ADP631

Formative Assessment 1

2025

Jaydon Ellick
20230443

Phase 1:

Research on existing municipality management systems:

Municipality management systems are software to help automate and streamline the functions of municipal governments. They help manage services, allows for engagement with their citizens and more important operations.

With them, managing and storing of their citizens information has never been easier and in turn the citizens can request services and allow the government to be in the know about problems. It is also useful for their own employees, allowing them to manage their staff's shifts, assignments salary etc. With all this data they can improve their area and offer quality service.

Popular systems include Trimble CityWorks, Accela, CityView.

With CityWorks, you can track assets such as roads, water system and more with the help of Geographic information system, helping for mapping where exactly most issues or other services occur. They allow municipalities to track and resolve service requests from their inhabitants and customers. They offer dashboards so that their staff can make data-driven decisions.

Citation:

Trimble. (n/d.). Cityworks – Public asset and work management software. Trimble Asset Lifecycle Management. Retrieved from https://assetlifecycle.trimble.com/en/products/software/cityworks

---

Accela offers a powerful cloud-based platform for local governments. It assists them with overseeing services like planning, public works, code enforcement. It streamlines workflows, increases transparency and allows customers to apply, pay and track services online.

Citation:

Accela. (n.d.). Powering digital government. Accela. Retrieved April 4, 2025, from https://www.accela.com

---

Cityview reduces calls to staff but also has online access to submit services, make payments and other applications. With streamlined workflow and business processes, it allows municipalities to be more efficient, which would reduce costs and allow them to invest in increasing their population's happiness. It is highly flexible and can offer numerous benefits to whatever municipalities need.

They can track and manage prime building places, view data on location-based land management activities with Cityview. Monitor and oversee enforcement events from the filing of a citizen complaint until the conclusion of the case. Allow easy animal licencing services for pet lovers, cemetery management and marriage or death registration.

Citation:

Municipal Software. (n.d.). CityView Suite. Retrieved April 1, 2025, from https://www.municipalsoftware.com/cityview-suite/

System requirements and functionalities.

The municipality system would have a method citizens can register with their information, update their profile and view pasts services and reports they have done. They should be allowed to submit service requests, track the status of these and notifications when the status changes. Their emails should be unique.

Staff should be able to register, with their details and be able to be assigned to services/requests. They should be able to track their pay, performance and attendance.

Citizens should also be able to make reports on issues such as safety concerns and be able to track the progress of this with staff being able to manage these and resolves these issues.

The municipality should be able to manage the staff information, updating, adding and deleting staff if needed. Their emails should be unique.

The websites should be fast and can handle multiple concurrent users. Implement role-based access to staff for important features on the website.

The website should be fully responsive on mobile devices and have an easy-to-understand user interface. Most actions load under a second.

Technical Requirements

ASP.NET EMPTY

Razor Pages, CSS

Entity Framework Core with SQL Server

xUnit with EF Core In-Memory DB for testing and in-memory to test without corrupting real data.

Real-time validation to ensure data integrity.

Project Proposal

System Objectives:

The goal of the system is to improve the management and operation of municipal services, allowing for efficient management of service requests, reports, staff and citizens. These being:

A system to manage citizen information such as their contact info, names, addresses and more.

A system to keep track of and resolve service requests from their citizens.

A system to keep track of staff of the municipality.

A system to allow citizens to report emergencies and other issues which can be managed by staff.

Scope:

Included:

Adding, updating, deleting and view citizen information will be included.

Creating, updating and resolving service requests.

Managing staff information and adding, removing, updating it.

Report handling: Adding, removing, seeing its details and changing its status.

Not Included:

Generating reports on service requests, incident reports and other data.

Logging hasn't been developed.

Payment integration isn't included.

It is only a website for now.

GPS integration hasn't been developed to pinpoint locations of reports and services.

Features:

Basic Citizen management.

Service Request Management.

Staff Management.

Reports Management.

Why is the System Useful:

It minimizes errors since data is validated, reducing common mistakes.

Efficiency is enhanced; staff can access date without needing to request them.

Transactions and other interactions are logged, with anyone being able to see the status updates making the new system more transparent for its citizens and other users.

It improved communication and lowers response time, customers can get auto updates and track progress. Staff can respond faster due to it being in real time.

Phase 2:

Key Entities

Table Citizens {

CitizenID integer [primary key]

FullName varchar [not null]

AddressID integer [not null]

PhoneNumber varchar [not null]

```
  Email varchar [unique]
  DateOfBirth datetime
  RegistrationDate datetime [default: 'current date']
}
Citizens may submit many reports and service requests.
Table ServiceRequests {
  RequestID integer [primary key]
  CitizenID integer [ref: > Citizens.CitizenID]
  ServiceType varchar [not null]
  RequestDate datetime [default: 'current date']
  Status varchar [default: 'Pending']
}
Citizens may submit many reports and service requests.
Table Staff {
  StaffID integer [primary key]
  FullName varchar [not null]
  Position varchar [not null]
  Department varchar [not null]
  Email varchar [unique, not null]
  PhoneNumber varchar [not null]
  HireDate datetime [not null]
}
Table Reports {
  ReportID integer [primary key]
  CitizenID integer [ref: > Citizens.CitizenID]
  ReportType varchar [not null]
  Details varchar [not null]
  SubmissionDate datetime [default: 'current date']
  Status varchar [default: 'Under Review']
} Citizens may submit many reports and service requests.
```

Relationships:

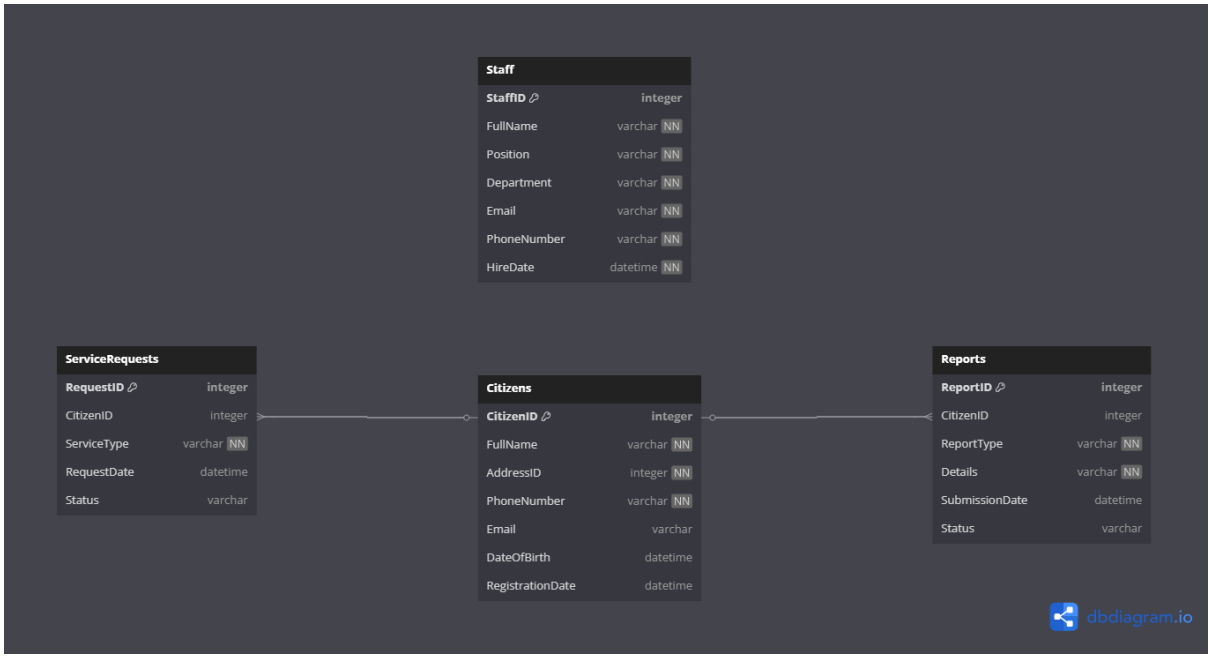
Citizens should have a one-to-many relationship with Service Request table. With the citizen being able to make multiple requests. Citizen Id being the foreign key in the request table.

Citizens should have a one-to-many relationship with the Reports table, with the citizen being able to make multiple reports. Citizen Id being the foreign key in the reports table.

Define database schema using SQL Server:

In separate file named ‘Phase2_DatabaseSchema.sql’.

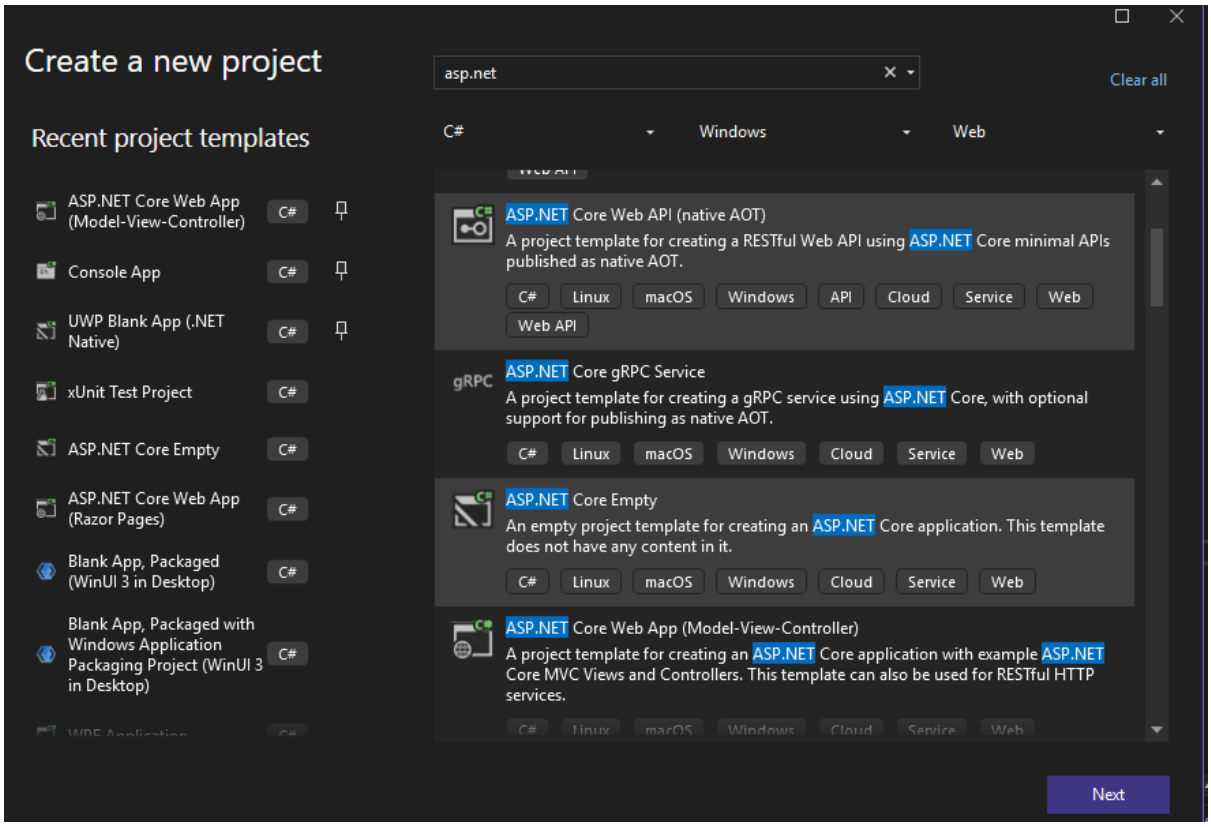
ERD Diagram:



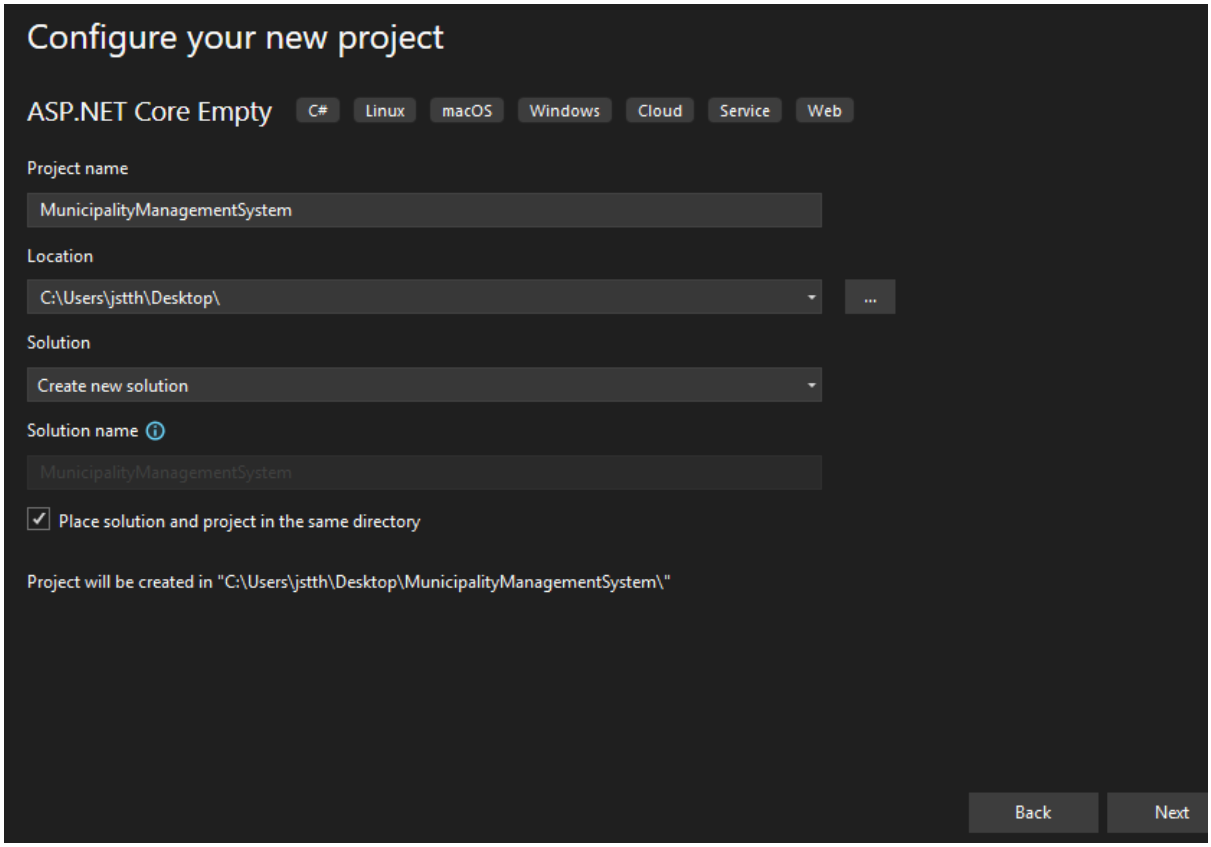
Phase 3:

Steps To Create the Project:

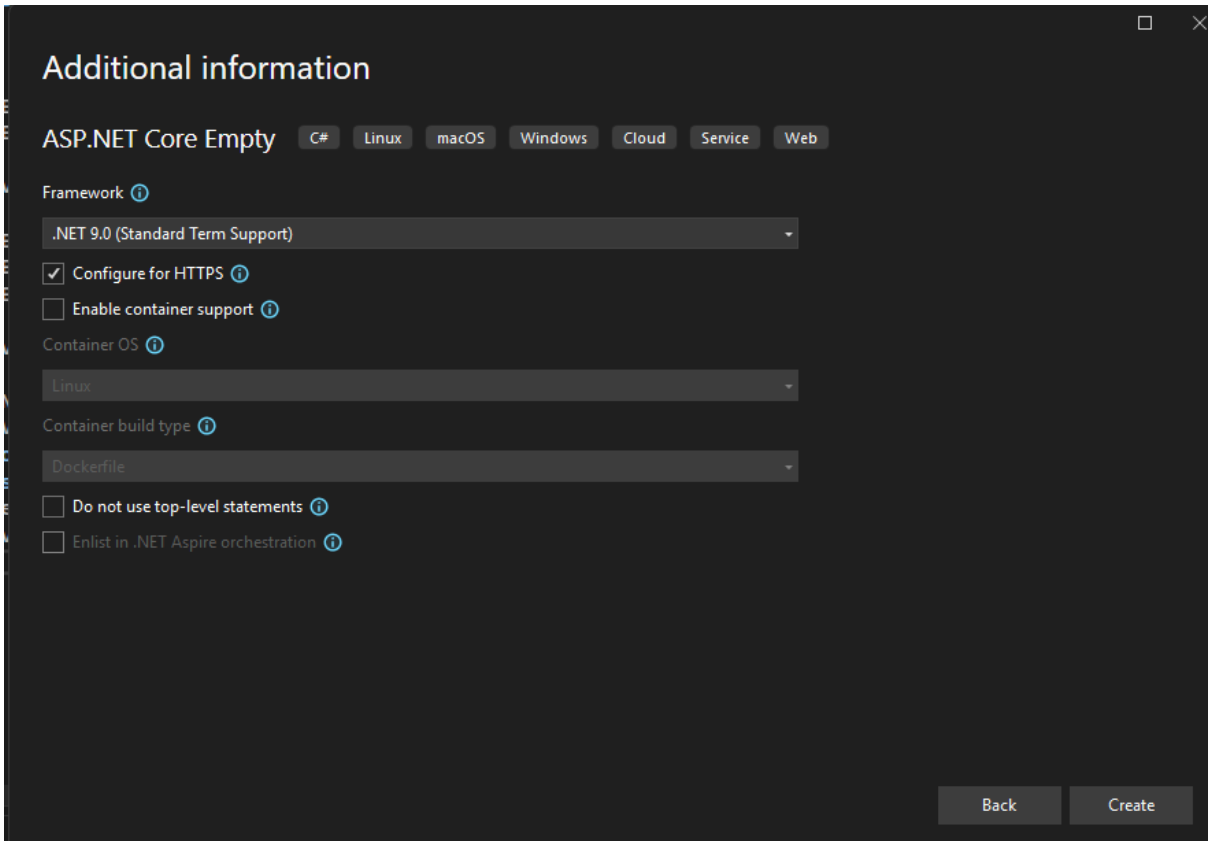
Open Visual Studio, select Asp.Net Empty



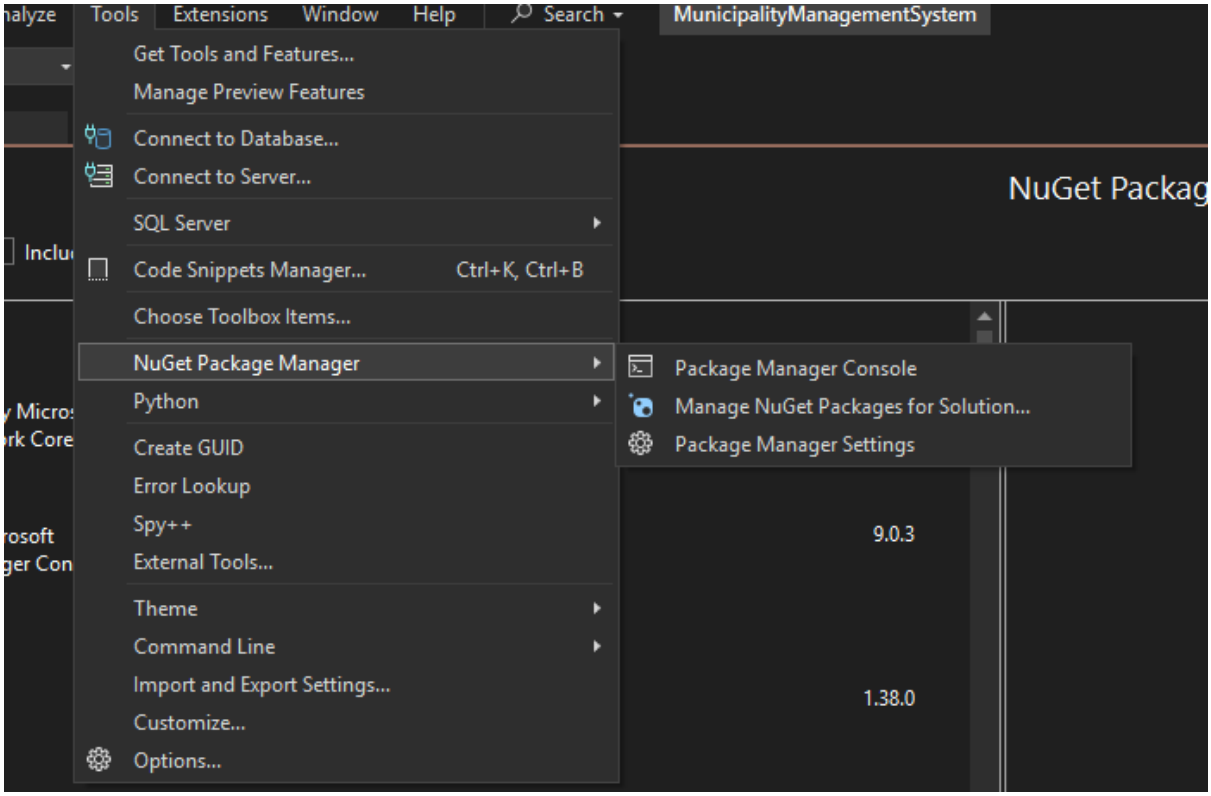
Configure the location and name of the project.



