# Code Listing

This document is intended to share the current code which has been used for the submitted iteration of the prototype. Most of the files created for the prototype will be listed here, however, their respective functionalities will be shown in the development log in addition to any observations to be recorded for feedback purposes.

There are 3 types of files in this code listing. First is a razor view / page, which is a HTML page displayed to the user which can generate content based on C# conditions and queries. This will be indicated with the (V) suffix. Secondly is a model or C# class, which is used to define how data is represented and stored by a database, for example: constraints, data types, and nullable properties. This will be indicated with the (M) suffix. Additionally, the controller is used to manipulate and connect models and their respective data to views for display / interaction for the user. Finally, JavaScript scripts are used to modify the look and content of the page dynamically, and will be indicated with the (J) suffix.

Contents

[Code Listing 1](#_Toc158822180)

[Shared 1](#_Toc158822181)

[Program.cs 1](#_Toc158822182)

[(V) \_Layout.cshtml 2](#_Toc158822183)

[(V) \_LoginPartial.cshtml 4](#_Toc158822184)

[(M) ApplicationDbContext.cs 5](#_Toc158822185)

[(M) ApplicationUser.cs 5](#_Toc158822186)

[(M) ErrorViewModel.cs 6](#_Toc158822187)

[(V) Register.cshtml 6](#_Toc158822188)

[(V) Login.cshtml 7](#_Toc158822189)

[(J) Site.js 8](#_Toc158822190)

[Advice 9](#_Toc158822191)

[(C) AdviceController.cs 9](#_Toc158822192)

[(M) Advice.cs 13](#_Toc158822193)

[(V) Create.cshtml 14](#_Toc158822194)

[(V) Delete.cshtml 15](#_Toc158822195)

[(V) Details.cshtml 16](#_Toc158822196)

[(V) Edit.cshtml 17](#_Toc158822197)

[(V) Index.cshtml 18](#_Toc158822198)

[(V) MyAdvice.cshtml 20](#_Toc158822199)

[Dashboard 21](#_Toc158822200)

[(C) DashboardController.cs 21](#_Toc158822201)

[(M) Save.cs 23](#_Toc158822202)

[(V) AirQuality.cshtml 23](#_Toc158822203)

[(V) Forecast.cshtml 24](#_Toc158822204)

[(J) Airquality.js 24](#_Toc158822205)

[(J) Forecast.js 26](#_Toc158822206)

[Health 27](#_Toc158822207)

[(C) HealthController.cs 27](#_Toc158822208)

[(M) Health.cs 30](#_Toc158822209)

[(V) Create.cshtml 31](#_Toc158822210)

[(V) Delete.cshtml 32](#_Toc158822211)

[(V) Details.cshtml 33](#_Toc158822212)

[(V) Edit.cshtml 34](#_Toc158822213)

[(V) Index.cshtml 34](#_Toc158822214)

[(V) MyDiary.cshtml 35](#_Toc158822215)

[Home 36](#_Toc158822216)

[(C) HomeController.cs 36](#_Toc158822217)

[(V) Index.cshtml 37](#_Toc158822218)

[(V) Privacy.cshtml 39](#_Toc158822219)

# Shared

This section consists of views and models which are not exclusive to any role or are accessed before a user is assigned a role. Most of these sections are fundamental for the functionality of the solution and may serve purposes such as configuration or initialisation of important scripts or content.

## Program.cs

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using HealthAdviceGroup.Areas.Identity.Data;

using HealthAdviceGroup.Data;

namespace GibJohn

{

public class Program

{

public static async Task Main(string[] args)

{

// Initialises the database and configures application

var builder = WebApplication.CreateBuilder(args);

var connectionString = builder.Configuration.GetConnectionString("ApplicationDbContextConnection") ?? throw new InvalidOperationException("Connection string 'ApplicationDbContextConnection' not found.");

// Adds the DbContext to the service container with SQL Server as the database provider

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(connectionString));

// Configures default identity options and adds identity services to the service container

builder.Services.AddDefaultIdentity<ApplicationUser>(options => options.SignIn.RequireConfirmedAccount = true)

.AddRoles<IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>();

// 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");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

// Adds HTTPS redirection and static files middleware to the pipeline

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication(); // Enables authentication middleware

app.UseAuthorization(); // Enables authorization middleware

// Maps controller routes and Razor Pages

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

app.MapRazorPages();

// Check if the role already exists, if not create the roles

using (var scope = app.Services.CreateScope())

{

var roleManager = scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();

var roles = new[] { "User", "Management" };

foreach (var role in roles)

{

if (!await roleManager.RoleExistsAsync(role))

{

await roleManager.CreateAsync(new IdentityRole(role));

}

}

}

DotNetEnv.Env.Load(); // Load environment variables from a .env file

app.Run(); // Runs the application

}

}

}

## (V) \_Layout.cshtml

<!DOCTYPE html>

<html lang="en">

<head>

<!-- Meta tags for character set and viewport -->

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"] - HealthAdviceGroup</title>

<!-- Link to local and third-party assets -->

<**link** rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

<**link** rel="stylesheet" **href**="~/css/site.css" **asp-append-version**="true" />

<**link** rel="stylesheet" **href**="~/HealthAdviceGroup.styles.css" **asp-append-version**="true" />

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.1/css/all.min.css" integrity="sha512-DTOQO9RWCH3ppGqcWaEA1BIZOC6xxalwEsw9c2QQeAIftl+Vegovlnee1c9QX4TctnWMn13TZye+giMm8e2LwA==" crossorigin="anonymous" referrerpolicy="no-referrer" />

</head>

<body>

<header>

<!-- Navigation bar -->

<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light border-bottom box-shadow mb-3" style="height: 5rem; font-size: 1.25rem; padding: 1.25rem;">

<div class="container-fluid">

<!-- Brand name with icon -->

<**a** class="navbar-brand" **asp-area**="" **asp-controller**="Home" **asp-action**="Index" style="padding-right: 1rem;"> Health Advice Group <i class="fa-solid fa-cloud-sun-rain"></i></**a**>

<!-- Toggle button for collapsed navigation bar -->

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"

aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<!-- Navigation options -->

<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">

<ul class="navbar-nav flex-grow-1">

<li class="nav-item">

<**a** class="nav-link" **asp-area**="" **asp-controller**="Home" **asp-action**="Index">Home</**a**>

</li>

<li class="nav-item">

<**a** class="nav-link" **asp-area**="" **asp-controller**="Home" **asp-action**="Privacy">Privacy</**a**>

</li>

<!-- Button to toggle theme -->

<li class="nav-item">

<button class="nav-link btn" onclick="setTheme();">Toggle Theme</button>

</li>

</ul>

</div>

<!-- View which provides links to login options -->

<**partial** **name**="\_LoginPartial" />

</div>

</nav>

</header>

<!-- Main content container -->

<div class="container">

<main role="main" class="pb-3">

<!-- This is where the main page will be rendered -->

@RenderBody()

</main>

</div>

<!-- Footer section -->

<footer class="border-top footer text-muted">

<div class="container">

&copy; 2024 - HealthAdviceGroup - <**a** **asp-area**="" **asp-controller**="Home" **asp-action**="Privacy">Privacy</**a**>

</div>

</footer>

<!-- JavaScript files -->

<**script** src="~/lib/jquery/dist/jquery.min.js"></**script**>

<**script** src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></**script**>

<**script** **src**="~/js/site.js" **asp-append-version**="true"></**script**>

<script>applyTheme();</script>

<!-- Render additional section scripts -->

@await RenderSectionAsync("Scripts", required: false)

