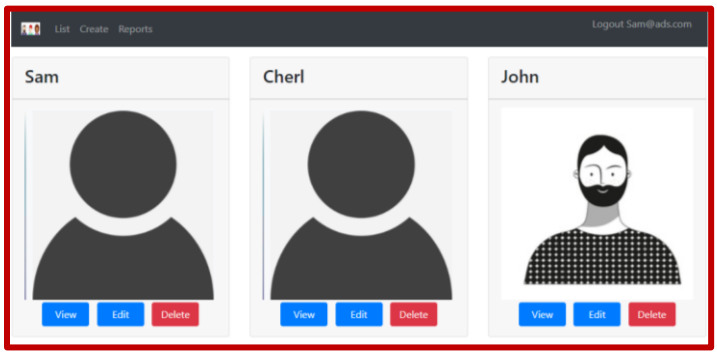


Figure 87

18.28 Change the **Name, Email** and **Department** of the **Test User** click **Update** to make the changes using the form in Figure 88.

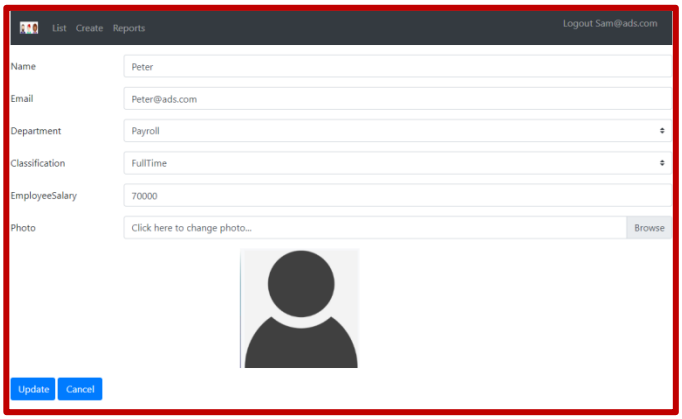


Figure 88

18.29 Observe the **Details View** has changed but the Username in the Logout field has not changed and the Department category has not changed to Payroll as shown in Figure 89

Figure 89

18.30 **Logout** of the application and log back in using the new **TestUser** name and email as shown in Figure 90

Figure 90

18.31 Observe the **User** is able to login Successfully as a member of the **Employee Role**. Observe the following appears in Figure 91

18.32 The Employee Role can be verified based on the restrictions and freedoms for this role but to be sure logout and log back into the application as Administrator.

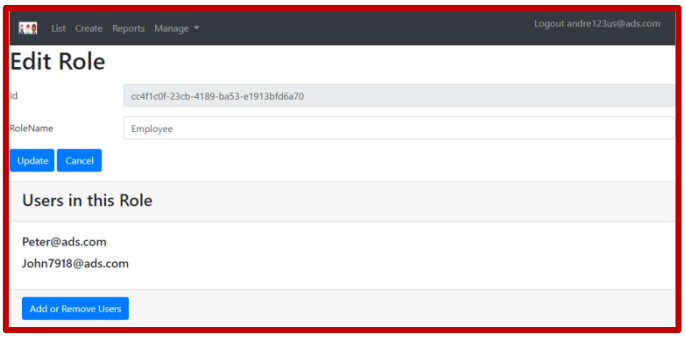


Figure 91

November 2, 2019
Prepared by Andre Masters
Revision 1.0

18.33 As a Administrator Role check to see if the Test User is a member of the Employee Role.

18.34 In **GitKracken** complete the standard workflow process of **Stage Comit** and **Push**. In this example we will be pushing up an updated database. Drill Down to the Database Stage Request screen as shown in Figure 92

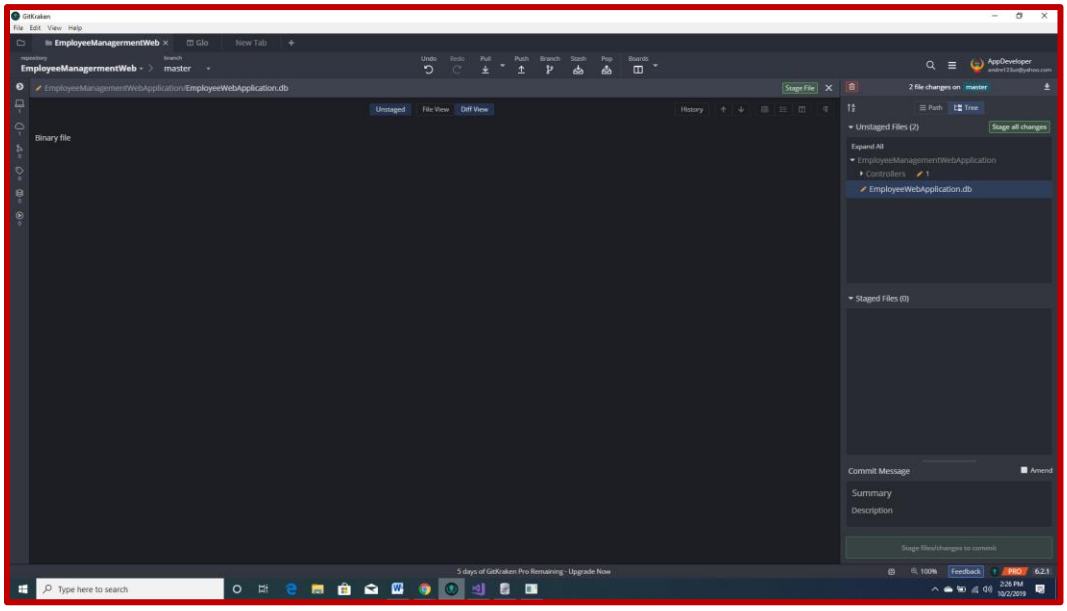


Figure 92

18.35 Hover the mouse by the **EmployeeWebApplicationDB** file to turn on the Stage File button. Click the button to stage the file. Observe the following appears in Figure 93.

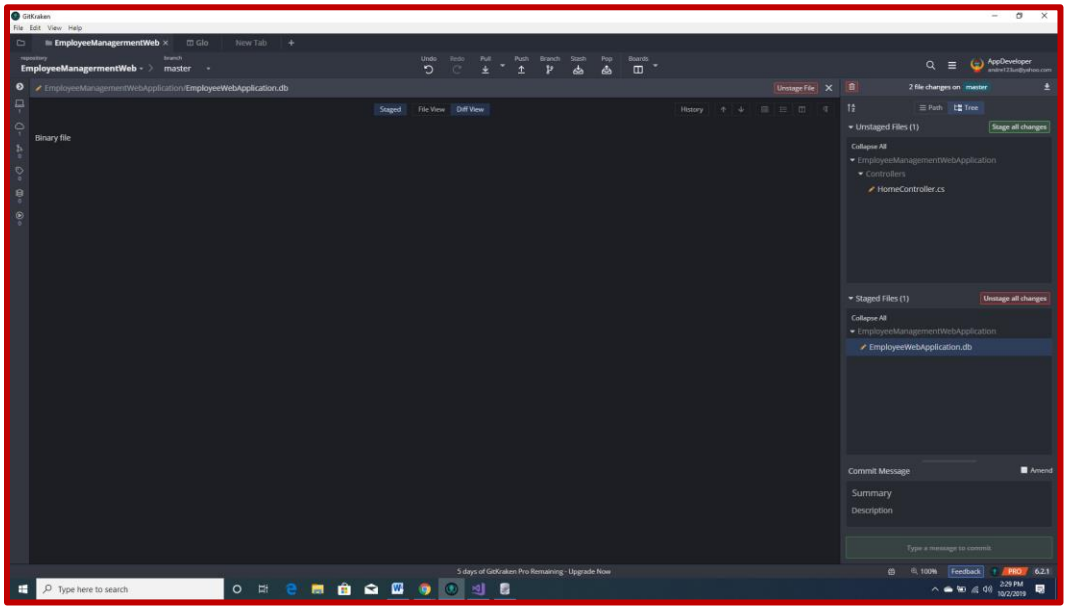


Figure 93

18.36 In the Summary Section enter **Completed UAT Testing** and in the description section enter the following comment:

“Tested the Employee Role using an existing User who happens to be a member of the Employee Role. Verified the Employee Data Name, Email, and Photo were changed while Leaving the remaining editbable fields untouched since the Employee Role does not have access these fields. All tests passed UAT”

18.37 Click Commit Changes to File button as shown in Figure 94.

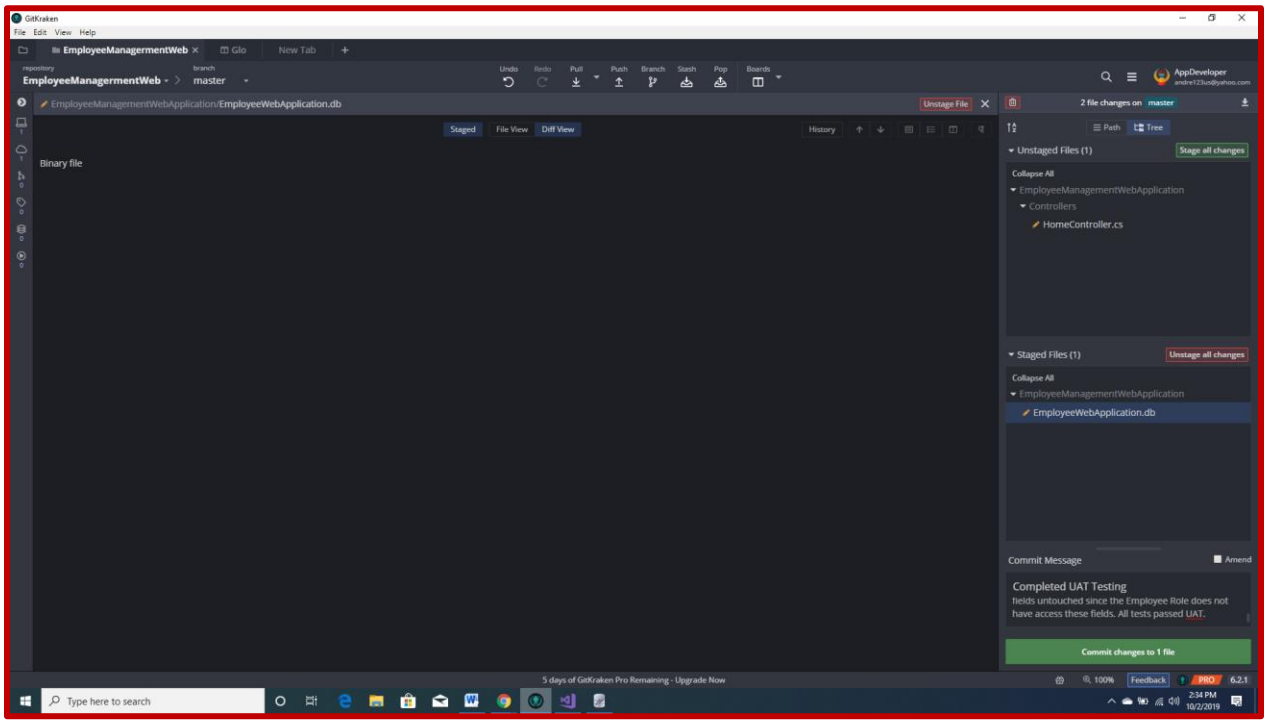


Figure 94

18.38 Observe the following screen appears in Figure 95 indicating WIP and an out of alignment between the local workstation and the remote repository.

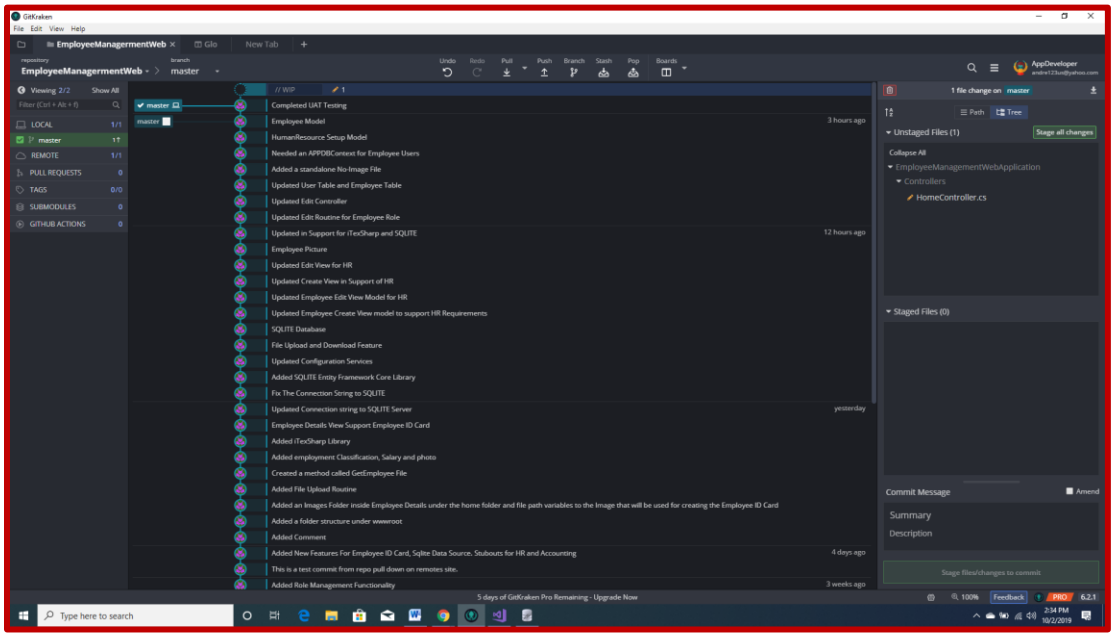


Figure 95

18.39 Push the changes to the Remot repository and observe the following appears in Figure 96.

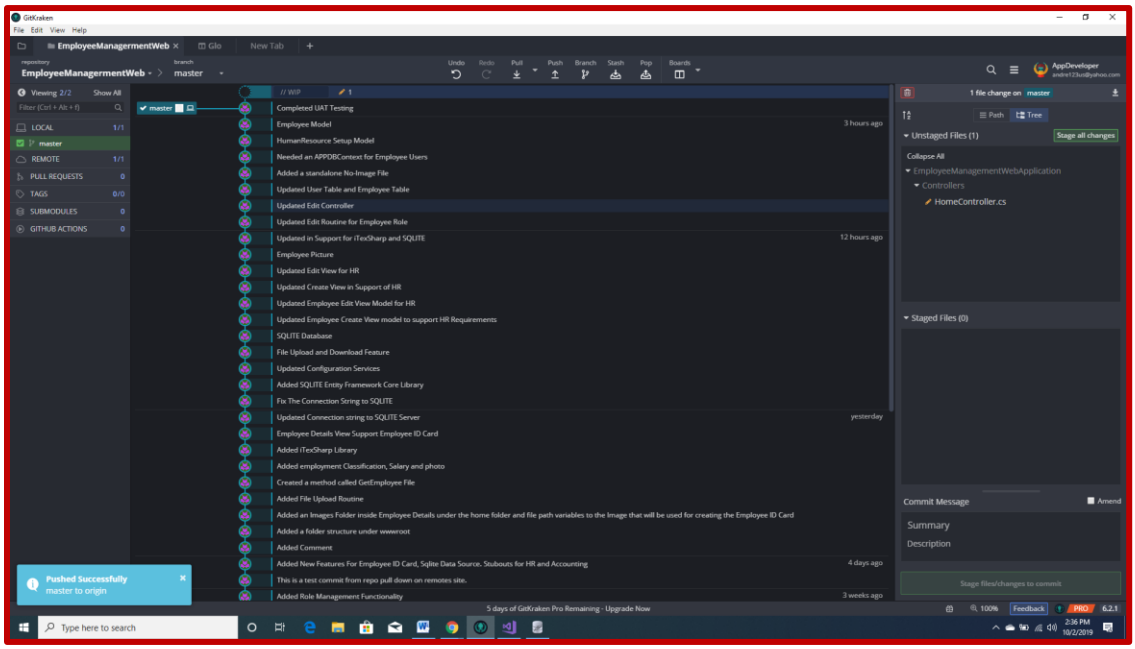


Figure 96

18.40 Next Stage the HomeController by selecting in the righthand pane in the screen shot of Figure 96. Observe the following occurs as shown in Figure 97. Push the changes to the Remot repository and observe the following appears as shown in Figure 97.

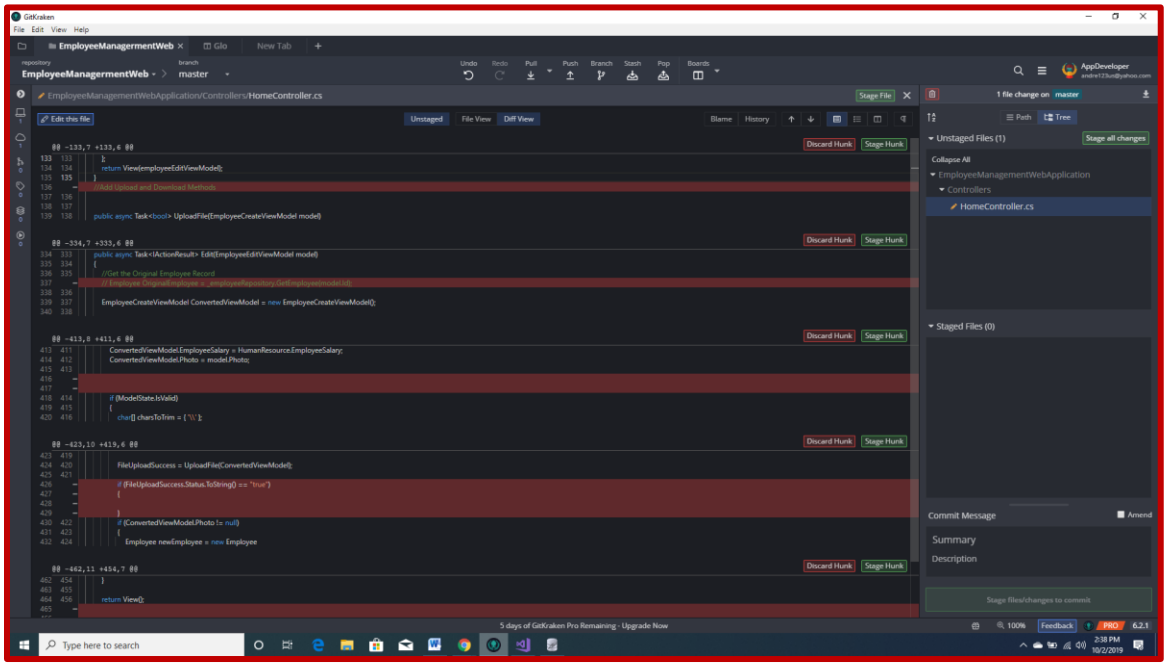


Figure 97

18.41 Since the solution passed UAT testing it is desirable to accept all changes so in this case we want to stage all changes. Add appropriate comments for the staging effort as shown in Figure 98.

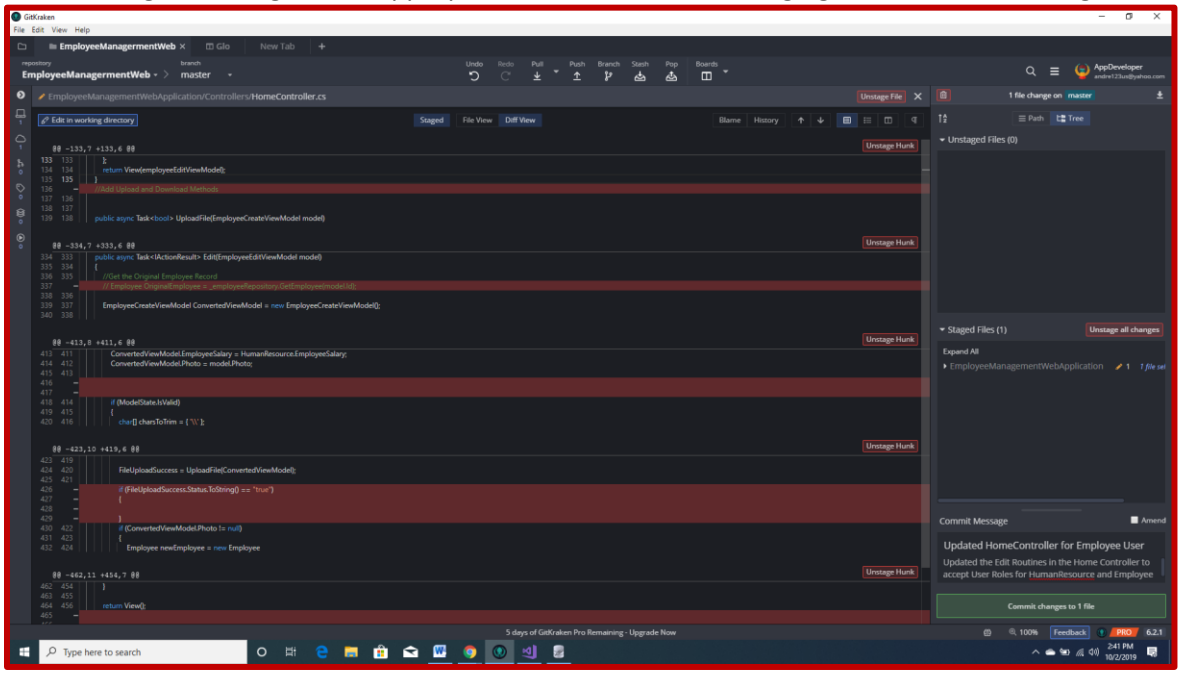


Figure 98

18.42 Click Commit changes for each feature and observe the following appears as shown in Figure 99.

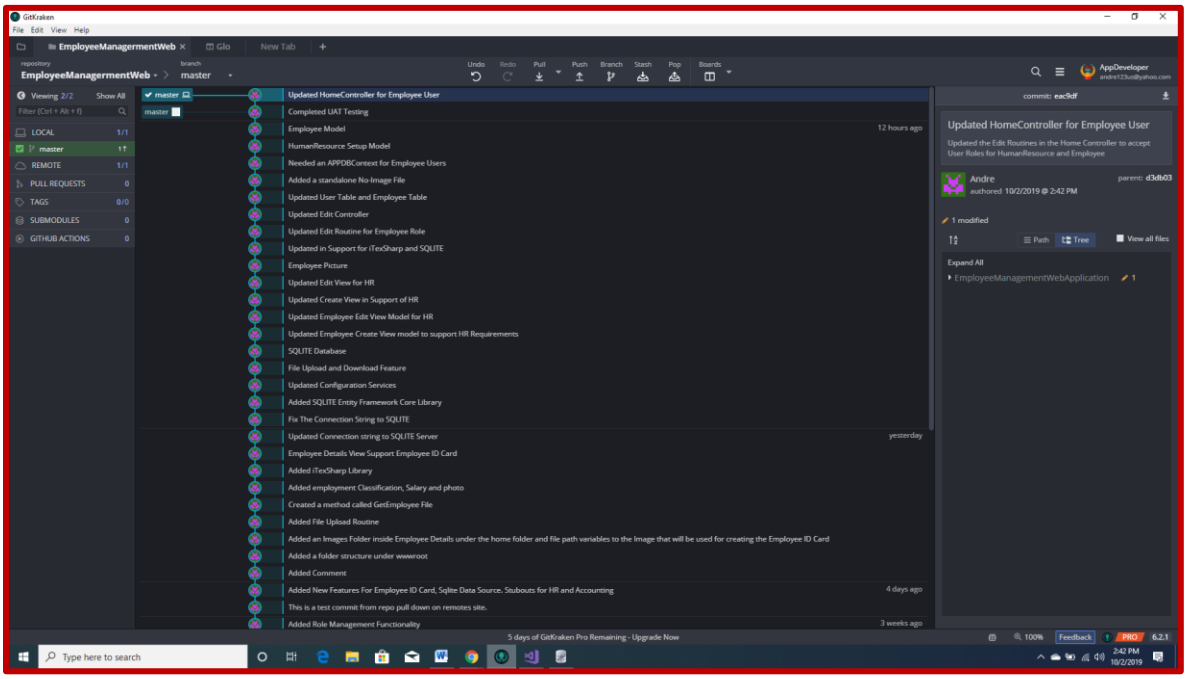


Figure 99

18.43 Complete the process by pushing the changes to the remot GitHub repository and observe the following occurs as shown below. Observe the remote and local repos are synchronized as shown in Figure 100.