Select .NET 9.0 as your framwork and leave the rest of the setting default.



Install the Nuget Packages, using the package manager console. Can be found in tools.



Install-Package Microsoft.EntityFrameworkCore

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools

Install-Package Microsoft.EntityFrameworkCore.Design

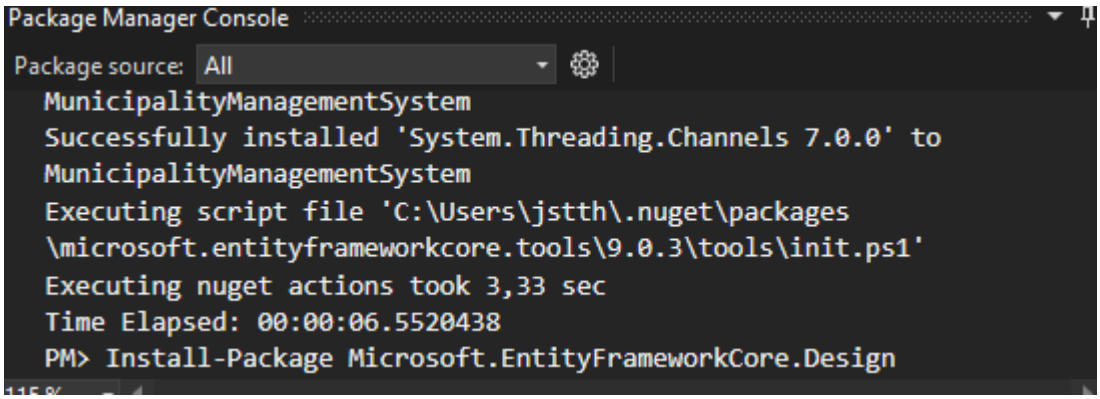
Install-Package Microsoft.VisualStudio.Web.CodeGeneration.Design

Install-Package Moq

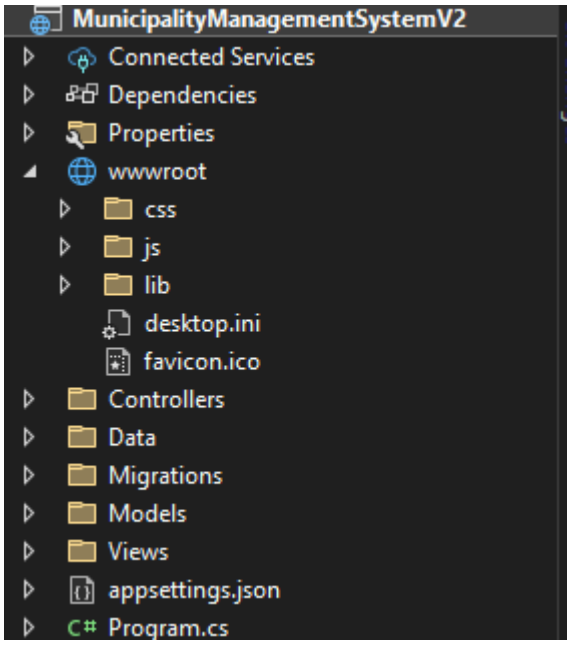
Install-Package xUnit

Install-Package xunit.runner.visualstudio

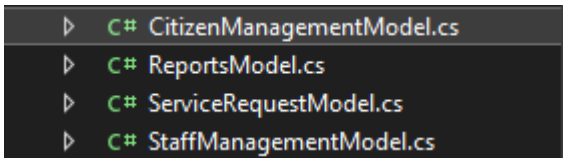
Run these commands in the package manager.



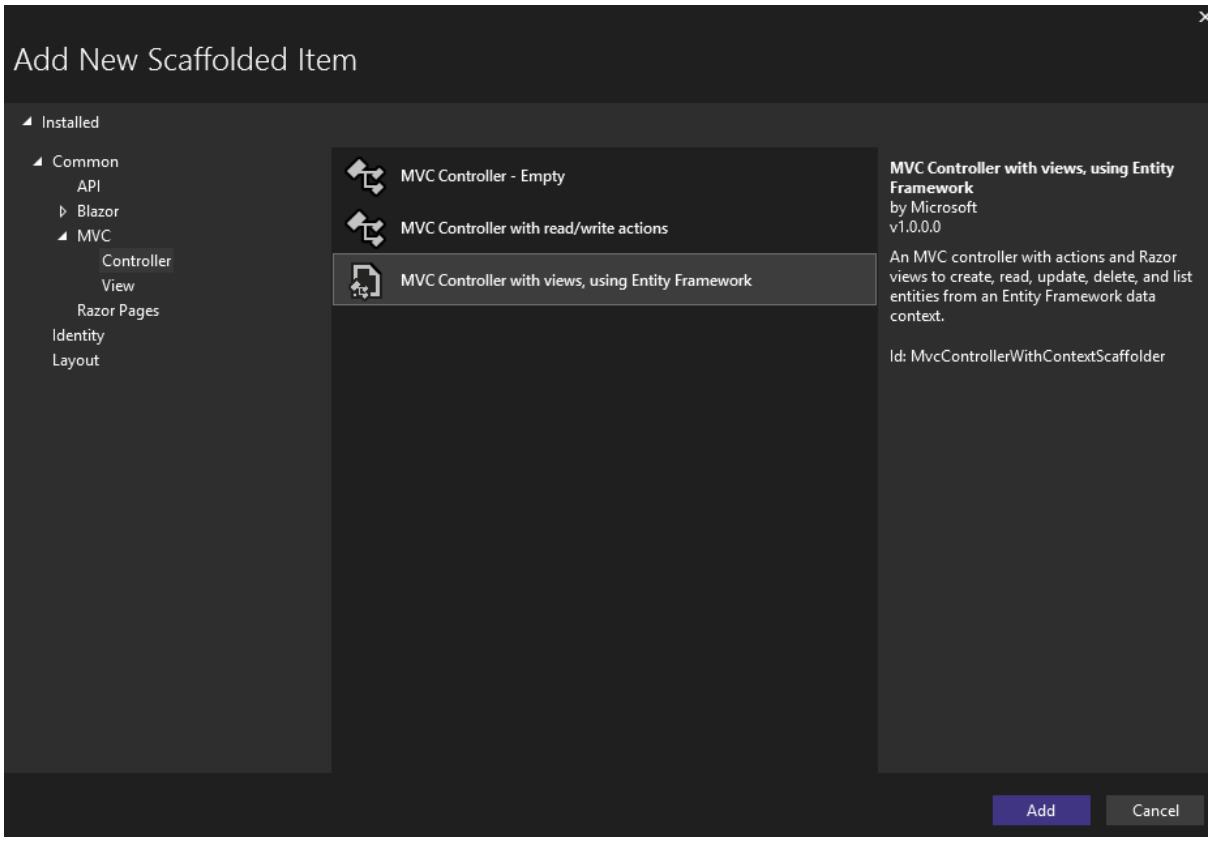
Create The models, views, data, controllers, wwroot folders. With the css, js and lib folders.



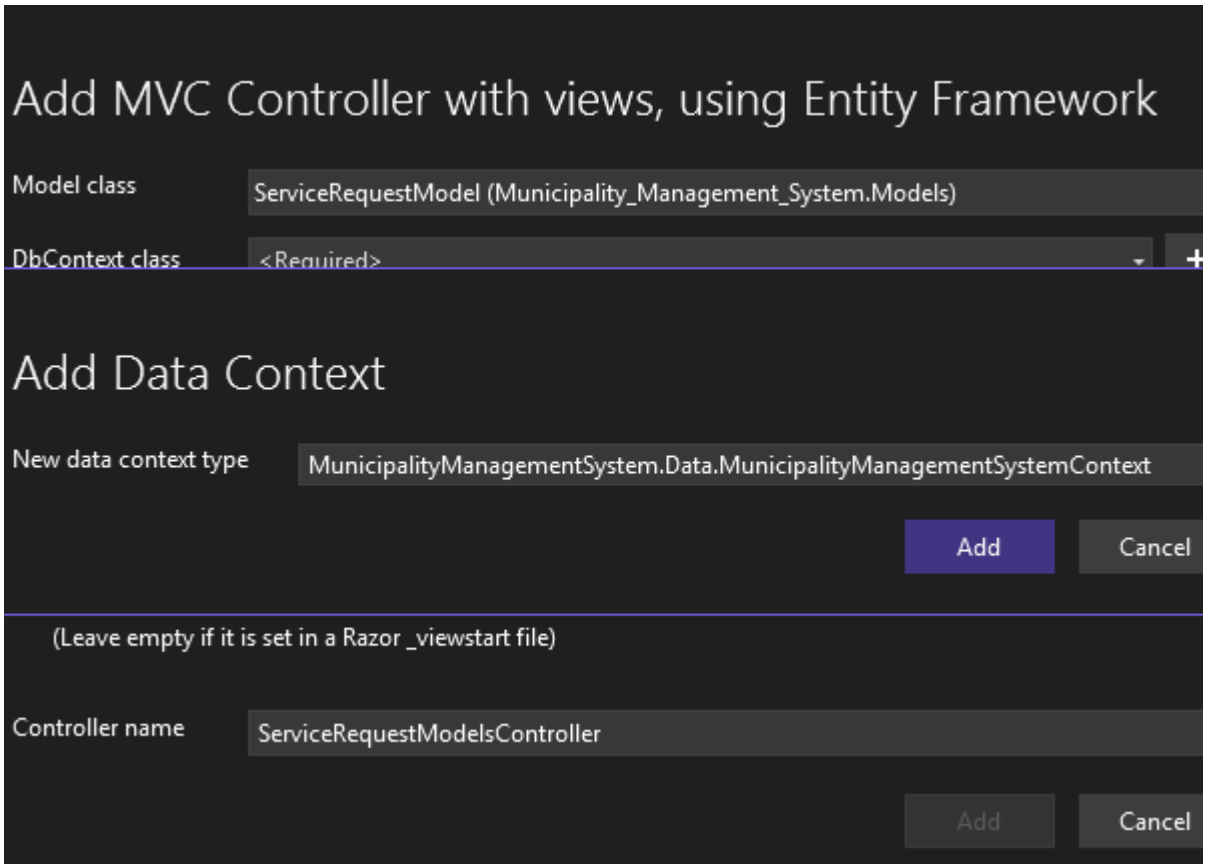
Create your models classes.



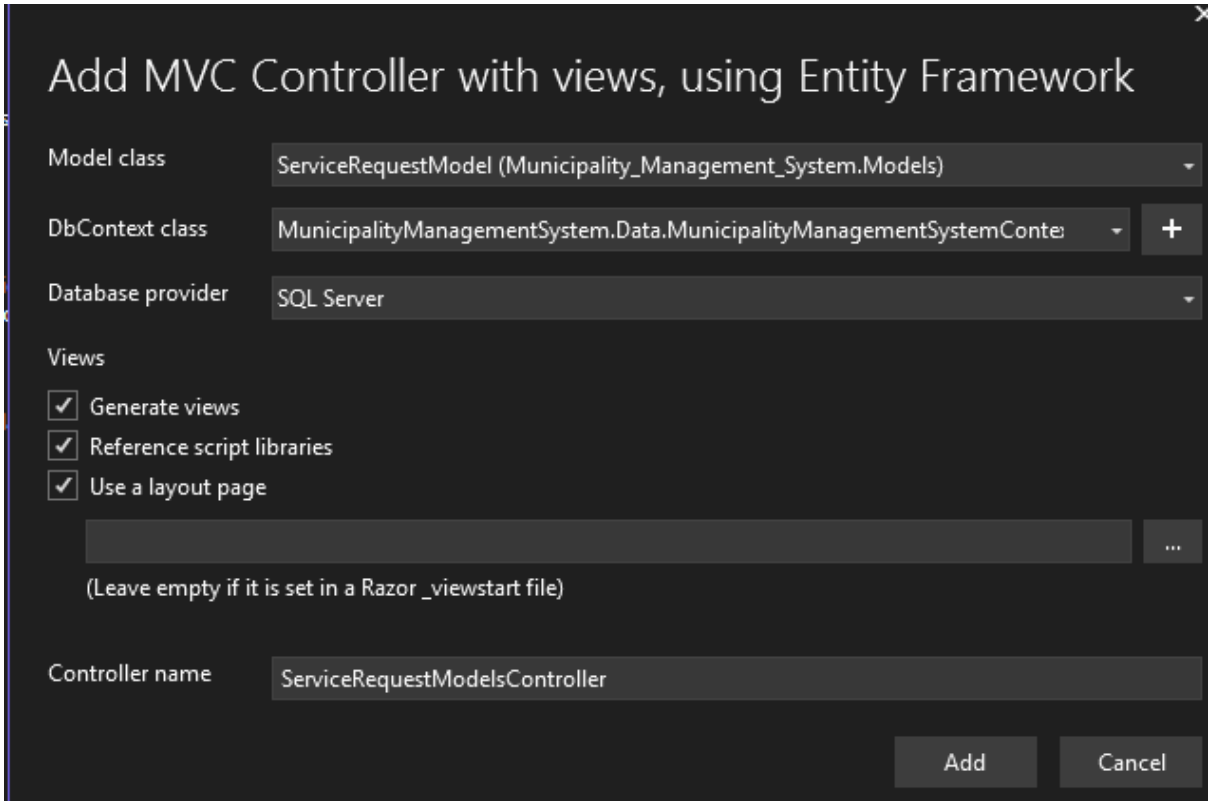
Create your controllers with the entity framework.



Select the model for it, give the controller a name and add the data context



Choose SQL Server, leave every thing in the views check marked.

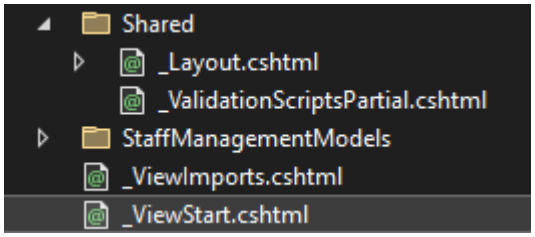


In The Package Manger console add these commands.

Add-Migrations ‘InitialCreate’

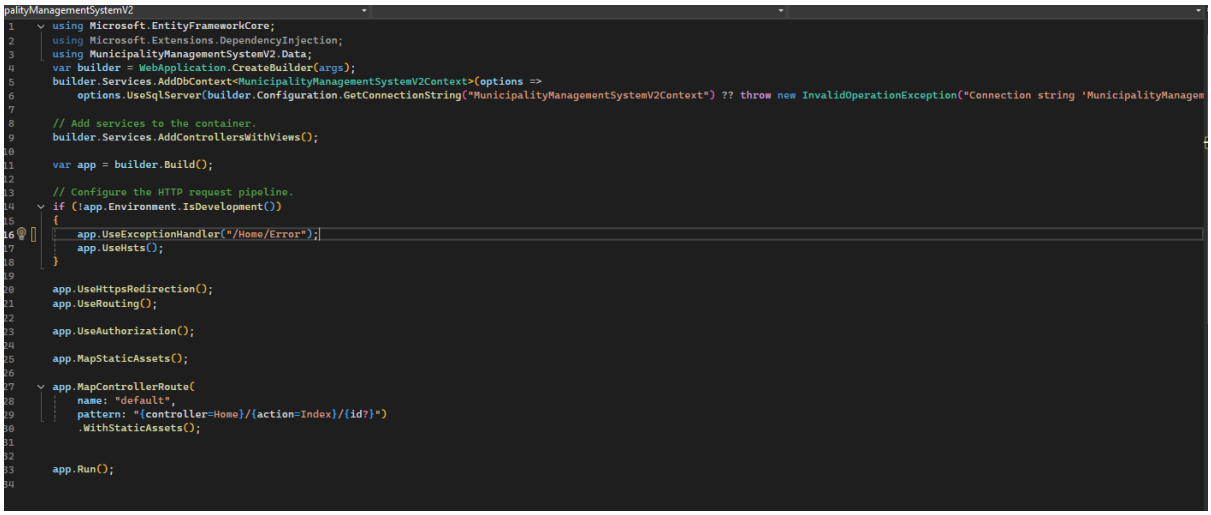
Update-Database

Create your layout view and viewstart cshtml files



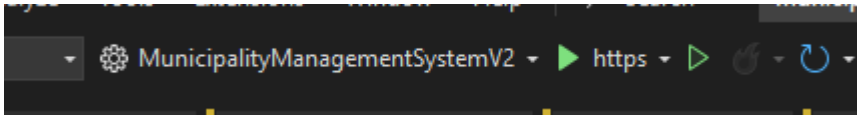
In the view start just specify your layout. Customize your views and create a home page with a home controller as well.

Modify the program.cs file with all the routing and specifying the connection string



```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using MunicipalityManagementSystemV2.Data;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddDbContext<MunicipalityManagementSystemV2Context>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("MunicipalityManagementSystemV2Context") ?? throw new InvalidOperationException("Connection string 'MunicipalityManagem

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseRouting();

app.UseAuthorization();

app.MapStaticAssets();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}")
    .WithStaticAssets();


app.Run();
```

Run your project and customize it.



Nugget Packages Installed and Why:

Microsoft Entity Framework: To work with databases in my project.

MicrosoftEntityFrameworkCore.Design (9.0.3): To use Migrations, generate database schemas and scaffolding

Microsoft.EntityFrameworkCore.InMemory (9.0.3): Allows me to use an in-memory database to test during my unit tests.

Microsoft.EntityFrameworkCore.SqlServer (9.0.3): Allows me to use sql server as my database provider in EF Core.

Microsoft.EntityFrameworkCore.Tools (9.0.3): Essential for migration and updating the database.

Microsoft.NET.Test.Sdk (17.13.0): It is required for using xUnit

Microsoft.VisualStudio.Web.CodeGeneration.Design (9.0.0): Helped scaffold controllers, view and models saving time.

xUnit (2.9.3): Used to write the test cases for my project.

xUnit Runner for Visual Studio (3.0.2): To run xUnit test inside Visual Studio's Test Explorer.

Code Snippets:

```
namespace Municipality_Management_System.Models
{
    21 references
    public class CitizenManagementModel
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)] //identity autoincrements.
        22 references | 0 1/3 passing
        public int CitizenID { get; set; }

        [Required] //Add error message for this as well later?
        [StringLength(100, ErrorMessage = ("Max character limit for name is 100."))]
        18 references | 0 1/3 passing
        public required string Fullname { get; set; }

        [Required]
        [StringLength(200, ErrorMessage = ("Max character limit for address is 200."))]
        18 references | 0 1/3 passing
        public required string Address { get; set; }

        [Required]
        [RegularExpression(@"^\d{10}$", ErrorMessage = "Phone number must be  10 digits.")]
        17 references | 0 1/3 passing
        public required string PhoneNumber { get; set; }

        //Not required
        [EmailAddress]
        22 references | 0 1/3 passing
        public string? Email { get; set; }

        16 references | 0 1/2 passing
        public DateTime? DateOfBirth { get; set; } //Not sure why this is DateTime. Not Like most people know what time they are born.

        16 references | 0 1/2 passing
        public DateTime RegistrationDate { get; set; } = DateTime.UtcNow; //utc to be consistent across time zones.

    }
}
```

Citizen Management class, the id will increment automatically with maximum string limits and error messages for the fields. Primary key is included and the specified fields are set to required or nullable as specified in the assignment. Also default values specified.

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

namespace Municipality_Management_System.Models
{
    21 references
    public class ReportsModel
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        19 references | 0 0/4 passing
        public int ReportID { get; set; }

        [ForeignKey("CitizenID")]
        15 references | 0 1/3 passing
        public int CitizenID { get; set; }

        [Required]
        14 references | 0 1/3 passing
        public required string ReportType { get; set; }

        [Required]
        14 references | 0 1/3 passing
        public required string Details { get; set; }

        11 references | 0 1/3 passing
        public DateTime SubmissionDate { get; set; } = DateTime.UtcNow;

        17 references | 0 1/3 passing
        public string Status { get; set; } = "Under Review";
    }
}
```

Similar to the citizen model, but this time included a foreign key to link the table to the citizen table.

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

namespace Municipality_Management_System.Models
{
    22 references
    public class ServiceRequestModel
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        21 references | 2/3 passing
        public int RequestID { get; set; }

        [ForeignKey("CitizenID")]
        18 references | 2/4 passing
        public int CitizenID { get; set; }

        [Required]
        [StringLength(50)]
        17 references | 2/4 passing
        public required string ServiceType { get; set; }

        [Required]
        17 references | 2/4 passing
        public DateTime RequestDate { get; set; } = DateTime.UtcNow;

        [StringLength(30)]
        22 references | 2/4 passing
        public string Status { get; set; } = "Pending";

    }
}
```

The service request model also has a foreign key linking the citizen id to the citizen management model. With similar fields as the previous models, with autoincrementing ID, default values for specified fields and data validation.

```cs
    21 references
    public class StaffManagementModel
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        19 references | 3/4 passing
        public int StaffID { get; set; }

        [Required]
        [StringLength(100)]
        19 references | 3/5 passing
        public required string FullName { get; set; }

        [Required]
        [StringLength(100)]
        17 references | 2/4 passing
        public required string Position { get; set; }

        [Required]
        [StringLength(100)]
        17 references | 2/4 passing
        public required string Department { get; set; }

        //Unlike citizen, email is required for staff.
        [Required]
        [EmailAddress]
        24 references | 2/4 passing
        public required string Email { get; set; }

        [Required]
        17 references | 2/4 passing
        public required DateTime HiredDate { get; set; }
    }
}
```

The staff model is as similar to the citizen model as it gets except this has required for the email and no phone number.