</body>

</html>

## (V) \_LoginPartial.cshtml

﻿@using Microsoft.AspNetCore.Identity

@using HealthAdviceGroup.Areas.Identity.Data

@inject SignInManager<ApplicationUser> SignInManager

@inject UserManager<ApplicationUser> UserManager

@{

// Retrieve the current user

var user = await UserManager.GetUserAsync(Context.User);

}

<ul class="navbar-nav">

<!-- Check if user is signed in -->

@if (SignInManager.IsSignedIn(User))

{

<!-- Display user's name and link to manage account -->

<li class="nav-item">

<**a** id="manage" class="nav-link" **asp-area**="Identity" **asp-page**="/Account/Manage/Index" title="Manage">Hello @user.FirstName @user.LastName </**a**>

</li>

<!-- Form for logout -->

<li class="nav-item">

<**form** id="logoutForm" class="form-inline" **asp-area**="Identity" **asp-page**="/Account/Logout" **asp-route-returnUrl**="@Url.Action("Index", "Home", new { area = "" })">

<button id="logout" type="submit" class="nav-link btn btn-link">Logout</button>

</**form**>

</li>

}

else

{

<!-- Link to register page -->

<li class="nav-item">

<**a** class="nav-link" id="register" **asp-area**="Identity" **asp-page**="/Account/Register">Register</**a**>

</li>

<!-- Link to login page -->

<li class="nav-item">

<**a** class="nav-link" id="login" **asp-area**="Identity" **asp-page**="/Account/Login">Login</**a**>

</li>

}

</ul>

## (M) ApplicationDbContext.cs

using HealthAdviceGroup.Areas.Identity.Data;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using HealthAdviceGroup.Models;

namespace HealthAdviceGroup.Data

{

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

// Apply custom configuration for ApplicationUser entity

builder.ApplyConfiguration(new ApplicationUserEntityConfiguration());

}

public class ApplicationUserEntityConfiguration : IEntityTypeConfiguration<ApplicationUser>

{

// Configure method to set constraints for ApplicationUser properties

public void Configure(EntityTypeBuilder<ApplicationUser> builder)

{

// Set maximum lengths for FirstName and LastName properties

builder.Property(u => u.FirstName).HasMaxLength(20);

builder.Property(u => u.LastName).HasMaxLength(20);

}

}

// DbSet for the Health, Advice, and Save entity in the database

public DbSet<HealthAdviceGroup.Models.Health> Health { get; set; } = default!;

public DbSet<HealthAdviceGroup.Models.Advice> Advice { get; set; } = default!;

public DbSet<HealthAdviceGroup.Models.Save> Save { get; set; } = default!;

}

}

## (M) ApplicationUser.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Identity;

namespace HealthAdviceGroup.Areas.Identity.Data;

// Add profile data for application users by adding properties to the ApplicationUser class

public class ApplicationUser : IdentityUser

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

## (M) ErrorViewModel.cs

namespace HealthAdviceGroup.Models

{

// Model representing error details for display in views

public class ErrorViewModel

{

// Property to store the request identifier associated with the error

public string? RequestId { get; set; }

// Property that returns true if the request identifier is not null or empty

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

}

}

## (V) Register.cshtml

@page

@model RegisterModel

@{

ViewData["Title"] = "Register";

}

<div class="row">

<div class="col-md-4">

</div>

<div class="col-md-4">

<h1>@ViewData["Title"]</h1>

<**form** id="registerForm" **asp-route-returnUrl**="@Model.ReturnUrl" method="post">

<h2>Create a new account.</h2>

<hr />

<!-- Display validation errors -->

<**div** **asp-validation-summary**="ModelOnly" class="text-danger" role="alert"></**div**>

<!-- Label, validation, and input field for FirstName-->

<div class="form-floating mb-3">

<**input** **asp-for**="Input.FirstName" class="form-control" autocomplete="given-name" aria-required="true" />

<**label** **asp-for**="Input.FirstName">First Name</**label**>

<**span** **asp-validation-for**="Input.FirstName" class="text-danger"></**span**>

</div>

<!-- Label, validation, and input field for LastName-->

<div class="form-floating mb-3">

<**input** **asp-for**="Input.LastName" class="form-control" autocomplete="given-name" aria-required="true" />

<**label** **asp-for**="Input.LastName">Last Name</**label**>

<**span** **asp-validation-for**="Input.LastName" class="text-danger"></**span**>

</div>

<!-- Label, validation, and input field for Email-->

<div class="form-floating mb-3">

<**input** **asp-for**="Input.Email" class="form-control" autocomplete="username" aria-required="true" />

<**label** **asp-for**="Input.Email">Email</**label**>

<**span** **asp-validation-for**="Input.Email" class="text-danger"></**span**>

</div>

<!-- Label, validation, and input field for Password-->

<div class="form-floating mb-3">

<**input** **asp-for**="Input.Password" class="form-control" autocomplete="new-password" aria-required="true" />

<**label** **asp-for**="Input.Password">Password</**label**>

<**span** **asp-validation-for**="Input.Password" class="text-danger"></**span**>

</div>

<!-- Label, validation, and input field for ConfirmPassword-->

<div class="form-floating mb-3">

<**input** **asp-for**="Input.ConfirmPassword" class="form-control" autocomplete="new-password" aria-required="true" />

<**label** **asp-for**="Input.ConfirmPassword">Confirm Password</**label**>

<**span** **asp-validation-for**="Input.ConfirmPassword" class="text-danger"></**span**>

</div>

<div>

<!-- User role input, this is what the average user will have -->

<label>

<input type="radio" name="role" value="user"/> Add User Role

</label>

</div>

<div>

<!-- Management role input for testing purposes -->

<label>

<input type="radio" name="role" value="management" /> Add Management Role

</label>

</div>

<button id="registerSubmit" type="submit" class="w-100 btn btn-lg btn-primary">Register</button>

</**form**>

</div>

<div class="col-md-4">

</div>

</div>

@section Scripts {

<**partial** **name**="\_ValidationScriptsPartial" />

}

## (V) Login.cshtml

@page

@model LoginModel

@{

ViewData["Title"] = "Log in";

}

<div class="container">

<div class="row">

<div class="col-md-4">

</div>

<div class="col-md-4">

<h1>@ViewData["Title"]</h1>

<section>

<**form** id="account" method="post">

<h2>Use an account to log in.</h2>

<hr />

<!-- Display login errors -->

<**div** **asp-validation-summary**="ModelOnly" class="text-danger" role="alert"></**div**>

<div class="form-floating mb-3">

<!-- Label, validation, and input for email -->

<**input** **asp-for**="Input.Email" class="form-control" autocomplete="username" aria-required="true" placeholder="name@example.com" />

<**label** **asp-for**="Input.Email" class="form-label">Email</**label**>

<**span** **asp-validation-for**="Input.Email" class="text-danger"></**span**>

</div>

<!-- Label, validation, and input for password -->

<div class="form-floating mb-3">

<**input** **asp-for**="Input.Password" class="form-control" autocomplete="current-password" aria-required="true" placeholder="password" />

<**label** **asp-for**="Input.Password" class="form-label">Password</**label**>

<**span** **asp-validation-for**="Input.Password" class="text-danger"></**span**>

</div>

<!-- Checkbox for remember me -->

<div class="checkbox mb-3">

<**label** **asp-for**="Input.RememberMe" class="form-label">

<**input** class="form-check-input" **asp-for**="Input.RememberMe" />

@Html.DisplayNameFor(m => m.Input.RememberMe)