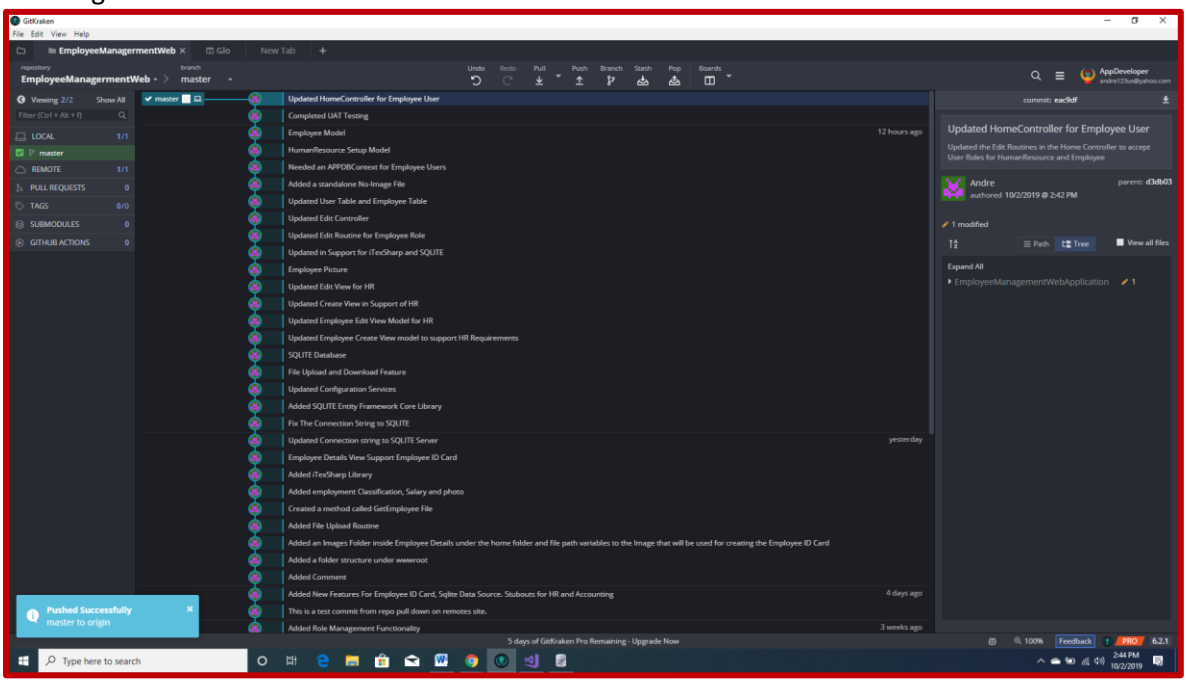


Figure 100

November 2, 2019
Prepared by Andre Masters
Revision 1.0

18.44 Login to the Github repo and see if indeed the project has been updated with the new changes taking affect as shown in Figure 101

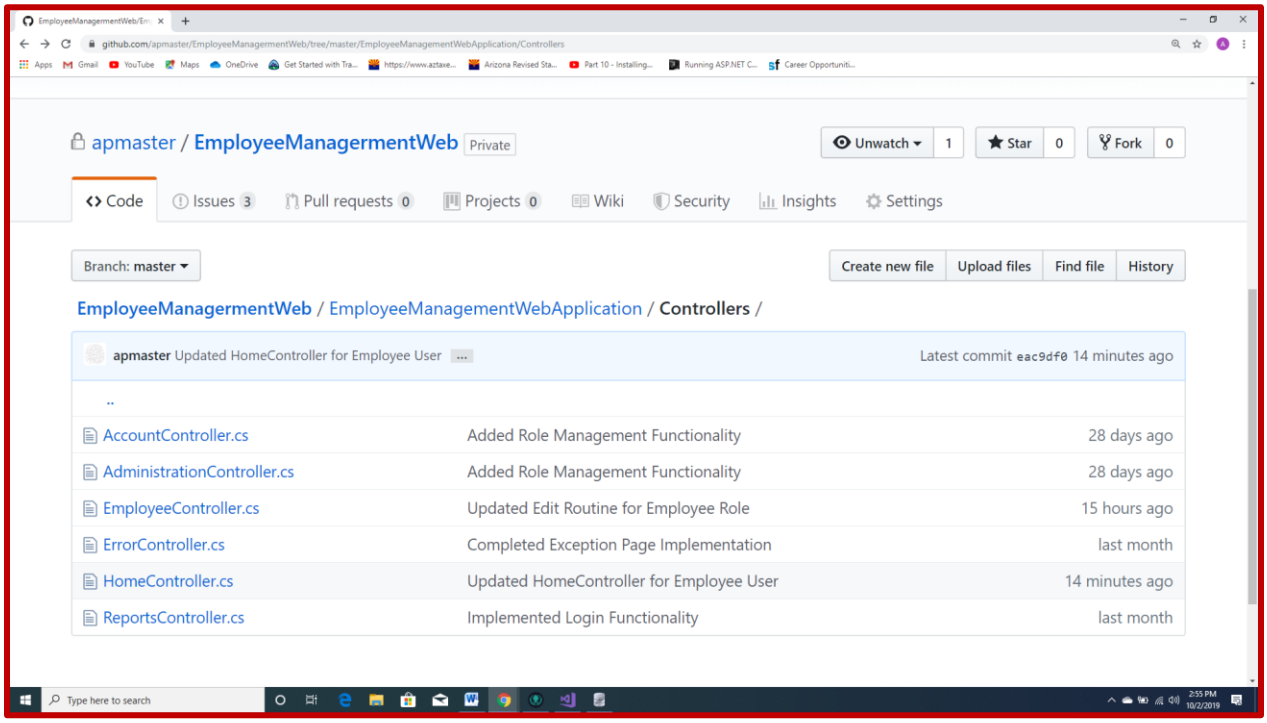


Figure 101

Human Resource Section

Once the RBAC section is completed, create an **HR User Group** that will have access to the **Reports Data-grid in section 11**.

This section will encompass a view containing more than one model. This will be accomplished using a Viewmodel. The Viewmodel will contain a **Employee Model** and an **HR Model** making the parent **ViewModel** a composite of **Employee** and **HR**.

19.0 Create a Story in **GitKracken** called **HR Feature** which contains a task called **Human Resource View Model** with a description as shown in Figure 102

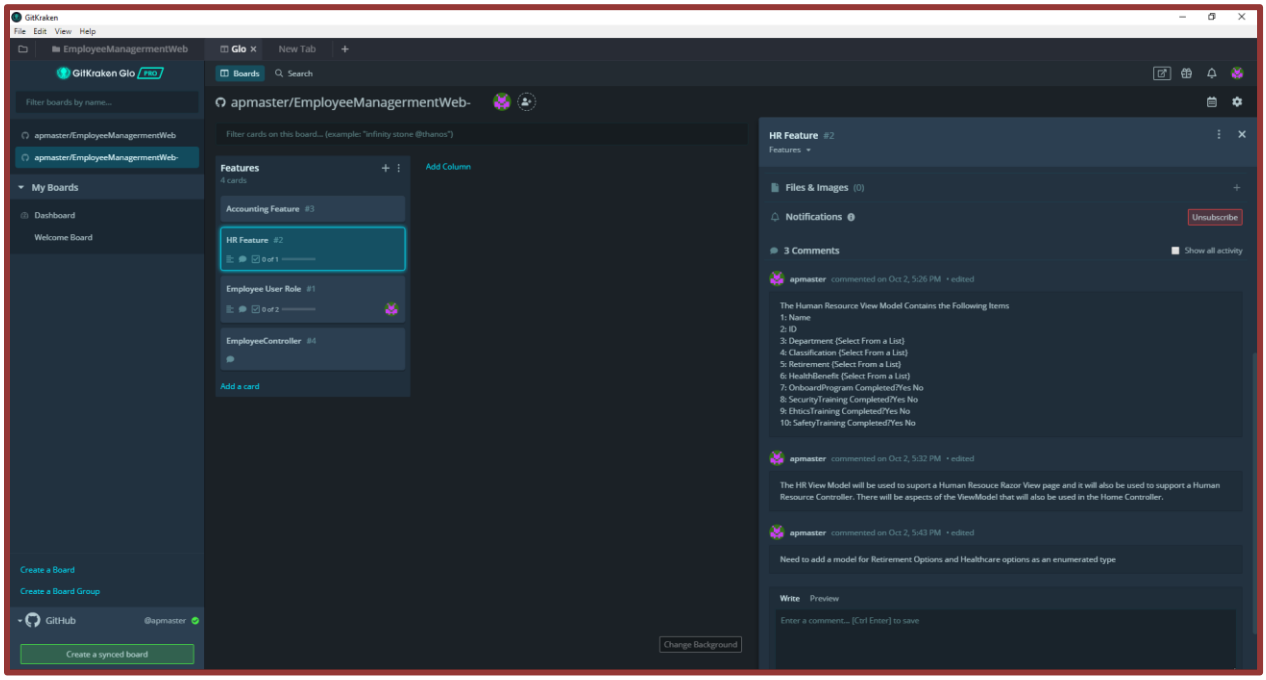


Figure 102

19.1 Create a Human Resource View Model object using the characteristics as shown below. Also Create a Supporting Model for Retirement and HealthBenefit.

```
namespace EmployeeManagementWebApplication.ViewModels
{
    public class HumanResourcesViewModel
    {
        //Employee
        //HumanResource
    }
}
```

19.2 Create a Human Resource Model object using the characteristics as shown below.

```
namespace EmployeeManagementWebApplication.Models
{
    public class HumanResource
    {
        public Retirement.Plan? RetirementPlan { get; set; }
        public HealthCare.Plan? HealthPlan { get; set; }
        public bool OnboardProgramComplete { get; set; }
        public bool SecurityTrainingComplete { get; set; }
        public bool EthicsTrainingComplete { get; set; }
        public bool SafetyTrainingComplete { get; set; }
    }
}
```

19.3 Create a Retirement Model with an enumerated type for plan categories as shown below.

```
namespace EmployeeManagementWebApplication.Models
{
    public class RetirementModel
    {
        public enum Plan
        {
            LowRisk,
            MediumRisk,
            HighRisk
        }
    }
}
```

19.4 Create a Healthcare Model with an enumerated type for plan categories as shown below.

```
namespace EmployeeManagementWebApplication.Models
{
    public class HealthCareModel
    {
       public enum Plan
        {
            HealthSavings,
            PPO_Plan,
            StandardPlan
        }
    }
}
```

19.5 Complete the details of the HumanResourceViewModel since the underlying models are completed.

```
namespace EmployeeManagementWebApplication.ViewModels
{
    public class HumanResourcesViewModel
    {
        public IEnumerable<Employee> ActiveEmployee {get;set;}
        public IEnumerable<HumanResource> EmployeeRequirements { get; set; }
    }
}
```

19.6 In the **AppDBContext** file add the following code construction whose aim is to create a human resource collection as shown in the redbox below.

```
namespace EmployeeManagementWebApplication.Models
{
    public class AppDbContext:IdentityDbContext<ApplicationUser>
    {
        public AppDbContext(DbContextOptions<AppDbContext> options): base(options)
        {

        }

        public DbSet<Employee> Employees { get; set; }
        public DbSet<ApplicationUser> AppUsers { get; set; }
        public DbSet<HumanResource> HRDepartment { get; set; }
    }
}
```

19.7 Add a migration called **HumanResourceFeature** table in the **Package Manager Console**.

Add-migration HumanResourceFeature

19.8 Observe the following appears as shown below for the **Up Condition**

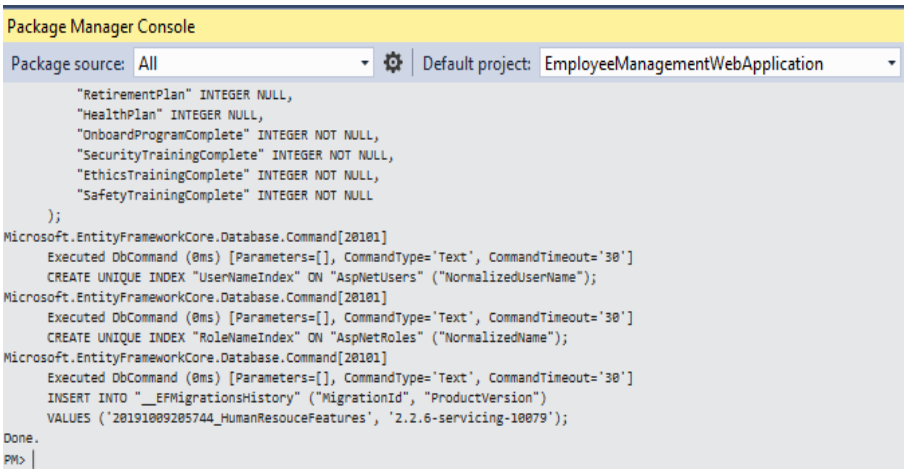
```
using Microsoft.EntityFrameworkCore.Migrations;

namespace EmployeeManagementWebApplication.Migrations
{
    public partial class HumanResouceFeatures : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropIndex(
                name: "UserNameIndex",
                table: "AspNetUsers");

            migrationBuilder.DropIndex(
                name: "RoleNameIndex",
                table: "AspNetRoles");

            migrationBuilder.CreateTable(
                name: "HRDepartment",
                columns: table => new
                {
                    Id = table.Column<int>(nullable: false)
                        .Annotation("Sqlite:Autoincrement", true),
                    RetirementPlan = table.Column<int>(nullable: true),
                    HealthPlan = table.Column<int>(nullable: true),
                    OnboardProgramComplete = table.Column<bool>(nullable: false),
                    SecurityTrainingComplete = table.Column<bool>(nullable: false),
                    EthicsTrainingComplete = table.Column<bool>(nullable: false),
                    SafetyTrainingComplete = table.Column<bool>(nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_HRDepartment", x => x.Id);
                });

            migrationBuilder.CreateIndex(
                name: "UserNameIndex",
                table: "AspNetUsers",
                column: "NormalizedUserName",
                unique: true);

            migrationBuilder.CreateIndex(
                name: "RoleNameIndex",
                table: "AspNetRoles",
                column: "NormalizedName",
                unique: true);
        }
```

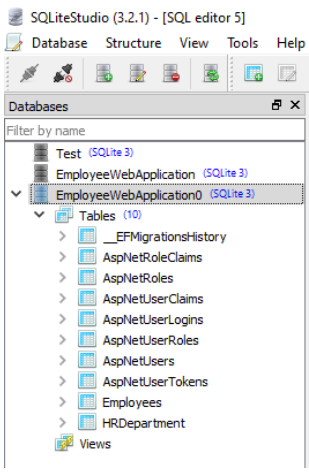
19.9 Observe the following appears as shown below for the **Down Condition**

```
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "HRDepartment");

            migrationBuilder.DropIndex(
                name: "UserNameIndex",
                table: "AspNetUsers");

            migrationBuilder.DropIndex(
                name: "RoleNameIndex",
                table: "AspNetRoles");

            migrationBuilder.CreateIndex(
                name: "UserNameIndex",
                table: "AspNetUsers",
                column: "NormalizedUserName",
                unique: true,
                filter: "[NormalizedUserName] IS NOT NULL");

            migrationBuilder.CreateIndex(
                name: "RoleNameIndex",
                table: "AspNetRoles",
                column: "NormalizedName",
                unique: true,
                filter: "[NormalizedName] IS NOT NULL");
        }
    }
}
```

19.10 Check the Migration code to verify the SQL statements are valid make the appropriate corrections prior to executing the update database command.

Update-database





NOTE:

There is a serious limitation using Migration feature of the EntityFramework Core in that if there is a complex SQL data structure or if the database technology does not support a minimum feature subset in MSSQL the process fails. For the sake of this tutorial, all of the tables in the current database were dropped. With a clean database, the Update – Database command will work at the cost of losing all of the existing data, if the database technology does not support the minimum feature set from MSSQL

November 2, 2019
Prepared by Andre Masters
Revision 1.0

19.11 Since we no longer have any data since the database is new, comment the following code in the **_Layout.cshtml** file.

```
@*@if (signInManager.IsSignedIn(User) && User.IsInRole("Admin"))
 {*@
     <li class="nav-item dropdown">
         <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownMenuLink"
            data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
             Manage
         </a>
         <div class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink">
             <a class="dropdown-item" asp-controller="Administration"
                asp-action="ListUsers">Users</a>
             <a class="dropdown-item" asp-controller="Administration"
                asp-action="ListRoles">Roles</a>
         </div>
     </li>
 @*}*@
```

19.12 In the **AdministrationController** file comment the following code construction. This will allow for the creation of an **Admin** User.

```
namespace EmployeeManagementWebApplication.Controllers
{

   // [Authorize(Roles = "Admin")]
    public class AdministrationController: Controller
    {
    }
}
```

19.13 Create an **Admin Role** and **Admin Account Credential** keep the account details in a secure storage location for future use.

19.14 Create an **Employee Role** and **Employee Account Credential** keep the account details in a secure storage location for future use.

19.15 Create a **Human Resource Role** and **Human Resource Account** credential keep the **Account Details** in a secure storage location for future use.

19.16 Compile the solution and verify there are no errors. This completes the HR Feature tasks for **HumanResourcesViewModel**, **HealthCareModel** and **RetirementModels**.

19.17 Now that we have a **Human Resource Table**, a SQL Human Resource Repository is needed in for CRUD operations. In the Models folder create a new File call **SQLHumanResourceRepository**.

```
namespace EmployeeManagementWebApplication.Models
{
    public class SQLEmployeeRepository : IEmployeeRepository
    {
        private readonly AppDbContext context;

        public SQLEmployeeRepository(AppDbContext context)
        {
            this.context = context;
        }
        public Employee Add(Employee employee)
        {

            context.Employees.Add(employee);
            context.SaveChanges();
            return employee;
        }

        public Employee Delete(int id)
        {
           Employee employee = context.Employees.Find(id);
            if(employee !=null)
            {
                context.Employees.Remove(employee);
                context.SaveChanges();
            }

            return employee;
        }

        public IEnumerable<Employee> GetAllEmployees()
        {
            return context.Employees;
        }
        public Employee GetEmployee(int id)
        {
            return context.Employees.Find(id);
        }

        public Employee Update(Employee employeeChanges)
        {
            var employee = context.Employees.Attach(employeeChanges);
            employee.State =
Microsoft.EntityFrameworkCore.EntityState.Modified;
            context.SaveChanges();

            return employeeChanges;
        }
    }
}
```

19.18 Create an **Interface** in the **Models** folder for the **Dependancy Injector** to be able to invert program control from the HumanResource controller to the Employee SQL repository.

```
namespace EmployeeManagementWebApplication.Models
{
    public interface IEmployeeRepository
    {
        Employee GetEmployee(int id);
        IEnumerable<Employee> GetAllEmployees();
        Employee Add(Employee employee);
        Employee Update(Employee employeeChanges);
        Employee Delete(int id);
    }
}
```

19.19 In the **Models** folder create a **SQLHumanResourceRepository** file using the following code construction below.

```
namespace EmployeeManagementWebApplication.Models
{
    public class SQLHumanResourceRepository : IHumanResourcesRepository
    {
        private readonly AppDbContext context;

        public SQLHumanResourceRepository(AppDbContext context)
        {
            this.context = context;
        }

        public IEnumerable<HumanResource> GetAllHREmployeeData()
        {
            return context.HRDepartment;
        }

        public HumanResource Update(HumanResource HRChanges)
        {
            var employee = context.HRDepartment.Attach(HRChanges);
            employee.State = Microsoft.EntityFrameworkCore.EntityState.Modified;
            context.SaveChanges();
            return HRChanges;
        }
    }
}
```

19.20 In the **Models** folder create an Interface in the Models folder using the code construction below.

```
    public interface IHumanResourcesRepository
    {
        IEnumerable<HumanResource> GetAllHREmployeeData();
        HumanResource Update(HumanResource HRChanges);
    }
```

19.21 In **GitKracken** create another task inside the **HR Feature** called **HumanResourcesView** enter the following information for the view details.