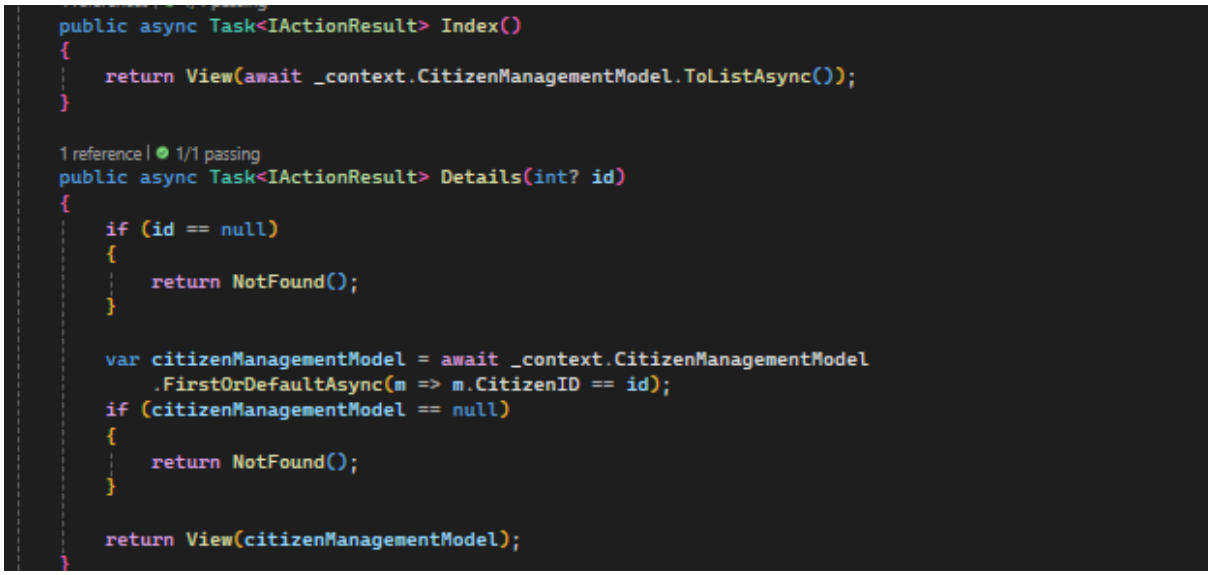
```cs
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using MunicipalityManagementSystemV2.Data;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddDbContext<MunicipalityManagementSystemV2Context>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("MunicipalityManagementSystemV2Context") ?? throw new InvalidOperationException("Connection string 'MunicipalityManagementSystemV2Context' not found.")));

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseRouting();

app.UseAuthorization();

app.MapStaticAssets();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}")
    .WithStaticAssets();


app.Run();
```

The project cs file configures the database context and tells the EF Core to use SQL server it fetches the connection string from the appsetting.json file.

Also enables MVC, allowing the app to handle web requests and finally compiles everything in the builder. Enforces HTTPS, and redirects to the Error page in case problems occur.

```
namespace MunicipalityManagementSystemV2.Data
{
    30 references
    public class MunicipalityManagementSystemV2Context : DbContext
    {
        4 references
        public MunicipalityManagementSystemV2Context (DbContextOptions<MunicipalityManagementSystemV2Context> options)
            : base(options)
        {
        }

        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<CitizenManagementModel>()
                .HasIndex(c => c.Email)
                .IsUnique();

            modelBuilder.Entity<StaffManagementModel>()
                .HasIndex(c => c.Email)
                .IsUnique();
        }


        19 references | 2/4 passing
        public DbSet<Municipality_Management_System.Models.CitizenManagementModel> CitizenManagementModel { get; set; } = default!;
        10 references
        public DbSet<Municipality_Management_System.Models.ReportsModel> ReportsModel { get; set; } = default!;
        16 references | 1/3 passing
        public DbSet<Municipality_Management_System.Models.ServiceRequestModel> ServiceRequestModel { get; set; } = default!;
        16 references | 3/3 passing
        public DbSet<Municipality_Management_System.Models.StaffManagementModel> StaffManagementModel { get; set; } = default!;
    }
```

Forces the email to be unique for the citizen management model and staff management model. Defines the 4 tables in the database, created after the models in the model folder.

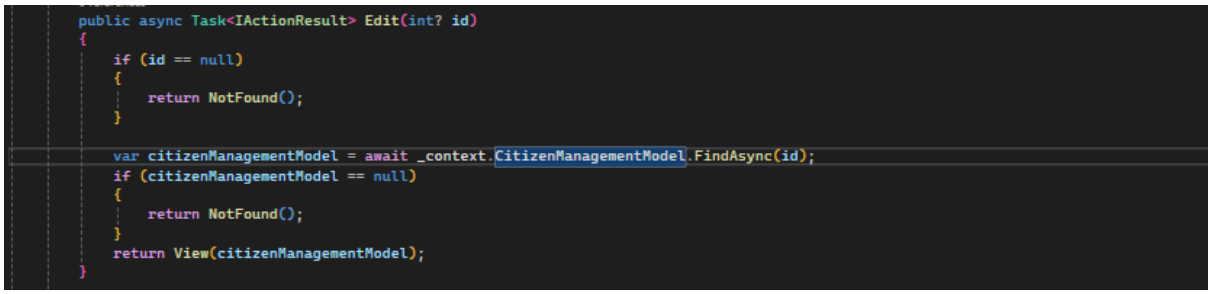


```
public async Task<IActionResult> Index()
{
    return View(await _context.CitizenManagementModel.ToListAsync());
}

1 reference | 1/1 passing
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var citizenManagementModel = await _context.CitizenManagementModel
        .FirstOrDefaultAsync(m => m.CitizenID == id);
    if (citizenManagementModel == null)
    {
        return NotFound();
    }

    return View(citizenManagementModel);
}
```

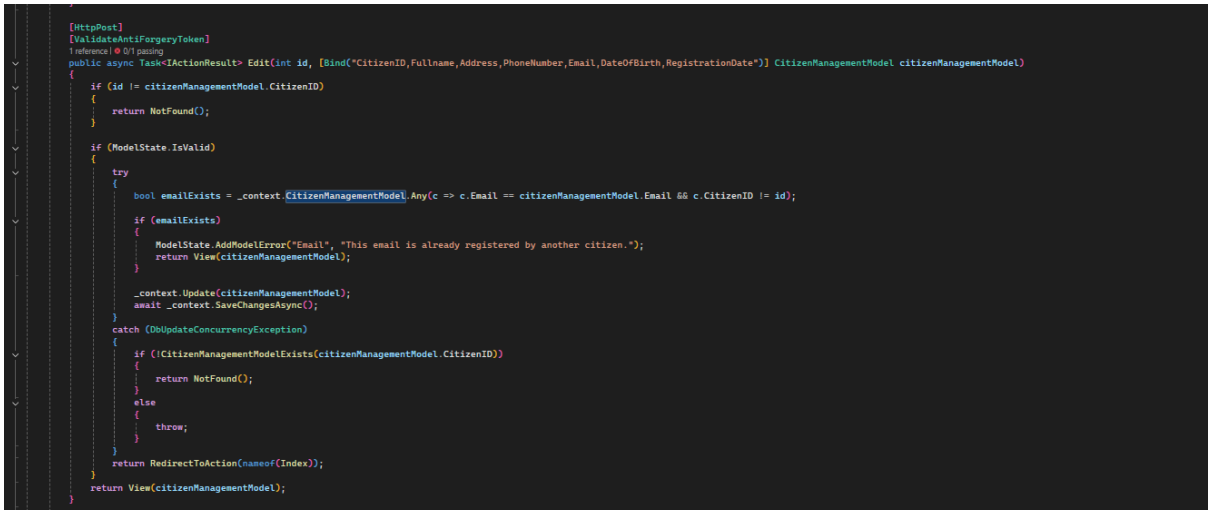
Index lists all of the records in the Citizen management table on the website and details lists all the details for a record matching the id.

```
0 references
public IActionResult Create()
{
    return View();
}


[HttpPost]
[ValidateAntiForgeryToken]
1 reference | 1/1 passing
public async Task<IActionResult> Create([Bind("CitizenID,Fullname,Address,PhoneNumber,Email,DateOfBirth,RegistrationDate")] CitizenManagementModel citizenManagementModel)
{
    if (ModelState.IsValid)
    {
        bool emailExists = _context.CitizenManagementModel.Any(c => c.Email == citizenManagementModel.Email);

        if (emailExists)
        {
            ModelState.AddModelError("Email", "This email is already registered.");
            return View(citizenManagementModel);
        }

        _context.Add(citizenManagementModel);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(citizenManagementModel);
}
```

Get and Post methods, to create the citizens which to add to the database. In the post, adds error handling for existing emails and if everything is valid saves it to the database.

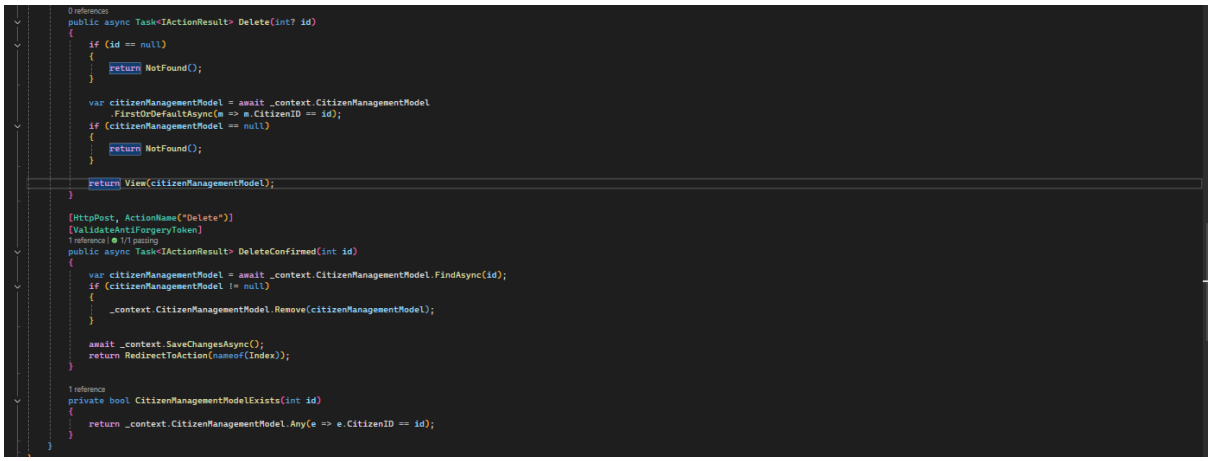


```
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var citizenManagementModel = await _context.CitizenManagementModel.FindAsync(id);
    if (citizenManagementModel == null)
    {
        return NotFound();
    }
    return View(citizenManagementModel);
}
```



```
[HttpPost]
[ValidateAntiForgeryToken]
1 reference | 0/1 passing
public async Task<IActionResult> Edit(int id, [Bind("CitizenID,Fullname,Address,PhoneNumber,Email,DateOfBirth,RegistrationDate")] CitizenManagementModel citizenManagementModel)
{
    if (id != citizenManagementModel.CitizenID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            bool emailExists = _context.CitizenManagementModel.Any(c => c.Email == citizenManagementModel.Email && c.CitizenID != id);

            if (emailExists)
            {
                ModelState.AddModelError("Email", "This email is already registered by another citizen.");
                return View(citizenManagementModel);
            }

            _context.Update(citizenManagementModel);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!CitizenManagementModelExists(citizenManagementModel.CitizenID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(citizenManagementModel);
}
```

Get and Posts method to edit an existing citizen through finding their id and even validating if the person is trying to update their email to one that another citizen already has. Binding allow them only to add fields matching that into it and saves the changes to the database.



```
0 references
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var citizenManagementModel = await _context.CitizenManagementModel
        .FirstOrDefaultAsync(m => m.CitizenID == id);
    if (citizenManagementModel == null)
    {
        return NotFound();
    }

    return View(citizenManagementModel);
}

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
1 reference | 1/1 passing
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var citizenManagementModel = await _context.CitizenManagementModel.FindAsync(id);
    if (citizenManagementModel != null)
    {
        _context.CitizenManagementModel.Remove(citizenManagementModel);
    }

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

1 reference
private bool CitizenManagementModelExists(int id)
{
    return _context.CitizenManagementModel.Any(e => e.CitizenID == id);
}
}
```

Get and Post methods to delete a citizen who matches the ID provided.

```
namespace MunicipalityManagementSystemV2.Controllers
{
    3 references
    public class ReportsModelsController : Controller
    {
        private readonly MunicipalityManagementSystemV2Context _context;

        1 reference
        public ReportsModelsController(MunicipalityManagementSystemV2Context context)
        {
            _context = context;
        }

        5 references | 0/1 passing
        public async Task<IActionResult> Index()
        {
            return View(await _context.ReportsModel.ToListAsync());
        }

        1 reference | 0/1 passing
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var reportsModel = await _context.ReportsModel
                .FirstOrDefaultAsync(m => m.ReportID == id);
            if (reportsModel == null)
            {
                return NotFound();
            }

            return View(reportsModel);
        }
```

Specifies the database I'm working with and like previously, index lists all the records for the reports table and details lists the detail of a single report.

```
        0 references
        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        2 references | 1/2 passing
        public async Task<IActionResult> Create([Bind("ReportID,CitizenID,ReportType,Details,SubmissionDate,Status")] ReportsModel reportsModel)
        {
            if (ModelState.IsValid)
            {
                // Check if the provided CitizenID exists
                bool citizenExists = await _context.CitizenManagementModel.AnyAsync(c => c.CitizenID == reportsModel.CitizenID);

                if (!citizenExists)
                {
                    ModelState.AddModelError("CitizenID", "Invalid Citizen ID. Please enter a valid Citizen.");
                    return View(reportsModel); // Return view if invalid

                }

                _context.Add(reportsModel);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }

            return View(reportsModel);
        }
```

Returns the view for the reports model, and creates the report on post. Also making sure a valid citizen id has been provided before saving the record to the database.

```
        1 reference | 0/1 passing
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var reportsModel = await _context.ReportsModel
                .FirstOrDefaultAsync(m => m.ReportID == id);
            if (reportsModel == null)
            {
                return NotFound();
            }

            return View(reportsModel);
        }

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        1 reference | 0/1 passing
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var reportsModel = await _context.ReportsModel.FindAsync(id);
            if (reportsModel != null)
            {
                _context.ReportsModel.Remove(reportsModel);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        1 reference | 0/1 passing
```

Returns the view to the page matching the id of the report to delete it. The post deletes the report matching said id and return the index page if successful.

```
1 reference | 0/1 passing
public async Task<IActionResult> ReviewReport(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var reportsModel = await _context.ReportsModel
        .FirstOrDefaultAsync(m => m.ReportID == id);
    if (reportsModel == null)
    {
        return NotFound();
    }

    return View(reportsModel);
}
```

Goes to the view of the report, where it will allow you to see the details of the review and change its status.

```
120
121
122         [HttpPost]
123         [ValidateAntiForgeryToken]
            0 references
124         public async Task<IActionResult> ReviewReport(int id, string status)
125         {
126             if (id == 0)
127             {
128                 return NotFound();
129             }
130
131             // Fetch the existing ReportsModel from the database
132             var reportsModel = await _context.ReportsModel.FindAsync(id);
133             if (reportsModel == null)
134             {
135                 return NotFound();
136             }
137
138             // Update only the Status field
139             reportsModel.Status = status;
140
141             if (ModelState.IsValid)
142             {
143                 try
144                 {
145                     // Update the model in the database
146                     _context.Update(reportsModel);
147                     await _context.SaveChangesAsync();
148                 }
149                 catch (DbUpdateConcurrencyException)
150                 {
151                     if (!ReportsModelExists(id))
152                     {
153                         return NotFound();
154                     }
155                     else
156                     {
157                         throw;
158                     }
159                 }
160             }
161
162             return RedirectToAction(nameof(Index));
163         }
164
```

Updates only the status field for the report and saves the changes to the database.

```
namespace MunicipalityManagementSystemV2.Controllers
{
    8 references
    public class ServiceRequestModelsController : Controller
    {
        private readonly MunicipalityManagementSystemV2Context _context;

        7 references | 4/7 passing
        public ServiceRequestModelsController(MunicipalityManagementSystemV2Context context)
        {
            _context = context;
        }

        // GET: ServiceRequestModels
        5 references | 0/1 passing
        public async Task<IActionResult> Index()
        {
            return View(await _context.ServiceRequestModel.ToListAsync());
        }

        // GET: ServiceRequestModels/Details/5
        2 references | 2/2 passing
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var serviceRequestModel = await _context.ServiceRequestModel
                .FirstOrDefaultAsync(m => m.RequestID == id);
            if (serviceRequestModel == null)
            {
                return NotFound();
            }

            return View(serviceRequestModel);
        }
```

Lists all services requests from the database on the webpage and details retrieves the specific request matching the ID to the view.

```
public IActionResult Create()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
2 references | 1/2 passing
public async Task<IActionResult> Create([Bind("RequestID,CitizenID,ServiceType,RequestDate,Status")] ServiceRequestModel serviceRequestModel)
{
    if (ModelState.IsValid)
    {
        // Check if the provided CitizenID exists
        bool citizenExists = await _context.CitizenManagementModel.AnyAsync(c => c.CitizenID == serviceRequestModel.CitizenID);

        if (!citizenExists)
        {
            ModelState.AddModelError("CitizenID", "Invalid Citizen ID.");
            return View(serviceRequestModel); // Return view if invalid

        }
        _context.Add(serviceRequestModel);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(serviceRequestModel);
}
```

Takes the user to the view to create the service request and the post adds the given information matching the fields in the parameters to the model. Adding and saving them to the database if successful and giving an error page if not.

```
0 references
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var serviceRequestModel = await _context.ServiceRequestModel
        .FirstOrDefaultAsync(m => m.RequestID == id);
    if (serviceRequestModel == null)
    {
        return NotFound();
    }

    return View(serviceRequestModel);
}

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
1 reference | 1/1 passing
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var serviceRequestModel = await _context.ServiceRequestModel.FindAsync(id);
    if (serviceRequestModel != null)
    {
        _context.ServiceRequestModel.Remove(serviceRequestModel);
    }

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

Gives the view to the page to delete the specified service request matching the id. And on post, allows the deletion of it. Saving the changes to the database in the end.

```
0 references
public async Task<IActionResult> UpdateStatus(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var serviceRequestModel = await _context.ServiceRequestModel.FindAsync(id);
    if (serviceRequestModel == null)
    {
        return NotFound();
    }

    return View(serviceRequestModel);
}
```

```
[HttpPost]
[ValidateAntiForgeryToken]
1 reference | 0/1 passing
public async Task<IActionResult> UpdateStatus(int id, string status)
{
    if (id == 0)
    {
        return NotFound();
    }

    // Fetch the existing ServiceRequestModel from the database
    var serviceRequestModel = await _context.ServiceRequestModel.FindAsync(id);
    if (serviceRequestModel == null)
    {
        return NotFound();
    }

    // Update only the Status field
    serviceRequestModel.Status = status;

    if (ModelState.IsValid)
    {
        try
        {
            // Update the model in the database
            _context.Update(serviceRequestModel);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!ServiceRequestModelExists(serviceRequestModel.RequestID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
    }

    return RedirectToAction(nameof(Index));
}
```

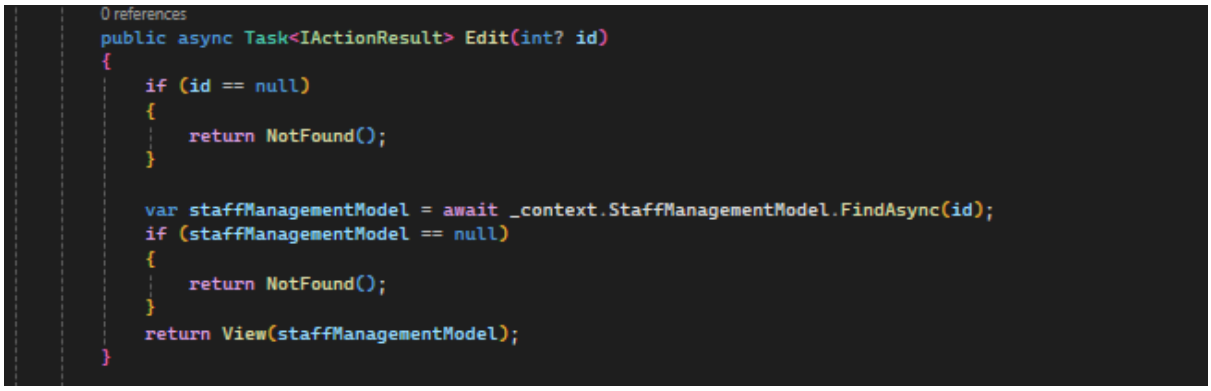
Gives the details in a view matching the specified service request where the staff can change the status of it.