</**label**>

</div>

<!-- Button to submit login form -->

<div>

<button id="login-submit" type="submit" class="w-100 btn btn-lg btn-primary">Log in</button>

</div>

<div>

<p>

<**a** id="forgot-password" **asp-page**="./ForgotPassword">Forgot your password?</**a**>

</p>

</div>

</**form**>

</section>

</div>

</div>

</div>

@section Scripts {

<**partial** **name**="\_ValidationScriptsPartial" />

}

## (J) Site.js

// Global variables to store latitude and longitude

let latitude;

let longitude;

// Function to convert temperature from Kelvin to Celsius

function toCelsius(kelvin) {

kelvin = parseFloat(kelvin);

return Math.round(kelvin - 273.15);

}

// Function to convert timestamp to time format (HH:MM)

function toTime(timestamp) {

let date = new Date(timestamp \* 1000); // Convert timestamp to Date object

let hours = date.getHours().toString().padStart(2, "0"); // Get hours with added zero if theres only one digit

let minutes = date.getMinutes().toString().padStart(2, "0"); // Get minutes with added zero if theres only one digit

return `${hours}:${minutes}`;

}

// Function to toggle between light and dark themes

function setTheme() {

var currentTheme = localStorage.getItem('theme');

if (currentTheme === 'light') {

localStorage.setItem('theme', 'dark');

} else {

localStorage.setItem('theme', 'light');

}

applyTheme();

}

// Function to apply the theme stored in local storage

function applyTheme() {

var theme = localStorage.getItem('theme');

if (theme === 'light') {

document.documentElement.setAttribute('data-bs-theme', 'light');

} else if (theme === 'dark') {

document.documentElement.setAttribute('data-bs-theme', 'dark');

}

}

// Function to update text content of an element with a given selector

function updateTextContent(selector, value) {

const element = document.querySelector(selector);

if (element) {

element.textContent = value;

}

}

# Advice

The advice section is designed for management / trustees to be able to create new and update existing advice for users to see and save. Additionally, should we have had time, advice relevant to the forecast / current conditions would also be displayed. However, due to time constraints, this has not yet been implemented.

## (C) AdviceController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using HealthAdviceGroup.Data;

using HealthAdviceGroup.Models;

using System.Security.Claims;

namespace HealthAdviceGroup.Controllers

{

public class AdviceController : Controller

{

private readonly IWebHostEnvironment \_webHostEnvironment;

private readonly ApplicationDbContext \_context;

public AdviceController(IWebHostEnvironment webHostEnvironment, ApplicationDbContext context)

{

\_webHostEnvironment = webHostEnvironment;

\_context = context;

}

// GET: Advice/Index

public async Task<IActionResult> Index()

{

// Display all advice entries

return View(await \_context.Advice.ToListAsync());

}

// GET: Advice/MyAdvice

public async Task<IActionResult> MyAdvice()

{

// Retrieve advice saved by the logged-in user

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

var adviceIds = await \_context.Save

.Where(s => s.UserId == userId)

.Select(s => s.AdviceId)

.ToListAsync();

var advice = await \_context.Advice

.Where(a => adviceIds.Contains(a.Id))

.ToListAsync();

return View(advice);

}

// GET: Advice/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

// Retrieve details of a specific advice entry

var advice = await \_context.Advice

.FirstOrDefaultAsync(m => m.Id == id);

if (advice == null)

{

return NotFound();

}

return View(advice);

}

// GET: Advice/Create

public IActionResult Create()

{

// Display the form for creating new advice

return View();

}

// POST: Advice/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(Advice advice)

{

if (ModelState.IsValid)

{

// If the form is valid, handle image upload and save the advice

if (advice.ImageFile != null)

{

string uploadsFolder = Path.Combine(\_webHostEnvironment.WebRootPath, "assets", "uploaded");

advice.ImagePath = Guid.NewGuid().ToString() + "\_" + advice.ImageFile.FileName;

string filePath = Path.Combine(uploadsFolder, advice.ImagePath);

advice.ImageFile.CopyTo(new FileStream(filePath, FileMode.Create));

}

\_context.Add(advice);

await \_context.SaveChangesAsync();

// Redirect to the dashboard after successful advice creation

return RedirectToAction("Index", "Home");

}

return View(advice);

}

// GET: Advice/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

// Display the form for editing a specific advice entry

var advice = await \_context.Advice.FindAsync(id);

if (advice == null)

{

return NotFound();

}

return View(advice);

}

// POST: Advice/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("ImagePath,Id,Title,Description")] Advice advice)

{

if (id != advice.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

// Update and save the edited advice entry

\_context.Update(advice);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!AdviceExists(advice.Id))

{

return NotFound();

}

else

{

throw;

}

}

// Redirect to the advice index after successful edit

return RedirectToAction(nameof(Index));

}

return View(advice);

}

// GET: Advice/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

// Display the confirmation page for deleting a specific advice entry

var advice = await \_context.Advice

.FirstOrDefaultAsync(m => m.Id == id);

if (advice == null)

{

return NotFound();

}

return View(advice);

}

// POST: Advice/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

// Delete the specified advice entry

var advice = await \_context.Advice.FindAsync(id);

if (advice != null)

{

\_context.Advice.Remove(advice);

}

await \_context.SaveChangesAsync();

// Redirect to the advice index after successful deletion

return RedirectToAction(nameof(Index));

}

// Helper method to check if an advice entry exists

private bool AdviceExists(int id)

{

return \_context.Advice.Any(e => e.Id == id);

}

// POST: Advice/SaveAdvice

public async Task<IActionResult> SaveAdvice(IFormCollection collection)

{

// Get user's ID so it is saved specifically to their account

// Get the value from the hidden input named "id"

int adviceId = int.Parse(collection["id"].ToString());

string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

var existingSave = await \_context.Save

.FirstOrDefaultAsync(s => s.UserId == userId && s.AdviceId == adviceId);

if (existingSave != null)

{

// If already saved, redirect to MyAdvice

return RedirectToAction("Index", "Advice");

}

else

{

// Create a new advice entry and save it to the user's saved advice list

var save = new Save

{

AdviceId = adviceId,

UserId = userId,

Date = DateTime.Now.ToString("dd/MM/yyyy"),

};

// Add the new entry to "Save" table

\_context.Save.Add(save);

await \_context.SaveChangesAsync();

Console.WriteLine("Successful save");

// Redirect to MyAdvice after successful save

return RedirectToAction("Index", "Advice");

}

}

// POST: Advice/DeleteSave

[HttpPost, ActionName("DeleteSave")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteSave(IFormCollection collection)

{

// Remove advice from the user's saved advice list

var id = int.Parse(collection["id"].ToString());

var save = await \_context.Save.FindAsync(id);

if (save != null)

{

\_context.Save.Remove(save);

}

await \_context.SaveChangesAsync();

// Redirect to MyAdvice after successful removal

return RedirectToAction(nameof(MyAdvice));

}

}

}

## (M) Advice.cs

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Diagnostics.CodeAnalysis;

using Microsoft.AspNetCore.Http; // Import necessary namespace for IFormFile

namespace HealthAdviceGroup.Models

{

// Model used to define and add constraints / conditions for data to be stored

public class Advice

{

// Primary key for the advice entry

public int Id { get; set; }

// Property for uploading an image associated with the advice (not saved to the database)