- ID Textfield
- Name Textfield
- Department Dropdown
- Classification Dropdown
- Retirement Dropdown
- HealthCare Dropdown
- OnboardProgramComplete Checkbox
- SecurityTrainingComplete Checkbox
- EthicsTrainingCompleteCheckbox
- SafetyTrainingCompleteCheckbox

19.22 In the **Views Folder** create a new folder for **Human Resources** feature.

19.23 Inside the **Views Folder** create another folder for **HumanReosurces** and create a single HumanResource dashboard as a razor view

```
@model HumanResourcesViewModel

@{
    ViewBag.Title = "Human Resources";
}
```

19.24 In the **_Layout View** make the following changes to the view to accommodate the **Human Resource View.**

```
 @if(signInManager.IsSignedIn(User) && User.IsInRole("HumanResources"))
  {
   <li class="nav-item">
       <a asp-action="HRDashboard" asp-controller="HumanResource" class="nav-link">Human Resources</a>
   </li>
  }
```

19.25 In the Controllers Folder create a HumanResourceController controller using the code construction below.

```
namespace EmployeeManagementWebApplication.Controllers
{
    public class HumanResourceController : Controller
    {
        public IActionResult HRDashboard()
        {
            return View();
        }
    }
}
```

19.26 In the Controllers Folder create a **HumanResourceController** controller using the code construction below.

```
namespace EmployeeManagementWebApplication.Controllers
{
    public class HumanResourceController : Controller
    {
        private IHumanResourcesRepository _HRemployeeRepository;
        private IEmployeeRepository _Employee;

        public HumanResourceController(IHumanResourcesRepository HRemployeeRepository, IEmployeeRepository Employee)
        {
            _HRemployeeRepository = HRemployeeRepository;
            _Employee = Employee;
        }

        [HttpGet]
        public IActionResult HRDashboard()
        {
            HumanResourcesViewModel EmployeeHRequirements = new HumanResourcesViewModel();

            var HR_Requirements = _HRemployeeRepository.GetAllHREmployeeData();
            var Employee = _Employee.GetAllEmployees();

            EmployeeHRequirements.EmployeeRequirements = HR_Requirements;
            EmployeeHRequirements.ActiveEmployee = Employee;

            return View(EmployeeHRequirements);
        }
    }
}
```

NOTE:

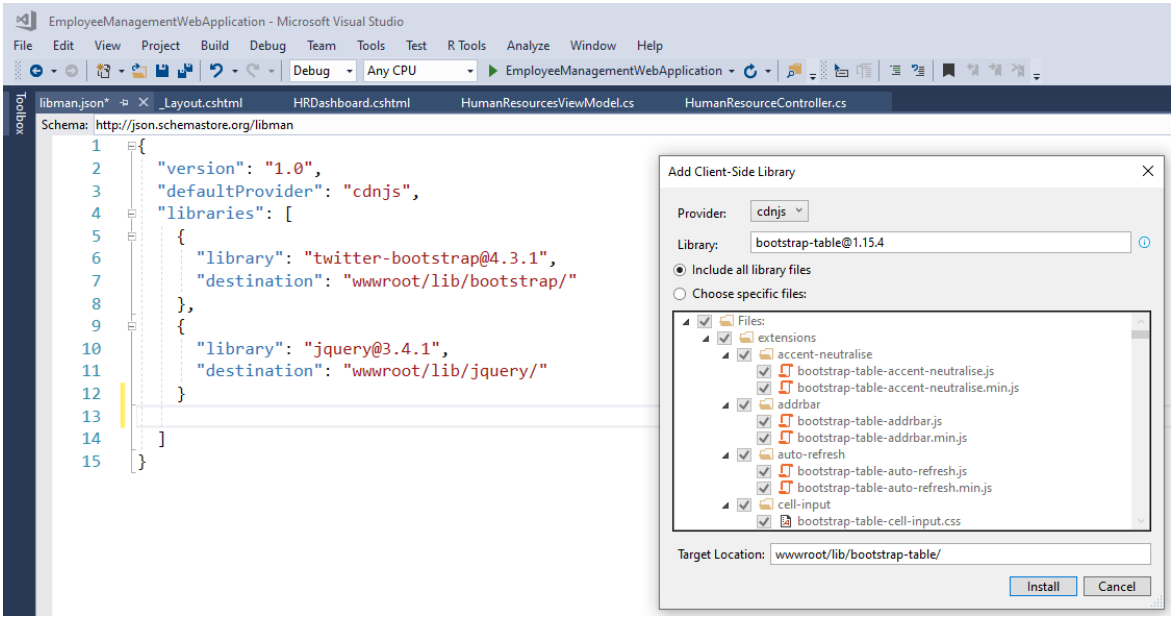
Since HumanResourcesViewModel is a composite view cosnsisting of two other models one for employee and another for HumanResource Requirements per employee. These two models will be displayed next two each other synchronized by a primary and foreign key relationship.

19.27 In the **StartUp.cs** file under **ConfigureServices** routine, make the following changes so the dependency injection service can invert control to the **SQLHumanResourceRepository** routines.

```
public void ConfigureServices(IServiceCollection services)
{
    //Section I Database Configurations
    var DevelopmentDatabase = _config.GetConnectionString("EmployeeDBConnection");
    var ProductionDatabase = "Filename=employees.db";

    services.AddDbContextPool<AppDbContext>(
             options => options.UseSqlite(DevelopmentDatabase));

    //Section II XML Configurations
    services.AddMvc().AddXmlSerializerFormatters();

    //Section IV Repository Section
    services.AddScoped<IEmployeeRepository, SQLEmployeeRepository>();
    services.AddScoped<IHumanResourcesRepository, SQLHumanResourceRepository>();

    //Section V Security Section
    services.AddIdentity<ApplicationUser, IdentityRole>()
            .AddEntityFrameworkStores<AppDbContext>();
    services.AddMvc(options =>
    {
        var policy = new AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();
        options.Filters.Add(new AuthorizeFilter(policy));
    });
}
```

19.28 In keeping the application responsive while and functional for the user, it is desirable to add a **Bootstrap Table Library**. Update the existing library as shown below.

ProjectName>Add > **Client-Side Library**.



19.29 Add a **Bootstrap Table library** as shown in the red box below.

19.30 In the **Views Folder** create a new folder called **HumanResource**. Inside the folder add a new View called **HRDashboard**.

19.31 In the **HRDashboard** view enter the following code construction for the Main Table structure.

```
@model HumanResourcesViewModel
@{
    ViewBag.Title = "Human Resources";
    int DepRowCounter = 1;
    int HRCounter = 1;
}
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Human Resource Dashboard</title>
    <script src="~/js/ProcessTableData.js"></script>
</head>
<body>
    <div class="table-responsive text-nowrap">
        <div class="row">
            <div class="col-sm-6">
                <table id="existingtable" class="table table-bordered">
                    <thead class="black white-text">
                        <tr>
                            <th scope="col">ID</th>
                            <th scope="col">Name</th>
                            <th scope="col">Email</th>
                            <th scope="col">Department</th>
                            <th scope="col">Classification</th>
                            <th scope="col">Pay</th>
                        </tr>
                    </thead>
                    @foreach (var item in Model.ActiveEmployee)
                    {
                    }
                </table>
            </div>
            <div class="col-sm-6">
                <table id="existingtable1" class="table table-bordered">
                    <thead class="black white-text">
                        <tr>
                            <th scope="col">Retirement</th>
                            <th scope="col">Health</th>
                            <th scope="col">Orientation</th>
                            <th scope="col">Security</th>
                            <th scope="col">Ethics</th>
                            <th scope="col">Safety</th>
                        </tr>
                    </thead>
                    @foreach (var item in Model.EmployeeRequirements)
                    {
                    }
                </table>
            </div>
        </div>
    </div>
    <button id="convert-table" class="btn btn-primary" onclick="showTableData()">Update</button>
    <div id="UpdateSuccess" hidden class="badge-success">
        <h1>Employee Records Updated Sucessfully</h1>
    </div>
    <div id="UpdateFailure" hidden class="badge-danger">
        <h1>Employee Records Update failed</h1>
    </div>
</body>
</html>
```

19.32 In the **HRDashboard** view inside the **ForEach Loop** of the **Model.ActiveEmployee** table enter the following code construction.

```
 <tr id="somerow">
     <td scope="col">@item.Id</td>
     <td scope="col" contenteditable="true">@item.Name</td>
     <td scope="col" contenteditable="true">@item.Email</td>
     <td scope="col" contenteditable="true">
         <select asp-for="@item.Department" id="SelDepartment-@DepRowCounter" class="custom-select mr-sm-2"
                 asp-items="Html.GetEnumSelectList<Department.Dept>()">
           <option value="">Please Select</option>
         </select>
            <span asp-validation-for="@item.Department" class="text-danger"></span>
     </td>
     <td scope="col" contenteditable="true">
         <select asp-for="@item.Classification" id="SelClassification-@DepRowCounter" class="custom-select mr-sm-2"
                 asp-items="Html.GetEnumSelectList<Classification.type>()">
              <option value="">Please Select</option>
         </select>
              <span asp-validation-for="@item.Classification" class="text-danger"></span>
     </td>
      <td scope="col" contenteditable="true">@item.EmployeeSalary</td>

      <td scope="col" hidden>@item.PhotoPath</td>

   </tr>
          DepRowCounter = DepRowCounter + 1;
```

19.33 In the **HRDashboard** view inside the **ForEach Loop** of the **Model.EmployeeRequirements** table enter the following code construction.

```
<tr id="HRRow">
    <td scope="col" hidden>@item.Id</td>
    <td scope="col" contenteditable="true">
        <select asp-for="@item.RetirementPlan" id="SelRetirement-@HRCounter" class="custom-select mr-sm-2"
                asp-items="Html.GetEnumSelectList<Retirement.Plan>()">
                    <option value="">Please Select</option>
        </select>
                <span asp-validation-for="@item.RetirementPlan" class="text-danger"></span>
   </td>
    <td scope="col" contenteditable="true">
        <select asp-for="@item.HealthPlan" id="SelHealthPlan-@HRCounter" class="custom-select mr-sm-2"
                asp-items="Html.GetEnumSelectList<HealthCare.Plan>()">
                    <option value="">Please Select</option>
        </select>
                <span asp-validation-for="@item.HealthPlan" class="text-danger"></span>
         </td>
<td scope="col" contenteditable="true"><input type="checkbox" id="OnboardStatus-@HRCounter" asp-
for=@item.OnboardProgramComplete /></td>
            <td scope="col" contenteditable="true"><input type="checkbox" id="SecurityTrainingStatus-@HRCounter" asp-
for=@item.SecurityTrainingComplete /></td>
            <td scope="col" contenteditable="true"><input type="checkbox" id="EthicsTrainingStatus-@HRCounter" asp-
for=@item.EthicsTrainingComplete /></td>
            <td scope="col" contenteditable="true"><input type="checkbox" id="SafetyTrainingStatus-@HRCounter" asp-
for=@item.SafetyTrainingComplete /></td>
</tr>

         HRCounter = HRCounter + 1;
```

NOTE:

One may wonder why this section repeats the same functionality from earlier parts of the tutorial. The answer is simply this section has several additional features; firstly all the employe data except photo path is displayed in tabular format. Next, access to this section is restricted to the HumamResource user group. Lastly, and most important of all, more than one employee record can be updated in a single transaction.

19.34 Compile the Application and execute it as a member of the HumanResource team. Observe the following appears as shown in Figure 103.

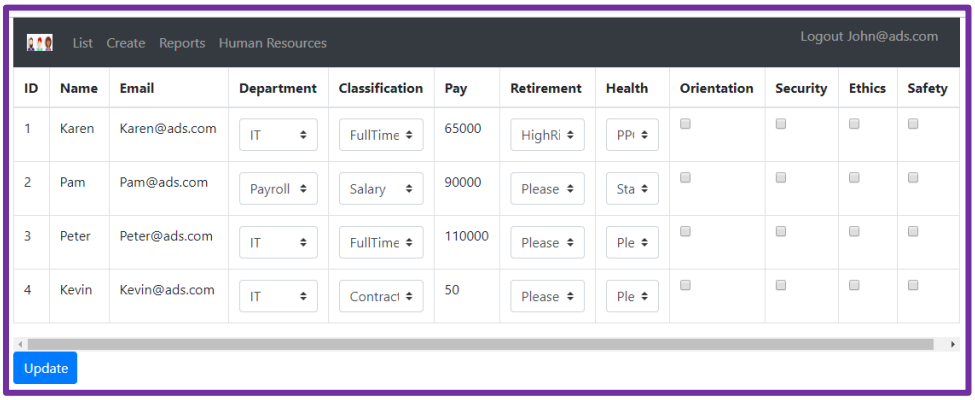


Figure 103

19.35 Check the code in using **GitKracken** and close out the task items in **Glo**.

19.36 The next task to complete is to synchronize the database structure for the Employee Table and **Human Resource Table**. In **Sqlite Explorer** Select the **Human Resource Table** by using the right mouse button to select the **Table Populate Option** as shown in Figure 104.

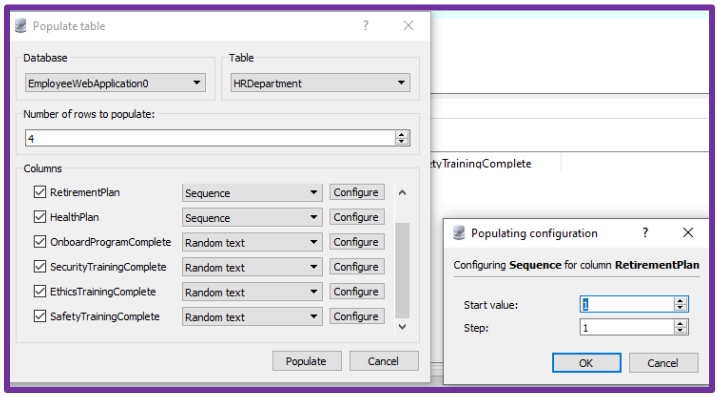


Figure 104

19.37 Add additional fields for the Human Resource section by entering data for each record. Observe the following applears as shown below.

19.38 Observe the following records appear as shown in Figure 105.

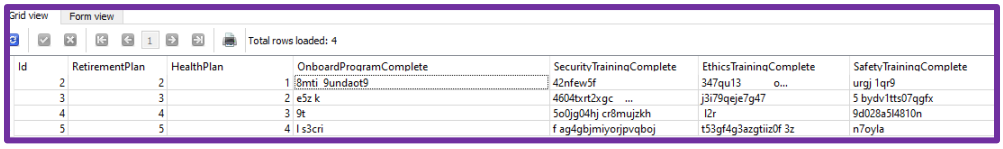


Figure 105

November 2, 2019
Prepared by Andre Masters
Revision 1.0

19.39 In **Sqlite Studio** edit the records for the **Human Resource Section.** Do this using the **Edit Values** as displayed in the editor feature as shown Figure 106

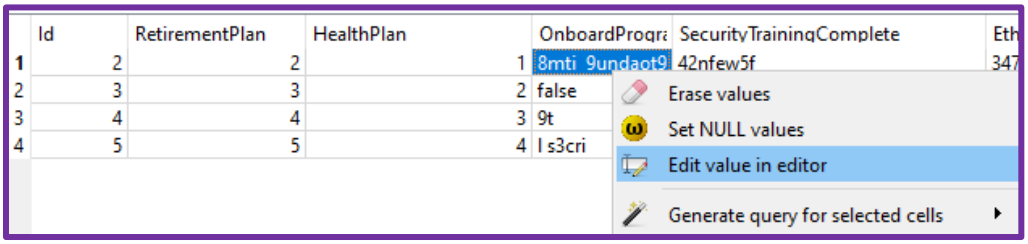


Figure 106

19.40 Make the following change for **Boolean Value** as shown in the **Edit Value Textbox** of Figure 107.

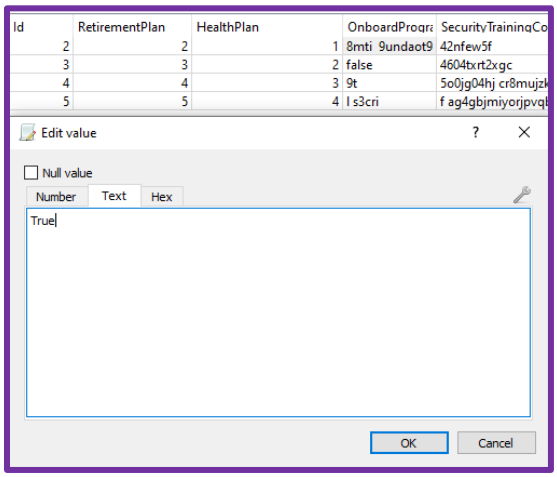


Figure 107

19.41 In order for the cnages to take affect select Commit for a single entry as shown in Figure 108.

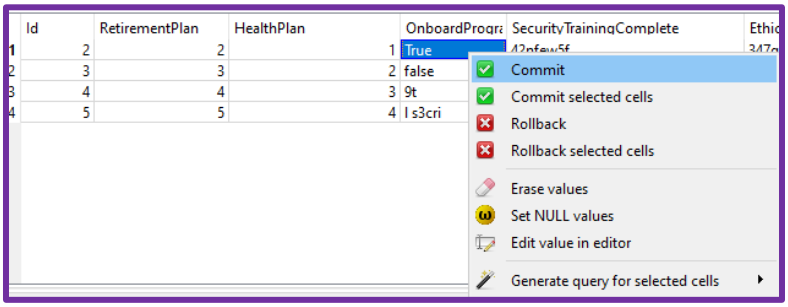


Figure 108

19.42 Use the Ctrl key to make multiple selections to commit all at once as shown in Figure 109.

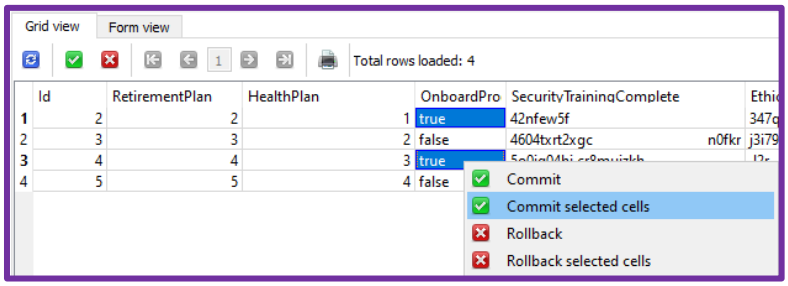


Figure 109

19.43 Make all the neccsesary changes to match the what is displayed in Figure 110

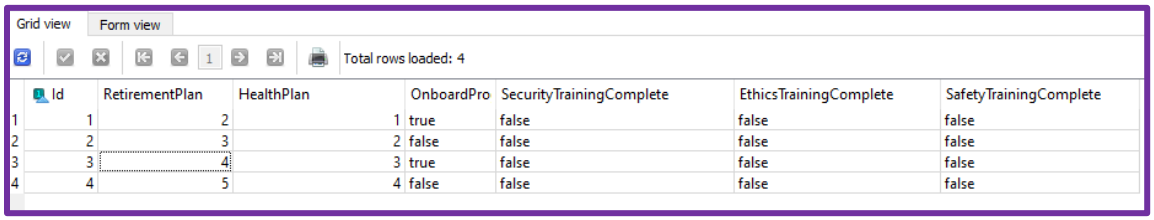


Figure 110

19.44 Run the application by pressing F5. Observe the following appears as shown in Figure 111.
In Figure 111 both tables are joined into one and all the check boxes are in the unchecked state.

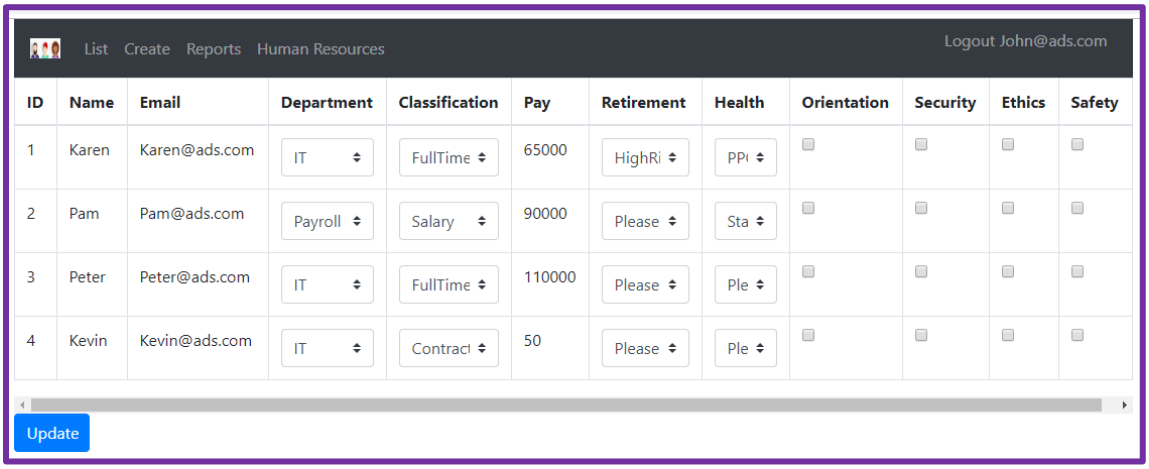


Figure 111

19.45 In the **Human Resource Table** make an Update by changing Karen's **Orientation Complete Status** from false to true. At this point the code construction for the Update button is not operational. All the fields should be editable this includes textboxes, selection lists/drop down dialogs and checkboxes as shown in Figure 112.