```
9 references | 6/9 passing
public StaffManagementModelsController(MunicipalityManagementSystemV2Context context)
{
    _context = context;
}

4 references | 0/1 passing
public async Task<IActionResult> Index()
{
    return View(await _context.StaffManagementModel.ToListAsync());
}

2 references | 2/2 passing
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var staffManagementModel = await _context.StaffManagementModel
        .FirstOrDefaultAsync(m => m.StaffID == id);
    if (staffManagementModel == null)
    {
        return NotFound();
    }

    return View(staffManagementModel);
}
```

The Index gives the view where all the records for the staff will be listed, the details method will list all of the details for the specified staff member matching the ID.

```
public IActionResult Create()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
2 references | 1/2 passing
public async Task<IActionResult> Create([Bind("StaffID,FullName,Position,Department,Email,HiredDate")] StaffManagementModel staffManagementModel)
{
    if (ModelState.IsValid)
    {
        // Check if email already exists
        bool emailExists = _context.StaffManagementModel.Any(s => s.Email == staffManagementModel.Email);
        if (emailExists)
        {
            ModelState.AddModelError("Email", "This email is already registered.");
            return View(staffManagementModel);
        }

        _context.Add(staffManagementModel);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(staffManagementModel);
}
```

Allows adding a new staff member, listing the view to add the data and on post saves it to the database if the data matches the fields in the parameters. Also enforces the unique of emails, so the program doesn't combust because of multiple similar emails.



```
0 references
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var staffManagementModel = await _context.StaffManagementModel.FindAsync(id);
    if (staffManagementModel == null)
    {
        return NotFound();
    }
    return View(staffManagementModel);
}
```

This method gives the view of the staff member that you wish to edit, along with their details and allow you to edit it.

```
[ValidateAntiForgeryToken]
2 references | 0 1/2 passing
public async Task<IActionResult> Edit(int id, [Bind("StaffID,FullName,Position,Department,Email,HiredDate")] StaffManagementModel staffManagementModel)
{
    if (id != staffManagementModel.StaffID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {

        try
        {
            bool emailExists = _context.StaffManagementModel.Any(s => s.Email == staffManagementModel.Email && s.StaffID != id);

            if (emailExists)
            {
                ModelState.AddModelError("Email", "This email is already registered by another staff member.");
                return View(staffManagementModel);
            }