[NotMapped]

[DisplayName("Upload Image")]

public IFormFile? ImageFile { get; set; }

// Path to the stored image in the server

public string? ImagePath { get; set; }

// Nullable property representing the temperature associated with the advice

[AllowNull]

[Range(-90, 60, ErrorMessage = "Temperature must be between -90 and 60 degrees Celsius")]

[DisplayName("Temperature (°C)")]

public double? Temperature { get; set; }

// Required property representing the title of the advice which will be shown on the all advice page

[StringLength(100, ErrorMessage = "Title must be between 5 and 100 characters", MinimumLength = 5)]

[Required]

public string Title { get; set; }

// Property representing the description of the advice which will be shown on the advice details page

public string Description { get; set; }

}

}

## (V) Create.cshtml

@model HealthAdviceGroup.Models.Advice

@{

ViewData["Title"] = "Create";

}

<h1>Create</h1>

<h4>Advice</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="Create" enctype="multipart/form-data">

<!-- Display any errors or rejected inputs from the user-->

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

<!-- Display name, input, and validation for the image file -->

<div class="form-group">

<**label** **asp-for**="ImageFile" class="control-label"></**label**>

<**input** **asp-for**="ImageFile" **type**="file" class="form-control" />

<**span** **asp-validation-for**="ImageFile" class="text-danger"></**span**>

</div>

<!-- Display name, input, and validation for the title -->

<div class="form-group">

<**label** **asp-for**="Title" class="control-label"></**label**>

<**input** **asp-for**="Title" class="form-control" />

<**span** **asp-validation-for**="Title" class="text-danger"></**span**>

</div>

<!-- Display name, input, and validation for the temperature -->

<div class="form-group">

<**label** **asp-for**="Temperature" class="control-label"></**label**>

<**input** **asp-for**="Temperature" class="form-control" />

<**span** **asp-validation-for**="Temperature" class="text-danger"></**span**>

</div>

<!-- Display name, text area, and validation for the description -->

<div class="form-group">

<**label** **asp-for**="Description" class="control-label"></**label**>

<**textarea** **asp-for**="Description" class="form-control" style="width: 20rem; height: 10rem;"></**textarea**>

<**span** **asp-validation-for**="Description" class="text-danger"></**span**>

</div>

<!-- Submit button to create the advice entry -->

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

<!-- Link to navigate back to the advice list -->

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

<!-- Include partial scripts for client-side validation -->

@section Scripts {

@{

await Html.RenderPartialAsync("\_ValidationScriptsPartial");

}

}

## (V) Delete.cshtml

@model HealthAdviceGroup.Models.Advice

@{

ViewData["Title"] = "Delete";

}

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>Advice</h4>

<hr />

<dl class="row">

<!-- Display the image path property -->

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.ImagePath)

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.ImagePath)

</dd>

<!-- Displays the title property -->

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.Title)

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Title)

</dd>

<!-- Display the description property -->

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.Description)

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Description)

</dd>

</dl>

<!-- Form for confirming the deletion -->

<**form** **asp-action**="Delete">

<!-- Hidden input field to store the Id of the advice entry to be deleted -->

<**input** **type**="hidden" **asp-for**="Id" />

<!-- Submit button to confirm the deletion -->

<input type="submit" value="Delete" class="btn btn-danger" /> |

<**a** **asp-action**="Index">Back to List</**a**>

</**form**>

</div>

## (V) Details.cshtml

@model HealthAdviceGroup.Models.Advice

@inject Microsoft.AspNetCore.Hosting.IWebHostEnvironment webHostEnvironment;

@{

ViewData["Title"] = "Details";

}

<h1>Details</h1>

<div>

<h4>Advice</h4>

<hr />

<dl class="row">

<!-- Display the image path property's name -->

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.ImagePath)

</dt>

<dd class="col-sm-10">

@{

// Generate the image path, using a placeholder if the file doesn't exist

string imagePath = string.IsNullOrEmpty(Model.ImagePath) || System.IO.File.Exists(webHostEnvironment.WebRootPath + "/assets/uploaded" + Model.ImagePath) ? Url.Content("~/assets/static/placeholder.png") : Url.Content("~/assets/uploaded/" + Model.ImagePath);

// Display the image

<img style="width: 30vh; height: 20vh;" src="@imagePath" />

}

</dd>

<!-- Display the title property's name -->

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.Title)

</dt>

<!-- Display the title property -->

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Title)

</dd>

<!-- Display the description property's name -->

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.Description)

</dt>

<!-- Display the description property -->

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Description)

</dd>

</dl>

</div>

<!-- Link to navigate back to the advice list -->

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

## (V) Edit.cshtml

@model HealthAdviceGroup.Models.Advice

@{

ViewData["Title"] = "Edit";

}

<h1>Edit</h1>

<h4>Advice</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="Edit" enctype="multipart/form-data">

<!-- Display any errors or rejected inputs from the user-->

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

<!-- Display name, input, and validation for the image file -->

<div class="form-group">

<**label** **asp-for**="ImageFile" class="control-label"></**label**>

<**input** **asp-for**="ImageFile" class="form-control" **type**="file" />

<**span** **asp-validation-for**="ImagePath" class="text-danger"></**span**>

</div>

<!-- Hidden input field to store the Id of the advice entry being edited -->

<**input** **type**="hidden" **asp-for**="Id" />

<!-- Display name, input, and validation for the title -->

<div class="form-group">

<**label** **asp-for**="Title" class="control-label"></**label**>

<**input** **asp-for**="Title" class="form-control" />

<**span** **asp-validation-for**="Title" class="text-danger"></**span**>

</div>

<!-- Display name, input, and validation for the temperature -->

<div class="form-group">

<**label** **asp-for**="Temperature" class="control-label"></**label**>

<**input** **asp-for**="Temperature" class="form-control" />

<**span** **asp-validation-for**="Temperature" class="text-danger"></**span**>

</div>

<!-- Display name, text area, and validation for the description -->

<div class="form-group">

<**label** **asp-for**="Description" class="control-label"></**label**>

<**input** **asp-for**="Description" class="form-control" />

<**span** **asp-validation-for**="Description" class="text-danger"></**span**>

</div>

<!-- Submit button to save the edited advice entry -->

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

<!-- Link to navigate back to the advice list -->

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

<!-- Include partial scripts for client-side validation -->

@section Scripts {

@{

await Html.RenderPartialAsync("\_ValidationScriptsPartial");

}

}

## (V) Index.cshtml

@using HealthAdviceGroup.Data

@using Microsoft.EntityFrameworkCore

@using System.Security.Claims

@model IEnumerable<HealthAdviceGroup.Models.Advice>

@inject Microsoft.AspNetCore.Hosting.IWebHostEnvironment webHostEnvironment;

@inject ApplicationDbContext applicationDbContext;

@{

ViewData["Title"] = "Index";

}

<h1>Index</h1>

<**a** **asp-action**="MyAdvice" **asp-controller**="Advice">Saved Advice</**a**>

<div class="container-fluid courses">

<div class="row">

@foreach (var item in Model)

// Generate new card for every database entry

{

// Determine the image path, using a placeholder if the file doesn't exist

string imagePath = string.IsNullOrEmpty(item.ImagePath) || System.IO.File.Exists(webHostEnvironment.WebRootPath + "/assets/uploaded" + item.ImagePath) ? Url.Content("~/assets/static/placeholder.png") : Url.Content("~/assets/uploaded/" + item.ImagePath);

