Locations Controller:  
using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MMPD.Data.Context;

using MMPD.Data.Models;

namespace MMPD.Api.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class LocationsController : ControllerBase

{

private readonly AppDbContext \_context;

public LocationsController(AppDbContext context)

{

\_context = context;

}

// GET: api/Locations

[HttpGet]

public async Task<ActionResult<IEnumerable<Location>>> GetLocations([FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

return await \_context.Locations

.Include(l => l.LocationType)

.Include(l => l.Departments.Where(d => d.Active == true))

.ThenInclude(d => d.Employees.Where(e => e.Active == true))

.Where(l => l.Active == true)

.OrderBy(l => l.LocName)

.ToListAsync();

}

// GET: api/Locations/5

[HttpGet("{id}")]

public async Task<ActionResult<Location>> GetLocation(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var location = await \_context.Locations

.Include(l => l.LocationType)

.Include(l => l.Departments.Where(d => d.Active == true))

.ThenInclude(d => d.Employees.Where(e => e.Active == true))

.FirstOrDefaultAsync(l => l.Id == id && l.Active == true);

if (location == null)

{

return NotFound();

}

return location;

}

// GET: api/Locations/type/corporate

[HttpGet("type/{locationType}")]

public async Task<ActionResult<IEnumerable<Location>>> GetLocationsByType(string locationType, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

// Map string to location type ID based on your ExportData class

var locationTypeId = locationType.ToLower() switch

{

"corporate" => 1,

"metalmart" or "metal-mart" or "metal mart" => 2,

"servicecenter" or "service-center" or "service center" => 3,

"plant" => 4,

\_ => -1

};

if (locationTypeId == -1)

{

return BadRequest("Invalid location type. Valid types: corporate, metalmart, servicecenter, plant");

}

return await \_context.Locations

.Include(l => l.LocationType)

.Include(l => l.Departments.Where(d => d.Active == true))

.ThenInclude(d => d.Employees.Where(e => e.Active == true))

.Where(l => l.Loctype == locationTypeId && l.Active == true)

.OrderBy(l => l.LocNum ?? 0)

.ThenBy(l => l.LocName)

.ToListAsync();

}

// GET: api/Locations/corporate (Matches your ExportData.FetchCorpDataAsync)

[HttpGet("corporate")]

public async Task<ActionResult<IEnumerable<Location>>> GetCorporateLocations([FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

return await GetLocationsByTypeId(1);

}

// GET: api/Locations/metalmart (Matches your ExportData.FetchMMDataAsync)

[HttpGet("metalmart")]

public async Task<ActionResult<IEnumerable<Location>>> GetMetalMartLocations([FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

return await GetLocationsByTypeId(2);

}

// GET: api/Locations/servicecenter (Matches your ExportData.FetchSCDataAsync)

[HttpGet("servicecenter")]

public async Task<ActionResult<IEnumerable<Location>>> GetServiceCenterLocations([FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

return await GetLocationsByTypeId(3);

}

// GET: api/Locations/plant (Matches your ExportData.FetchPlantDataAsync)

[HttpGet("plant")]

public async Task<ActionResult<IEnumerable<Location>>> GetPlantLocations([FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

return await GetLocationsByTypeId(4);

}

// GET: api/Locations/5/departments

[HttpGet("{id}/departments")]

public async Task<ActionResult<IEnumerable<Department>>> GetLocationDepartments(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var location = await \_context.Locations.FindAsync(id);

if (location == null || location.Active != true)

{

return NotFound();

}

return await \_context.Departments

.Include(d => d.Employees.Where(e => e.Active == true))

.Where(d => d.Location == id && d.Active == true)

.OrderBy(d => d.DeptName)

.ToListAsync();

}

// GET: api/Locations/5/employees

[HttpGet("{id}/employees")]

public async Task<ActionResult<IEnumerable<Employee>>> GetLocationEmployees(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var location = await \_context.Locations.FindAsync(id);

if (location == null || location.Active != true)

{

return NotFound();

}

return await \_context.Employees

.Include(e => e.EmpDepartment)

.Where(e => e.Location == id && e.Active == true)

.OrderBy(e => e.FirstName)

.ThenBy(e => e.LastName)

.ToListAsync();

}

// PUT: api/Locations/5

[HttpPut("{id}")]

public async Task<IActionResult> PutLocation(int id, Location location, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

if (id != location.Id)

{

return BadRequest("ID mismatch");

}

var existingLocation = await \_context.Locations.AsNoTracking().FirstOrDefaultAsync(l => l.Id == id);

if (existingLocation == null)

{

return NotFound();

}

\_context.Entry(location).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!LocationExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// POST: api/Locations

[HttpPost]

public async Task<ActionResult<Location>> PostLocation(Location location, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

// Set defaults for new location

location.Active = true;

location.RecordAdd = DateTime.UtcNow;

// Validate location type exists

var locationTypeExists = await \_context.Loctypes.AnyAsync(lt => lt.Id == location.Loctype);

if (!locationTypeExists)

{

return BadRequest("Invalid location type ID");

}

\_context.Locations.Add(location);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetLocation", new { id = location.Id, apiKey }, location);

}

// DELETE: api/Locations/5 (Soft Delete)

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteLocation(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var location = await \_context.Locations

.Include(l => l.Departments)

.ThenInclude(d => d.Employees)

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

if (location == null)

{

return NotFound();

}

// Check if location has active departments or employees

var hasActiveDepartments = location.Departments.Any(d => d.Active == true);

var hasActiveEmployees = location.Departments.Any(d => d.Employees.Any(e => e.Active == true));

if (hasActiveDepartments || hasActiveEmployees)