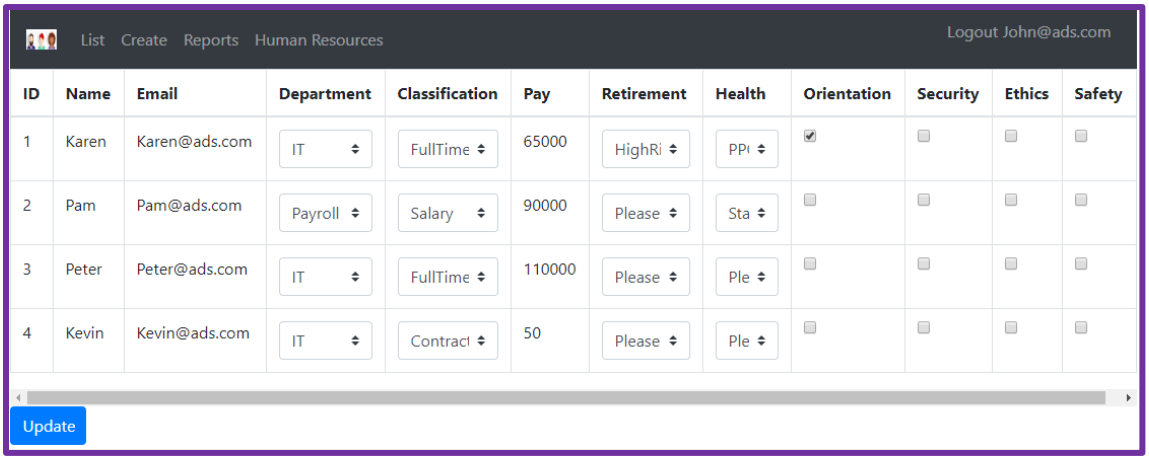


Figure 112

NOTE:

Boolean values in the database need to be stated as 1 for True and 0 for false. Any input that is not a 1 will be treated as false.

19.46 In the **Human Resource Controller** stubout the **POST** version of the table update feature.

```
[HttpPost]
public IActionResult HRDashboard([FromBody]  List<List<JObject>> data1)
{
    return View();
}
```

19.47 In the **js** subfolder of the **wwwroot** main folder create a file called **ProcessTableData.js.** In the file enter the following code construction.

```
function showTableData() {
    //Step 1 Create Data structures for iterating the HR + Employee  variables for parsing the HTML DOM
    var myEmployees = new Array();
    var myHrs = new Array();
    var EmployeeHR = new Array();
    var jsonemployee;
    var jsonhr;
    var Employeetbl = document.getElementById("existingtable");
    var HRtbl = document.getElementById("existingtable1");
    var StatusPass = document.getElementById("UpdateSuccess");
    var StatusFail = document.getElementById("UpdateFailure");

    //Step 2 Iterate through the Employee Table
    for (var i = 1; i < Employeetbl.rows.length; i++) {
        var row = Employeetbl.rows[i];
        var myEmployee = { 'Id': 'Name', 'Email': 'Department', 'Classification': 'EmployeeSalary', 'ExistingPhoto': '' };
        myEmployee.Id = row.cells[0].innerHTML;
        myEmployee.Name = row.cells[1].innerHTML;
        myEmployee.Email = row.cells[2].innerHTML;
        myEmployee.Department = $('select[id$=SelDepartment-'+i+'] :selected').text();
        myEmployee.Classification = $('select[id$=SelClassification-'+i+'] :selected').text();
        myEmployee.EmployeeSalary = row.cells[5].innerHTML;
        myEmployee.PhotoPath = row.cells[6].innerHTML;
        myEmployees.push(myEmployee);

    }

    //Step 3 Assign the Employee Table data to an Array and json objects display them in the Browser Console
    jsonemployee = JSON.stringify(myEmployees);
    console.log(myEmployees);
    console.log(jsonemployee);

    //Step 4 Iterate through the Human Resource Table
    for (var lcv = 1; lcv < HRtbl.rows.length; lcv++) {
        var hrow = HRtbl.rows[lcv];
        var myHr = { 'Id': 'RetirementPlan', 'HealthPlan': 'OnboardProgramComplete', 'SecurityTrainingComplete': 'EthicsTrainingComplete',
'SafetyTrainingComplete': '' };
        myHr.Id = hrow.cells[0].innerHTML;
        myHr.RetirementPlan = $('select[id$=SelRetirement-'+lcv+'] :selected').text();
        myHr.HealthPlan = $('select[id$=SelHealthPlan-'+lcv+'] :selected').text();
        myHr.OnboardProgramComplete = $('[id$=OnboardStatus-'+lcv+']').is(':checked');
        myHr.SecurityTrainingComplete = $('[id$=SecurityTrainingStatus-'+lcv+']').is(':checked');
        myHr.EthicsTrainingComplete = $('[id$=EthicsTrainingStatus-'+lcv+']').is(':checked');
        myHr.SafetyTrainingComplete = $('[id$=SafetyTrainingStatus-'+lcv+']').is(':checked');
        myHrs.push(myHr);
    }

    //Step 5 Assign the Human Resource Table data to an Array and json objects display them in the Browser Console
    jsonhr = JSON.stringify(myHrs);
    console.log(myHrs);
    console.log(jsonhr);

    //Step 6 Build the Comtainer Array by supplying it Human Resource and Employee Array Data
    EmployeeHR.push(myEmployees);
    EmployeeHR.push(myHrs);
    EmployeeHR = JSON.stringify(EmployeeHR);

    //Step 7 Set the Status Flags
    StatusPass.hidden = true;
    StatusFail.hidden = true;

    //Step 8 Send the Payload to the HumanResource Controler and HRDashboard action method
    $.ajax({
        contentType: "application/json; charset=utf-8",
        dataType: 'json',
        type: 'POST',
        url: '/HumanResource/HRDashboard',
        data: EmployeeHR,
        cache: false,
    });

}
```

19.48 In the **HumanResourceController** file enter the following code construction for the **HRDashboard** routine **HttpPost** version as shown in red box. See step 19.26 for the HttpGet version of this routine.

```
[HttpPost]
public IActionResult HRDashboard([FromBody]  List<List<JObject>> data1)
{
    List<JObject> EmployeeRequirement = data1.FirstOrDefault();
    List<JObject> HumanResources = data1.LastOrDefault();

        foreach (JObject EmployeeLiveData in EmployeeRequirement)
        {
              Employee UpdatedEmployee = new Employee();
              JObject EmployeeData = JObject.Parse(EmployeeLiveData.ToString());
              UpdatedEmployee.Id = (Int32)EmployeeData["Id"];
              UpdatedEmployee.Email = (string)EmployeeData["Email"];
              UpdatedEmployee.Name = (string)EmployeeData["Name"];
              UpdatedEmployee.EmployeeSalary = (double)EmployeeData["EmployeeSalary"];
              UpdatedEmployee.Department = EmployeeLiveData["Department"].ToObject<Department.Dept>();
              UpdatedEmployee.Classification = EmployeeData["Classification"].ToObject<Classification.type>();
              UpdatedEmployee.PhotoPath = (string)EmployeeData["PhotoPath"];

             //push Each Record to the Employee Database
              _EmployeeRepo.Update(UpdatedEmployee);
         }

        foreach(JObject HumanResourceLiveData in HumanResources)
        {
             HumanResource HRecord = new HumanResource();
             JObject HR_Data = JObject.Parse(HumanResourceLiveData.ToString());
             HRecord.Id = (Int32)HR_Data["Id"];
             HRecord.RetirementPlan = HR_Data["RetirementPlan"].ToObject<Retirement.Plan>();
             HRecord.HealthPlan = HR_Data["HealthPlan"].ToObject<HealthCare.Plan>();
             HRecord.OnboardProgramComplete = (bool)HR_Data["OnboardProgramComplete"];
             HRecord.SafetyTrainingComplete = (bool)HR_Data["SafetyTrainingComplete"];
             HRecord.SecurityTrainingComplete = (bool)HR_Data["SecurityTrainingComplete"];
             HRecord.EthicsTrainingComplete = (bool)HR_Data["EthicsTrainingComplete"];
             //push each record to the Human Resource Database
            _HRemployeeRepository.Update(HRecord);
        }

    HumanResourcesViewModel EmployeeHRequirements = new HumanResourcesViewModel();

    var HR_Requirements = _HRemployeeRepository.GetAllHREmployeeData();
    var Employee = _EmployeeRepo.GetAllEmployees();

    EmployeeHRequirements.EmployeeRequirements = HR_Requirements;
    EmployeeHRequirements.ActiveEmployee = Employee;

    return View(EmployeeHRequirements);
}
```

19.49 In the **ProcessTableData.js** file add the following code construction for reporting the HTTP status responses allowing only Success to be displayed for HTTP Status 200 and 201.

```
statusCode: {
    200: function () {
        StatusPass.hidden = false;
        StatusFail.hidden = true;
    },
    201: function () {
        StatusPass.hidden = false;
        StatusFail.hidden = true;
    },
    400: function () {
        StatusPass.hidden = true;
        StatusFail.hidden = false;
    },
    404: function () {
        StatusPass.hidden = true;
        StatusFail.hidden = false;
    },
    500: function () {
        StatusPass.hidden = true;
        StatusFail.hidden = false;
    }
```

19.50 Verify the application runs success fully by pressing F5 and observe the following appears as shown in Figure 113

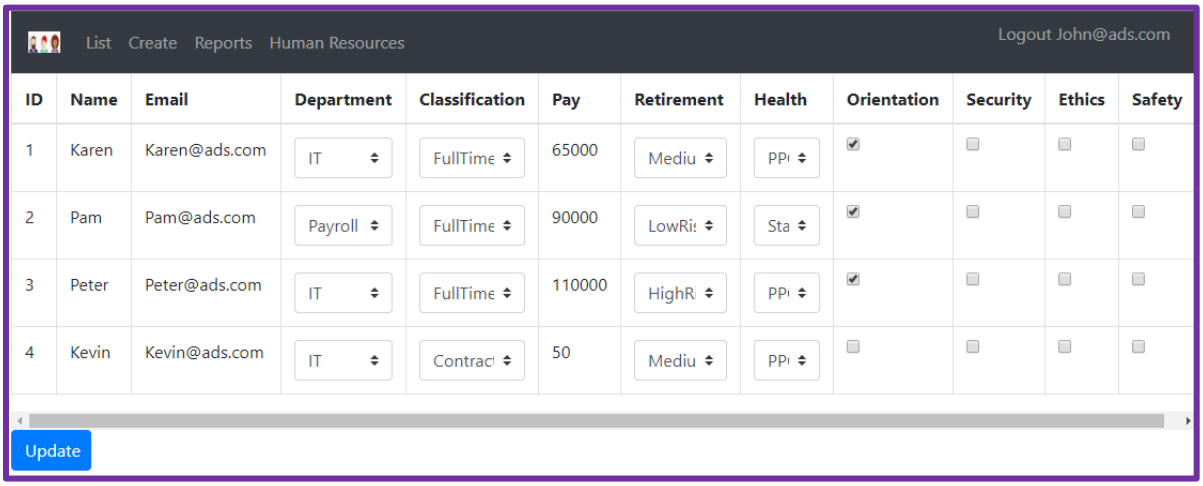


Figure 113

UAT Testing

19.51 Make the following changes for the indicated Employees as shown below and press the Update button.

Karen

- New Department is Payroll
- New Classification is Contractor
- New Pay is 60000
- New Retirement is LowRisk
- New Health is Standard
- New Orientation Unchecked
- New Security Checked
- New Ethics Checked
- New Safety Checked

Peter

- New Department is HumanResources
- New Classification is Salary
- New Pay is 100000
- New Retirement is LowRisk
- New Health is Standard
- New Orientation Unchecked
- New Security is checked
- New Ethics is checked
- New Safety is checked

November 2, 2019
Prepared by Andre Masters
Revision 1.0

19.52 Observe the following appears as shown in Figure 114. Before Pressing the Update Button

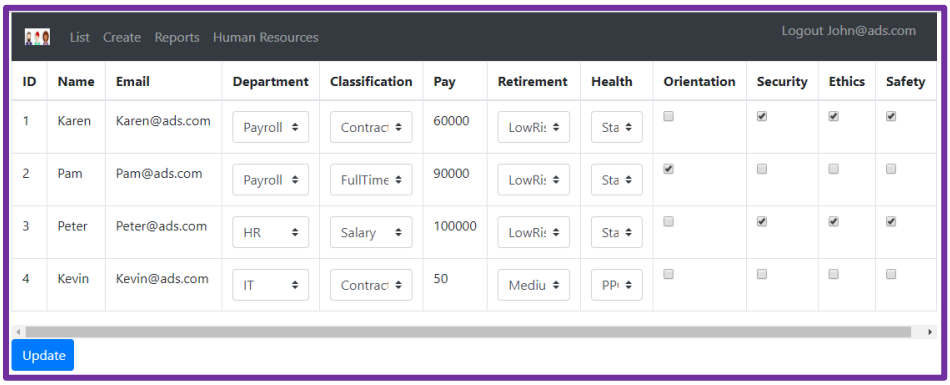


Figure 114

19.53 After Pressing the Update Button observe the following appears as shown in Figure 115

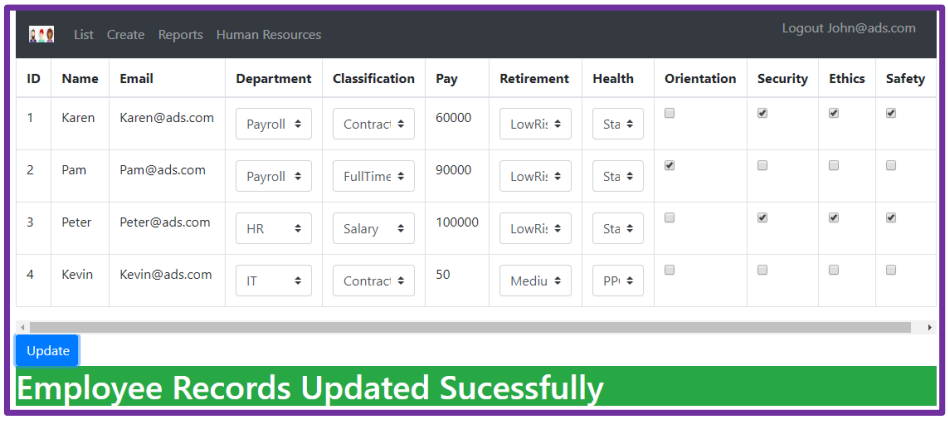


Figure 115

19.54 Redo the previous step by taking the opposite actions for both Peter and Karen. Then select the List option on the main menu then select **Human Resources**. Observe the banner for Employee Records Updated Successfully disappears as shown in Figure 116.

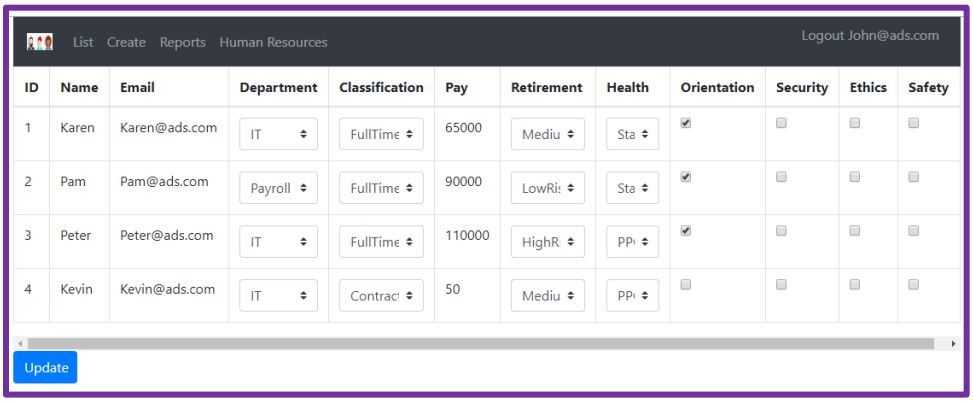


Figure 116

19.55 Check in the code using the steps outlined in the Employee Section for Git Kracken. Make clear and concise notes for each item that gets checked. Do not do a bulk code check in with a single comment.

NOTE:

There typically in a more secure environment there is an antiforgery token that gets applied to POST actions. The data payload is encrypted between the client and server end of the application.

There is a strange sytax error that gets thrown in the ajax call. Why this occurs may be due to a mis compilation of the javascript code in the browser resulting in an exception in the base javascript library.

Ideally it would have been cleaner to develop this as a two way binding of the HumanResourcesViewModel a model that contains two models of which each contain enums as complex types resulting in three layers of abstraction.

HUMAN RESOURCE SECTION IMPLEMENTATION COMPLETE

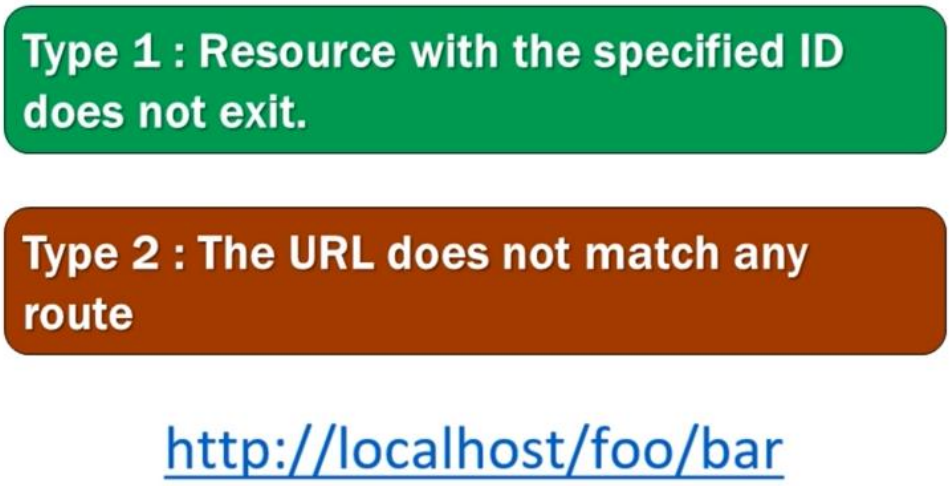
November 2, 2019
Prepared by Andre Masters
Revision 1.0

Error Handling Process

This section contains instructions for implementing error handling for page not found, loggin error, database errors and input handing that is not covered by validation.

Section I 404 Not Found Error

Not found error pages can be applied to any condition for which a given url does not return a valid page view. The Details controller will be modified to support navigation to an error page. The error page will be a razor view that will be made available to the project. This section will address two types of page not found errors one for url not found and one for index not found.



20.0 In the **Details Routine** in the **Home controller** modify the existing code with the statements below.

```
public ViewResult Details(int? id)
{
    Employee employee = _employeeRepository.GetEmployee(id.Value);

    if(employee == null)
    {
        Response.StatusCode = 404;
        return View("EmployeeNotFound", id.Value);
    }
    EmployeeDetailsViewModel homeDetailsViewModel = new EmployeeDetailsViewModel()
    {
        Employee = employee,
        PageTitle = "Employee Details"
    };
    return View(homeDetailsViewModel);
}
```

20.1 In the Launch Settings file change the ASP.Net Core Environment variable from development to production.

```
"ASPNETCORE_ENVIRONMENT": "Production”
```

20.2 In the Home Folder for Views, open the file EmployeeNotFound, and enter the following code statements below.

```
@model int

@{
    ViewBag.Title = "404 Error";
}

<div class="alert alert-danger mt-1 mb-1">
    <h4>404 Error Not Found</h4>
    <hr />
    <h5>
        Employee with id = @Model  cannot be found
    </h5>
</div>

<a asp-action="index" asp-controller="home" class="btn btn-outline-success" style="width:auto">
click here to see the list of all employees
</a>
```

20.3 In the Error Controller, open the file and enter the following code statements.

```
[Route("Error/{statuscode}")]
public IActionResult HttpStatusCodeHandler(int statuscode)
{
    switch (statuscode)
    {
        case 404:
            ViewBag.ErrorMessage = "Resource Not Found";
            break;
    }

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

20.4 In the Startup Class enter the following statement in the Configure routine.