// Extract adviceId and userId for saving or viewing advice

int adviceId = item.Id;

string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

// Check if the advice is already saved by the user

var existingSave = await applicationDbContext.Save

.FirstOrDefaultAsync(s => s.UserId == userId && s.AdviceId == adviceId);

<div class="col-lg-3 col-md-5 col-sm-12 spacing">

<div class="card" style="height: 15rem;">

<!-- Display the advice image and title -->

<img class="card-img-top" src=@imagePath style="width: 100%; height: 10rem;" />

<h3 class="card-title text-center">@item.Title</h3>

<!-- Actions based on user role -->

@if (User.IsInRole("Management"))

{

<!-- Display links for Management roles -->

<p class="text-center">

<**a** **asp-action**="Details" **asp-route-id**="@item.Id">Details</**a**> |

<**a** **asp-action**="Edit" **asp-route-id**="@item.Id" class="text-success">Edit</**a**> |

<**a** **asp-action**="Delete" **asp-route-id**="@item.Id" class="text-danger">Delete</**a**>

</p>

}

else

{

<!-- Display options for regular users -->

<div class="d-flex flex-row justify-content-evenly">

<**a** **asp-action**="Details" **asp-route-id**="@item.Id">Details</**a**>

@if (existingSave == null)

{

<!-- Form to save advice if it's not already saved -->

<**form** **asp-action**="SaveAdvice" **asp-controller**="Advice" method="post">

<input type="hidden" name="id" value="@item.Id" />

<button type="submit">Save</button>

</**form**>

}

else

{

<!-- Display message if the advice is already saved -->

<p>Already Saved!</p>

}

</div>

}

</div>

</div>

}

</div>

</div>

## (V) MyAdvice.cshtml

@using HealthAdviceGroup.Data

@using Microsoft.EntityFrameworkCore

@using System.Security.Claims

@model IEnumerable<HealthAdviceGroup.Models.Advice>

@inject Microsoft.AspNetCore.Hosting.IWebHostEnvironment webHostEnvironment;

@inject ApplicationDbContext applicationDbContext;

@{

ViewData["Title"] = "My Advice";

}

<h1>Saved Advice</h1>

<**a** **asp-action**="Index" **asp-controller**="Advice">All Advice</**a**>

<div class="container-fluid courses">

<div class="row">

@foreach (var item in Model)

{

<!-- Determine the path to the uploaded image, using a spaceholder if it is empty-->

string imagePath = string.IsNullOrEmpty(item.ImagePath) || System.IO.File.Exists(webHostEnvironment.WebRootPath + "/assets/uploaded" + item.ImagePath) ? Url.Content("~/assets/static/placeholder.png") : Url.Content("~/assets/uploaded/" + item.ImagePath);

<!-- Extract adviceId and userId for saving or viewing advice -->

int adviceId = item.Id;

string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

<div class="col-lg-3 col-md-5 col-sm-12 spacing">

<div class="card" style="height: 15rem;">

<!-- Display the advice image and title -->

<img class="card-img-top" src=@imagePath style="width: 100%; height: 10rem;" />

<h3 class="card-title text-center">@item.Title</h3>

<div class="d-flex flex-row justify-content-evenly">

<!-- Details option to view more -->

<**a** **asp-action**="Details" **asp-route-id**="@item.Id">Details</**a**>

<!-- Action to unsave advice -->

<**form** **asp-action**="DeleteSave" **asp-controller**="Advice" method="post">

<input type="hidden" name="id" value="@item.Id" />

<button type="submit" class="btn-danger">Unsave</button>

</**form**>

</div>

</div>

</div>

}

</div>

</div>

# Dashboard

The dashboard section is designed to display relevant information to the user based off their current location and the current time. This should allow users to make relevant health decisions. In this section, I managed to implement server-side API-calling by passing arguments and responses between the controller and scripts using ajax calls.

## (C) DashboardController.cs

using HealthAdviceGroup.Data;

using HealthAdviceGroup.Models;

using Microsoft.AspNetCore.Mvc;

using System.Diagnostics;

using System.Net.Http;

using System.Text.Json;

using System.Threading.Tasks;

namespace HealthAdviceGroup.Controllers

{

public class DashboardController : Controller

{

private readonly ILogger<DashboardController> \_logger;

private readonly ApplicationDbContext \_context;

public DashboardController(ILogger<DashboardController> logger, ApplicationDbContext context)

{

\_logger = logger;

\_context = context;

}

// POST: Dashboard/GetOWMData

[HttpPost]

public async Task<ActionResult> GetOWMData(double latitude, double longitude, string option)

{

// Using HttpClient to interact with the OpenWeatherMap API

using (var client = new HttpClient())

{

// Set the base address for the OpenWeatherMap API

client.BaseAddress = new Uri("https://api.openweathermap.org/data/2.5/");

var response = new HttpResponseMessage();

// Check the chosen option to determine the type of API call

if (option.ToLower() == "weather")

{

// Make a request to retrieve current weather data

response = await client.GetAsync($"weather?lat={latitude}&lon={longitude}&appid={Environment.GetEnvironmentVariable("API\_KEY")}");

if (response.IsSuccessStatusCode)

{

// Return the weather data if successful

return Ok(await response.Content.ReadAsStringAsync());

}

else

{

// Return an error message if the request is not successful

return Json("Error");

}

}

else if (option.ToLower() == "air\_pollution")

{

// Make a request to retrieve air pollution data

response = await client.GetAsync($"air\_pollution?lat={latitude}&lon={longitude}&appid={Environment.GetEnvironmentVariable("API\_KEY")}");

if (response.IsSuccessStatusCode)

{

// Return the air pollution data if successful

return Ok(await response.Content.ReadAsStringAsync());

}

else

{

// Return an error message if the request is not successful

return Json("Error");

}

}

else

{

// Return "Null" if the option is not recognized

return Json("Null");

}

}

}

// GET: Dashboard/Forecast

public IActionResult Forecast()

{

return View();

}

// GET: Dashboard/AirQuality

public IActionResult AirQuality()

{

return View();

}

// GET: Dashboard/Privacy

public IActionResult Privacy()

{

return View();

}

// GET: Dashboard/Error

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

// Return the Error view with the request identifier

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

## (M) Save.cs

using System.ComponentModel.DataAnnotations.Schema;

namespace HealthAdviceGroup.Models

{

// Model representing Advice entries saved by a User

public class Save

{

// Primary Id used to identify any advice saved by a user

public int Id { get; set; }

// The id of the advice entry that the user has saved

public int AdviceId{ get; set; }

// The id of the user that saved the advice

public string UserId{ get; set; }

// The date that the advice was saved

public string Date { get; set; }

}

}

## (V) AirQuality.cshtml

@{

ViewData["Title"] = "Air Quality";

}

<div class="container">

<div style="float: left;">

<h4 class="display-4">Pollutants (μg/m3)</h4>

<!-- Location of air quality data (not implemented yet) -->

<h3 class="location-text">Placeholder</h3>

<!-- Amount of pollutants in air -->

<h4 class="co-text">Loading</h4>

<h4 class="no2-text">Loading</h4>

<h4 class="o3-text">Loading</h4>

<h4 class="pm2\_5-text">Loading</h4>

<h4 class="pm10-text">Loading</h4>

<h4 class="so4-text">Loading</h4>

</div>

<div style="float: right;">

<h4 class="display-4">Air Quality Index</h4>

<!-- AQI for each polutant -->

<h3 class="aqi-text" >Loading</h3>

<h4 class="co-aqi">Loading</h4>

<h4 class="no2-aqi">Loading</h4>

<h4 class="o3-aqi">Loading</h4>

<h4 class="pm2\_5-aqi">Loading</h4>

<h4 class="pm10-aqi">Loading</h4>

<h4 class="so4-aqi">Loading</h4>

</div>

</div>

<!-- Placeholder chart for pollutant concentration / history -->

<div class="container">

<div style="chart">

<img src="https://sta.laits.utexas.edu/wp-content/uploads/files/charts.png" style="width: 75%;" />