            _context.Update(staffManagementModel);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!StaffManagementModelExists(staffManagementModel.StaffID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(staffManagementModel);
}
```

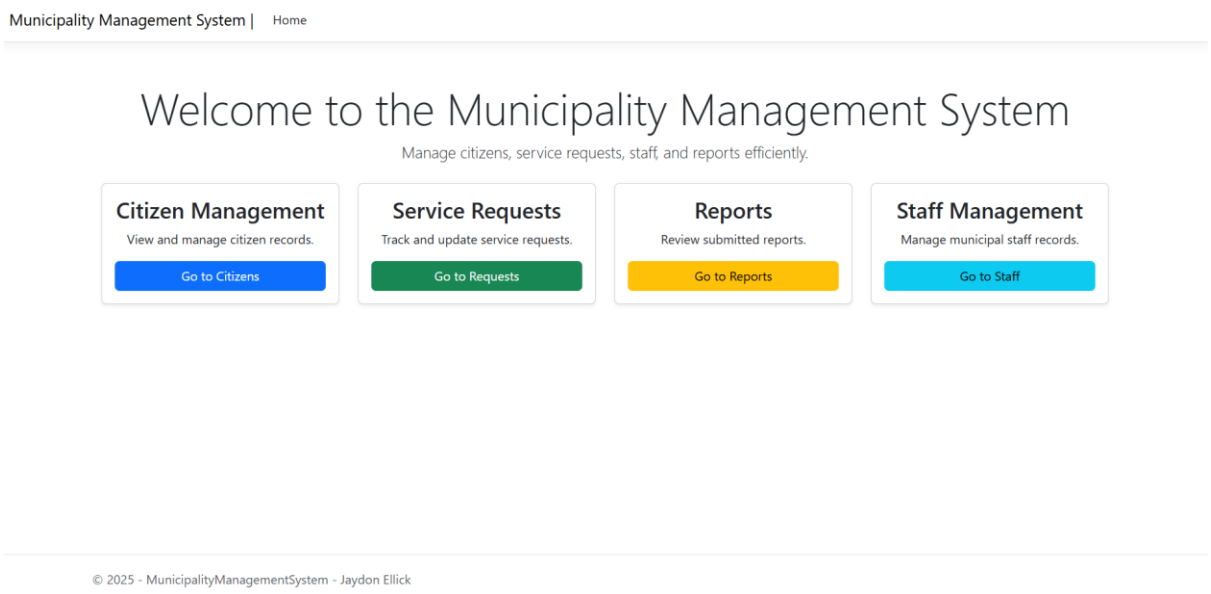
This post method adds the changes to the databases if the new data values matches the fields in the parameters and if the email address if unique for the staff.

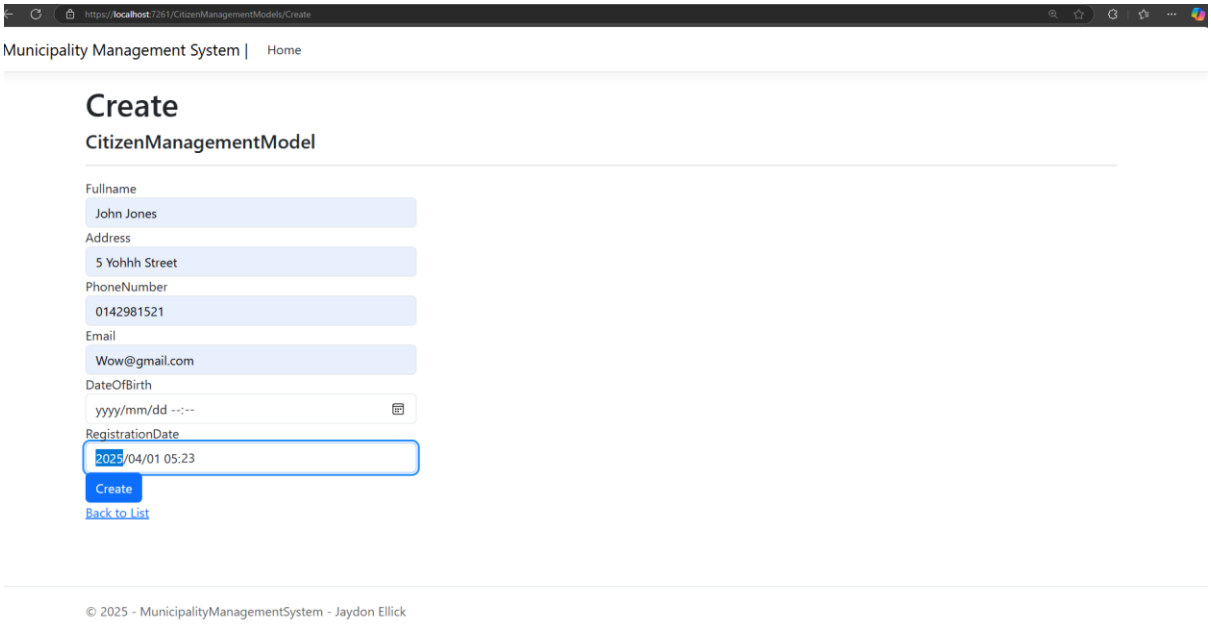
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "MunicipalityManagementSystemV2Context": "Server=(localdb)\\mssqllocaldb;Database=MunicipalityManagementSystemV2Context-3b3d003b-76d7-4aec-bee0-855bb11102d4;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

Specifies which database to the connect to, in this case just a local SQL database.s

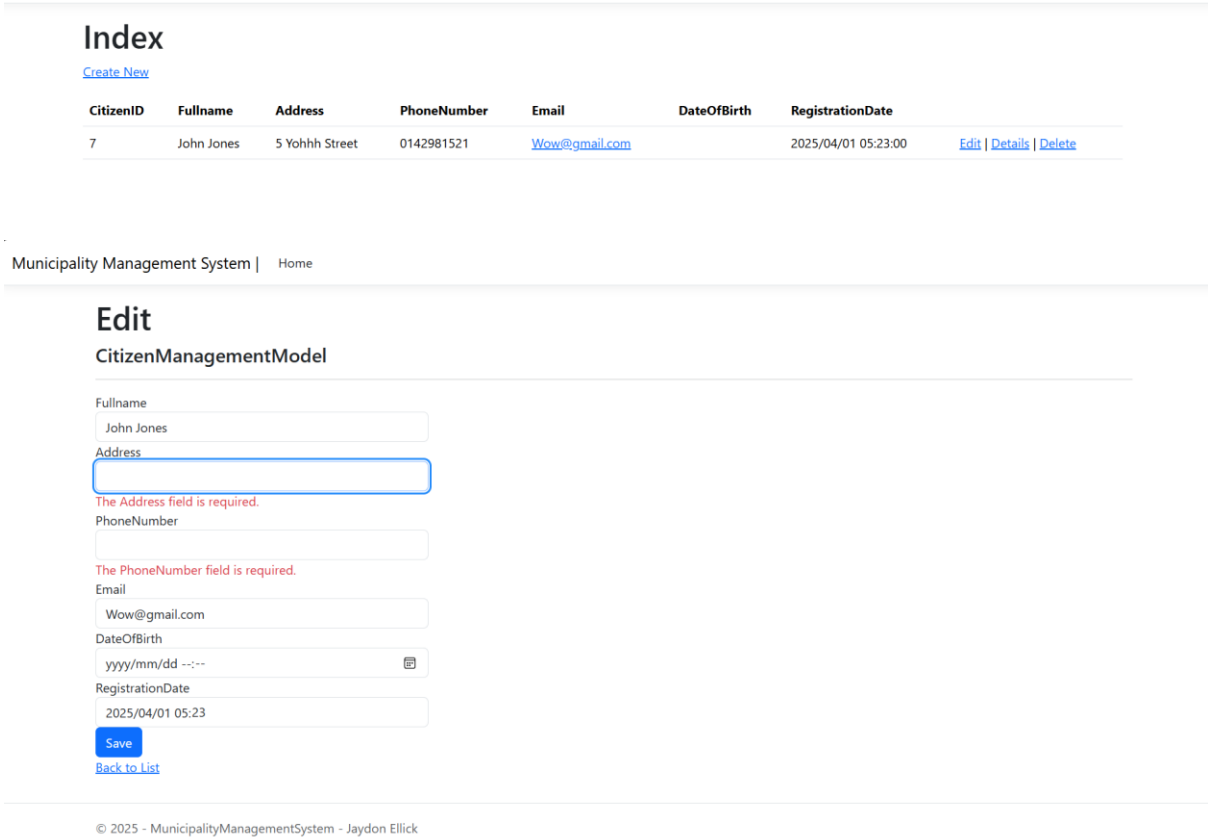
---

Working Features Screenshots:

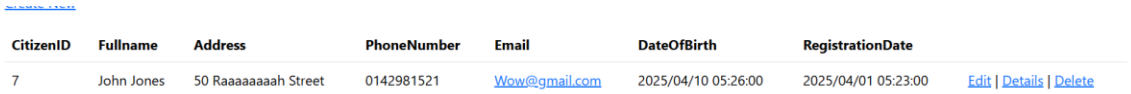




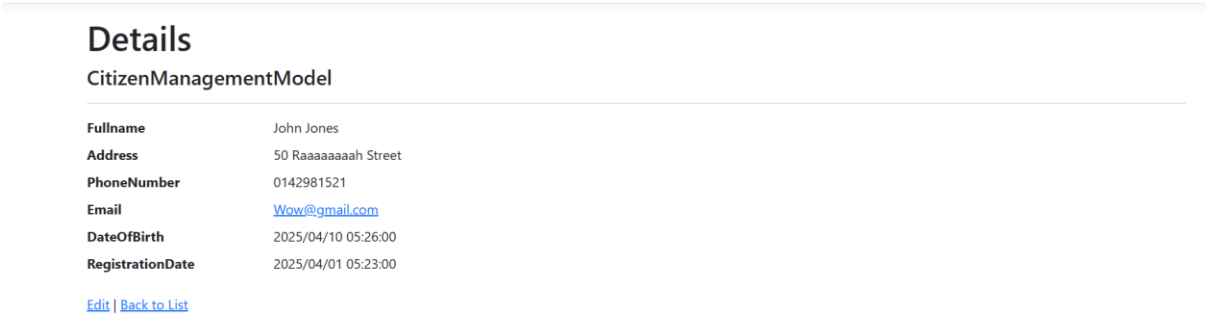
Date of birth isn't required so no error and registration date would give today date. The model name, in this case 'CitizenManagementModel', has been changed to just Citizen and the same for the other models. I'm just not willing to screenshot it again for something so small.



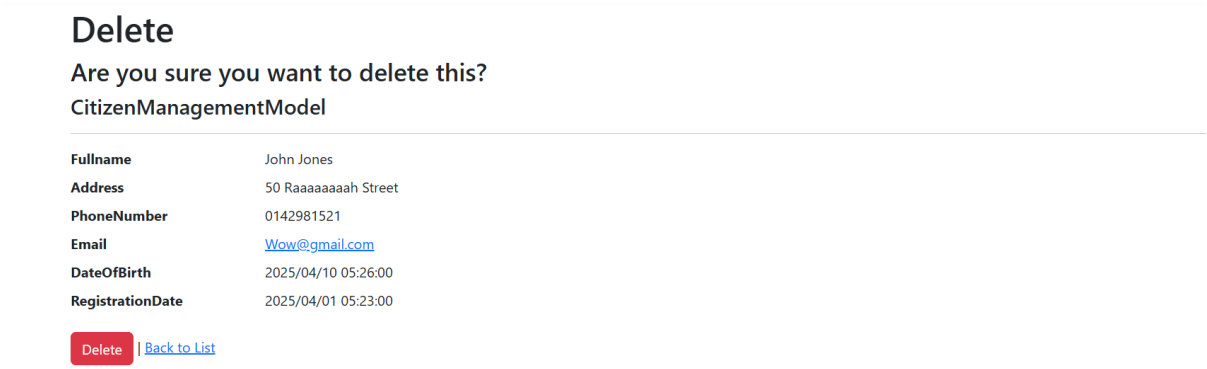
Required fields give me error.



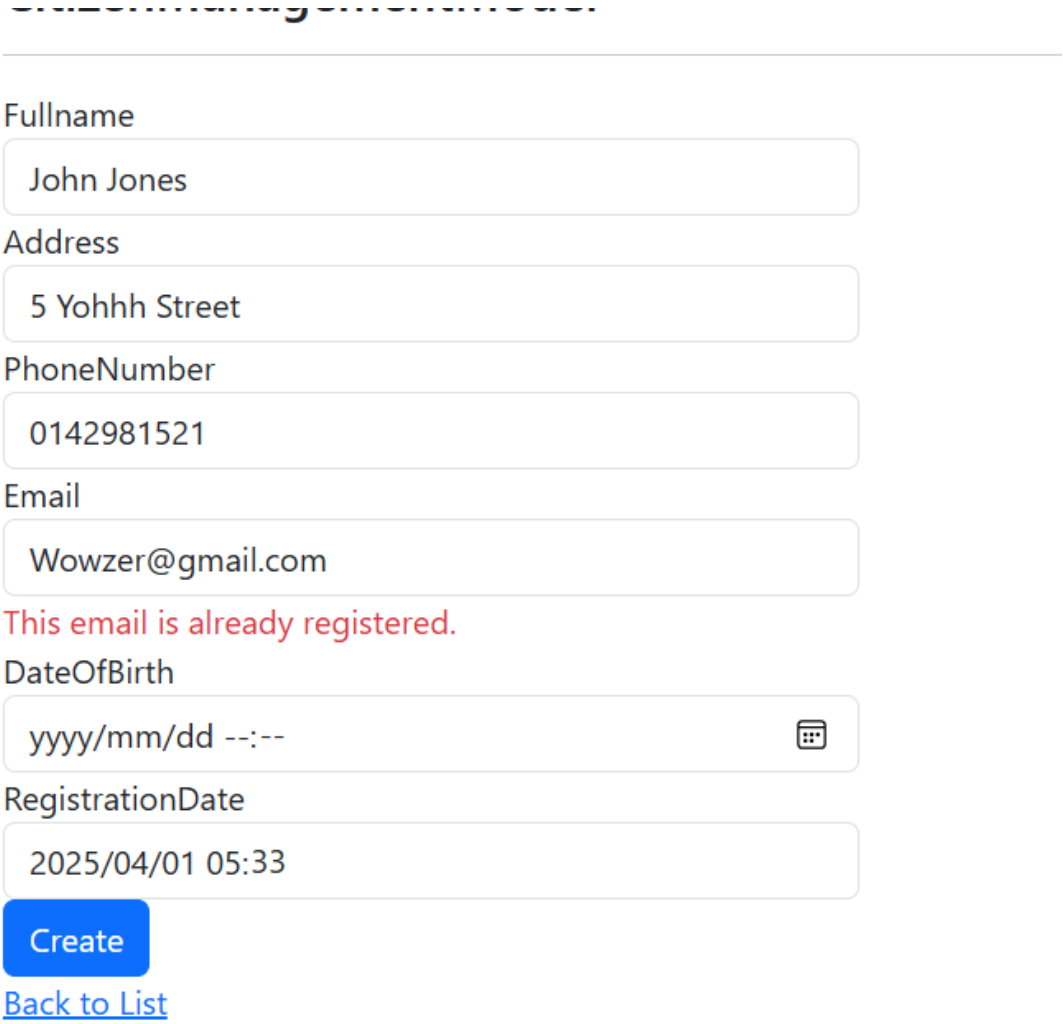
The phone number, street address and date of birth was edited.



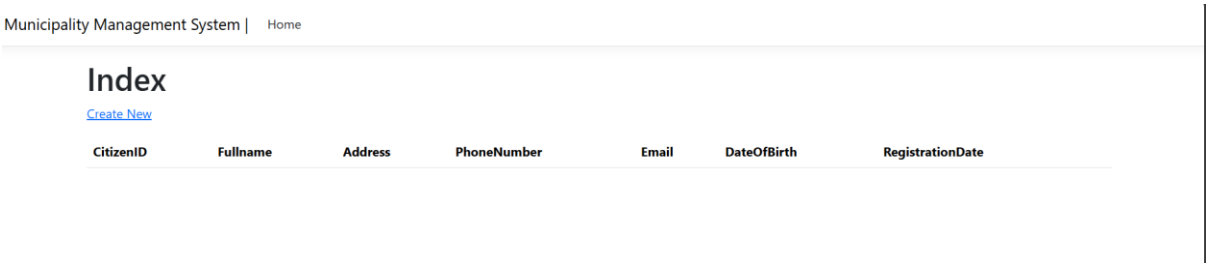
Details page



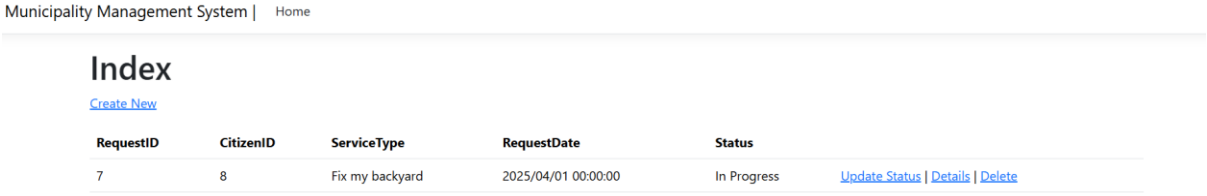
Delete Page

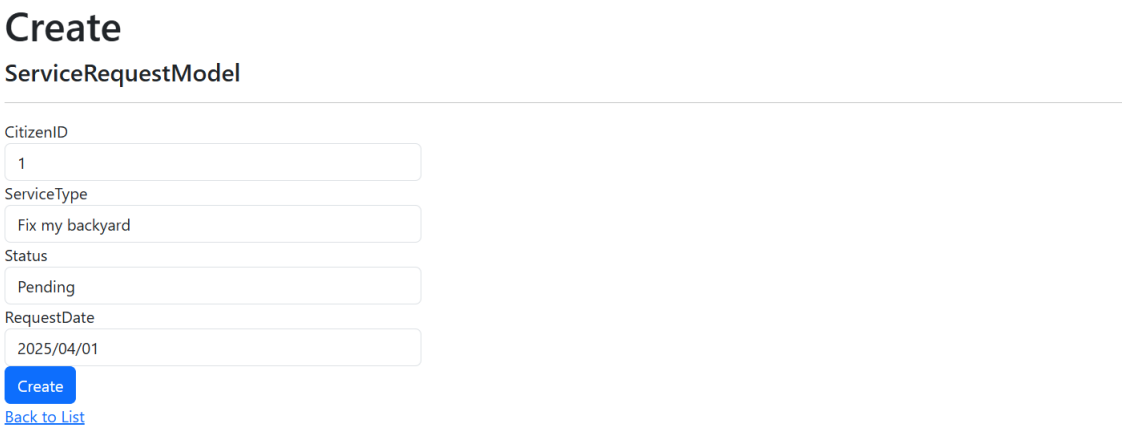


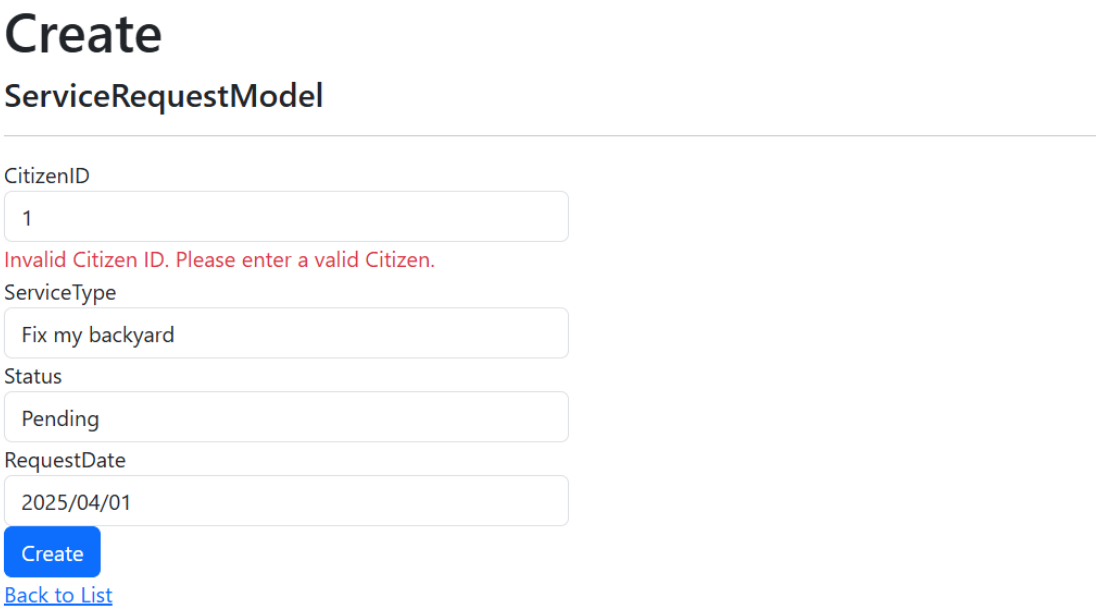
Also the email has to be unique.



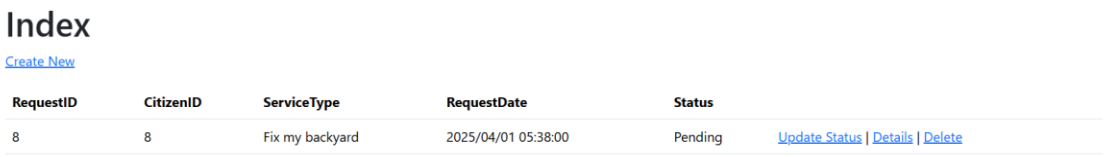
The results of the delete, all records are gone.



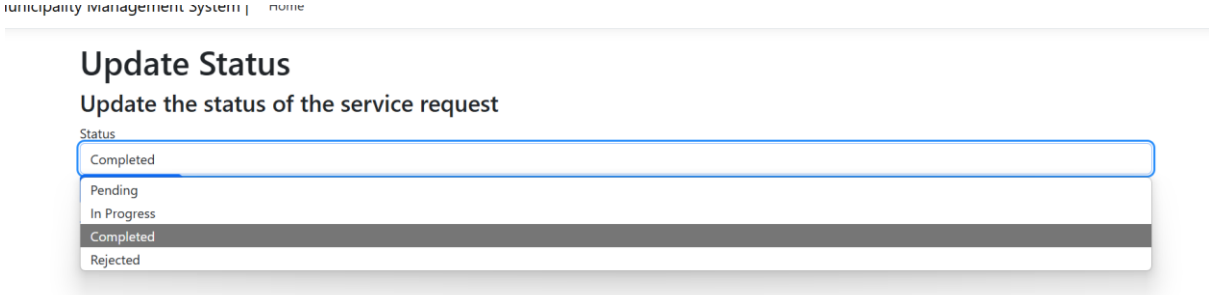




Invalid since the person got deleted and no one has the id of 1.

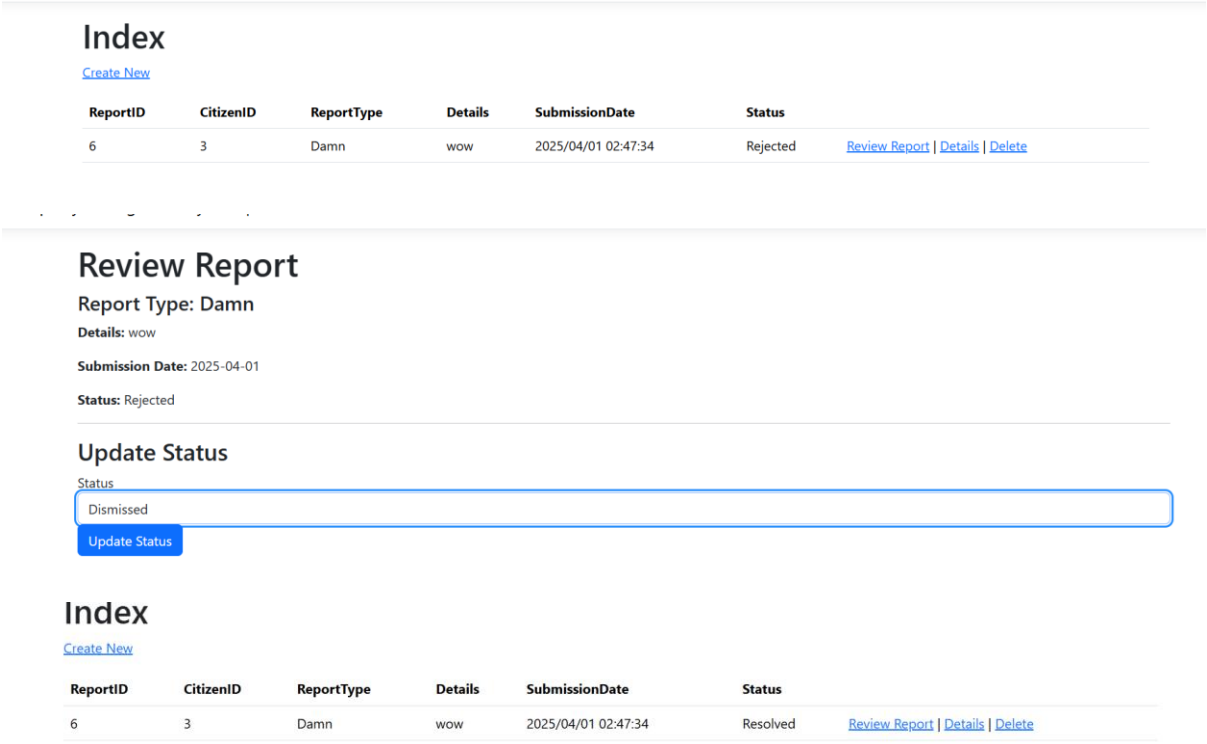


8 is a valid Id so the service request went through.



RequestID	CitizenID	ServiceType	RequestDate	Status	
8	8	Fix my backyard	2025/04/01 05:38:00	Completed	Update Status | Details | Delete

Status went from pending to completed.



Review status went from Rejected to Resolved.

StaffManagementModel

FullName

Jory Jones

Position

Driver

Department

Drivers

Email

Damn@gmail.com

HiredDate

2025/04/01

Save

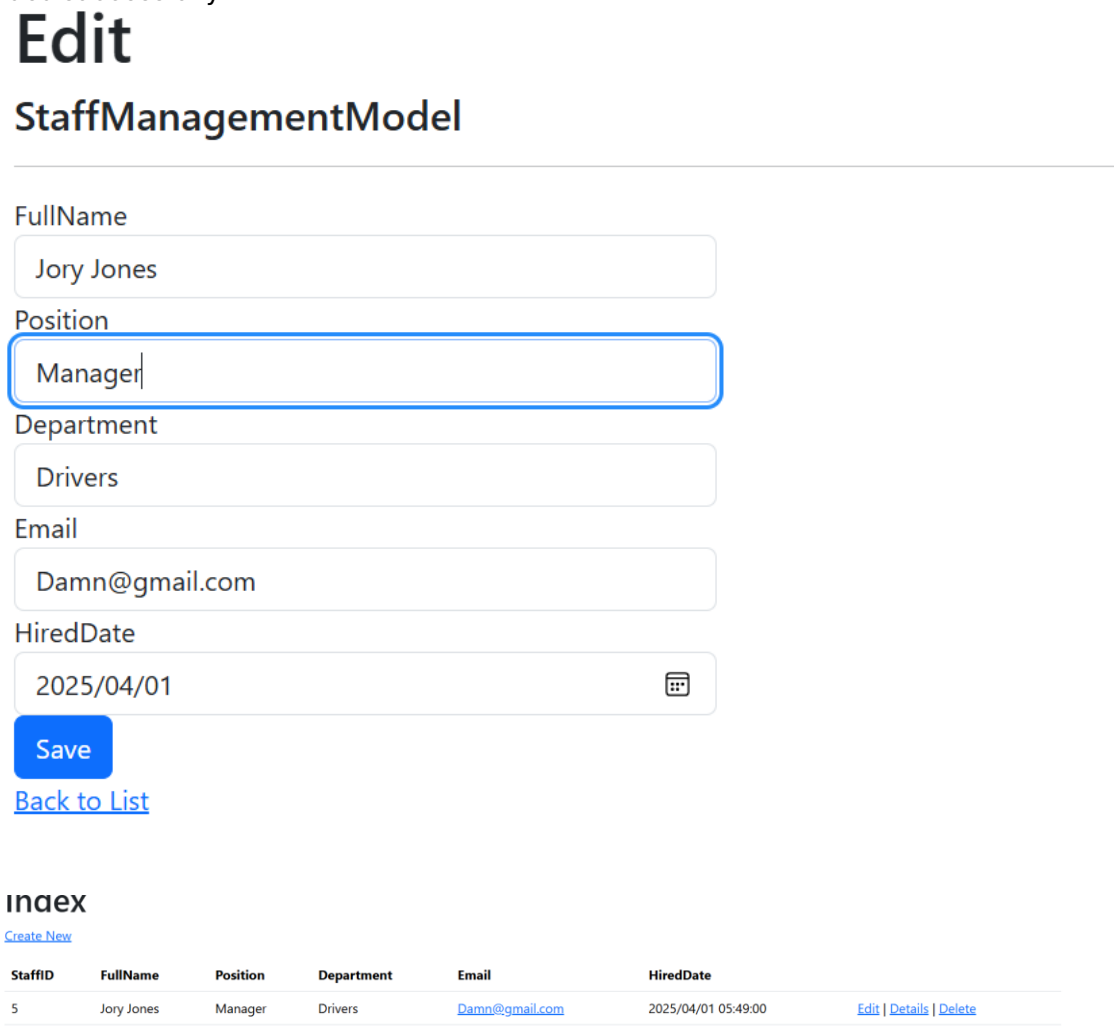
Back to List

Index

Create New

StaffID	FullName	Position	Department	Email	HiredDate	
5	Jory Jones	Driver	Drivers	Damn@gmail.com	2025/04/01 00:00:00	Edit | Details | Delete

Staff added successfully.’



Edit works successfully. Went from just a driver to Manager.

Phase 4:

Test Cases and Plan:

Plan:

The testing type used was manual and Unit Testing through xUnit. The citizens management, staff management, services requests and report management was tested.

I used Visual Studio, Entity Framework Core In-Memory Database and my Firefox’ browser developer tools.

Citizen

Test Case ID: TC001

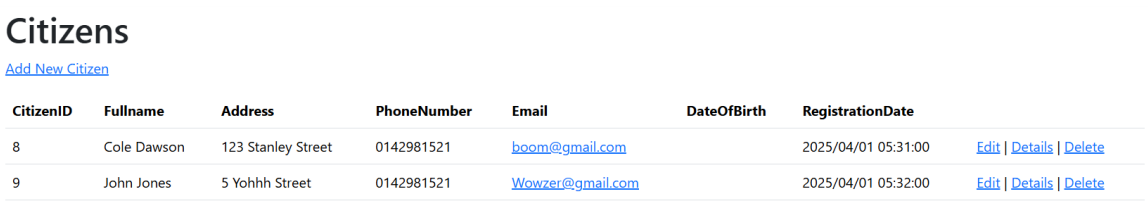
Title: View All Citizens

Test Steps: Navigate to /CitizenManagementModels

Test Data (Input): N/A

Expected Output: List of all citizens appears

Status: Pass



Citizens

Add New Citizen

CitizenID	Fullname	Address	PhoneNumber	Email	DateOfBirth	RegistrationDate	
8	Cole Dawson	123 Stanley Street	0142981521	boom@gmail.com		2025/04/01 05:31:00	Edit | Details | Delete
9	John Jones	5 Yohhh Street	0142981521	Wowzer@gmail.com		2025/04/01 05:32:00	Edit | Details | Delete

---

Test Case ID: TC002

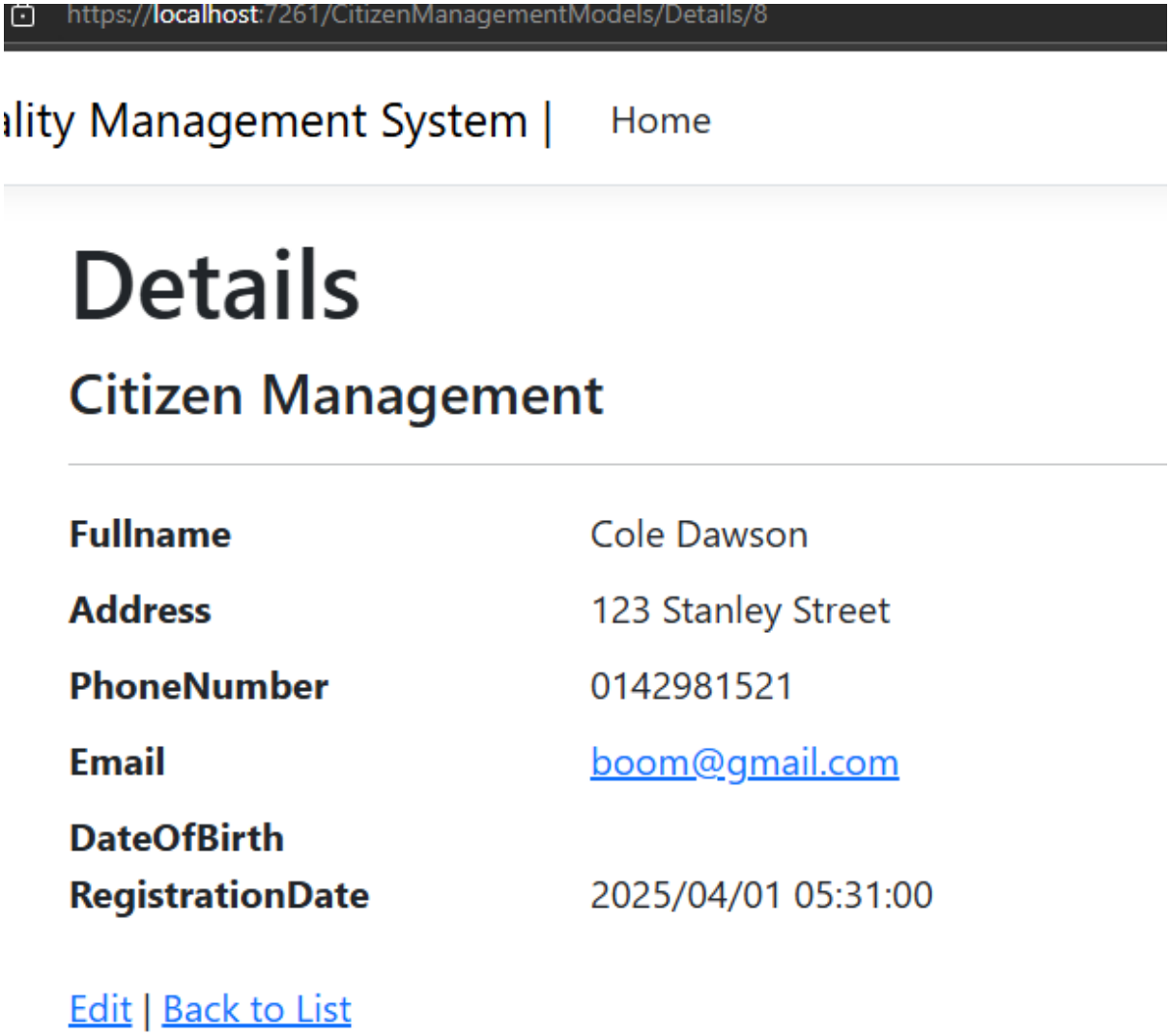
Title: View Citizen Details (Valid ID)

Test Steps: Click Details for citizen with ID = 8

Test Data (Input): CitizenID = 8

Expected Output: Full details of citizen #1 shown

Status: Pass



Test Case ID: TC003

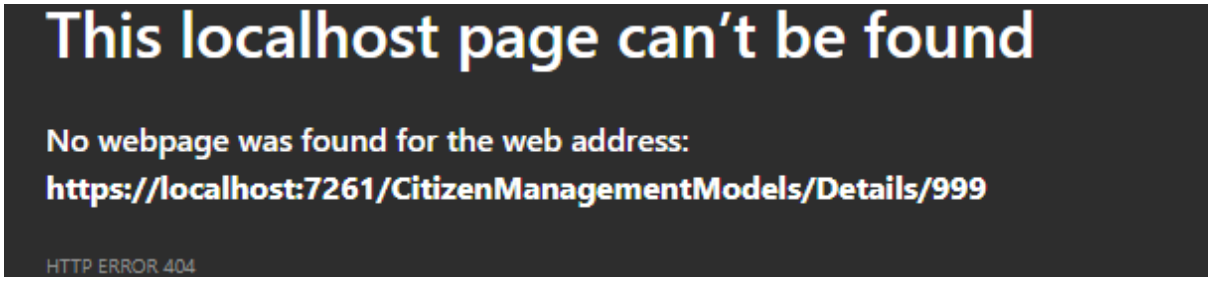
Title: View Citizen Details (Invalid ID)

Test Steps: Navigate to /Details/999

Test Data (Input): CitizenID = 999

Expected Output: NotFound or error page

Status: Pass



Test Case ID: TC004

Title: Create Citizen (Valid Input)

Test Steps: Fill form and submit

Test Data (Input): Fullname: "Amy Strawson", Email: "amy.strawson@gmail.com", etc.

Expected Output: Citizen created and redirected to Index

Status: Pass

Test Case ID: TC005

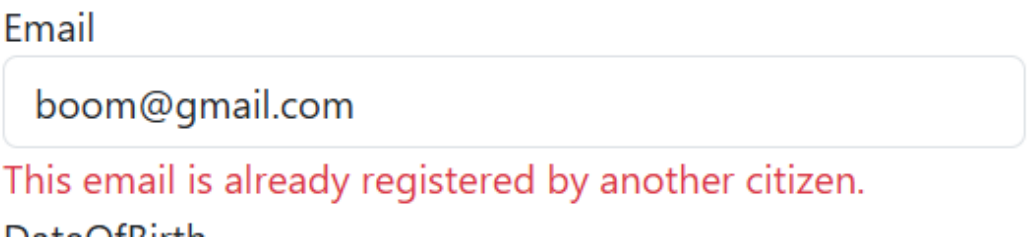
Title: Create Citizen (Duplicate Email)

Test Steps: Submit with an existing email

Test Data (Input): Email: "boom@gmail.com"

Expected Output: Duplicate email error

Status: Pass



Test Case ID: TC006

Title: Edit Citizen (Valid Update)