```
app.UseExceptionHandler("/Error");
app.UseStatusCodePagesWithReExecute("/Error/{0}");
```

20.5 Run the application select an employee detail and replace it with an employee ID that does not exist and observe the following appears as shown in Figure 117.

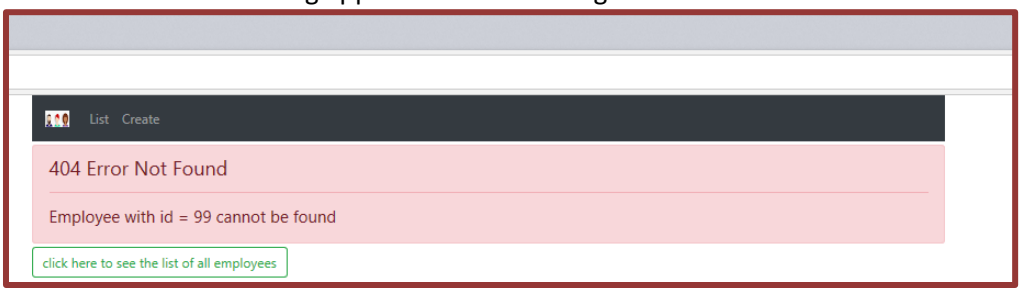


Figure 117

20.6 Click on the link to observe the user gets returned to the main page as shown in Figure 118

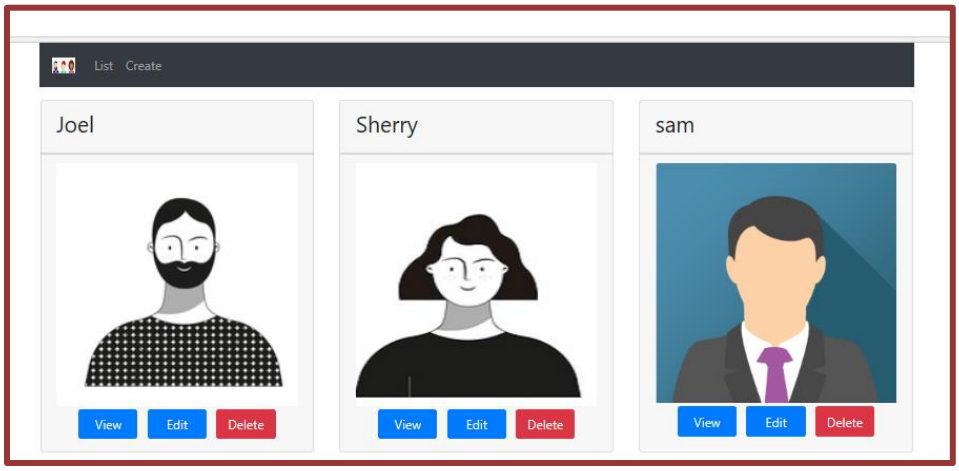


Figure 118

20.7 In the URL add a link that does not exist observe the following appears as shown in Figure 119 and verify the redirect link takes the user to the main page as shown in step 20.5.

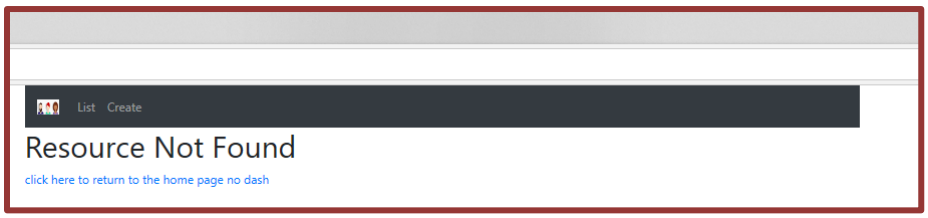


Figure 119

NOTE:

In the startup class, there are two options to use for error handling. The first option called ***UseStatusCodePagesWithReExecute*** *will call the 404 error page as a direct http request, registering it in the browser as a true 404 error. The other option is called* ***UseStatusCodePages,*** *this option will show as a 302 success in the browser because the true error page gets served as a valid http request, but rather was the result of a bad htttp request.*

Section II Delete Role Error

This section deals with errors associated with login related errors. If a given user is logged in already

20.8 Create a new View in the Shared Views folder, call this view Error.cshtml. Enter the following code construction as shown below.

```
@if (ViewBag.ErrorTitle == null)
{
    <h3>
        An Error Occured while processing your request.
        A Support ticket has been filed with the support team
    </h3>
}
else
{
    <h1 class="text-danger">@ViewBag.ErrorTitle</h1>
    <h3 class="text-danger">@ViewBag.ErrorMessage</h3>
}
```

NOTE:

A role may only be deleted when there are no users in the role otherwise the result will be a foreign key to primary key constraint violation. The effect of this violation will terminate the SQL operation while the user needs to be instructed in developing a solution to the problem. Add the following changes to the code below in yellow highlight.

20.9 Create a new **View** in the **Shared Views Folder**, call this view **Error.cshtml**. Enter the following code construction as shown below.

EXISTING DELETE ROLE METHOD

```
[HttpPost]
  public async Task<IActionResult> DeleteRole(string id)
  {
      var role = await roleManager.FindByIdAsync(id);

      if (role == null)
      {
          ViewBag.ErrorMessage = $"Role with Id = {id} Cannot be found";
          return View("NotFound");
      }
      else
      {
          var result = await roleManager.DeleteAsync(role);
          if(result.Succeeded)
          {
              return RedirectToAction("ListRoles");
          }
          foreach(var error in result.Errors)
          {
              ModelState.AddModelError("", error.Description);
          }
      }
      return View("ListRoles");
  }
```

NEW DELETE ROLE METHOD

```
[HttpPost]
public async Task<IActionResult> DeleteRole(string id)
{
    var role = await roleManager.FindByIdAsync(id);
    string rolecheck = role.ToString();
    var user = userManager.GetUsersInRoleAsync(rolecheck).Result;

    if (role == null)
    {
        ViewBag.ErrorMessage = $"Role with Id = {id} Cannot be found";
        return View("NotFound");
    }
    else
    {
        try
        {
            if(user.Count == 0)
            {
                var result = await roleManager.DeleteAsync(role);
                if (result.Succeeded)
                {
                    return RedirectToAction("ListRoles");
                }
                foreach (var error in result.Errors)
                {
                    ModelState.AddModelError("", error.Description);
                }
            }
            else
            {
                ViewBag.ErrorTitle = $"{role.Name} is in use";
                ViewBag.ErrorMessage = $"The {role.Name} role cannot be deleted since there are existing Users "+
                                       $"assigned to this role. To delete the Role first " +
                                       $"remove all existing Users and then delete the role.";
                return View("Error");
            }
        }
        catch( DbUpdateException ex)
        {
            ViewBag.ErrorTitle = $"{role.Name} is in use";
            ViewBag.ErrorMessage = $"{role.Name} Database Failure " + ex.Message;
            return View("Error");
        }
    }
    return View("ListRoles");
}
```

20.10 Make the following changes to the ListRoles to allow for the Delete Operation of a given role.

```
@model IEnumerable<IdentityRole>
@{
    ViewBag.Title = "All Roles";
}
<h1>All Roles</h1>
@if (Model.Any())
{

    <a class="btn btn-primary"style="width:auto" asp-action="CreateRole" asp-controller="administration">Add New Role</a>

    @foreach(var role in Model)
    {
        <div class="card mb-3">
            <div class="card-header">
                Role Id: @role.Id
            </div>
            <div class="card-body"style="font-size:large">
                @role.Name
            </div>
            <div class="card-footer">

                <form asp-action="DeleteRole" asp-route-id="@role.Id" method="post">
                    <a class="btn btn-primary" asp-action="EditRole"
                       asp-controller="Administration" asp-route-id="@role.Id">
                        Edit
                    </a>
                    <span id="confirmDeleteSpan_@role.Id" style="display:none">
                        <span>Are you sure you want to delete?</span>
                        <button type="submit" class="btn btn-danger">Yes</button>
                        <a href="#" class="btn btn-primary"
                           onclick="confirmDelete('@role.Id', false)">No</a>
                    </span>

                    <span id="deleteSpan_@role.Id">
                        <a href="#" class="btn btn-danger"
                           onclick="confirmDelete('@role.Id', true)">Delete</a>
                    </span>
                </form>
            </div>
        </div>
    }
}

else
{
    <div class="card">
        <div class="card-header">
            No roles created yet
        </div>
        <div class="card-body">
            <h5 class="card-title"> Use the button below to create a role</h5>
            <a class="btn btn-primary" style="width:auto"
               asp-controller="administration" asp-action="CreateRole">Create Role</a>
        </div>
    </div>
}
@section scripts{
    <script src="~/js/ConfirmDelete.js"></script>
}
```

20.11 Implement the following code construction for **ConfirmDelete.js** file as shown below.

```
function confirmDelete(uniqueId, isTrue) {
    var deleteSpan = 'deleteSpan_' + uniqueId;
    var confirmDeleteSpan = 'confirmDeleteSpan_' + uniqueId;
    if (isTrue) {
        $('#' + deleteSpan).hide();
        $('#' + confirmDeleteSpan).show();
    } else {
        $('#' + deleteSpan).show();
        $('#' + confirmDeleteSpan).hide();
    }
}
```

20.12 Run the application by pressing F5 login as an **Admin** user and observe the following appears as shown in Figure 120

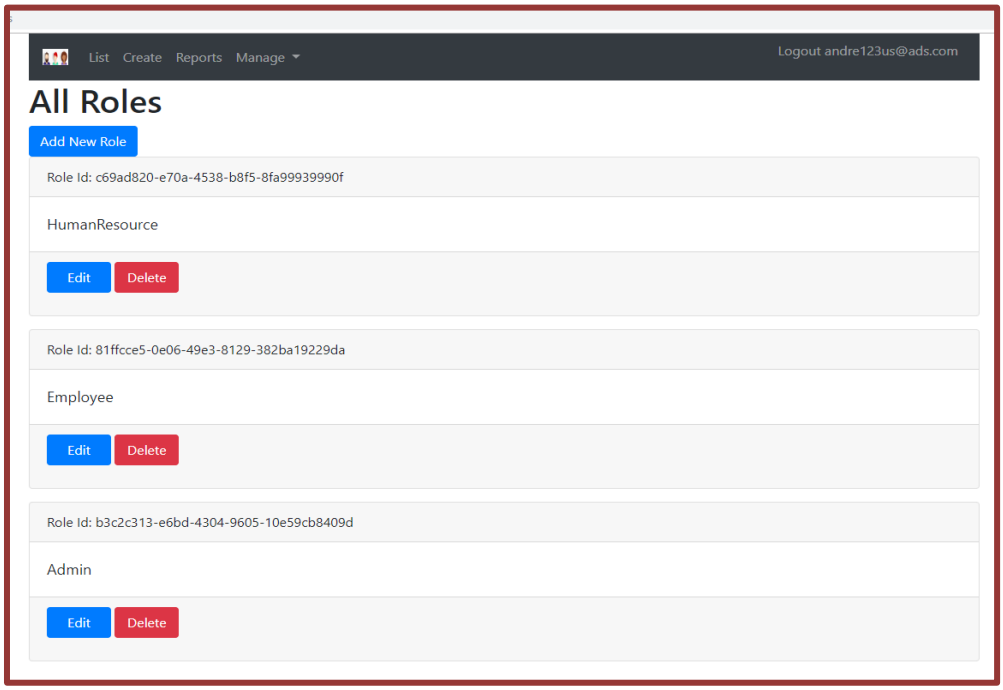


Figure 120

20.13 Try to delete the Admin Role by pressing the Delete Button observe the following appears as Shown in Figure 121

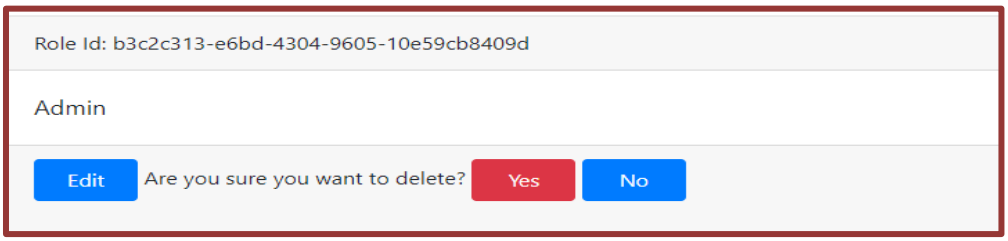


Figure 121

20.14 Press the Yes Button and observe the following Error file appears as as shown in Figure 122

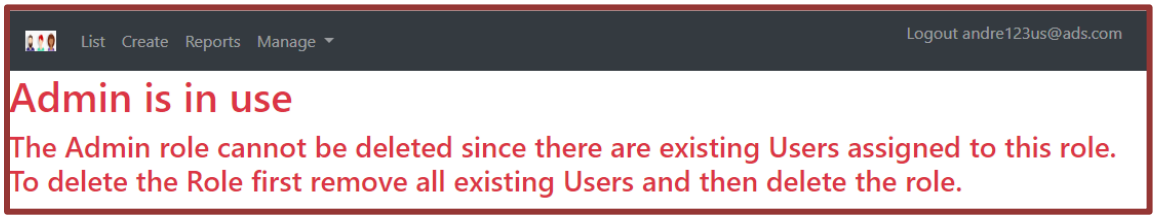


Figure 122

Deployment to Raspberry PI

This purpose of this section is to create the capability for creating and executing an image for a given container. The container would then be deployed to a repository and then pulled down onto the target device in QA or Production. In this case an image will be created from the Employee Management as a published deployment. The image will be housed in a docker container that will be stored in a repository in Docker Hub. The container will operate based on instruction in a Docker File.

In order to for the application to function in a linux environment it will need to be cross compiled into an ARM 32 chipset architecture within visual studio. The section is broken down into three parts, docker setup and configuration for the workstation and raspberry pi.

Section I Setup Raspberry PI

The raspberry pi used in this in example is a PI-3 with a debian linux distribution type rasperian stretch version 9. The architecture is a 64 bit ARM cortex processor version 8. Disk space is 32GB which is plenty for the current application. This segment will be mostly commands that will be entered in the raspberry pi command line through an SSH connection . In order to complete all the steps in this section the user will need to have root privileges. Begin the process by completing all the steps outlined below.

21.0 Setup Raspberry PI using the Rasperian Stretch Linux distro version 9. Verify the Setup meets this requirement as shown in Figure 123

```
sudo: /etc/os-release: command not found
pi@raspberrypi:~ $ cat /etc/os-release
PRETTY_NAME="Raspbian GNU/Linux 9 (stretch)"
NAME="Raspbian GNU/Linux"
VERSION_ID="9"
VERSION="9 (stretch)"
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
pi@raspberrypi:~ $
```

Figure 123

21.1 Update the system using the following command.

```
sudo apt update
```

21.2 Install the **Apt** support for https using the following command