</div>

</div>

<!-- Link to script which gets location and calculates pollutant AQIs -->

<**script** src="~/js/airquality.js"></**script**>

## (V) Forecast.cshtml

@{

ViewData["Title"] = "Forecast";

}

<div class="container">

<div class="left" style="float: left;">

<!-- Left side of the screen, displays information relevant to current weather -->

<h3 class="location-text">Loading</h3>

<img class="icon-img" id="weather-img" src="https://tse2.mm.bing.net/th/id/OIP.F00dCf4bXxX0J-qEEf4qIQHaD6?rs=1&pid=ImgDetMain"/>

<h3 class="d-inline-block float-md-right condition-text">Loading</h3>

<h4 class="temperature-text">Loading</h4>

<h4 class="lastupdated-text">Loading</h4>

</div>

<!-- Right side of the screen, displays information relevant to other conditions / entire day -->

<div class="right" style="float: right;">

<h3><a href="#">Search for another location</a></h3>

<h4 class="humidity-text">Loading</h4>

<h4 class="airpressure-text">Loading</h4>

<h4 class="sunrise-text">Loading</h4>

<h4 class="sunset-text">Loading</h4>

</div>

</div>

<!-- Link to script which calculates temperature and updates text -->

<**script** src="~/js/forecast.js"></**script**>

## (J) Airquality.js

// Array to store pollution conditions

let conditions = ["Good", "Fair", "Moderate", "Poor", "Very Poor"];

// Object to store pollutant ranges

let pollutantRanges = {

"SO2": {

1: [0, 20],

2: [20, 80],

3: [80, 250],

4: [250, 350],

5: [350, Infinity]

},

"NO2": {

1: [0, 40],

2: [40, 70],

3: [70, 150],

4: [150, 200],

5: [200, Infinity]

},

"PM10": {

1: [0, 20],

2: [20, 50],

3: [50, 100],

4: [100, 200],

5: [200, Infinity]

},

"PM2.5": {

1: [0, 10],

2: [10, 25],

3: [25, 50],

4: [50, 75],

5: [75, Infinity]

},

"O3": {

1: [0, 60],

2: [60, 100],

3: [100, 140],

4: [140, 180],

5: [180, Infinity]

},

"CO": {

1: [0, 4400],

2: [4400, 9400],

3: [9400, 12400],

4: [12400, 15400],

5: [15400, Infinity]

}

};

// Function to get current location and fetch air pollution data

navigator.geolocation.getCurrentPosition(function (position) {

// Extract latitude and longitude

latitude = position.coords.latitude;

longitude = position.coords.longitude;

// AJAX call to retrieve air pollution data

$.ajax({

type: "POST",

url: "http://localhost:5074/Dashboard/GetOWMData",

data: { latitude: latitude, longitude: longitude, option: "air\_pollution" },

// Success callback function to update pollution HUD

success: function (data) {

updatePollutionHUD(data);

},

// Error callback function to handle errors

error: function (error) {

console.log(error);

alert("You need to allow location services or enter a location to use this feature.")

}

});

});

// Function to calculate Air Quality Index (AQI) based on pollutant concentration

function calculateAQI(pollutant, concentration) {

for (let i = 1; i <= 5; i++) {

if (concentration >= pollutantRanges[pollutant][i][0] && concentration < pollutantRanges[pollutant][i][1]) {

return conditions[i - 1];

}

}

return "Error";

}

// Function to update pollution HUD elements with data

function updatePollutionHUD(data) {

const parsedData = JSON.parse(data);

const components = parsedData.list[0].components;

// Object to store pollutant elements and their corresponding values

const pollutantElements = {

".co-text": components.co,

".no2-text": components.no2,

".o3-text": components.o3,

".pm2\_5-text": components.pm2\_5,

".pm10-text": components.pm10,

".so4-text": components.so2,

".aqi-text": parsedData.list[0].main.aqi

};

// Update pollutant elements with values

for (const [selector, value] of Object.entries(pollutantElements)) {

updateTextContent(selector, value);

}

// Object to store AQI elements and their corresponding values

const aqiElements = {

".co-aqi": calculateAQI("CO", components.co),

".no2-aqi": calculateAQI("NO2", components.no2),

".o3-aqi": calculateAQI("O3", components.o3),

".pm2\_5-aqi": calculateAQI("PM2.5", components.pm2\_5),

".pm10-aqi": calculateAQI("PM10", components.pm10),

".so4-aqi": calculateAQI("SO2", components.so2)

};

// Update AQI elements with values

for (const [selector, value] of Object.entries(aqiElements)) {

updateTextContent(selector, value);

}

}

## (J) Forecast.js

// Function to get current location and fetch weather data

navigator.geolocation.getCurrentPosition(function (position) {

// Extract latitude and longitude from position object

latitude = position.coords.latitude;

longitude = position.coords.longitude;

// AJAX call to retrieve weather data

$.ajax({

type: "POST",

url: "http://localhost:5074/Dashboard/GetOWMData",

data: { latitude: latitude, longitude: longitude, option: "weather" },

// Success callback function to update weather HUD

success: function (data) {

updateWeatherHUD(data);

},

// Error callback function to handle errors

error: function (error) {

console.log(error);

alert("You need to allow location services or enter a location to use this feature.")

}

});

});

// Function to update weather HUD elements with data

function updateWeatherHUD(data) {

// Parse JSON data received from AJAX call

const parsedData = JSON.parse(data);

console.log(parsedData);

// Update weather HUD elements with corresponding data

updateTextContent(".location-text", parsedData.name);

updateTextContent(".icon-img", `http://openweathermap.org/img/w/${parsedData.weather[0].icon}.png`);

updateTextContent(".condition-text", parsedData.weather[0].main);

updateTextContent(".temperature-text", toCelsius(parsedData.main.temp));

updateTextContent(".lastupdated-text", toTime(parsedData.dt));

updateTextContent(".humidity-text", parsedData.main.humidity);

updateTextContent(".airpressure-text", parsedData.main.pressure);

updateTextContent(".sunrise-text", toTime(parsedData.sys.sunrise));

updateTextContent(".sunset-text", toTime(parsedData.sys.sunset));

}

# Health

The health section is designed for users to input and keep track of health metrics such as calories, steps, and water drunk on a daily basis. Users also have the ability to view previous days which they have made entries.