Test Steps: Change name and submit

Test Data (Input): New Name: "Amy Strawson The Second" etc

Expected Output: Record updated, redirected to Index

Status: Pass

Test Case ID: TC007

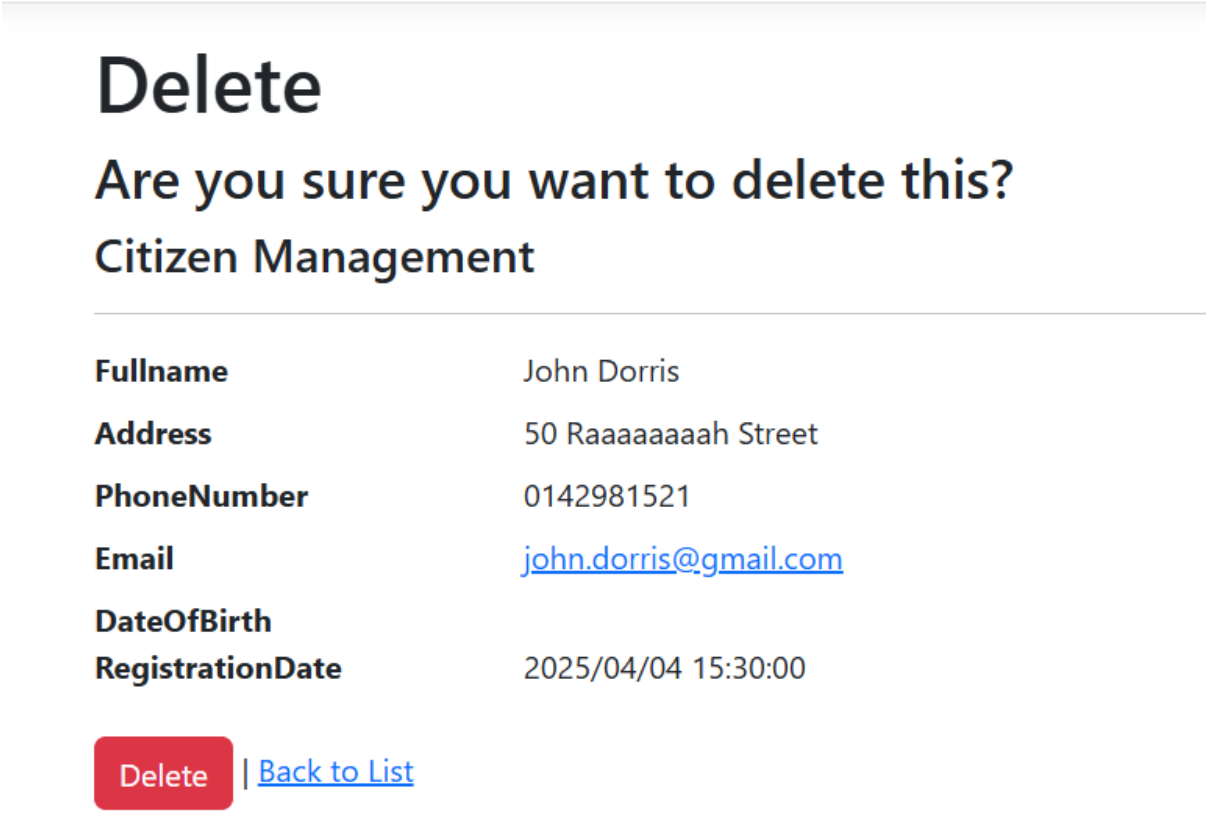
Title: Delete Citizen (Valid ID)

Test Steps: Confirm delete on citizen ID = 11

Test Data (Input): CitizenID = 11

Expected Output: Citizen deleted and redirected

Status: Pass



Reports:

Test Case ID: TC008

Title: View All Reports

Test Steps: Navigate to /ReportsModels

Test Data (Input): N/A

Expected Output: List of all reports appears

Status: Pass

ReportID	CitizenID	ReportType	Details	SubmissionDate	Status	
6	3	Damn	wow	2025/04/01 02:47:34	Pending	Review Report | Details | Delete

---

Test Case ID: TC009

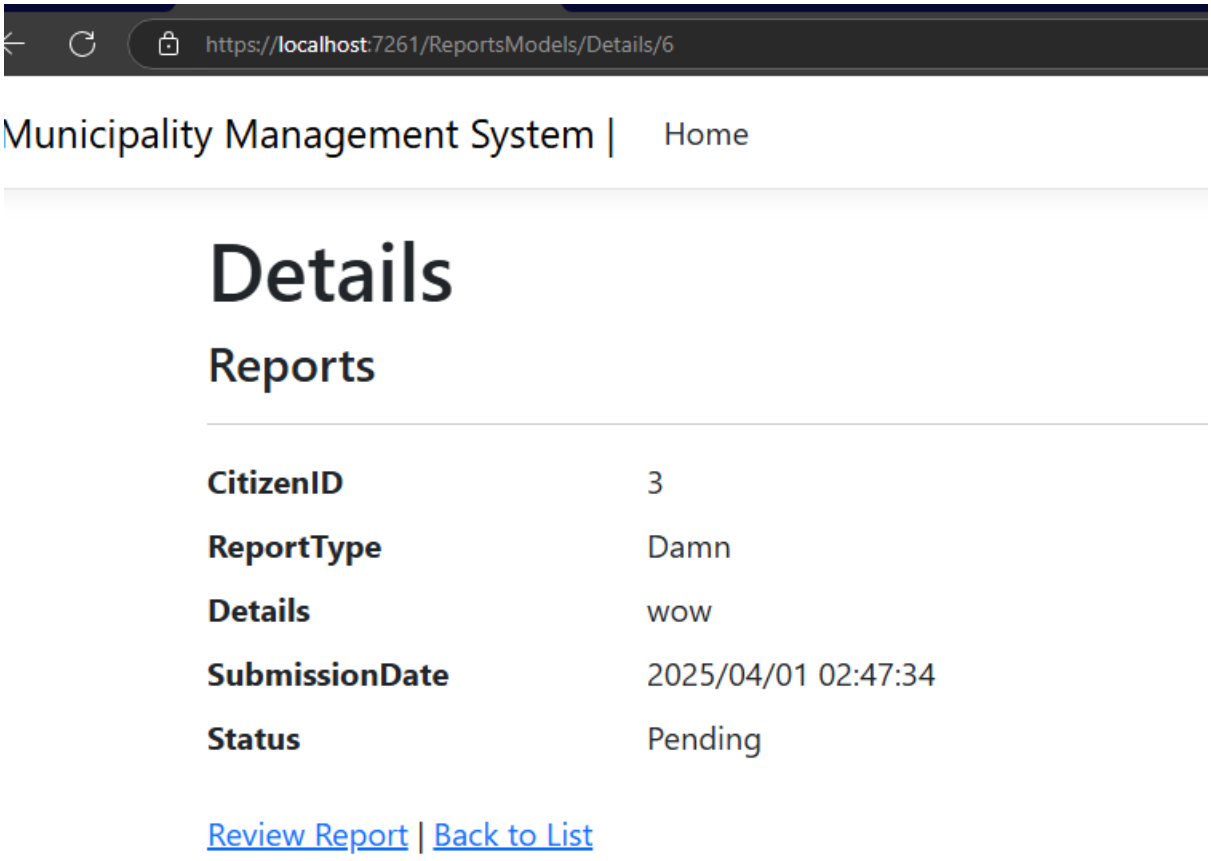
Title: View Report Details (Valid ID)

Test Steps: Click Details for report with ID = 6

Test Data (Input): ReportID = 6

Expected Output: Full details of report #1 shown

Status: Pass



Test Case ID: TC010

Title: Create Report (Valid Input)

Test Steps: Fill form with valid details and submit

Test Data (Input): CitizinID: 8, ReportType: "Incident", Details: "Ahh no", Date: Today etc

Expected Output: Report created and redirected to Index

Status: Pass

7	8	Incident	Ahhhh	2025/04/04 13:36:41	Pending	Review Report | Details | Delete

Test Case ID: TC011

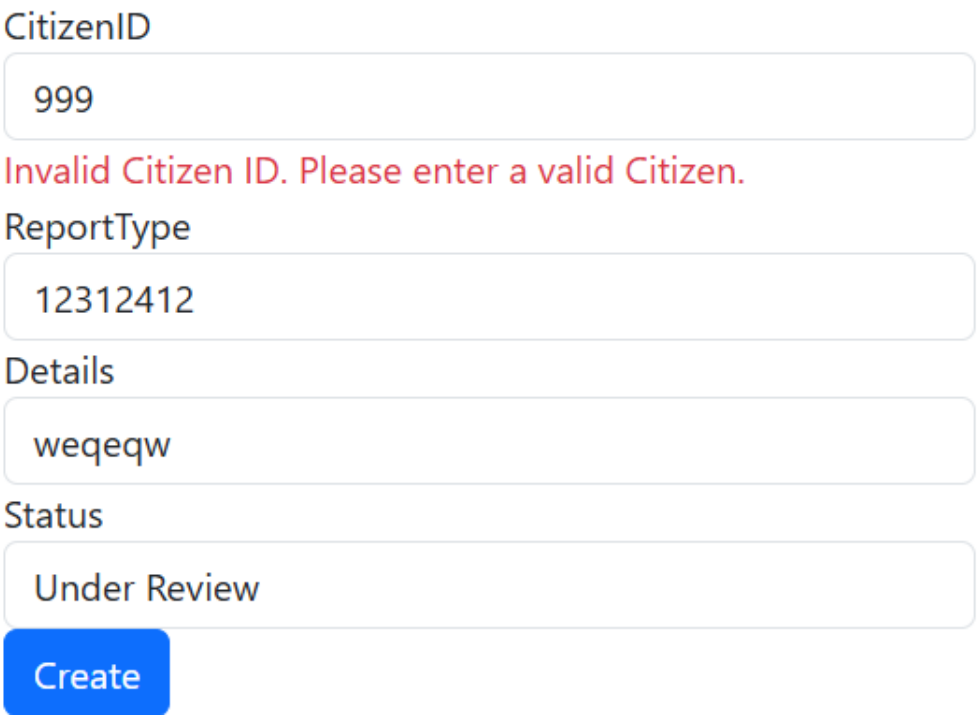
Title: Create Report (Invalid ID ☹️)

Test Steps: Submit form with invalid id

Test Data (Input): ReportType: "Incident", Details: "Save me." ID=999

Expected Output: Validation error messages appear

Status: Pass



Test Case ID: TC012

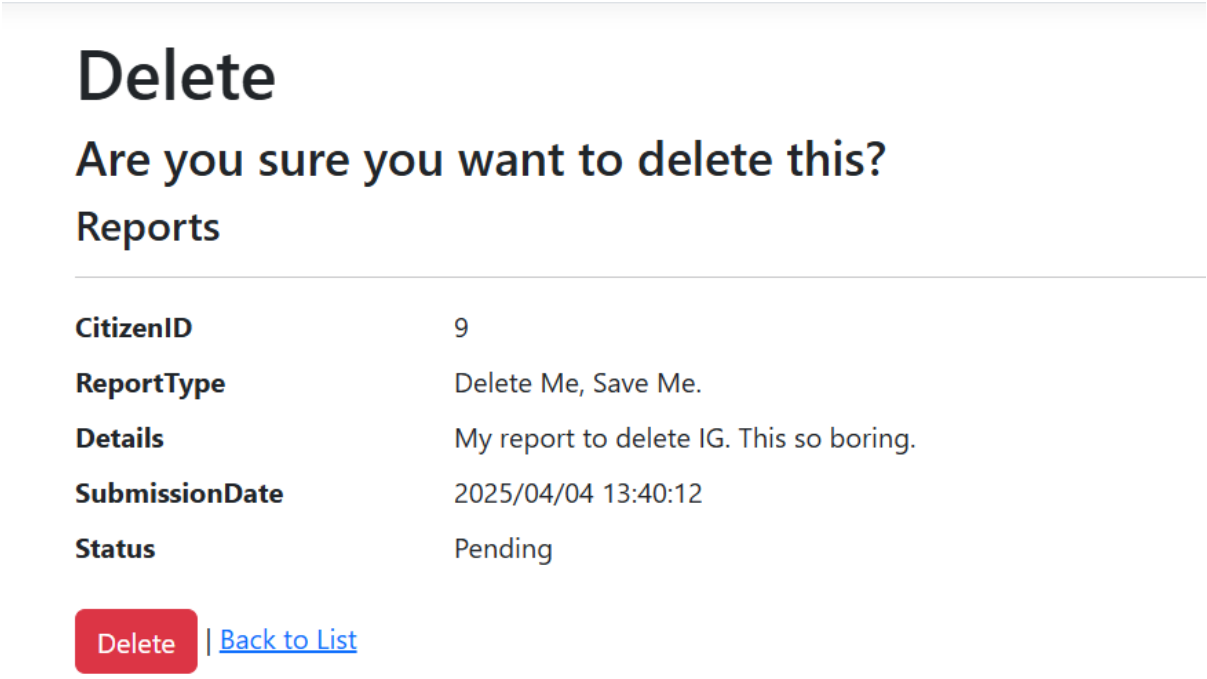
Title: Delete Report (Valid ID)

Test Steps: Navigate to Delete action with a valid report ID such 8

Test Data (Input): ReportID = 8

Expected Output: The report is fetched and displayed for confirmation.

Status: Pass



Test Case ID: TC013

Title: Delete Confirmed Report (Valid ID)

Test Steps: Confirm the deletion of the report with a valid ReportID , in this case 8

Test Data (Input): ReportID = 8

Expected Output: Report is deleted successfully, and the user is redirected to the Index page.

Status: Pass

ReportID	CitizenID	ReportType	Details	SubmissionDate	Status	
6	3	Damn	wow	2025/04/01 02:47:34	Pending	Review Report | Details | Delete
7	8	Incident	Ahhhh	2025/04/04 13:36:41	Pending	Review Report | Details | Delete

My boy is gone.

Services:

Test Case ID: TC014

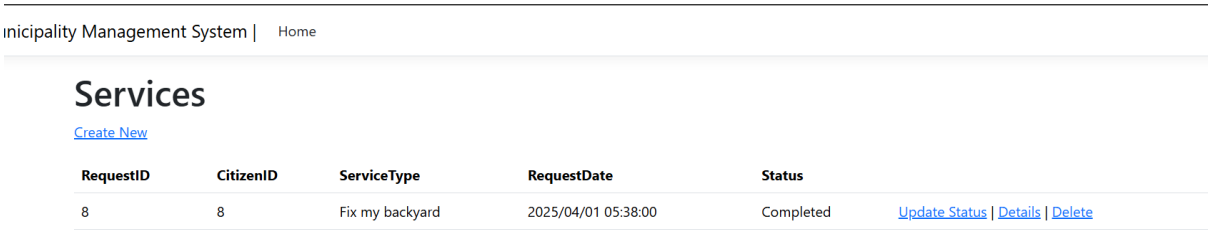
Title: View All Service Requests

Test Steps: Navigate to the Index /ServiceRequestModels

Test Data (Input): N/A

Expected Output: The list of service requests is displayed.

Status: Pass



Test Case ID: TC015

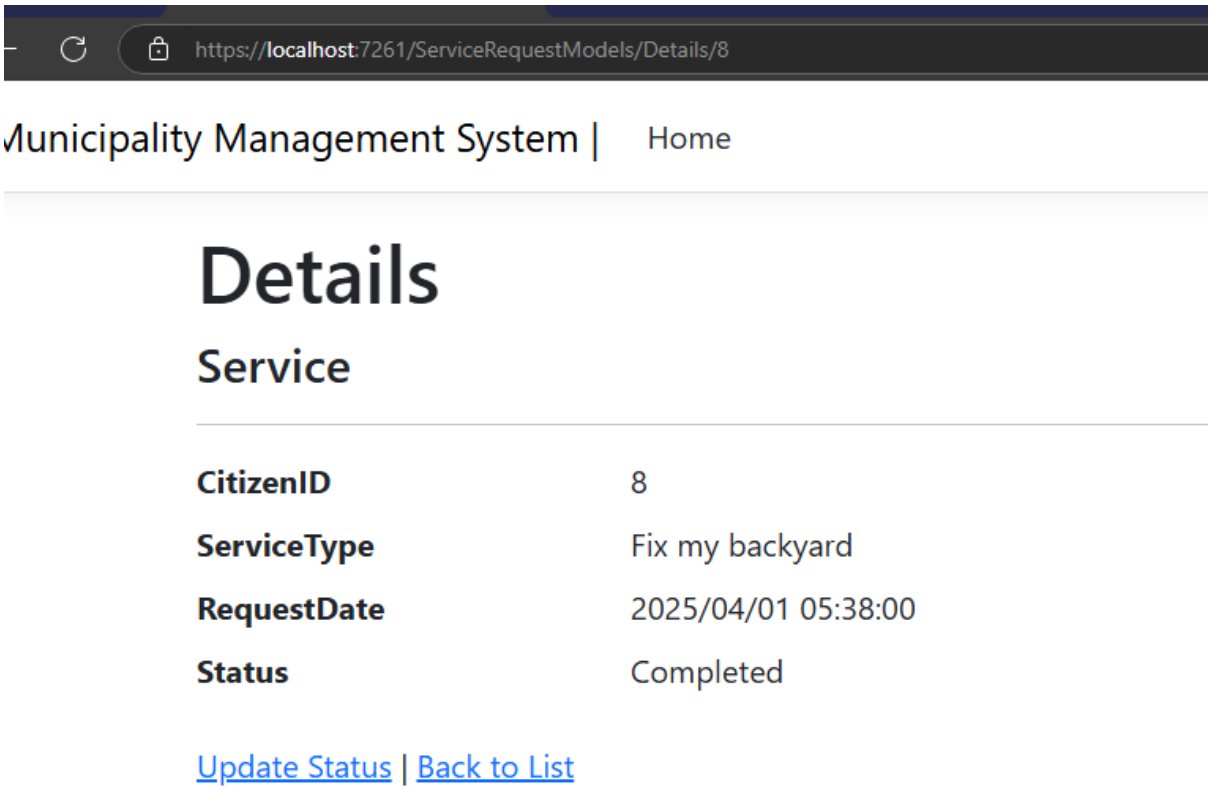
Title: View Service Request Details (Valid ID)

Test Steps: Navigate to the Details/8 in the web bar.

Test Data (Input): RequestID = 8

Expected Output: The details of the service request with the given RequestID are displayed.

Status: Pass



Test Case ID: TC016

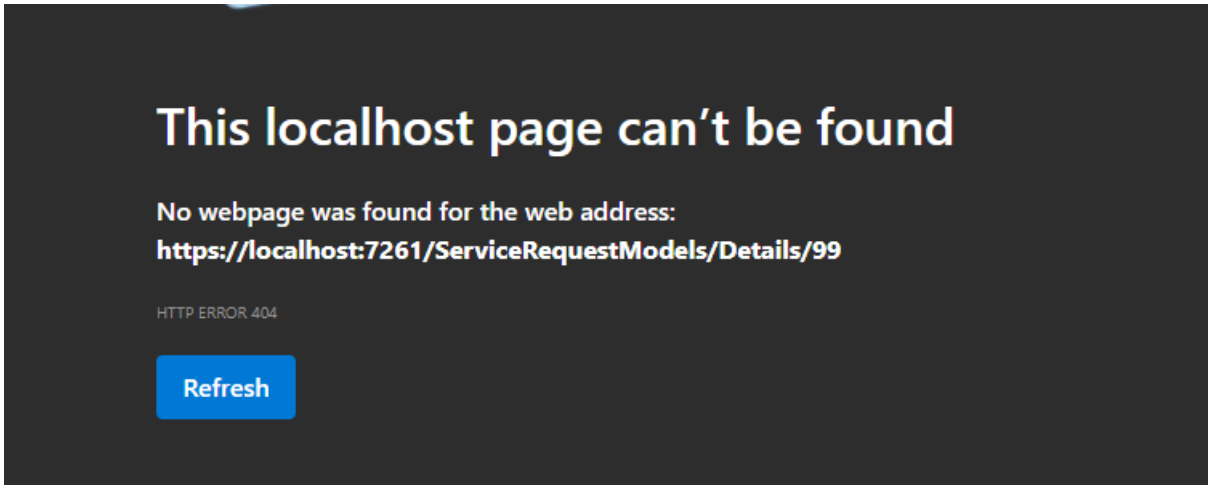
Title: View Service Request Details (Invalid ID)

Test Steps: Navigate to the Details/99 in the web bar on top

Test Data (Input): RequestID = 99

Expected Output: A "Not Found" page is displayed.

Status: Pass



Test Case ID: TC017

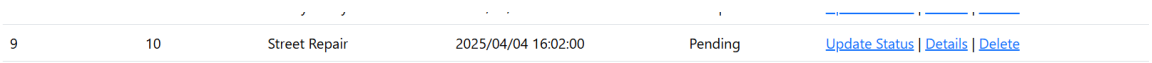
Title: Create Service Request (Valid CitizenID)

Test Steps: Submit a new service request with a valid CitizenID.

Test Data (Input): CitizenID = 10, ServiceType = "Street Repair", Status = "Pending"

Expected Output: The service request is created and the user is redirected to the Index page.

Status: Pass



Test Case ID: TC018

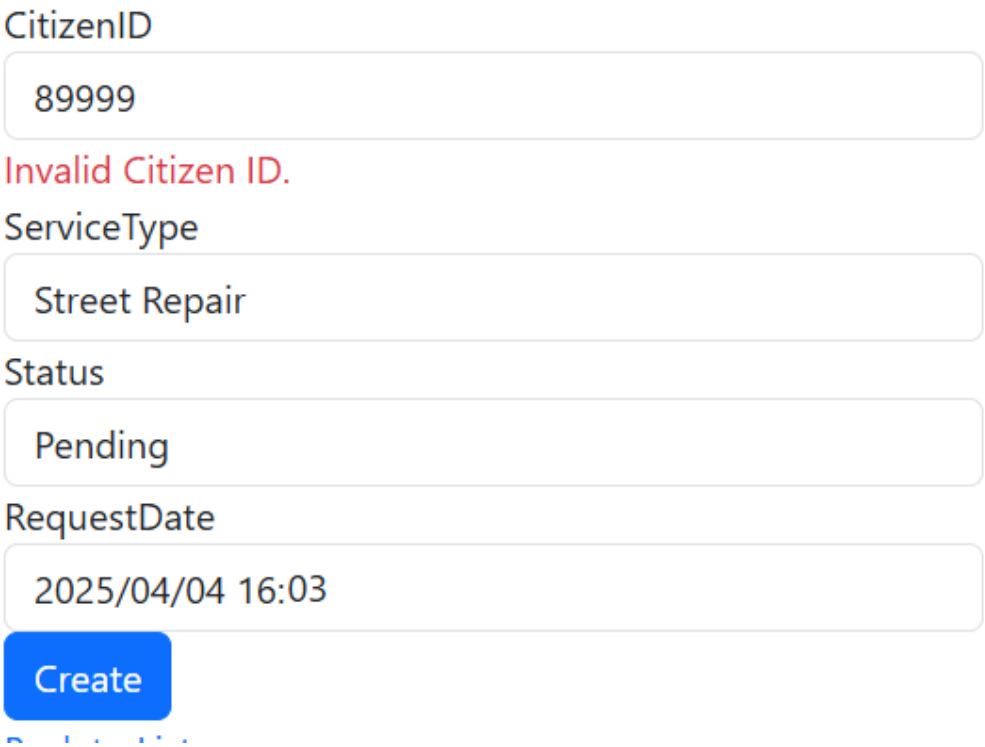
Title: Create Service Request (Invalid CitizenID)

Test Steps: Submit a new service request with an invalid CitizenID.

Test Data (Input): CitizenID = 89999, ServiceType = "Street Repair", Status = "Pending"

Expected Output: A validation error is shown, indicating the CitizenID is invalid.

Status: Pass



Test Case ID: TC019

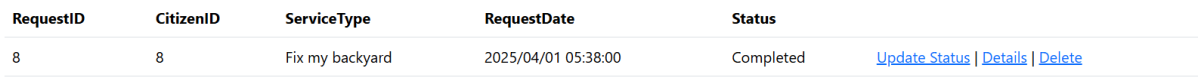
Title: Delete Service Request (Valid ID)

Test Steps: Delete a service request with a valid RequestID.

Test Data (Input): RequestID = 10

Expected Output: The service request is deleted and the user is redirected to the Index page.

Status: Pass



RequestID	CitizenID	ServiceType	RequestDate	Status	
8	8	Fix my backyard	2025/04/01 05:38:00	Completed	Update Status | Details | Delete

---

Test Case ID: TC020

Title: Update Service Request Status (Valid ID)

Test Steps: Update the status of a service request with valid ID in this case RequestID 8.