```
sudo apt-get install -y apt-transport-https ca-certificates
wget software-properties-common
```

21.3 Observe the following occurs in the raspberry pi as shown in Figure 124

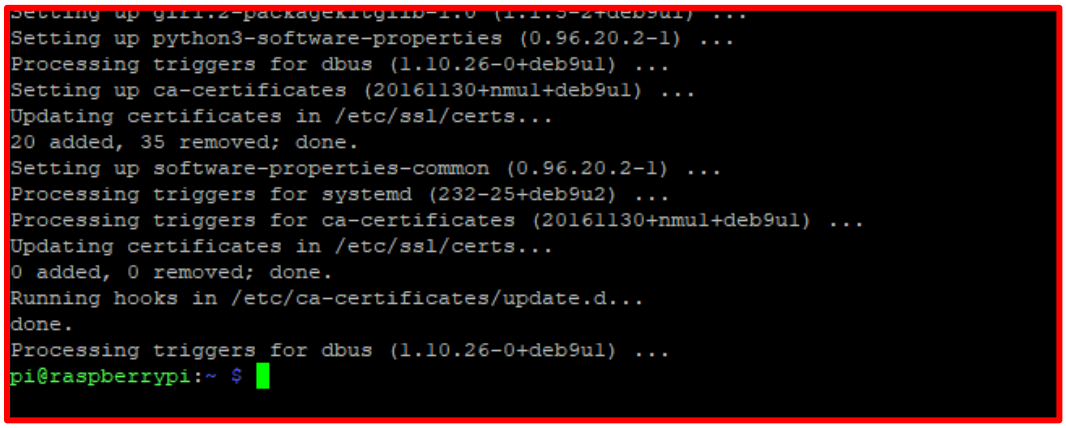


Figure 124

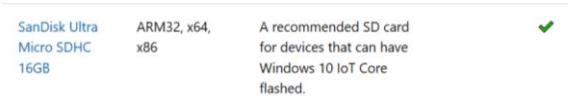
Setup Raspberry PI 2/3 - Windows IOT Dashboard

The steps for configuring a Windows IOT deployment for Raspberry PI are fairly straight forward. The purpose for using a Windows IOT environment is that it allows for the use of windows IOT libraries and features in building an ecosystem of endpoint devices using the raspberry pi as a central control entity.

22.0 Pre-Requisistes: Windows 10 Workstation download a custom image 17763.107, extract the fflu files.

22.1 Run the **Window_10_IOT_Core_for_RPi.msi** file and observe the following appears as shown on the Windows 10 local workstation.

C:\Program Files (x86)\Microsoft IoT\FFU\RaspberryPi2\flash.ffu



22.2 Run the SD Card Formatter observe the following screen appears as a shown in Figure 126 and observe the notes contained therein.

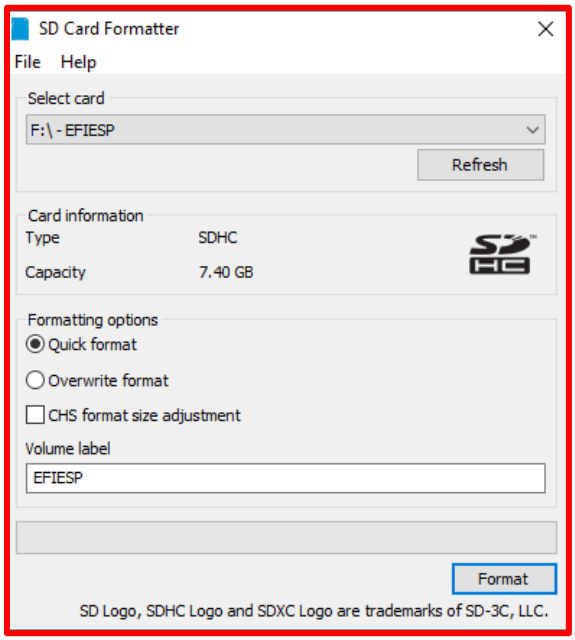


Figure 126

22.3 Run the SD Card Formatter observe the following screen appears as a shown in Figure 127 when complete.

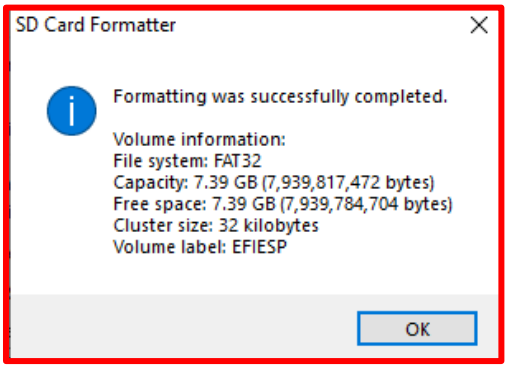


Figure 127

22.4 Make sure the SD card you are using contains a minimum disk space of 8GB. Down load the Windows IOT Core Dashboard to your local workstation PC from the following URL.

https://developer.microsoft.com/en-us/windows/iot/docs/iotdashboard

22.5 Verify the following screen appears as a shown in Figure 128 and observe the notes contained therein.

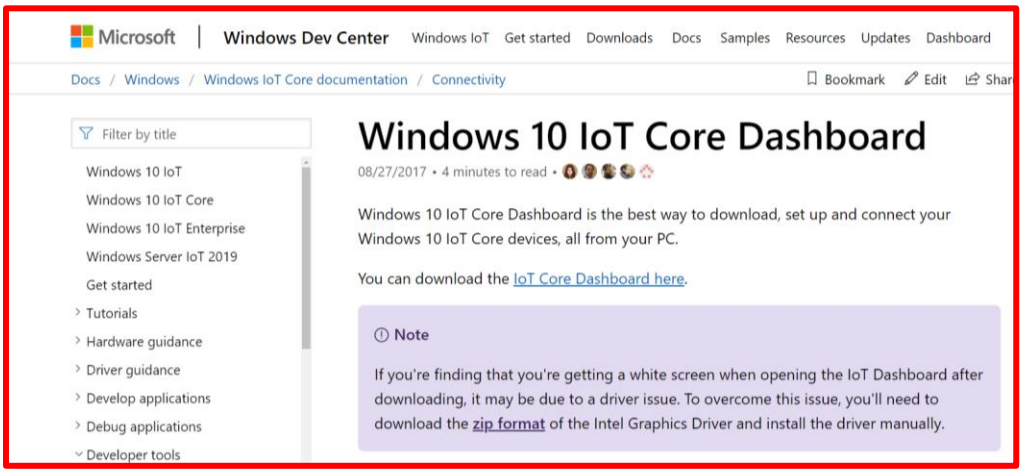


Figure 128

22.6 Install the IoT Dashboard observe the following appears as shown in Figure 129.

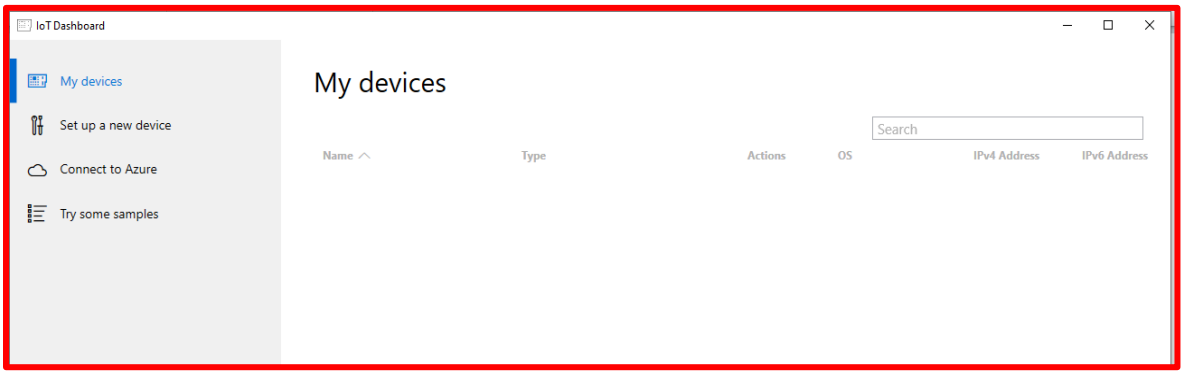


Figure 129

22.7 Insert an SD card into the Workstation PC and select Setup a New device on the Windows IOT Dashboard, observe the following appears as shown in Figure 130

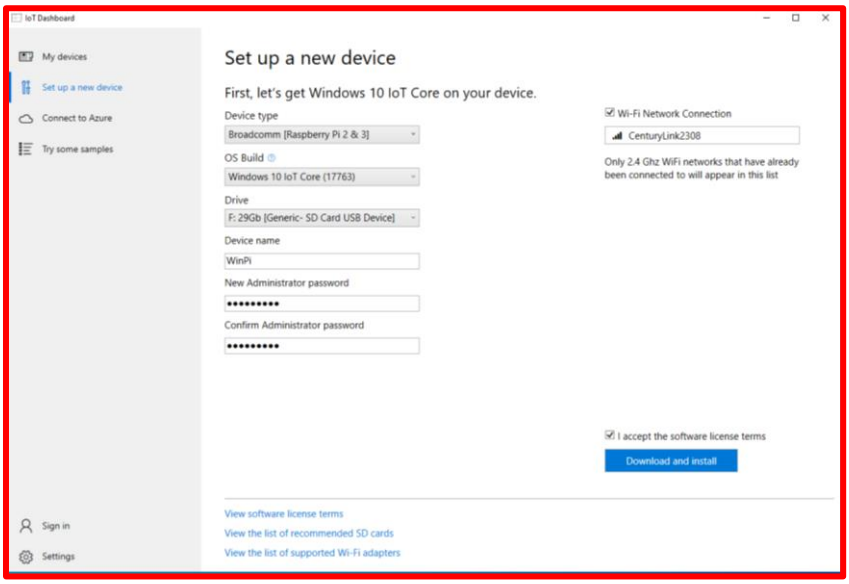


Figure 130

22.8 Create an Administrator Password in this case simply use the following as an example.

Dexfexpeg

22.8 Observe the following appears as shown in Figure 131.

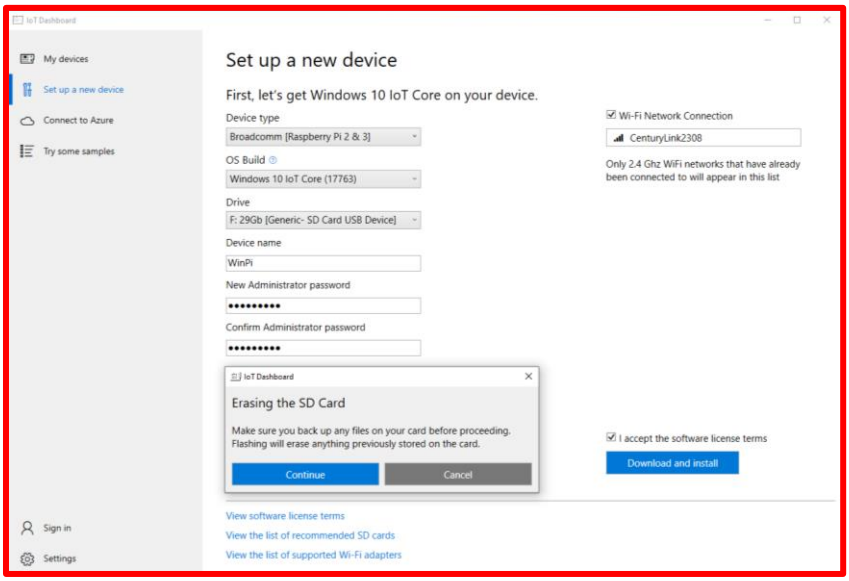


Figure 131

NOTE:

Wi-Fi Network is already setup and configured automatically and will be used as the default for the environment in which the device lives.

Make sure the IOT Dashboard is selected as a trusted application if running Accronis or similar workstation protection program.

22.9 The following screen should appear as displayed below indicating the status of the current installation at the cleaning stage as shown in Figure 132.

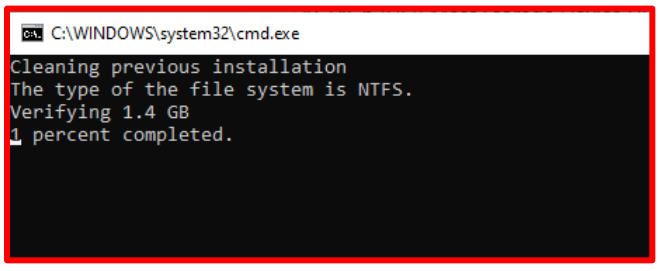


Figure 132

22.10 The following screen should appear as displayed below indicating the status of the current installation as shown in Figure 133.

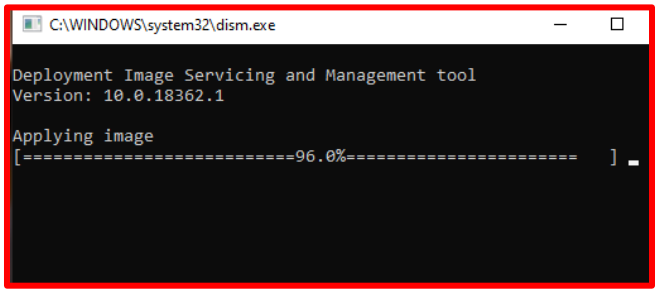


Figure 133

22.11 Once the installation completes the following screen should appear as shown in the **IoT Dashboard** in Figure 134.

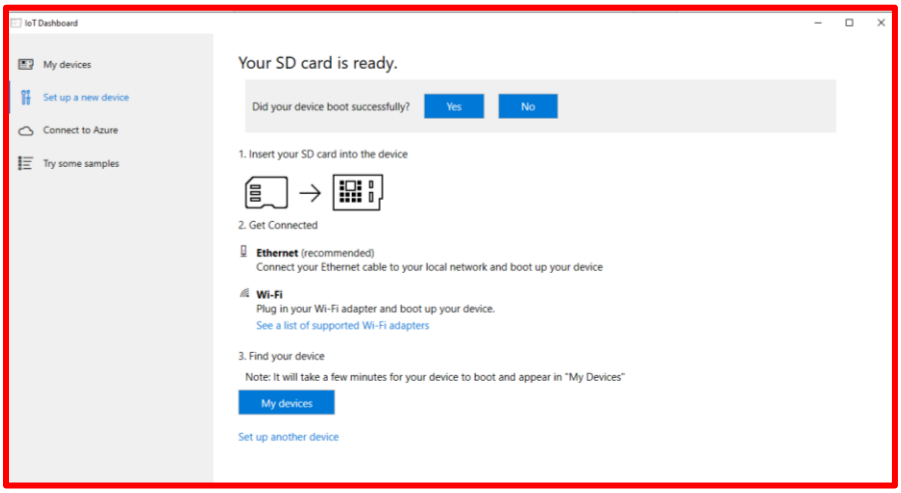


Figure 134

NOTE:

Once the installation completes there may appear several dialog windows with a message asking you to format the disk. Do not format the disk, it will erase all the data resulting in a do over. Remove the disk using the eject feature in the local workstation computer.

22.12 Mount the **Raspberry Pi** onto the **RPI Test jig** connect it to the Ethernet router and power it up. Observe the following screen appears as shown in Figure 135. The Screen is an HDMI monitor that is connected to the Raspberry Pi HDMI port.

Figure 135

22.13 In the **IoT Dashboard** select my devices and observe the following device appears as shown In Figure 136.

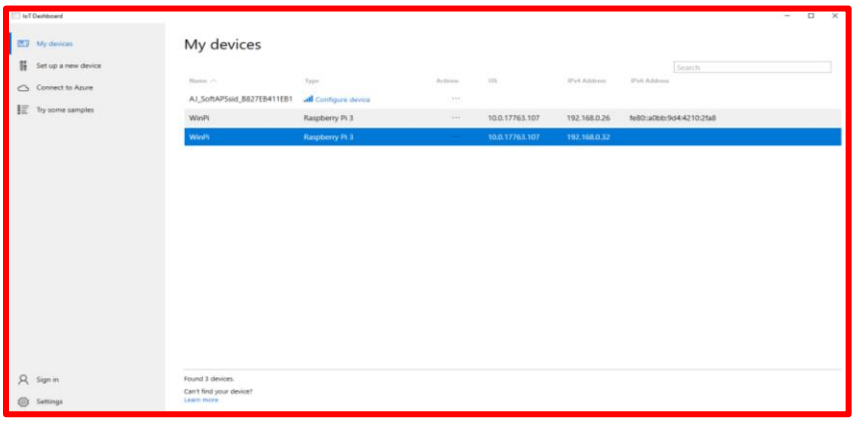


Figure 136

NOTE:

`The current image is for noncommercial use and does not have the security protection that is common in the IOT space these days. In the Appendix is a list of secure custom images containing a .ffl extension type.`

22.14 Verify the **WiFi Dongle** works in addition to verifying the Ethernet connection functions in the test jig. In the **MyDevices** of the **IoT Dashboard**, the **Raspberry PI** using the right mouse button select the **Power Shell** option. Observe the following appears as shown in Figure 137. In Figure 137 be sure to pay close attention to the IP address matches the same address as what the router has assigned to the raspberry pi.

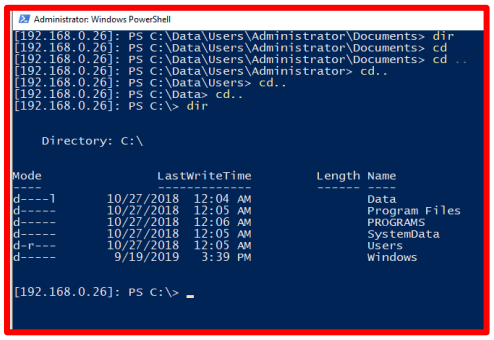


Figure 137

Application Deployment of MVC Application to a Windows IOT Device

Application deployment in this case will be a published project that will execute IIE server within the device. Deployment of the application will encompass a solution for the following sections.

23.0 Open the current solution using Visual Studio Code. Make the following change to the **launch settings** file as shown below in the red box.

```
"EmployeeManagementWebApplication": {
  "commandName": "Project",
  "launchBrowser": true,
  "environmentVariables": {
    "ASPNETCORE_ENVIRONMENT": "Development"
  },
  "applicationUrl": "http://*:89"
```

23.1 On the workstation computer, make the following change to the **Program.cs** file.

```
public class Program
{
    public static void Main(string[] args)
    {

        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
        .UseUrls("http://*:89")
        .UseStartup<Startup>();
}
```

23.2 Enter the following command to publish the solution for deployment
Observe the following occurs as shown in Figure 138

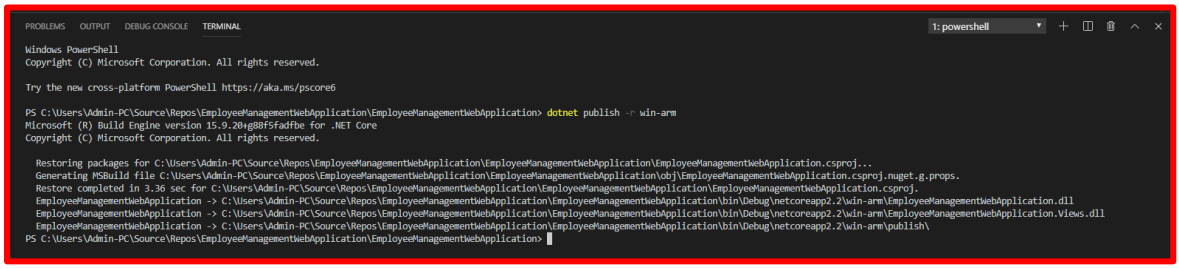


Figure 138

23.3 Log into the raspberry Pi remotely using the Windows IoT Dashboard, select launch PowerShell Enter the Raspberry Pi credentials navigate to the home directory and enter the following command

Start c:\windows\System32\ftpd.exe

23.4 Enter the following command to verify the ftp service is running observe the following appears as shown in Figure 139 in the red box.

get-process

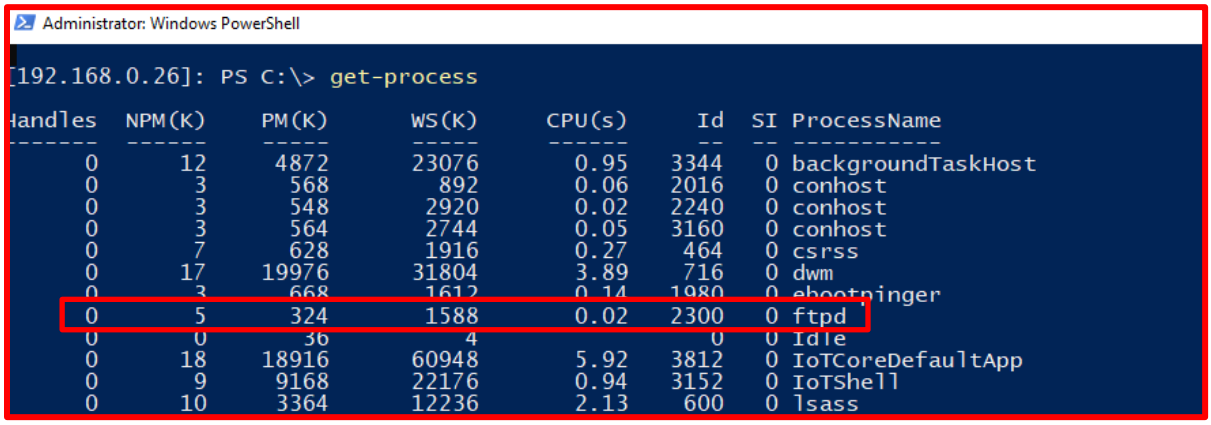


Figure 139

23.5 In file explorer enter the following command observe the following appears as shown in Figure 140.

ftp:\192.168.0.26

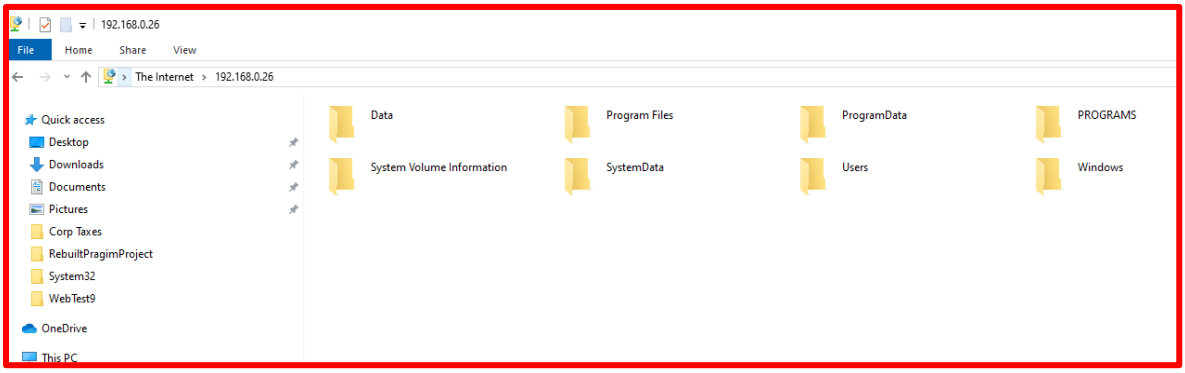


Figure 140

NOTE:

In order to load a page for MVC or any website the respective port must be opened in the IOT Raspberry PI. Apply the following command to open Port 89. If the command executes successfully a message will appear as OK in the command line.

23.6 Before Migration takes place, the Admin User needs to be logged in so that new users and new roles can be added. Once the migration takes place update the Users and Roles by replicating the steps in 23.26 – 23.27.

23.7 In the Power Shell console enter the following command as shown below.

netsh advfirewall firewall add rule name="Open Port 89" dir=in action=allow protocol=TCP localport=89

```
netsh advfirewall firewall add rule name="Open Port 88" dir=in action=allow protocol=TCP localport=88
```

23.8 Run the Application by navigating to the correct directory structure and run the application. Verify it runs successfully as shown below.

```
[192.168.0.32]: PS C:\Win-Arm\Publish> .\EmployeeManagementWebApplication.exe
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Data\Users\Administrator\AppData\Local\ASP.NET\DataProtection-Keys' as key r
pository and Windows DPAPI to encrypt keys at rest.
Hosting environment: Production
Content root path: C:\Win-Arm\Publish
Now listening on: http://[::]:89
Application started. Press Ctrl+C to shut down.
```

```
Administrator: Windows PowerShell
[192.168.0.26]: PS C:\Data\Users\Administrator\Documents> dir
[192.168.0.26]: PS C:\Data\Users\Administrator\Documents> cd
[192.168.0.26]: PS C:\Data\Users\Administrator\Documents> cd ..
[192.168.0.26]: PS C:\Data\Users\Administrator> cd..
[192.168.0.26]: PS C:\Data\Users> cd..
[192.168.0.26]: PS C:\Data> cd..
[192.168.0.26]: PS C:\> dir


    Directory: C:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----l        10/27/2018  12:04 AM                Data
d-----        10/27/2018  12:05 AM                Program Files
d-----        10/27/2018  12:06 AM                PROGRAMS
d-----        10/27/2018  12:05 AM                SystemData
d-r---        10/27/2018  12:05 AM                Users
d-----         9/19/2019   3:39 PM                Windows


[192.168.0.26]: PS C:\>
```

NOTE:

Steps 23.9 – 23.16 may not be necessary if SQLITE is already installed and configured in an earlier section of the procedure.

23.9 On the workstation computer add the following **Nuget Package** for the Sqlite database framework for entity framework core using the following library as shown in Figure 141.

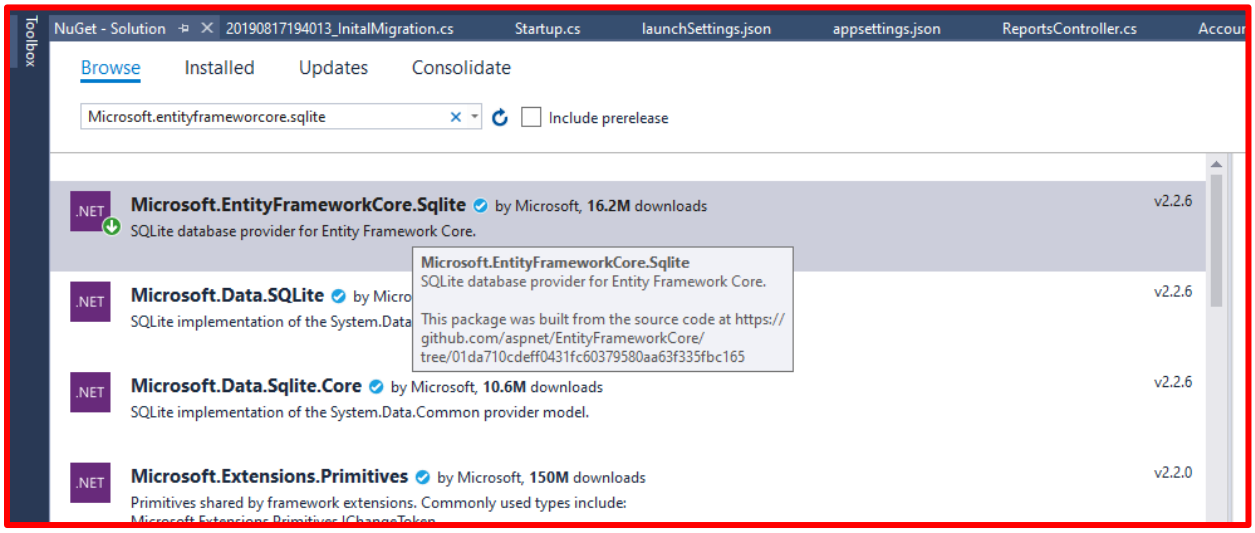


Figure 141

23.10 On the workstation computer make the following changes to the Startup class as shown in the redbox below. Change it from **UseSqlServer** to **UseSqlite**.