## (C) HealthController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using HealthAdviceGroup.Data;

using HealthAdviceGroup.Models;

using System.Security.Claims;

using Microsoft.AspNetCore.Mvc.ModelBinding;

namespace HealthAdviceGroup.Controllers

{

public class HealthController : Controller

{

private readonly ApplicationDbContext \_context;

public HealthController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Health/Index

public async Task<IActionResult> Index()

{

// Retrieve today's health data for the logged-in user

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

var date = DateTime.Now.ToString("dd/MM/yyyy");

var dailyEntry = await \_context.Health

.Where(u => u.UserId == userId)

.Where(u => u.Date == date)

.ToListAsync();

// Return the data to the view

return View(dailyEntry);

}

// GET: Health/MyDiary

public async Task<IActionResult> MyDiary()

{

// Retrieve all health entries for the logged-in user

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

var entries = await \_context.Health

.Where(u => u.UserId == userId)

.ToListAsync();

// Return the data to the view

return View(entries);

}

// GET: Health/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

// Retrieve health details for the specified ID

var health = await \_context.Health

.FirstOrDefaultAsync(m => m.Id == id);

// If no health object found, return NotFound

if (health == null)

{

return NotFound();

}

// Return the health object to the view

return View(health);

}

// GET: Health/Create

public IActionResult Create()

{

// Passes the User's ID and Date to the view for creating a new entry

ViewBag.UserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

ViewBag.Date = DateTime.Now.ToString("dd/MM/yyyy");

return View();

}

// POST: Health/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(Health health)

{

// Validate and save a new health entry

if (ModelState.IsValid)

{

\_context.Add(health);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

// If validation fails, return to the Create view with the userID and date values for making a new entry

ViewBag.UserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

ViewBag.Date = DateTime.Now.ToString("dd/MM/yyyy");

return View(health);

}

// GET: Health/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

// Retrieve health details for editing

var health = await \_context.Health.FindAsync(id);

// If no health object found, return NotFound

if (health == null)

{

return NotFound();

}

// Return the health object to the view for editing

return View(health);

}

// POST: Health/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,UserId,Steps,Calories,Water")] Health health)

{

// Validate and save edited health entry

if (id != health.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(health);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!HealthExists(health.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

// If validation fails, return to the Edit view with current values

return View(health);

}

// GET: Health/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

// Retrieve health details for deletion

var health = await \_context.Health

.FirstOrDefaultAsync(m => m.Id == id);

// If no health object found, return NotFound

if (health == null)

{

return NotFound();

}

// Return the health object to the view for confirmation of deletion

return View(health);

}

// POST: Health/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

// Delete the specified health entry

var health = await \_context.Health.FindAsync(id);

if (health != null)

{

\_context.Health.Remove(health);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

// Helper method to check if a health entry exists

private bool HealthExists(int id)

{

return \_context.Health.Any(e => e.Id == id);

}

}

}

## (M) Health.cs

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace HealthAdviceGroup.Models

{

// Model representing health-related data entries

public class Health

{

// Primary key for the health data entry, automatically generated by the database

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

// User ID associated with the health data entry, generated automatically.

[Required]

public string UserId { get; set; }

// Date of the health data entry, generated on the server's side.

[Required]

public string Date { get; set; }

// Number of steps taken, required and must be between 0 and 200,000

[Required]

[Range(0, 200000, ErrorMessage = "Number must be between 0 and 200,000")]

public int Steps { get; set; }

// Number of calories burned, required and must be between 0 and 50,000

[Required]

[Range(0, 50000, ErrorMessage = "Number must be between 0 and 50,000")]

public int Calories { get; set; }

// Amount of water consumed in cups, required and must be between 0 and 100

[Required]

[Range(0, 100, ErrorMessage = "Number must be between 0 and 100.")]

[DisplayName("Cups of Water")]

public int Water { get; set; }

}

}

## (V) Create.cshtml

@model HealthAdviceGroup.Models.Health

@{

ViewData["Title"] = "Create";

}

<h1>Create</h1>

<h4>Health</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="Create">

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

<div class="form-group">

<!-- Hidden field to store the user's id to identify who tracked it -->

<**input** **asp-for**="UserId" **type**="hidden" **value**="@ViewBag.UserId" />

</div>

<div class="form-group">

<!-- Hidden field to store the date to determine whether it should be showed on index or diary -->

<**input** **asp-for**="Date" **type**="hidden" **value**="@ViewBag.Date" />

</div>

<!-- Display name, validation, and input for steps with default value set to 0 -->

<div class="form-group">

<**label** **asp-for**="Steps" class="control-label"></**label**>

<**input** **asp-for**="Steps" class="form-control" **value**=0 />

<**span** **asp-validation-for**="Steps" class="text-danger"></**span**>

</div>

<!-- Display name, validation, and input for calories with default value set to 0 -->

<div class="form-group">

<**label** **asp-for**="Calories" class="control-label"></**label**>

<**input** **asp-for**="Calories" class="form-control" **value**=0 />

<**span** **asp-validation-for**="Calories" class="text-danger"></**span**>

</div>

<!-- Display name, validation, and input for steps with default value set to 0 -->

<div class="form-group">

<**label** **asp-for**="Water" class="control-label"></**label**>

<**input** **asp-for**="Water" class="form-control" **value**=0 />

<**span** **asp-validation-for**="Water" class="text-danger"></**span**>

</div>

<!-- Submit button to create the health tracker entry -->

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

<div>

<!-- Link to navigate back to the list page -->

<**a** **asp-action**="Index">Back to List</**a**>

</div>

@section Scripts {

@{

await Html.RenderPartialAsync("\_ValidationScriptsPartial");

}

}

## (V) Delete.cshtml

@model HealthAdviceGroup.Models.Health

@{

ViewData["Title"] = "Delete";

}

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>Health</h4>

<hr />

<dl class="row">

<!-- Display steps' title property -->

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Steps)

</dt>

<!-- Display steps' value -->

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Steps)

</dd>

<!-- Display calories' title property -->

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Calories)

</dt>

<!-- Display calories' value -->

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Calories)

</dd>

<!-- Display water's title property -->

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Water)

</dt>

<!-- Display water's value -->

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Water)

</dd>

</dl>

<**form** **asp-action**="Delete">

<**input** **type**="hidden" **asp-for**="Id" />

<input type="submit" value="Delete" class="btn btn-danger" /> |

<**a** **asp-action**="Index">Back to List</**a**>

</**form**>

</div>

## (V) Details.cshtml

@model HealthAdviceGroup.Models.Health

@{

ViewData["Title"] = "Details";

}

<h1>Details</h1>

<div>

<h4>Health</h4>

<hr />

<dl class="row">

<!-- Display the steps' title property -->

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Steps)

</dt>

<!-- Display the steps' value -->

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Steps)

</dd>

<!-- Display the calories' title property -->

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Calories)

</dt>

<!-- Display the calories' value -->

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Calories)

</dd>

<!-- Display the water's title property -->

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Water)

</dt>

<!-- Display the water's value -->

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Water)

</dd>

</dl>

</div>

<div>

<**a** **asp-action**="Edit" **asp-route-id**="@Model?.Id">Edit</**a**> |

<**a** **asp-action**="Index">Back to List</**a**>

</div>

## (V) Edit.cshtml

@model HealthAdviceGroup.Models.Health

@{

ViewData["Title"] = "Edit";

}

<h1>Edit</h1>

<h4>Health</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="Edit">

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

<!-- Hidden fields not needed as userID and date shouldnt be edited -->

<!-- Label, validation, and input for steps field -->

<div class="form-group">

<**label** **asp-for**="Steps" class="control-label"></**label**>

<**input** **asp-for**="Steps" class="form-control" />

<**span** **asp-validation-for**="Steps" class="text-danger"></**span**>

</div>

<!-- Label, validation, and input for calories field -->

<div class="form-group">

<**label** **asp-for**="Calories" class="control-label"></**label**>

<**input** **asp-for**="Calories" class="form-control" />

<**span** **asp-validation-for**="Calories" class="text-danger"></**span**>

</div>

<!-- Label, validation, and input for water field -->

<div class="form-group">

<**label** **asp-for**="Water" class="control-label"></**label**>

<**input** **asp-for**="Water" class="form-control" />

<**span** **asp-validation-for**="Water" class="text-danger"></**span**>

</div>

<!-- Button to submit changes to entry -->

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

## (V) Index.cshtml

@model IEnumerable<HealthAdviceGroup.Models.Health>

@{

ViewData["Title"] = "Index";

var item = Model.FirstOrDefault();