Test Data (Input): RequestID = 8, New Status = "Pending"

Expected Output: The service request status is updated, and the user is redirected to the Index page.

Status: Pass

RequestID	CitizenID	ServiceType	RequestDate	Status	
8	8	Fix my backyard	2025/04/01 05:38:00	Pending	Update Status | Details | Delete

Went from completed to pending.

Staff:

Test Case ID: TC021

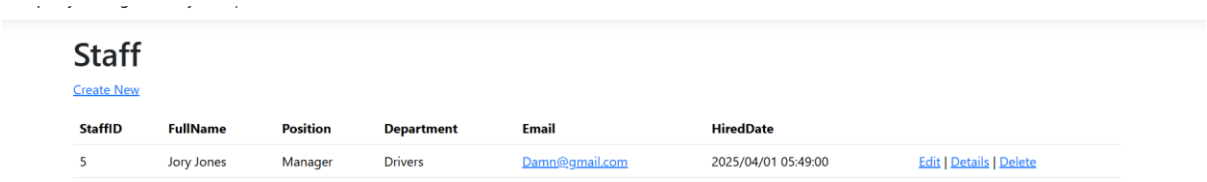
Title: Index (Returns View With Staff List)

Test Steps: Call the Index method on the StaffManagementModelsController.

Test Data (Input): None

Expected Output: The action returns a view with a list containing 1 staff member.

Status: Pass



Test Case ID: TC022

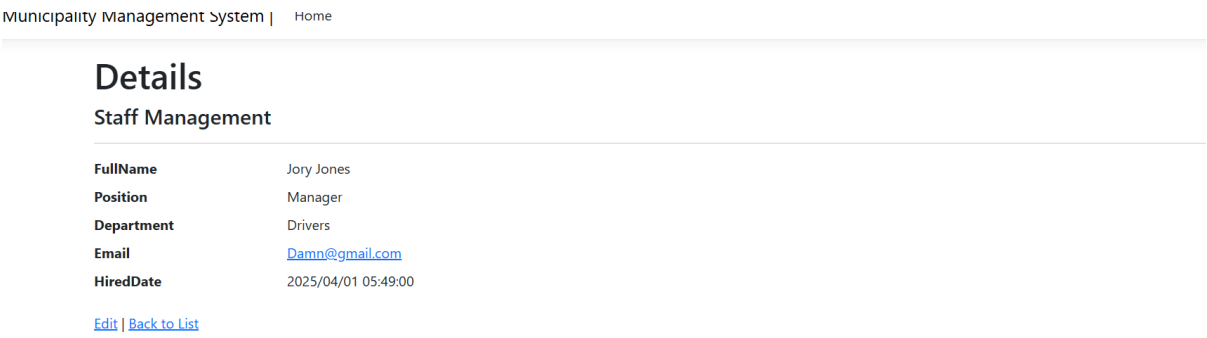
Title: Details (Valid ID Returns View With Staff Details)

Test Steps: Call the Details method with a valid StaffID, in this case 5

Test Data (Input): StaffID = 5

Expected Output: The action returns a view with the staff details for the given ID.

Status: Pass



Test Case ID: TC023

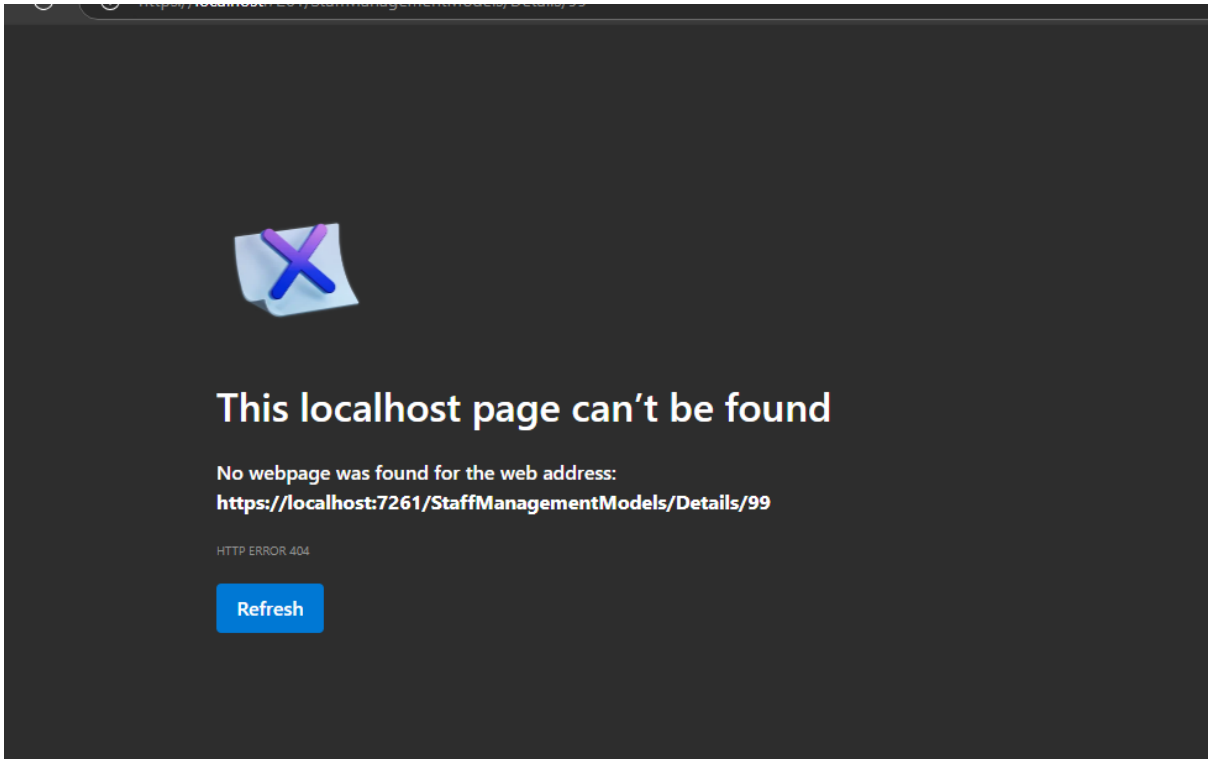
Title: Details (Invalid ID Returns Not Found)

Test Steps: Call the Details method with an invalid StaffID .

Test Data (Input): StaffID = 99

Expected Output: The action returns a NotFoundResult indicating the staff ID doesn't exist.

Status: Pass



Test Case ID: TC024

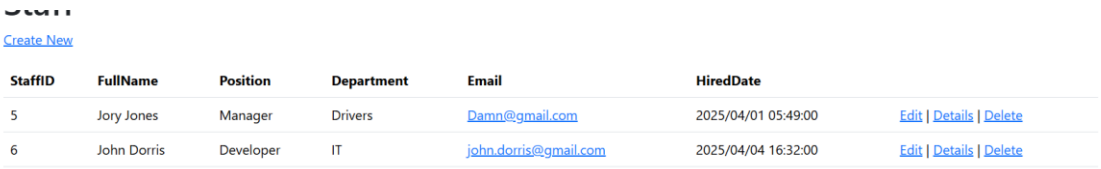
Title: Create (Valid Model Redirects To Index)

Test Steps: Call the Create method with a valid StaffManagementModel.

Test Data (Input): FullName = "John Dorris", Position = "Developer", Department = "IT", Email = "john.dorris @gmail.com", HiredDate = DateTime.Now

Expected Output: The action redirects to the Index page, and the staff member is added to the database.

Status: Pass



Test Case ID: TC025

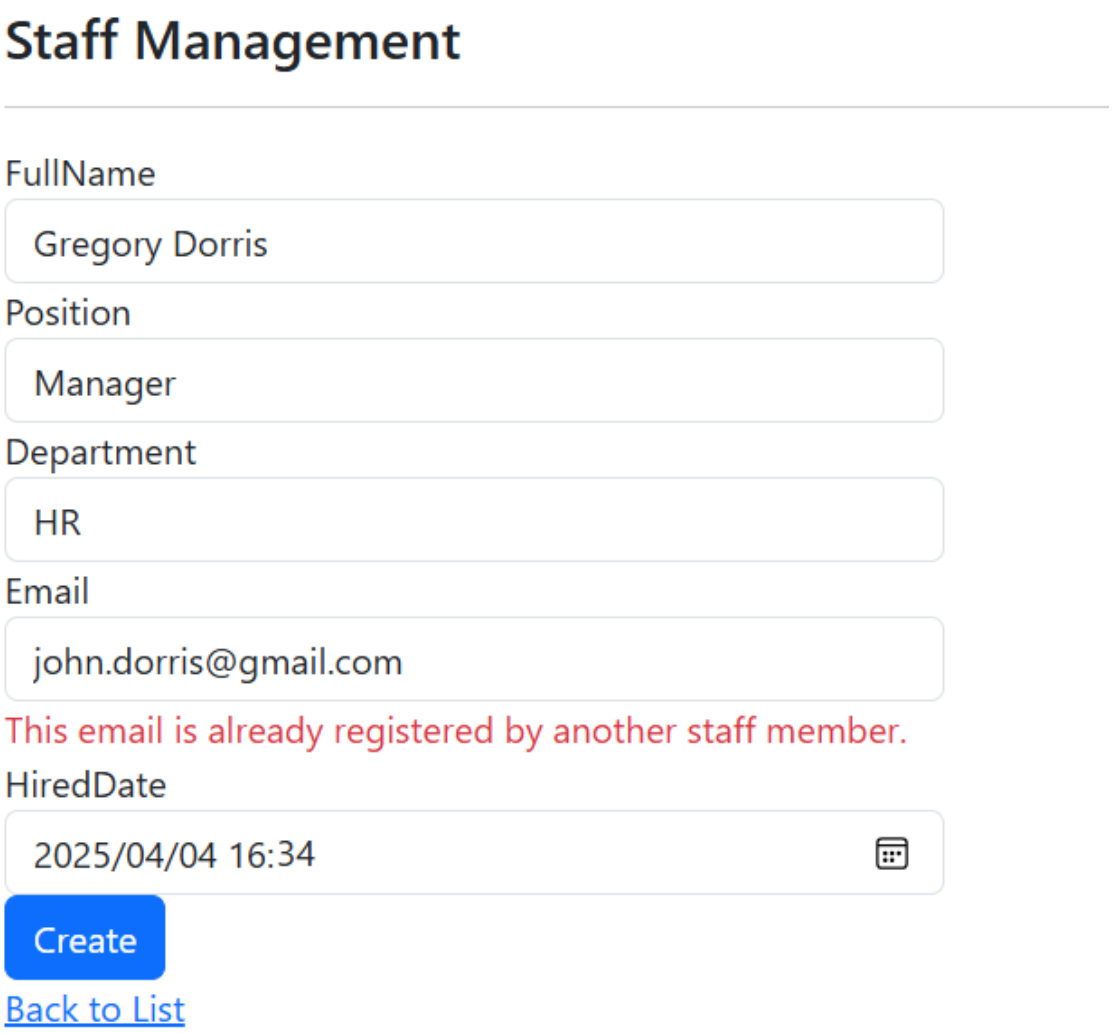
Title: Create (Existing Email Returns View With Error)

Test Steps: Call the Create method with a StaffManagementModel containing an email that already exists in the database.

Test Data (Input): FullName = "Gregory Dorris", Position = "Manager", Department = "HR", Email = "john.dorris@gmail.com", HiredDate = DateTime.Now

Expected Output: The action returns the view with an error indicating that the email is already registered.

Status: Pass



Test Case ID: TC026

Title: Edit (Valid ID Returns View With Staff Details)

Test Steps: Call the Edit method with a valid StaffID in this case 5 and updated staff data.

Test Data (Input): StaffID = 5, FullName = "Jory Jones The Second"

Expected Output: The action redirects to the Index page, and the staff details are updated.

Status: Pass

7	Gregory Dorris	Manager	HR		gregory.dorris@gmail.com	2025/04/04 16:34:00	Edit | Details | Delete

7	Gregory Dorris The Second	Manager	HR	gregory.dorris@gmail.com	2025/04/04 16:39:00	Edit | Details | Delete

---

Test Case ID: TC027

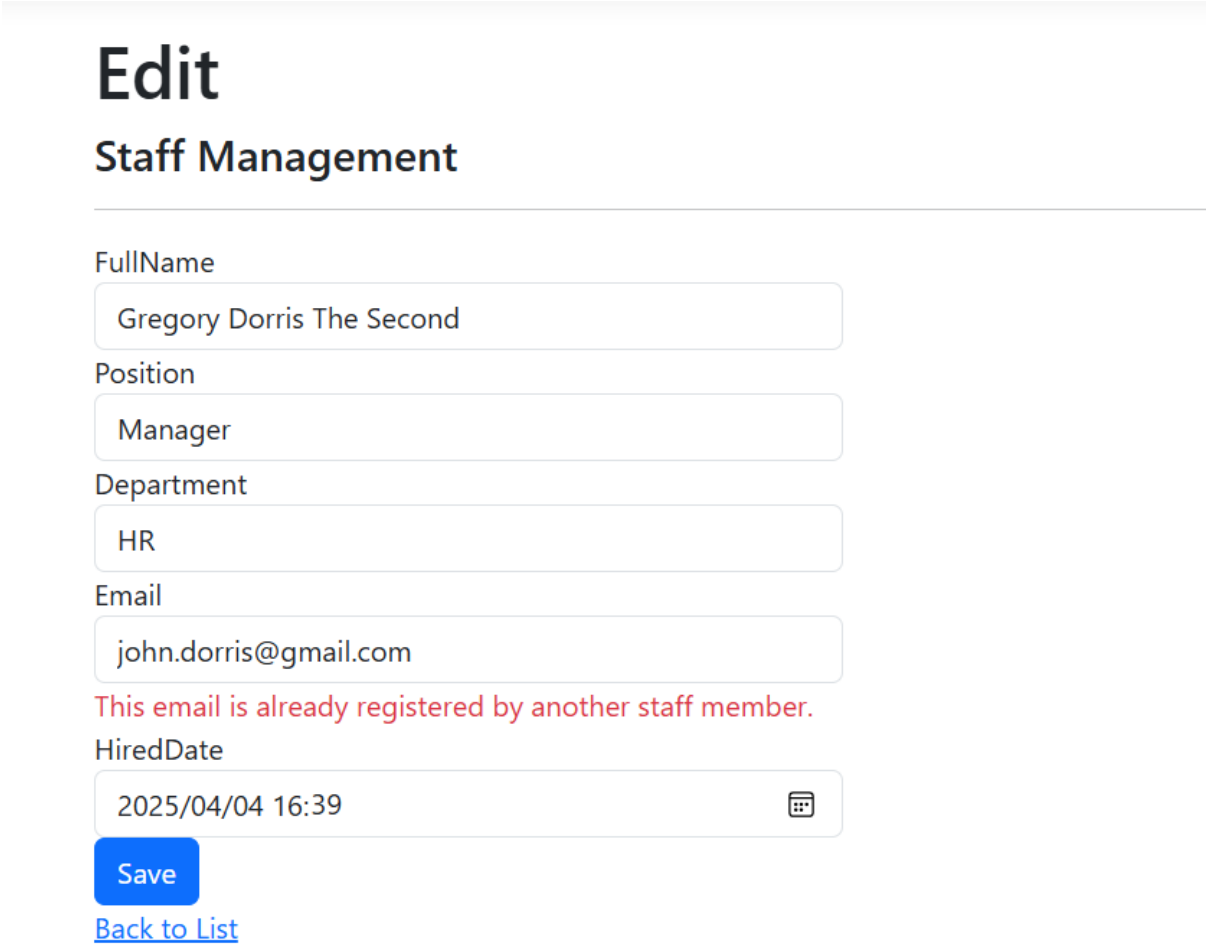
Title: Edit (Invalid Email Returns View With Error)

Test Steps: Call the Edit method with a valid StaffID and a new email that already exists in the database.

Test Data (Input): StaffID = 2, Email = "john.dorris@gmail.com"

Expected Output: The action returns the view with an error message indicating the email is already registered by another staff member.

Status: Pass



Test Case ID: TC028

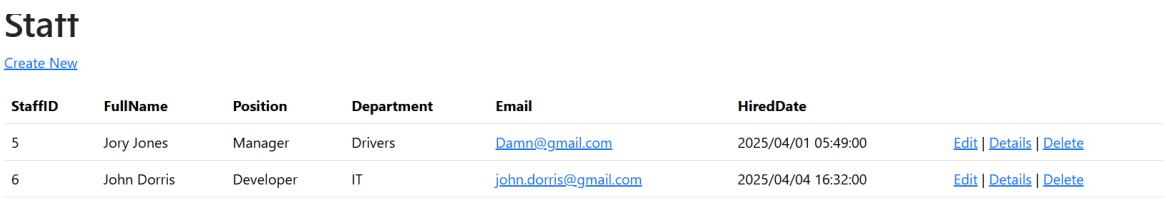
Title: Delete (Valid ID Redirects To Index)

Test Steps: Call the DeleteConfirmed method with a valid StaffID to delete a staff member.

Test Data (Input): StaffID = 7

Expected Output: The action redirects to the Index page, and the staff member is deleted from the database.

Status: Pass



My goat Gregory Is gone.

Test Case ID: TC029

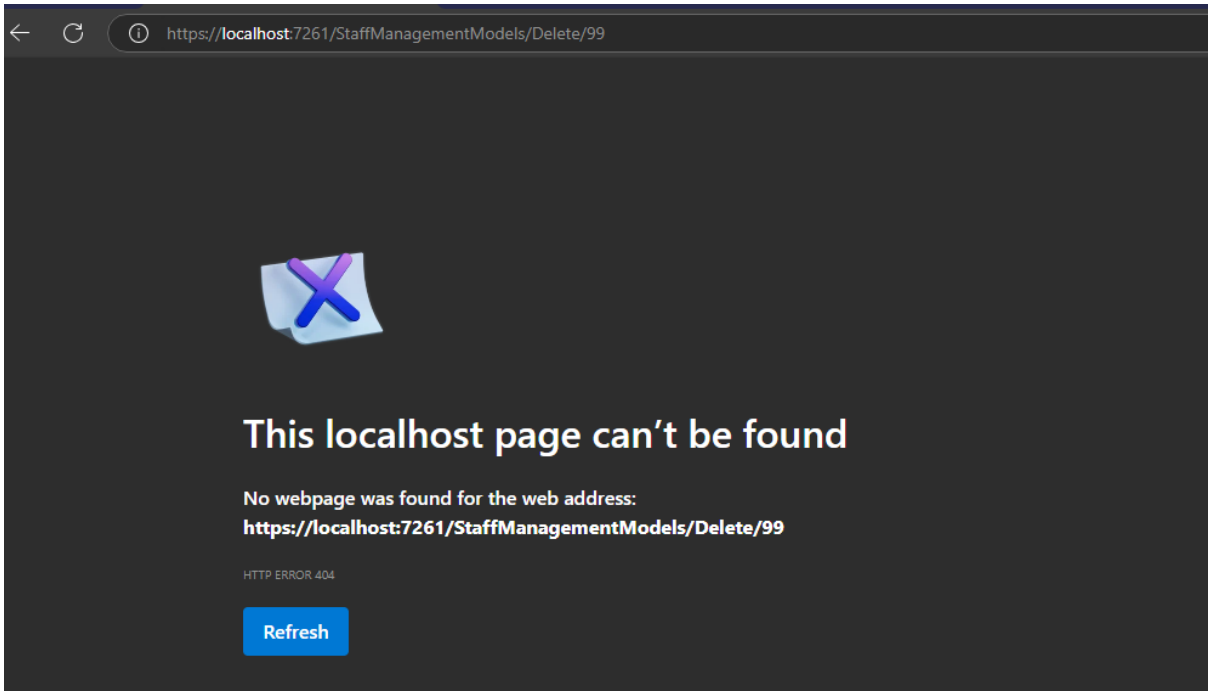
Title: Delete (Invalid ID Returns Not Found)

Test Steps: Call the Delete method with an invalid StaffID.

Test Data (Input): StaffID = 99

Expected Output: The action returns a NotFoundResult indicating the staff member was not found.

Status: Pass



Bug Reports and Resolutions:

The seeded database data carried over between tests, resulting in test failures.

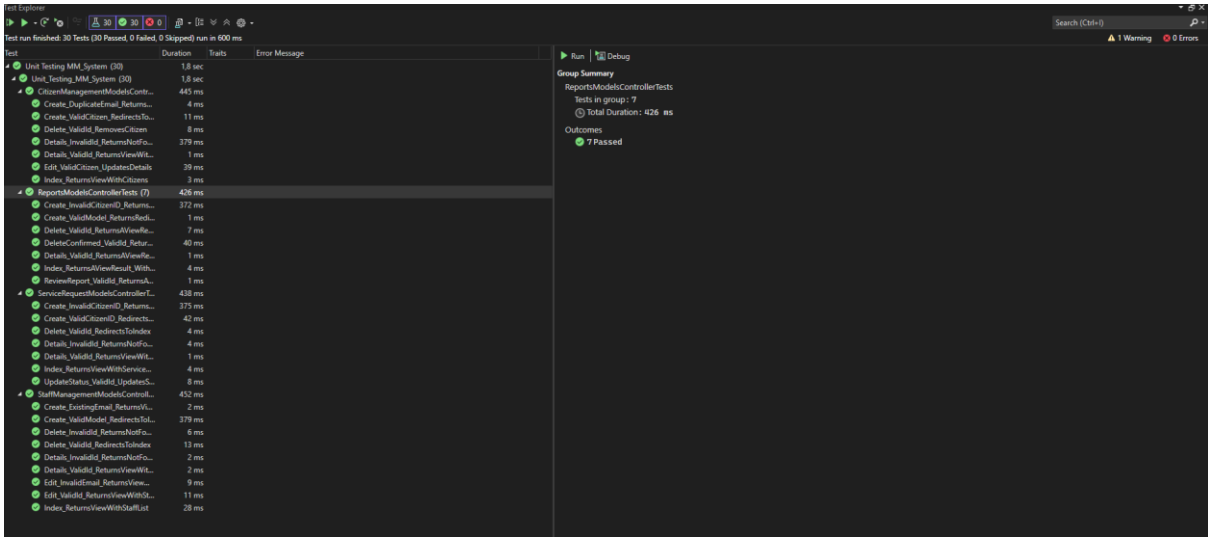
```
var options = new DbContextOptionsBuilder<MunicipalityManagementSystemV2Context>()
    .UseInMemoryDatabase(databaseName: "CitizenTestDB_" + Guid.NewGuid().ToString())
    .Options;
```

The Guid.NewGuid generated a unique database name each time, so every test run was fresh with no leftover data from others test.

Some entities couldn't be tracked because another instance with the same key was already tracked.