```
services.AddDbContextPool<AppDbContext>(
        options => options.UseSqlite(_config.GetConnectionString("EmployeeDBConnection")));
```

23.11 On the workstation computer create an initial migration for executing the database structure for **Sqlite** database.

23.12 On the workstation computer change the details for the **EmployeeDBConnection** to reflect the **Sqlite** database.

23.13 On the workstation computer update the database with the changes listed in the initial migration.

23.14 Run the application on the local workstation observe the following as shown in Figure 142.

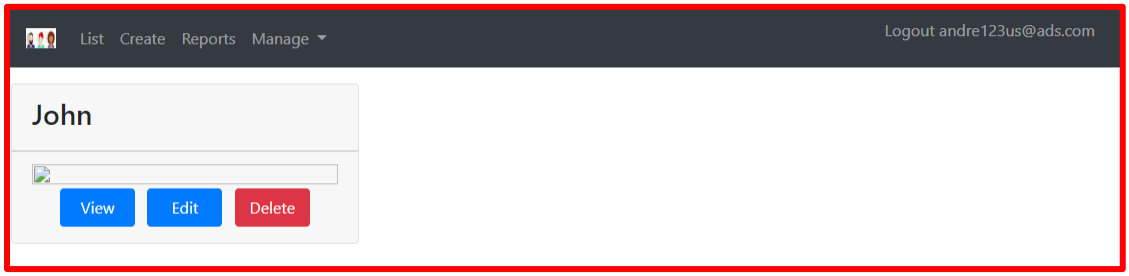


Figure 142

23.15 Add new users but do not add a picture to it observe the following appears as shown in Figure 143

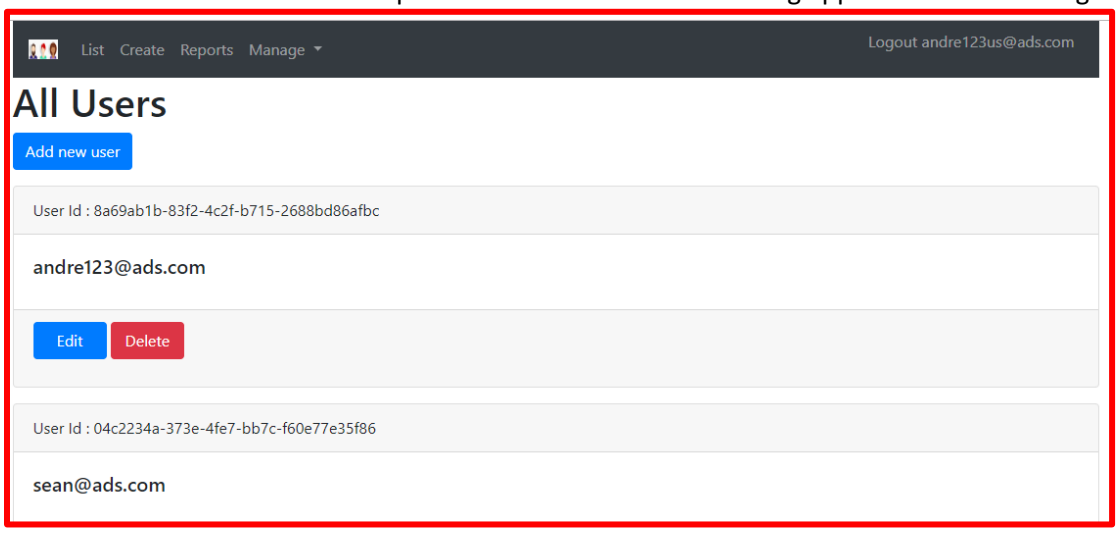


Figure 143

23.16 Verify the users exist in the **SQLite** database by using **SQLite Studio** as shown below.

Employees Table

	Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfir	PasswordHash	SecurityStamp	ConcurrencyStamp	PhoneNumI	PhoneNumI	TwoFactorI	LockoutEn
1	8a69ab1b-83f2-4c2f-b715-2688bd86afbc	andre123@ads.com	ANDRE123@ADS.COM	andre123@ads.com	ANDRE123@ADS.COM	0	AQAAAAEAACcQAAAAEAf0rIt...	PE2KMRWM76ISFSY6VCJ2CGF3ZZ23W3IB	c57737a3-75bb-4e39-b3d2-78db5474fcde	NULL	0	0	NULL
2	04c2234a-373e-4fe7-bb7c-f60e77e35f86	sean@ads.com	SEAN@ADS.COM	sean@ads.com	SEAN@ADS.COM	0	AQAAAAEAACcQAAAAEN9oQ...	U46XZVUY5FFLQSKS4JDBTTYEBNOND32Z	6dca79ef-a12c-4320-8388-d735abdddd31	NULL	0	0	NULL
3	b162b359-9625-4a13-bcfb-aee5cb5319f3	Thore@ads.com	THORE@ADS.COM	Thore@ads.com	THORE@ADS.COM	0	AQAAAAEAACcQAAAAEJvIA7...	XVX4BL4MTVLRV4QB7I6H7SHGDHRSSCSO	e6590c6f-6c3f-4338-9213-e3892b744dc0	NULL	0	0	NULL

AspNetUsers Table

Id	Name	Email	Departmen	PhotoPath	Classificatic	EmployeeS
1	John	John@ads.com	IT			

23.17 Change the **Startup.cs** class to reflect support for Sqlite as shown below, replace **DevelopmentDatabase** with **ProductionDatabase** statement.

```
public void ConfigureServices(IServiceCollection services)
{
    var DevelopmentDatabase = _config.GetConnectionString("EmployeeDBConnection");
    var ProductionDatabase = "Filename=employees.db";

    services.AddDbContextPool<AppDbContext>(
             options => options.UseSqlite(DevelopmentDatabase));

    services.AddMvc().AddXmlSerializerFormatters();

    services.AddScoped<IEmployeeRepository, SQLEmployeeRepository>();
    services.AddIdentity<ApplicationUser, IdentityRole>()
            .AddEntityFrameworkStores<AppDbContext>();

    services.AddMvc(options =>
    {
        var policy = new AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();
        options.Filters.Add(new AuthorizeFilter(policy));
    })

}
```

23.18 Publish the project for deployment to the raspberry pi. Use Visual Studio Code to make the deployment. Enter the following command at the solution root directory

Dotnet publish –r win10-arm

23.19 Navigate to the publish file directory make a copy of the folder containing all the program contents and copy it to the root directory of the raspberry pi as shown in Figure 144.

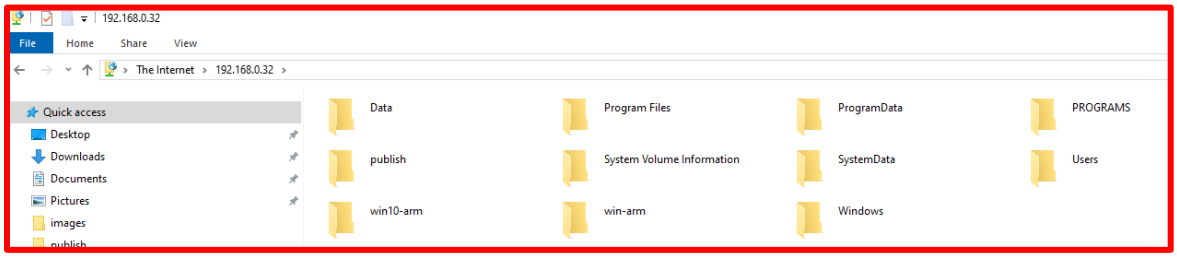


Figure 144

23.20 Copy the current **sqlite database** from the project and place it in the following directory location as shown in Figure 145

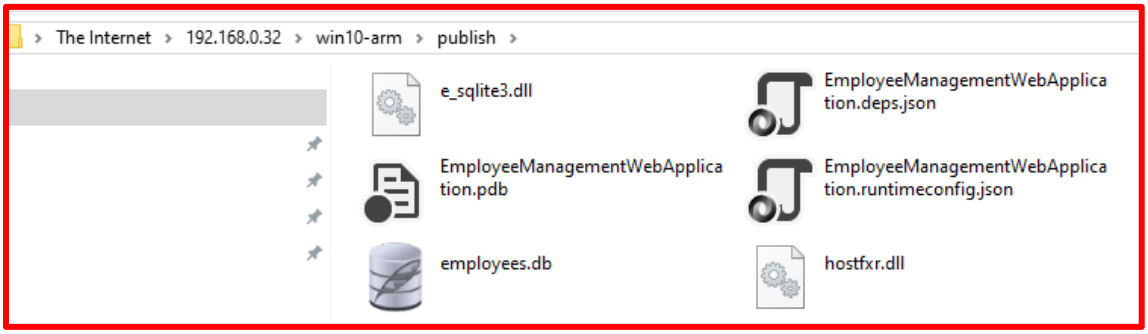


Figure 145

23.21 Open the Power Shell command navigate to the following file location enter the following command to kickoff the Employee Application. Observe the following appears in Figure 146.

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----l        10/27/2018  12:04 AM                Data
d-----        10/27/2018  12:05 AM                Program Files
d-----        10/27/2018  12:06 AM                PROGRAMS
d-----         9/20/2019   2:50 PM                publish
d-----        10/27/2018  12:05 AM                SystemData
d-r---        10/27/2018  12:05 AM                Users
d-----         9/23/2019  11:06 AM                win-arm
d-----         9/23/2019   2:56 PM                win10-arm
d-----         9/19/2019   3:39 PM                Windows


[192.168.0.32]: PS C:\> cd win10-arm
[192.168.0.32]: PS C:\win10-arm> cd publish
[192.168.0.32]: PS C:\win10-arm\publish> .\EmployeeManagementWebApplication.exe
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Data\Users\Administrator\AppData\Local\ASP.NET\DataProtection-Keys'
as key repository and Windows DPAPI to encrypt keys at rest.
Hosting environment: Production
Content root path: C:\win10-arm\publish
Now listening on: http://[::]:89
Application started. Press Ctrl+C to shut down.
```

Figure 146

23.22 In the browser navigate to the following URL: 192.168.0.32:89 observe the following appears as shown in Figure 147.

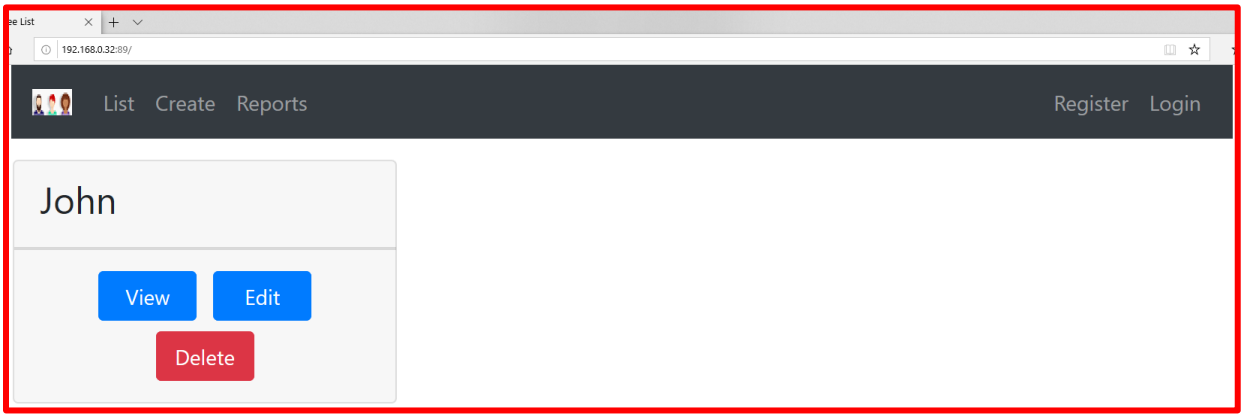


Figure 147

23.23 Test the Crud Operations on the database by Registering and logging in the user as displayed in the accompanying screen shots.

NOTE:

Since user andre123us@ads.com exists one should be able to login as this user. Since this user is not an admin user,the roles selection is not available for assignment of roles and claims as in Figure 148.

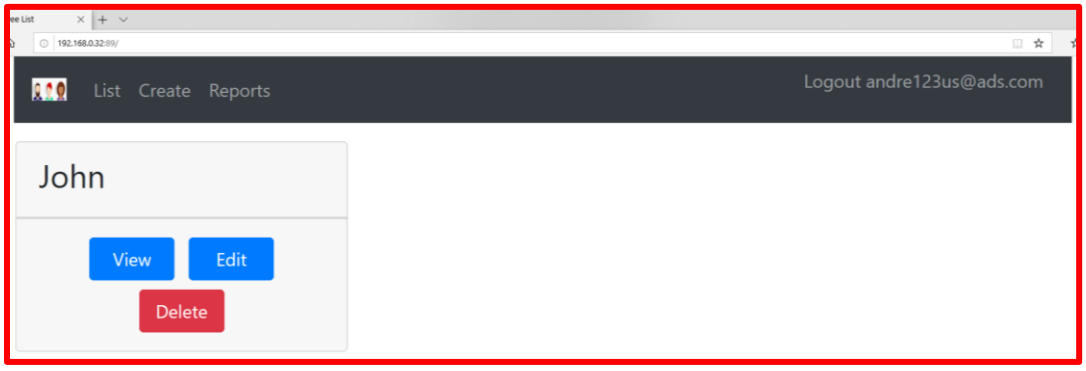


Figure 148

23.24 Register as a new user using the registration form as shown in Figure 149.

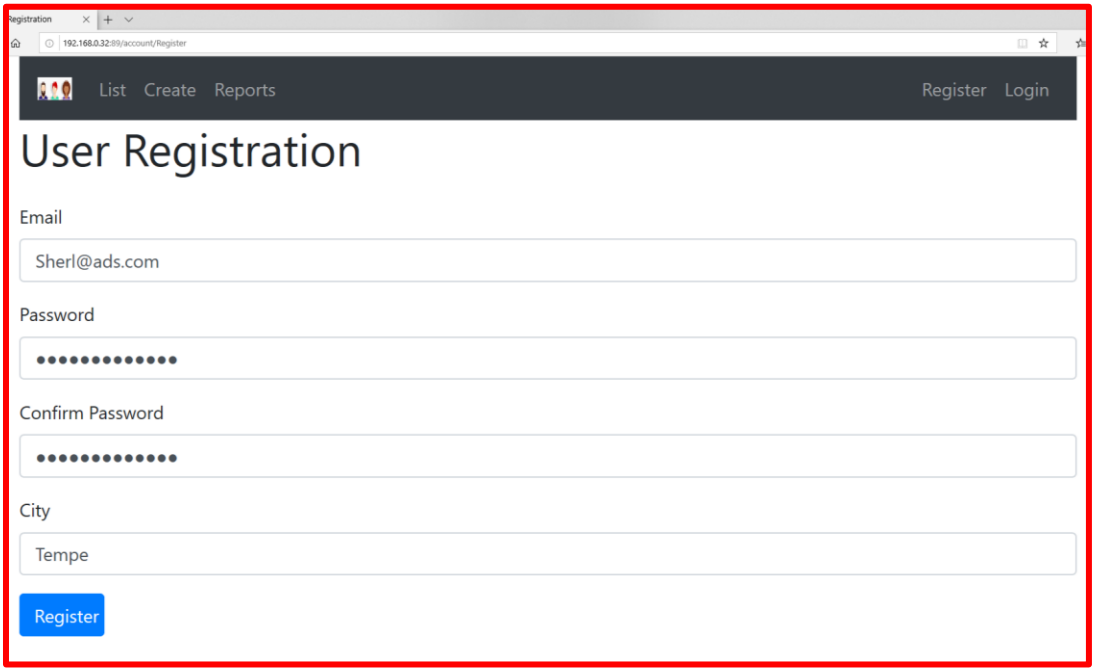


Figure 149

23.25 Once this user is registered, observe the following appears as shown in Figure 150.

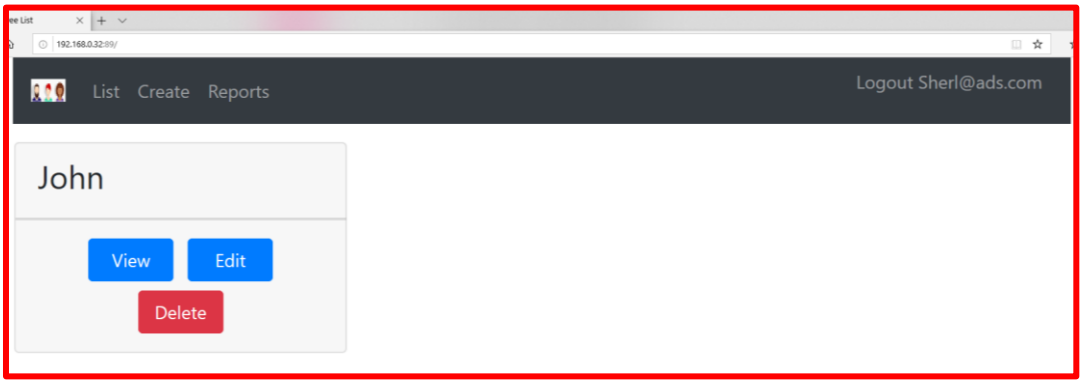


Figure 150

23.26 Enter the following command to terminate the application execution in the Raspberry pi

Ctrl + C

23.27 Observe the following appears as shown in Figure 151. Close the browser once the application stops running.

```
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[2]
      Request finished in 59.6545ms 304 text/css
The remote pipeline has been stopped.
    + CategoryInfo          : ResourceUnavailable: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : System.Management.Automation.Remoting.PSRemotingDataStructureException
```

Figure 151

23.28 Close out the **Power Shell** window and close out the Ftp explorer before powering off the raspberry pi.

23.29 Replace the **Launch Settings** file with the following content as shown below and change the Startup class configuration services setting from production mode to development mode.

NOTE:

This is normally not required if the data source is MSSQL but since this project is using Sqlite, and MSSQL as part of the software development process of this project, the data source definition in Configuration Services will need to be changed using the defined objects for Development and Production.

Development Launch Settings

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:50286",
      "sslPort": 0
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "EmployeeManagementWebApplication": {
      "commandName": "Project",
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

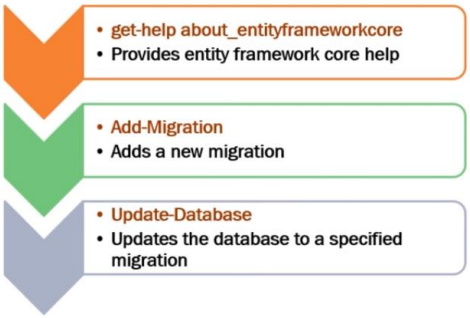
Appendix A {ASP.Net Core Summary Notes}

A-1 Configure Services Section DI container states

Service Type	In the scope of a given http request	Across different http requests
Scoped Service	Same Instance	New Instance
Transient Service	New Instance	New Instance
Singleton Service	Same Instance	Same Instance

A-2 Configure Service Definitions

A-2 Migration process



November 2, 2019
Prepared by Andre Masters
Revision 1.0

Appendix B {ASP.Net Core Entity Framework Summary Notes}

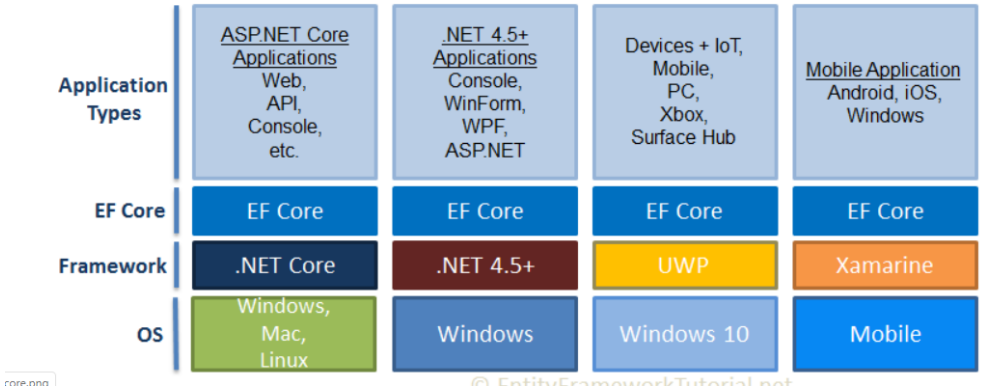
Entity Framework Core

Entity Framework Core is the new version of Entity Framework after EF 6.x. It is open-source, lightweight, extensible and a cross-platform version of Entity Framework data access technology.

Entity Framework is an Object/Relational Mapping (O/RM) framework. It is an enhancement to ADO.NET that gives developers an automated mechanism for accessing & storing the data in the database.

EF Core is intended to be used with .NET Core applications. However, it can also be used with standard .NET 4.5+ framework based applications.

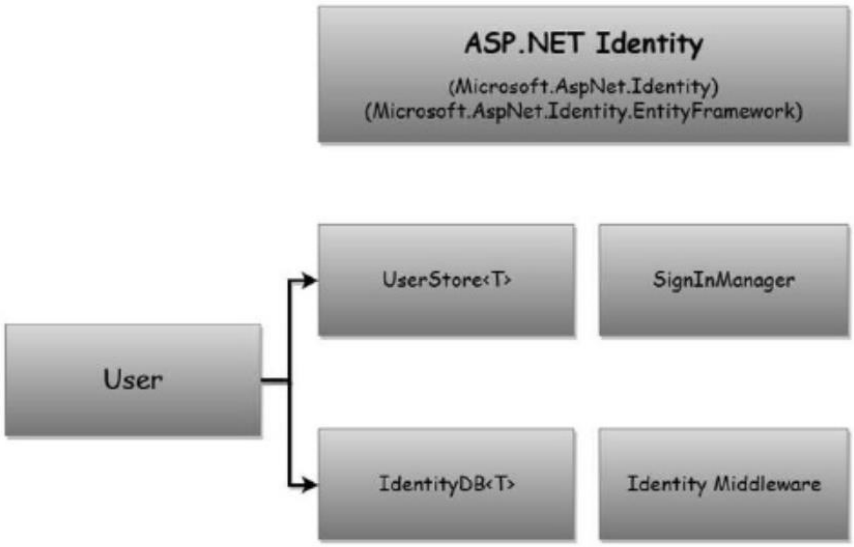
The following figure illustrates the supported application types, .NET Frameworks and OSs.



Appendix C {ASP.Net Core Identity Framework Summary Notes}

ASP.NET Core Identity is a membership system which allows you to add login functionality to your application. Users can create an account and login with a user name and password or they can use an external login providers such as Facebook, Google, Microsoft Account, Twitter and more.

You can configure ASP.NET Core Identity to use a SQL Server database to store user names, passwords, and profile data. Alternatively, you can use your own persistent store to store data in another other persistent storage, such as Azure Table Storage.



Appendix D {Raspberry PI Summary Notes}

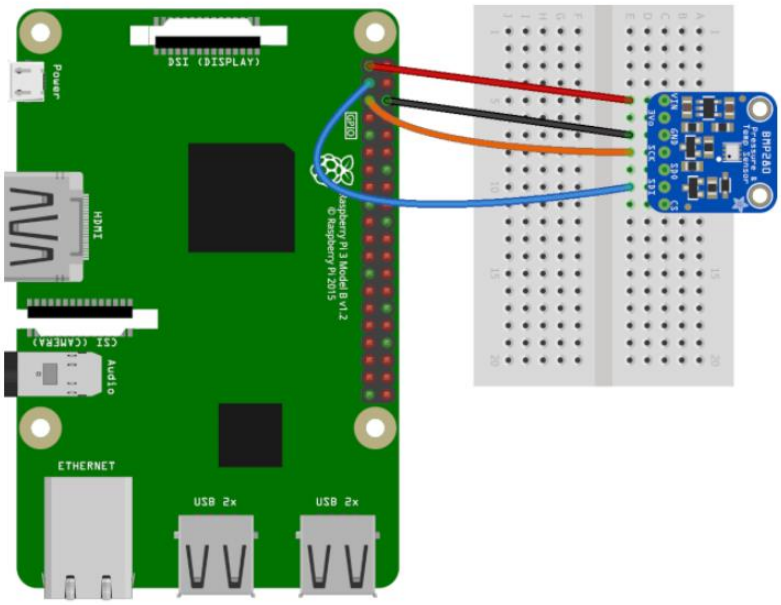
The .NET Core IoT Library is supported on Linux, and Windows IoT Core, across ARM and Intel processor architectures. See the .NET Core IoT Library Roadmap for more information.