}

<h1>Index</h1>

<div class="container">

<div class="row">

<!-- Displays the steps as a card in a column of a grid -->

<div class="col-md-6 col-lg-4 text-center">

<div class="card" style="height: 15rem; width: 20rem;">

<h2 class="card-title">Steps</h2>

<!-- Displays number of steps if entry has been created, otherwise displays 0 -->

<h3 class="card-body">@(item?.Steps ?? 0) / 6000 steps</h3>

</div>

</div>

<div class="col-md-6 col-lg-4 text-center">

<a></a>

<div class="card" style="height: 15rem; width: 20rem;">

<h2 class="card-title">Calories</h2>

<!-- Displays number of calories if entry has been created, otherwise displays 0 -->

<h3 class="card-body">@(item?.Calories ?? 0) / 2000 calories</h3>

</div>

</div>

<div class="col-md-6 col-lg-4 text-center">

<div class="card" style="height: 15rem; width: 20rem;">

<h2 class="card-title">Water</h2>

<!-- Displays number of cups of water if entry has been created, otherwise displays 0 -->

<h3 class="card-body">@(item?.Water ?? 0) / 8 cups</h3>

</div>

</div>

</div>

</div>

@{

// If there is no entry for today, display a link to create one, otherwise display a link to edit the entry

if(item == null) {

<**a** **asp-action**="Create" class="display-6" style="float: left;">Log today's entry</**a**>

} else {

<**a** **asp-action**="Edit" **asp-route-id**=@item.Id class="display-6">Edit todays entry</**a**>

}

<**a** **asp-controller**="Health" **asp-action**="MyDiary" class="display-6" style="float: right;">View previous entries</**a**>

}

## (V) MyDiary.cshtml

@model IEnumerable<HealthAdviceGroup.Models.Health>

@{

ViewData["Title"] = "Index";

}

<h1>Index</h1>

<table class="table">

<thead>

<!-- Table headers -->

<tr>

<th>Date</th>

<th>Steps</th>

<th>Calories</th>

<th>Cups of Water</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

// Create a row for every health diary entry, which maps the date, steps, calories, and water -->

{

<tr>

<td>@item.Date</td>

<td>@item.Steps</td>

<td>@item.Calories</td>

<td>@item.Water</td>

<td>

<**a** **asp-action**="Edit" **asp-route-id**="@item.Id">Edit</**a**> |

<**a** **asp-action**="Delete" **asp-route-id**="@item.Id">Delete</**a**>

</td>

</tr>

}

</tbody>

</table>

# Home

This section mainly contains “landing pages” and similar features which either do not require an account or will cater towards any account type or users who are not signed in. Namely, a landing page and a page which details our privacy policy.

## (C) HomeController.cs

using HealthAdviceGroup.Models;

using Microsoft.AspNetCore.Mvc;

using System.Diagnostics;

namespace HealthAdviceGroup.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

// GET: Home/Index

public IActionResult Index()

{

return View();

}

// GET: Home/Privacy

public IActionResult Privacy()

{

return View();

}

// GET: Home/Error

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

// Display the Error view with the request identifier

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

## (V) Index.cshtml

@{

ViewData["Title"] = "Home Page";

}

<div class="text-center">

@{

// This is displayed to the user if they arent logged in

if(!User.Identity.IsAuthenticated)

{

<h1 class="display-6">Please <**a** class="" id="login" **asp-area**="Identity" **asp-page**="/Account/Login">log in</**a**> to access our advice and forecasts.</h1>

}

else // if user is logged in, this is displayed

{

<h1 class="display-6">Welcome to Health Advice Group</h1>

<p class="lead">Please select a service from the menu below..</p>

}

if (User.IsInRole("User"))

{

<!--User dashboard / features-->

<div class="container">

<div class="row">

<!--Weather forecast option-->

<div class="col-md-6 col-lg-4">

<div class="card" style="width: 18rem; margin: 1rem;">

<div class="card-body">

<h5 class="card-title">Weather Forecast</h5>

<p class="card-text">Shows the weather at the user's location</p>

<**a** **asp-controller**="Dashboard" **asp-action**="Forecast" class="btn btn-primary">Go</**a**>

</div>

</div>

</div>

<!--Air quality option-->

<div class="col-md-6 col-lg-4">

<div class="card" style="width: 18rem; margin: 1rem;">

<div class="card-body">

<h5 class="card-title">Air Quality</h5>

<p class="card-text">Shows the air quality at the user's location</p>

<**a** **asp-controller**="Dashboard" **asp-action**="AirQuality" class="btn btn-primary">Go</**a**>

</div>

</div>

</div>

<!--Health tracker option-->

<div class="col-md-6 col-lg-4">

<div class="card" style="width: 18rem; margin: 1rem;">

<div class="card-body">

<h5 class="card-title">Health Tracker</h5>

<p class="card-text">Diary for tracking calories, water, and steps.</p>

<**a** **asp-controller**="Health" **asp-action**="Index" class="btn btn-primary">Go</**a**>

</div>

</div>

</div>

<!--View advice option-->

<div class="col-md-6 col-lg-4">

<div class="card" style="width: 18rem; margin: 1rem;">

<div class="card-body">

<h5 class="card-title">View Advice</h5>

<p class="card-text">Shows advice given by Health Advice Group.</p>

<**a** **asp-controller**="Advice" **asp-action**="Index" class="btn btn-primary">Go</**a**>

</div>

</div>

</div>

</div>

</div>

}

if (User.IsInRole("Management"))

{

<!--Management dashboard / features-->

<div class="container">

<div class="row">

<!--Add advice option-->

<div class="col-md-6 col-lg-4">

<div class="card" style="width: 18rem; margin: 1rem;">

<div class="card-body">

<h5 class="card-title">Add Advice</h5>

<p class="card-text">Create new advice articles for users to see.</p>

<**a** **asp-controller**="Advice" **asp-action**="Create" class="btn btn-primary">Go</**a**>

</div>

</div>

</div>

<!--Health tracker option-->

<div class="col-md-6 col-lg-4">

<div class="card" style="width: 18rem; margin: 1rem;">

<div class="card-body">

<h5 class="card-title">Manage Advice</h5>

<p class="card-text">View and manage existing advice created by management.</p>

<**a** **asp-controller**="Advice" **asp-action**="Index" class="btn btn-primary">Go</**a**>

</div>

</div>

</div>

<!--View advice option-->

<div class="col-md-6 col-lg-4">

<div class="card" style="width: 18rem; margin: 1rem;">

<div class="card-body">

<h5 class="card-title">Admin Dashboard</h5>

<p class="card-text">View advice impressions and site metrics.</p>

<a onclick="alert('This feature has not been implemented yet. ');" class="btn btn-primary">Go</a>

</div>

</div>

</div>

</div>

</div>

}

}

</div>

## (V) Privacy.cshtml

@{

ViewData["Title"] = "Privacy Policy";

}

<h1>@ViewData["Title"]</h1>

<p>Awaiting Health Advice Group to contact us regarding their policies. </p>