```
// Retrieve existing citizen
var existingCitizen = await dbContext.CitizenManagementModel.FindAsync(1);

if (existingCitizen != null)
{
    dbContext.Entry(existingCitizen).State = EntityState.Detached; // Detach the existing entity. Fixing the issue of two entities already being tracked.
}
```

Screenshots of unit test results:



Explanations in comments.

Citizen Tests:

```
namespace Unit_Testing_MM_System
{
    0 references
    public class CitizenManagementModelsControllerTests
    {
        //uses EF to create fake database
        7 references | 7/7 passing
        private async Task<MunicipalityManagementSystemV2Context> GetDatabaseContext()
        {
            var options = new DbContextOptionsBuilder<MunicipalityManagementSystemV2Context>()
                .UseInMemoryDatabase(databaseName: "CitizenTestDB_" + Guid.NewGuid().ToString())
                .Options;

            var databaseContext = new MunicipalityManagementSystemV2Context(options);
            databaseContext.Database.EnsureCreated(); // Only uses it if already created

            //create test citizen if the fake database is empty
            if (!databaseContext.CitizenManagementModel.Any())
            {
                databaseContext.CitizenManagementModel.Add(new CitizenManagementModel
                {
                    CitizenID = 1,
                    Fullname = "John Dorris",
                    Address = "123 Main Street",
                    PhoneNumber = "0833902460",
                    Email = "johndoe@gmail.com",
                    DateOfBirth = new System.DateTime(1990, 5, 15),
                    RegistrationDate = System.DateTime.Now
                });

                await databaseContext.SaveChangesAsync();
            }
            return databaseContext;
        }
```

```
        //Makes sure the index view reuturns the list of citizens and not empty.
        [Fact]
        | 0 references
        public async Task Index_ReturnsViewWithCitizens()
        {
            var dbContext = await GetDatabaseContext(); //creates context with fake data.
            var controller = new CitizenManagementModelsController(dbContext); //puts context into controller.

            var result = await controller.Index(); //call the index action

            var viewResult = Assert.IsType<ViewResult>(result); //Makes sure it is returning a view
            var model = Assert.IsAssignableFrom<List<CitizenManagementModel>>(viewResult.Model); //Make sure model is a list of citizens
            Assert.Single(model); //It only expects one record,
        }

        //Tests getting citizen details when valid id is used.
        [Fact]
        | 0 references
        public async Task Details_ValidId_ReturnsViewWithCitizen()
        {
            var dbContext = await GetDatabaseContext();
            var controller = new CitizenManagementModelsController(dbContext);

            var result = await controller.Details(1); //Call details with valid ID.

            var viewResult = Assert.IsType<ViewResult>(result);
            var model = Assert.IsType<CitizenManagementModel>(viewResult.Model); //Makes sure it returns a citizen model.
            Assert.Equal(1, model.CitizenID); // Confirm correct citizen was returned
        }

        //Tests for getting details of invalid id
        [Fact]
        | 0 references
        public async Task Details_InvalidId_ReturnsNotFound()
        {
            var dbContext = await GetDatabaseContext();
            var controller = new CitizenManagementModelsController(dbContext);

            var result = await controller.Details(999); // Non-existent ID

            Assert.IsType<NotFoundResult>(result);
        }
```

```
    // Tests creating a new valid citizen and redirects properly.
    [Fact]
    | 0 references
    public async Task Create_ValidCitizen_RedirectsToIndex()
    {
        var dbContext = await GetDatabaseContext();
        var controller = new CitizenManagementModelsController(dbContext);

        var newCitizen = new CitizenManagementModel
        {
            Fullname = "Amy Strawson",
            Address = "456 Elm Street",
            PhoneNumber = "0833902430",
            Email = "amy.strawson@gmail.com",
            DateOfBirth = new System.DateTime(1995, 8, 20),
            RegistrationDate = System.DateTime.Now
        };

        var result = await controller.Create(newCitizen); //calls the create post.

        var redirectResult = Assert.IsType<RedirectToActionResult>(result);
        Assert.Equal("Index", redirectResult.ActionName);  //makes sure redirect happens.

        Assert.Equal(2, dbContext.CitizenManagementModel.Count()); //Confirms there is now 2 citizens.
    }
```

```
// Tests updating a citizen and saves the changes.
[Fact]
0 references
public async Task Edit_ValidCitizen_UpdatesDetails()
{
    var dbContext = await GetDatabaseContext();
    var controller = new CitizenManagementModelsController(dbContext);

    // Retrieve existing citizen
    var existingCitizen = await dbContext.CitizenManagementModel.FindAsync(1);

    if (existingCitizen != null)
    {
        dbContext.Entry(existingCitizen).State = EntityState.Detached; // Detach the existing entity. Fixing the issue of two entities already being tracked.
    }

    var updatedCitizen = new CitizenManagementModel
    {
        CitizenID = 1, // Should be existing record other fail.
        Fullname = "John Dorris The Second",
        Address = "789 Dorris Street",
        PhoneNumber = "0833902461",
        Email = "johndorris_updated@gmail.com",
        DateOfBirth = new System.DateTime(1990, 5, 15),
        RegistrationDate = System.DateTime.Now
    };

    var result = await controller.Edit(1, updatedCitizen); //calls edit post methdo

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName); // confirms if redirects after update

    var citizenInDb = await dbContext.CitizenManagementModel.FindAsync(1);
    Assert.Equal("John Dorris The Second", citizenInDb.Fullname); //check if name update
    Assert.Equal("789 Dorris Street", citizenInDb.Address); //check if address updated.
}
```

```
    // Tests deleting a valid citizen
    [Fact]
    0 references
    public async Task Delete_ValidId_RemovesCitizen()
    {
        var dbContext = await GetDatabaseContext();
        var controller = new CitizenManagementModelsController(dbContext);

        var result = await controller.DeleteConfirmed(1); //call delete confirmed method.

        var redirectResult = Assert.IsType<RedirectToActionResult>(result);
        Assert.Equal("Index", redirectResult.ActionName); //makes sure redirect after delete

        var citizenInDb = await dbContext.CitizenManagementModel.FindAsync(1);
        Assert.Null(citizenInDb); // Ensure citizen was deleted
    }

    [Fact]
    0 references
    public async Task Create_DuplicateEmail_ReturnsError()
    {
        var dbContext = await GetDatabaseContext();
        var controller = new CitizenManagementModelsController(dbContext);

        // Add a new citizen with the same email as the one added at the start.
        var duplicateCitizen = new CitizenManagementModel
        {
            Fullname = "I don't care",
            Address = "321 IDC Street",
            PhoneNumber = "0833901111",
            Email = "johndoe@gmail.com", // dupe email
            DateOfBirth = new System.DateTime(1985, 4, 10),
            RegistrationDate = System.DateTime.Now
        };

        if (dbContext.CitizenManagementModel.Any(c => c.Email == duplicateCitizen.Email))
        {
            controller.ModelState.AddModelError("Email", "Email already exists");
        }

        var result = await controller.Create(duplicateCitizen);

        var viewResult = Assert.IsType<ViewResult>(result);
        Assert.False(controller.ModelState.IsValid);
        Assert.True(controller.ModelState.ContainsKey("Email"));
    }
}
```

Service Requests:

```
{
    0 references
    public class ServiceRequestModelsControllerTests
    {
        // Method to create a fresh controller and context for each test. Used a better(?) way to do it in my citizens and staff, but not gonna change it up now.
        7 references | 7/7 passing
        private async Task<MunicipalityManagementSystemV2Context> GetDatabaseContext(string dbName)
        {
            var options = new DbContextOptionsBuilder<MunicipalityManagementSystemV2Context>()
                .UseInMemoryDatabase(databaseName: dbName)
                .Options;

            var databaseContext = new MunicipalityManagementSystemV2Context(options);
            databaseContext.Database.EnsureCreated();

            if (!databaseContext.ServiceRequestModel.Any())
            {
                databaseContext.ServiceRequestModel.Add(new ServiceRequestModel
                {
                    RequestID = 1,
                    CitizenID = 101,
                    ServiceType = "Water Supply Issue",
                    RequestDate = System.DateTime.Now,
                    Status = "Pending"
                });

                await databaseContext.SaveChangesAsync();
            }
            return databaseContext;
        }
```

```
        [Fact]
        | 1 reference | 1/1 passing
        public async Task Index_ReturnsViewWithServiceRequests()
        {
            var dbContext = await GetDatabaseContext(nameof(Index_ReturnsViewWithServiceRequests));
            var controller = new ServiceRequestModelsController(dbContext);

            var result = await controller.Index();

            var viewResult = Assert.IsType<ViewResult>(result);
            var model = Assert.IsAssignableFrom<List<ServiceRequestModel>>(viewResult.Model);
            Assert.Single(model); // Ensure one request exists
        }

        [Fact]
        | 1 reference | 1/1 passing
        public async Task Details_ValidId_ReturnsViewWithServiceRequest()
        {
            var dbContext = await GetDatabaseContext(nameof(Details_ValidId_ReturnsViewWithServiceRequest));
            var controller = new ServiceRequestModelsController(dbContext);

            var result = await controller.Details(1);

            var viewResult = Assert.IsType<ViewResult>(result);
            var model = Assert.IsType<ServiceRequestModel>(viewResult.Model);
            Assert.Equal(1, model.RequestID); // Check if correct service request is retrieved
        }

        [Fact]
        | 1 reference | 1/1 passing
        public async Task Details_InvalidId_ReturnsNotFound()
        {
            var dbContext = await GetDatabaseContext(nameof(Details_InvalidId_ReturnsNotFound));
            var controller = new ServiceRequestModelsController(dbContext);

            var result = await controller.Details(99);

            Assert.IsType<NotFoundResult>(result); // Check if NotFoundResult is returned
        }
```

```
        [Fact]
        | 1 reference | 1/1 passing
        public async Task Create_ValidCitizenID_RedirectsToIndex()
        {
            var dbContext = await GetDatabaseContext(nameof(Create_ValidCitizenID_RedirectsToIndex));
            var controller = new ServiceRequestModelsController(dbContext);

            var citizen = new CitizenManagementModel
            {
                CitizenID = 101,
                Fullname = "John Dorris",
                Address = "123 Main St",
                PhoneNumber = "08392450",
                Email = "john.dorris@gmail.com"
            };
            dbContext.CitizenManagementModel.Add(citizen);
            await dbContext.SaveChangesAsync();

            // Ensure no service requests are in the database before adding a new one
            dbContext.ServiceRequestModel.RemoveRange(dbContext.ServiceRequestModel); // Remove any existing requests
            await dbContext.SaveChangesAsync();

            var serviceRequest = new ServiceRequestModel
            {
                CitizenID = 101,
                ServiceType = "Street Repair",
                RequestDate = System.DateTime.Now,
                Status = "Pending"
            };

            var result = await controller.Create(serviceRequest);

            var redirectResult = Assert.IsType<RedirectToActionResult>(result);
            Assert.Equal("Index", redirectResult.ActionName); // Should redirect to index
            Assert.Single(dbContext.ServiceRequestModel); // Ensure service request is saved
        }
```

```
[Fact]
| 1 reference | 1/1 passing
public async Task Create_InvalidCitizenID_ReturnsViewWithError()
{
    var dbContext = await GetDatabaseContext(nameof(Create_InvalidCitizenID_ReturnsViewWithError));
    var controller = new ServiceRequestModelsController(dbContext);

    var serviceRequest = new ServiceRequestModel
    {
        CitizenID = 9999, // Invalid CitizenID
        ServiceType = "Street Repair",
        RequestDate = System.DateTime.Now,
        Status = "Pending"
    };

    var result = await controller.Create(serviceRequest);

    var viewResult = Assert.IsType<ViewResult>(result);
    var modelState = viewResult.ViewData.ModelState;
    Assert.True(modelState.ContainsKey("CitizenID"));
    Assert.Contains("Invalid Citizen ID.", modelState["CitizenID"].Errors.Select(e => e.ErrorMessage).ToList()); // Ensure correct error message is returned
}
```

```
[Fact]
| 1 reference | 1/1 passing
public async Task Delete_ValidId_RedirectsToIndex()
{
    var dbContext = await GetDatabaseContext(nameof(Delete_ValidId_RedirectsToIndex));
    var controller = new ServiceRequestModelsController(dbContext);

    var serviceRequest = new ServiceRequestModel
    {
        CitizenID = 101,
        ServiceType = "Water Supply Issue",
        RequestDate = DateTime.Now,
        Status = "Pending"
    };
    dbContext.ServiceRequestModel.Add(serviceRequest);
    await dbContext.SaveChangesAsync();

    var result = await controller.DeleteConfirmed(serviceRequest.RequestID);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName); // Redirect to index after deletion

    var deletedRequest = await dbContext.ServiceRequestModel
        .FirstOrDefaultAsync(r => r.RequestID == serviceRequest.RequestID);
    Assert.Null(deletedRequest); // Ensure the service request is deleted
}
```

```
[Fact]
| 1 reference | 1/1 passing
public async Task UpdateStatus_ValidId_UpdatesStatusAndRedirects()
{
    var dbContext = await GetDatabaseContext(nameof(UpdateStatus_ValidId_UpdatesStatusAndRedirects));
    var controller = new ServiceRequestModelsController(dbContext);

    // Retrieve the existing service request to update it.
    var serviceRequest = await dbContext.ServiceRequestModel.FindAsync(1);

    //Detach the entities to avoid conflicts with tracking
    var trackedEntity = dbContext.Entry(serviceRequest);
    trackedEntity.State = EntityState.Detached;

    var result = await controller.UpdateStatus(1, "Completed"); //update the service

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName); // Redirect to index after status update

    // Ensure the service request's status was updated
    var updatedServiceRequest = await dbContext.ServiceRequestModel.FindAsync(1);
    Assert.NotNull(updatedServiceRequest); // Ensure the request exists
    Assert.Equal("Completed", updatedServiceRequest.Status); // Ensure status was updated
}
```

Reports:

```
public class ReportsModelsControllerTests
{
    //Method to create a fresh controller and context for each test. Fixes the issue that running the test individually would work but not the entire file.
    7 references | 7/7 passing
    private ReportsModelsController GetControllerWithFreshContext(string dbName)
    {
        var options = new DbContextOptionsBuilder<MunicipalityManagementSystemV2Context>()
            .UseInMemoryDatabase(databaseName: dbName)
            .Options;

        var context = new MunicipalityManagementSystemV2Context(options);

        // New citizen. Whatup
        context.CitizenManagementModel.Add(new CitizenManagementModel
        {
            CitizenID = 1,
            Fullname = "John Dean",
            Address = "12 Main St",
            PhoneNumber = "08338902405",
            Email = "JohnDean@gmail.com",
            DateOfBirth = new DateTime(1990, 1, 1),
            RegistrationDate = DateTime.Now
        });

        // Creates new report
        context.ReportsModel.Add(new ReportsModel
        {
            ReportID = 2,
            CitizenID = 1,
            ReportType = "Incident",
            Details = "Details about the incident.",
            SubmissionDate = DateTime.Now,
            Status = "Under Review"
        });

        context.SaveChanges();
        return new ReportsModelsController(context);
    }
```

```
    //Checks if returns view of list of reports.
    [Fact]
    1 reference | 1/1 passing
    public async Task Index_ReturnsAViewResult_WithAListOfReports()
    {
        var controller = GetControllerWithFreshContext(nameof(Index_ReturnsAViewResult_WithAListOfReports));

        var result = await controller.Index();

        var viewResult = Assert.IsType<ViewResult>(result);
        var model = Assert.IsAssignableFrom<IEnumerable<ReportsModel>>(viewResult.ViewData.Model);
        Assert.NotEmpty(model);
    }

    //Checks if report with valid id returns valid details for it.
    [Fact]
    1 reference | 1/1 passing
    public async Task Details_ValidId_ReturnsAViewResult_WithAReport()
    {
        var controller = GetControllerWithFreshContext(nameof(Details_ValidId_ReturnsAViewResult_WithAReport));

        var result = await controller.Details(2);

        var viewResult = Assert.IsType<ViewResult>(result);
        var model = Assert.IsAssignableFrom<ReportsModel>(viewResult.ViewData.Model);
        Assert.Equal(2, model.ReportID);
    }

    [Fact]
    1 reference | 1/1 passing
    public async Task Create_ValidModel_ReturnsRedirectToActionResult()
    {
        var controller = GetControllerWithFreshContext(nameof(Create_ValidModel_ReturnsRedirectToActionResult));

        var report = new ReportsModel
        {
            CitizenID = 1,
            ReportType = "Incident",
            Details = "Details about the incident.",
            SubmissionDate = DateTime.Now,
            Status = "Under Review"
        };

        var result = await controller.Create(report);

        var redirectResult = Assert.IsType<RedirectToActionResult>(result);
        Assert.Equal("Index", redirectResult.ActionName);
    }
```

```
[Fact]
| 1 reference | 1/1 passing
public async Task Create_InvalidCitizenID_ReturnsViewWithModelError()
{
    var controller = GetControllerWithFreshContext(nameof(Create_InvalidCitizenID_ReturnsViewWithModelError));

    var report = new ReportsModel
    {
        CitizenID = 9999,
        ReportType = "Incident",
        Details = "Details about the incident.",
        SubmissionDate = DateTime.Now,
        Status = "Under Review"
    };

    var result = await controller.Create(report);

    var viewResult = Assert.IsType<ViewResult>(result);
    Assert.True(viewResult.ViewData.ModelState.ContainsKey("CitizenID"));
}

[Fact]
| 1 reference | 1/1 passing
public async Task Delete_ValidId_ReturnsAViewResult_WithAReport()
{
    var controller = GetControllerWithFreshContext(nameof(Delete_ValidId_ReturnsAViewResult_WithAReport));

    var result = await controller.Delete(2);

    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsAssignableFrom<ReportsModel>(viewResult.ViewData.Model);
    Assert.Equal(2, model.ReportID);
}

[Fact]
| 1 reference | 1/1 passing
public async Task DeleteConfirmed_ValidId_ReturnsRedirectToActionResult()
{
    var controller = GetControllerWithFreshContext(nameof(DeleteConfirmed_ValidId_ReturnsRedirectToActionResult));

    var result = await controller.DeleteConfirmed(2);

    var redirectResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Index", redirectResult.ActionName);
}
```

```
[Fact]
| 1 reference | 1/1 passing
public async Task ReviewReport_ValidId_ReturnsAViewResult_WithAReport()
{
    var controller = GetControllerWithFreshContext(nameof(ReviewReport_ValidId_ReturnsAViewResult_WithAReport));

    var result = await controller.ReviewReport(2);

    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsAssignableFrom<ReportsModel>(viewResult.ViewData.Model);
    Assert.Equal(2, model.ReportID);
}
```

Phase 5:

Power Point in separate file called ‘Phase5_Presentation’

GITHUB LINK: https://github.com/JaydonEllick/ADP631-FA1-Municipality-System