System.Device.Gpio

The System.Device.Gpio package supports general-purpose I/O (GPIO) pins, PWM, I2C, SPI and related interfaces for interacting with low-level hardware pins to control hardware sensors, displays and input devices on single-board-computers; Raspberry Pi, BeagleBoard, HummingBoard, ODROID, and other single-board-computers that are supported by Linux and Windows 10 IoT Core.

Iot.Device.Bindings

The .NET Core IoT Repository contains IoT.Device.Bindings, a growing set of community-maintained device bindings for IoT components that you can use with your .NET Core applications. If you can't find what you need then porting your own C/C++ driver libraries to .NET Core and C# is pretty straight forward too.



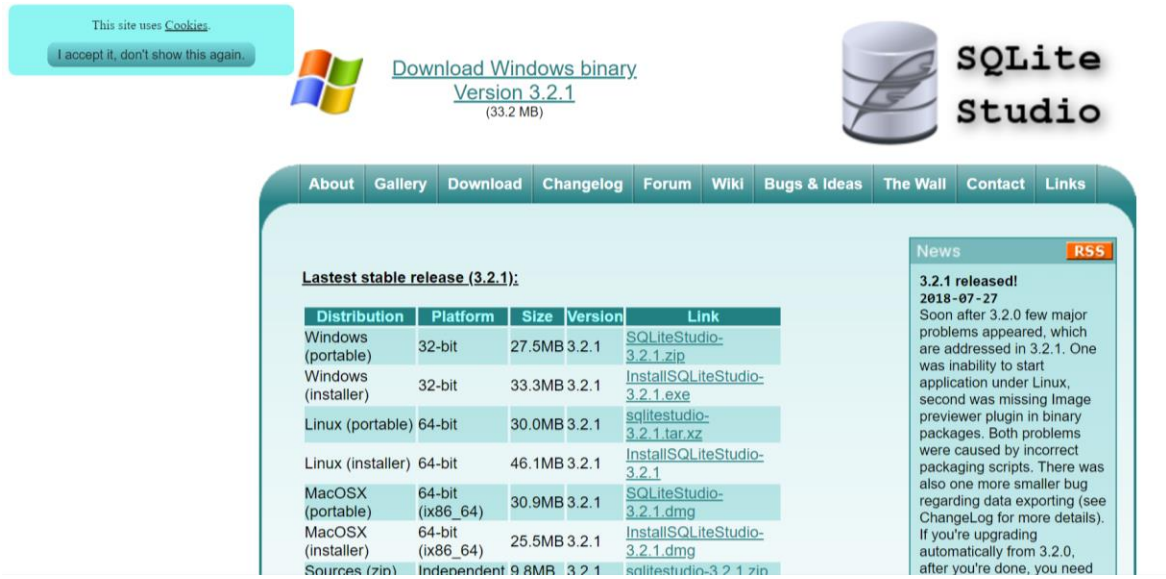
Appendix E {SQL Server Summary Notes}

Create SQL Test Cases using MSSQL. Create the following Schema file as shown below. Create a SpCreateEmployeeTable as shown below.

```
alter procedure SpCreateEmployeeTable
as
Begin
if OBJECT_ID('Employee') is not null
drop table Employee
else
Create table Employee(
ID int primary key identity(1,1),
EmployeeName varchar(30) not null,
Department int,
PhotoPath varchar(90),
Classification int,
Salary decimal,
);
End

alter table Employee
Add Constraint CHK_EmployeeSalary check (Salary > 0.0 and Salary < 250000.0)
alter table Employee
Add Constraint CHK_Classification check (Classification > 0)
alter table Employee
Add Constraint CHK_Department check (Department > 0)


alter procedure SpFillEmployeeTable(@EmployeeName as Varchar(30),@Department as int,@PhotoPath as
varchar(90),@Classification as int,@Salary as decimal)
as
begin
insert into Employee(EmployeeName, Department, PhotoPath, Classification, Salary)
            values(@EmployeeName,@Department,@PhotoPath,@Classification,@Salary)
select * from Employee
end

exec SpCreateEmployeeTable
exec SpFillEmployeeTable @EmployeeName = 'Paul',
                         @Department = -7,
                                                    @PhotoPath =
'\Home\EmployeeDetails\Images\MonoMan.png',
                                                    @Classification = 2,
                                                    @Salary = -55000;

select * from Employee
describe Employee
delete from Employee
where Department < 0
```

November 2, 2019
Prepared by Andre Masters
Revision 1.0

Appendix G {SQLite Server Summary Notes}



NOTE:

SQLITE Studio does not support stored procedures. It also does not have allany of the features of MSSQL since it's a file based database. That is why from an architecture perspective performing database logic in the database is not a sustainable strategy.

November 2, 2019
Prepared by Andre Masters
Revision 1.0

Appendix I {iText Color Options}

NOTE:

Select any color in the list, remove the last value 0xff, when using the **SetRGBColorFill** option of the PdfContentByte.

```
    NAMES["aliceblue"] = new int[] { 0xf0, 0xf8, 0xff, 0xff };

    NAMES["antiquewhite"] = new int[] { 0xfa, 0xeb, 0xd7, 0xff };

    NAMES["aqua"] = new int[] { 0x00, 0xff, 0xff, 0xff };

    NAMES["aquamarine"] = new int[] { 0x7f, 0xff, 0xd4, 0xff };

    NAMES["azure"] = new int[] { 0xf0, 0xff, 0xff, 0xff };

    NAMES["beige"] = new int[] { 0xf5, 0xf5, 0xdc, 0xff };

    NAMES["bisque"] = new int[] { 0xff, 0xe4, 0xc4, 0xff };

    NAMES["black"] = new int[] { 0x00, 0x00, 0x00, 0xff };

    NAMES["blanchedalmond"] = new int[] { 0xff, 0xeb, 0xcd, 0xff };

    NAMES["blue"] = new int[] { 0x00, 0x00, 0xff, 0xff };

    NAMES["blueviolet"] = new int[] { 0x8a, 0x2b, 0xe2, 0xff };

    NAMES["brown"] = new int[] { 0xa5, 0x2a, 0x2a, 0xff };

    NAMES["burlywood"] = new int[] { 0xde, 0xb8, 0x87, 0xff };

    NAMES["cadetblue"] = new int[] { 0x5f, 0x9e, 0xa0, 0xff };

    NAMES["chartreuse"] = new int[] { 0x7f, 0xff, 0x00, 0xff };

    NAMES["chocolate"] = new int[] { 0xd2, 0x69, 0x1e, 0xff };

    NAMES["coral"] = new int[] { 0xff, 0x7f, 0x50, 0xff };

    NAMES["cornflowerblue"] = new int[] { 0x64, 0x95, 0xed, 0xff };

    NAMES["cornsilk"] = new int[] { 0xff, 0xf8, 0xdc, 0xff };

    NAMES["crimson"] = new int[] { 0xdc, 0x14, 0x3c, 0xff };

    NAMES["cyan"] = new int[] { 0x00, 0xff, 0xff, 0xff };

    NAMES["darkblue"] = new int[] { 0x00, 0x00, 0x8b, 0xff };
```

```
    NAMES["darkcyan"] = new int[] { 0x00, 0x8b, 0x8b, 0xff };
    NAMES["darkgoldenrod"] = new int[] { 0xb8, 0x86, 0x0b, 0xff };
    NAMES["darkgray"] = new int[] { 0xa9, 0xa9, 0xa9, 0xff };
    NAMES["darkgreen"] = new int[] { 0x00, 0x64, 0x00, 0xff };
    NAMES["darkkhaki"] = new int[] { 0xbd, 0xb7, 0x6b, 0xff };
    NAMES["darkmagenta"] = new int[] { 0x8b, 0x00, 0x8b, 0xff };
    NAMES["darkolivegreen"] = new int[] { 0x55, 0x6b, 0x2f, 0xff };
    NAMES["darkorange"] = new int[] { 0xff, 0x8c, 0x00, 0xff };
    NAMES["darkorchid"] = new int[] { 0x99, 0x32, 0xcc, 0xff };
    NAMES["darkred"] = new int[] { 0x8b, 0x00, 0x00, 0xff };
    NAMES["darksalmon"] = new int[] { 0xe9, 0x96, 0x7a, 0xff };
    NAMES["darkseagreen"] = new int[] { 0x8f, 0xbc, 0x8f, 0xff };
    NAMES["darkslateblue"] = new int[] { 0x48, 0x3d, 0x8b, 0xff };
    NAMES["darkslategray"] = new int[] { 0x2f, 0x4f, 0x4f, 0xff };
    NAMES["darkturquoise"] = new int[] { 0x00, 0xce, 0xd1, 0xff };
    NAMES["darkviolet"] = new int[] { 0x94, 0x00, 0xd3, 0xff };
    NAMES["deeppink"] = new int[] { 0xff, 0x14, 0x93, 0xff };
    NAMES["deepskyblue"] = new int[] { 0x00, 0xbf, 0xff, 0xff };
    NAMES["dimgray"] = new int[] { 0x69, 0x69, 0x69, 0xff };
    NAMES["dodgerblue"] = new int[] { 0x1e, 0x90, 0xff, 0xff };
    NAMES["firebrick"] = new int[] { 0xb2, 0x22, 0x22, 0xff };
    NAMES["floralwhite"] = new int[] { 0xff, 0xfa, 0xf0, 0xff };
    NAMES["forestgreen"] = new int[] { 0x22, 0x8b, 0x22, 0xff };
    NAMES["fuchsia"] = new int[] { 0xff, 0x00, 0xff, 0xff };
    NAMES["gainsboro"] = new int[] { 0xdc, 0xdc, 0xdc, 0xff };
```

```
    NAMES["ghostwhite"] = new int[] { 0xf8, 0xf8, 0xff, 0xff };
    NAMES["gold"] = new int[] { 0xff, 0xd7, 0x00, 0xff };
    NAMES["goldenrod"] = new int[] { 0xda, 0xa5, 0x20, 0xff };
    NAMES["gray"] = new int[] { 0x80, 0x80, 0x80, 0xff };
    NAMES["green"] = new int[] { 0x00, 0x80, 0x00, 0xff };
    NAMES["greenyellow"] = new int[] { 0xad, 0xff, 0x2f, 0xff };
    NAMES["honeydew"] = new int[] { 0xf0, 0xff, 0xf0, 0xff };
    NAMES["hotpink"] = new int[] { 0xff, 0x69, 0xb4, 0xff };
    NAMES["indianred"] = new int[] { 0xcd, 0x5c, 0x5c, 0xff };
    NAMES["indigo"] = new int[] { 0x4b, 0x00, 0x82, 0xff };
    NAMES["ivory"] = new int[] { 0xff, 0xff, 0xf0, 0xff };
    NAMES["khaki"] = new int[] { 0xf0, 0xe6, 0x8c, 0xff };
    NAMES["lavender"] = new int[] { 0xe6, 0xe6, 0xfa, 0xff };
    NAMES["lavenderblush"] = new int[] { 0xff, 0xf0, 0xf5, 0xff };
    NAMES["lawngreen"] = new int[] { 0x7c, 0xfc, 0x00, 0xff };
    NAMES["lemonchiffon"] = new int[] { 0xff, 0xfa, 0xcd, 0xff };
    NAMES["lightblue"] = new int[] { 0xad, 0xd8, 0xe6, 0xff };
    NAMES["lightcoral"] = new int[] { 0xf0, 0x80, 0x80, 0xff };
    NAMES["lightcyan"] = new int[] { 0xe0, 0xff, 0xff, 0xff };
    NAMES["lightgoldenrodyellow"] = new int[] { 0xfa, 0xfa, 0xd2, 0xff };
    NAMES["lightgreen"] = new int[] { 0x90, 0xee, 0x90, 0xff };
    NAMES["lightgrey"] = new int[] { 0xd3, 0xd3, 0xd3, 0xff };
    NAMES["lightpink"] = new int[] { 0xff, 0xb6, 0xc1, 0xff };
    NAMES["lightsalmon"] = new int[] { 0xff, 0xa0, 0x7a, 0xff };
    NAMES["lightseagreen"] = new int[] { 0x20, 0xb2, 0xaa, 0xff };
```

```
    NAMES["lightskyblue"] = new int[] { 0x87, 0xce, 0xfa, 0xff };
    NAMES["lightslategray"] = new int[] { 0x77, 0x88, 0x99, 0xff };
    NAMES["lightsteelblue"] = new int[] { 0xb0, 0xc4, 0xde, 0xff };
    NAMES["lightyellow"] = new int[] { 0xff, 0xff, 0xe0, 0xff };
    NAMES["lime"] = new int[] { 0x00, 0xff, 0x00, 0xff };
    NAMES["limegreen"] = new int[] { 0x32, 0xcd, 0x32, 0xff };
    NAMES["linen"] = new int[] { 0xfa, 0xf0, 0xe6, 0xff };
    NAMES["magenta"] = new int[] { 0xff, 0x00, 0xff, 0xff };
    NAMES["maroon"] = new int[] { 0x80, 0x00, 0x00, 0xff };
    NAMES["mediumaquamarine"] = new int[] { 0x66, 0xcd, 0xaa, 0xff };
    NAMES["mediumblue"] = new int[] { 0x00, 0x00, 0xcd, 0xff };
    NAMES["mediumorchid"] = new int[] { 0xba, 0x55, 0xd3, 0xff };
    NAMES["mediumpurple"] = new int[] { 0x93, 0x70, 0xdb, 0xff };
    NAMES["mediumseagreen"] = new int[] { 0x3c, 0xb3, 0x71, 0xff };
    NAMES["mediumslateblue"] = new int[] { 0x7b, 0x68, 0xee, 0xff };
    NAMES["mediumspringgreen"] = new int[] { 0x00, 0xfa, 0x9a, 0xff };
    NAMES["mediumturquoise"] = new int[] { 0x48, 0xd1, 0xcc, 0xff };
    NAMES["mediumvioletred"] = new int[] { 0xc7, 0x15, 0x85, 0xff };
    NAMES["midnightblue"] = new int[] { 0x19, 0x19, 0x70, 0xff };
    NAMES["mintcream"] = new int[] { 0xf5, 0xff, 0xfa, 0xff };
    NAMES["mistyrose"] = new int[] { 0xff, 0xe4, 0xe1, 0xff };
    NAMES["moccasin"] = new int[] { 0xff, 0xe4, 0xb5, 0xff };
    NAMES["navajowhite"] = new int[] { 0xff, 0xde, 0xad, 0xff };
    NAMES["navy"] = new int[] { 0x00, 0x00, 0x80, 0xff };
    NAMES["oldlace"] = new int[] { 0xfd, 0xf5, 0xe6, 0xff };
```

```
    NAMES["olive"] = new int[] { 0x80, 0x80, 0x00, 0xff };
    NAMES["olivedrab"] = new int[] { 0x6b, 0x8e, 0x23, 0xff };
    NAMES["orange"] = new int[] { 0xff, 0xa5, 0x00, 0xff };
    NAMES["orangered"] = new int[] { 0xff, 0x45, 0x00, 0xff };
    NAMES["orchid"] = new int[] { 0xda, 0x70, 0xd6, 0xff };
    NAMES["palegoldenrod"] = new int[] { 0xee, 0xe8, 0xaa, 0xff };
    NAMES["palegreen"] = new int[] { 0x98, 0xfb, 0x98, 0xff };
    NAMES["paleturquoise"] = new int[] { 0xaf, 0xee, 0xee, 0xff };
    NAMES["palevioletred"] = new int[] { 0xdb, 0x70, 0x93, 0xff };
    NAMES["papayawhip"] = new int[] { 0xff, 0xef, 0xd5, 0xff };
    NAMES["peachpuff"] = new int[] { 0xff, 0xda, 0xb9, 0xff };
    NAMES["peru"] = new int[] { 0xcd, 0x85, 0x3f, 0xff };
    NAMES["pink"] = new int[] { 0xff, 0xc0, 0xcb, 0xff };
    NAMES["plum"] = new int[] { 0xdd, 0xa0, 0xdd, 0xff };
    NAMES["powderblue"] = new int[] { 0xb0, 0xe0, 0xe6, 0xff };
    NAMES["purple"] = new int[] { 0x80, 0x00, 0x80, 0xff };
    NAMES["red"] = new int[] { 0xff, 0x00, 0x00, 0xff };
    NAMES["rosybrown"] = new int[] { 0xbc, 0x8f, 0x8f, 0xff };
    NAMES["royalblue"] = new int[] { 0x41, 0x69, 0xe1, 0xff };
    NAMES["saddlebrown"] = new int[] { 0x8b, 0x45, 0x13, 0xff };
    NAMES["salmon"] = new int[] { 0xfa, 0x80, 0x72, 0xff };
    NAMES["sandybrown"] = new int[] { 0xf4, 0xa4, 0x60, 0xff };
    NAMES["seagreen"] = new int[] { 0x2e, 0x8b, 0x57, 0xff };
    NAMES["seashell"] = new int[] { 0xff, 0xf5, 0xee, 0xff };
    NAMES["sienna"] = new int[] { 0xa0, 0x52, 0x2d, 0xff };
```

```
NAMES["silver"] = new int[] { 0xc0, 0xc0, 0xc0, 0xff };
NAMES["skyblue"] = new int[] { 0x87, 0xce, 0xeb, 0xff };
NAMES["slateblue"] = new int[] { 0x6a, 0x5a, 0xcd, 0xff };
NAMES["slategray"] = new int[] { 0x70, 0x80, 0x90, 0xff };
NAMES["snow"] = new int[] { 0xff, 0xfa, 0xfa, 0xff };
NAMES["springgreen"] = new int[] { 0x00, 0xff, 0x7f, 0xff };
NAMES["steelblue"] = new int[] { 0x46, 0x82, 0xb4, 0xff };
NAMES["tan"] = new int[] { 0xd2, 0xb4, 0x8c, 0xff };
NAMES["teal"] = new int[] { 0x00, 0x80, 0x80, 0xff };
NAMES["thistle"] = new int[] { 0xd8, 0xbf, 0xd8, 0xff };
NAMES["tomato"] = new int[] { 0xff, 0x63, 0x47, 0xff };
NAMES["transparent"] = new int[] { 0xff, 0xff, 0xff, 0x00 };
NAMES["turquoise"] = new int[] { 0x40, 0xe0, 0xd0, 0xff };
NAMES["violet"] = new int[] { 0xee, 0x82, 0xee, 0xff };
NAMES["wheat"] = new int[] { 0xf5, 0xde, 0xb3, 0xff };
NAMES["white"] = new int[] { 0xff, 0xff, 0xff, 0xff };
NAMES["whitesmoke"] = new int[] { 0xf5, 0xf5, 0xf5, 0xff };
NAMES["yellow"] = new int[] { 0xff, 0xff, 0x00, 0xff };
NAMES["yellowgreen"] = new int[] { 0x9a, 0xcd, 0x32, 0xff };
```

Credits and References

This procedure is based on the teaching materials offered by PragimTechnologies video series for ASP.Net Core 2.2. Most of the slides and content came from this site. Also, credit is due to the following websites and authors

- C# Corner – Call Web API using JQuery AJAX in ASP.NET Core {Nishan} www.c-sharpcorner.com
- Code Project - Scaffolding ASP.NET Core 2 with CatFactory {Hherzl} www.codeproject.com
- Code Affection – Mosh "Creating web services and consuming with Angular 7" video series.
- Pro ASP.NET Core 2 {Scott Freeman}
- "ASP.Net Core 2.2 & 3 REST API" Nick Chapsas video series
- Raspberry Pi Foundation https://www.raspberrypi.org/
- Edureka "How to install Windows 10 on a Raspberry Pi 3b+" video series
- Windows IOT "Getting Started – Windows 10 IOT Core + Raspberry Pi 3 video series
- Iot Dashboard - https://developer.microsoft.com/en-us/windows/iot/docs/iotdashboard
- Binary Intellect Use Ajax To Perform CRUD Operations In ASP.NET Core Razor Page www.binaryintellect.com
- Ken Haggerty "ASP.NET Core 2.2 – AJAX get ad Post Methods" www.kenhaggerty.com
- Sitepoint "An Editable Grid with jQuery, Bootstrap and Shield" www.sitepoint.com/editable-grid-jquery-shield-ui-lite/
- Gijgo "Boostrap Editable Table" www.gijgo.com/grid/demos/boostrap-grid-inline-edit
- Github https://github.com/apmaster/EmployeeManagementDcoumentation
- Github https://github.com/apmaster/EmployeeManagermentWeb
- Gitkracken – www.gitkracken.com
- Itexsharp – www.itexsharp.com
- Tutorialspoint – SQLite – Overview – Tutorialspoint www.tutorialspoint.com/sqlite/sqlite_overview.htm
- SQLite – sqlite.org
- SQLite Studio - https://sqlitestudio.pl/index.rvt
- Microsoft SQL Server - https://www.microsoft.com/en-us/sql-server/sql-server-2017
- Tutorialspoint – LEARN MS SQL SERVER database management system https://www.tutorialspoint.com/ms_sql_server/ms_sql_server_overview.htm
- jakeydocs.readthedocs.io https://jakeydocs.readthedocs.io/en/latest/security/authentication/identity.html

November 2, 2019
Prepared by Andre Masters
Revision 1.0

November 2, 2019
Prepared by Andre Masters
Revision 1.0