{

return BadRequest("Cannot delete location with active departments or employees. Please reassign or deactivate them first.");

}

// Soft delete - set Active to false

location.Active = false;

await \_context.SaveChangesAsync();

return NoContent();

}

// POST: api/Locations/5/restore

[HttpPost("{id}/restore")]

public async Task<IActionResult> RestoreLocation(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var location = await \_context.Locations.FindAsync(id);

if (location == null)

{

return NotFound();

}

location.Active = true;

await \_context.SaveChangesAsync();

return NoContent();

}

private async Task<List<Location>> GetLocationsByTypeId(int locationTypeId)

{

return await \_context.Locations

.Include(l => l.LocationType)

.Include(l => l.Departments.Where(d => d.Active == true))

.ThenInclude(d => d.Employees.Where(e => e.Active == true))

.Where(l => l.Loctype == locationTypeId && l.Active == true)

.OrderBy(l => l.LocNum ?? 0)

.ThenBy(l => l.LocName)

.ToListAsync();

}

private bool LocationExists(int id)

{

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

}

private bool IsValidApiKey(string? apiKey)

{

if (string.IsNullOrEmpty(apiKey))

return false;

var validKeys = new[]

{

"maui-app-key-2024",

"crud-web-app-key-2024"

};

return validKeys.Contains(apiKey);

}

}

}  
  
Departments Controller:  
using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MMPD.Data.Context;

using MMPD.Data.Models;

namespace MMPD.Api.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class DepartmentsController : ControllerBase

{

private readonly AppDbContext \_context;

public DepartmentsController(AppDbContext context)

{

\_context = context;

}

// GET: api/Departments

[HttpGet]

public async Task<ActionResult<IEnumerable<Department>>> GetDepartments([FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

return await \_context.Departments

.Include(d => d.DeptLocation) // Include location info

.Include(d => d.Employees.Where(e => e.Active == true)) // Include active employees

.Where(d => d.Active == true) // Only active departments

.OrderBy(d => d.DeptName)

.ToListAsync();

}

// GET: api/Departments/5

[HttpGet("{id}")]

public async Task<ActionResult<Department>> GetDepartment(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var department = await \_context.Departments

.Include(d => d.DeptLocation)

.Include(d => d.Employees.Where(e => e.Active == true))

.FirstOrDefaultAsync(d => d.Id == id && d.Active == true);

if (department == null)

{

return NotFound();

}

return department;

}

// GET: api/Departments/location/5

[HttpGet("location/{locationId}")]

public async Task<ActionResult<IEnumerable<Department>>> GetDepartmentsByLocation(int locationId, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

return await \_context.Departments

.Include(d => d.DeptLocation)

.Include(d => d.Employees.Where(e => e.Active == true))

.Where(d => d.Location == locationId && d.Active == true)

.OrderBy(d => d.DeptName)

.ToListAsync();

}

// GET: api/Departments/5/employees

[HttpGet("{id}/employees")]

public async Task<ActionResult<IEnumerable<Employee>>> GetDepartmentEmployees(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var department = await \_context.Departments.FindAsync(id);

if (department == null || department.Active != true)

{

return NotFound();

}

return await \_context.Employees

.Include(e => e.EmpLocation)

.Include(e => e.EmpDepartment)

.Where(e => e.Department == id && e.Active == true)

.OrderBy(e => e.FirstName)

.ThenBy(e => e.LastName)

.ToListAsync();

}

// PUT: api/Departments/5

[HttpPut("{id}")]

public async Task<IActionResult> PutDepartment(int id, Department department, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

if (id != department.Id)

{

return BadRequest("ID mismatch");

}

var existingDepartment = await \_context.Departments.AsNoTracking().FirstOrDefaultAsync(d => d.Id == id);

if (existingDepartment == null)

{

return NotFound();

}

\_context.Entry(department).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!DepartmentExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// POST: api/Departments

[HttpPost]

public async Task<ActionResult<Department>> PostDepartment(Department department, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

// Set defaults for new department

department.Active = true;

department.RecordAdd = DateTime.UtcNow;

// Validate location exists

var locationExists = await \_context.Locations.AnyAsync(l => l.Id == department.Location && l.Active == true);

if (!locationExists)

{

return BadRequest("Invalid location ID or location is inactive");

}

\_context.Departments.Add(department);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetDepartment", new { id = department.Id, apiKey }, department);

}

// DELETE: api/Departments/5 (Soft Delete)

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteDepartment(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var department = await \_context.Departments

.Include(d => d.Employees)

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

if (department == null)

{

return NotFound();

}

// Check if department has active employees

var hasActiveEmployees = department.Employees.Any(e => e.Active == true);

if (hasActiveEmployees)

{

return BadRequest("Cannot delete department with active employees. Please reassign or deactivate employees first.");

}

// Soft delete - set Active to false

department.Active = false;

await \_context.SaveChangesAsync();

return NoContent();

}

// POST: api/Departments/5/restore

[HttpPost("{id}/restore")]

public async Task<IActionResult> RestoreDepartment(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var department = await \_context.Departments.FindAsync(id);

if (department == null)

{

return NotFound();

}

// Ensure the location is still active

var location = await \_context.Locations.FindAsync(department.Location);

if (location == null || location.Active != true)

{

return BadRequest("Cannot restore department because its location is inactive");

}

department.Active = true;

await \_context.SaveChangesAsync();

return NoContent();

}

private bool DepartmentExists(int id)

{

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

}

private bool IsValidApiKey(string? apiKey)

{

if (string.IsNullOrEmpty(apiKey))

return false;

var validKeys = new[]

{

"maui-app-key-2024",

"crud-web-app-key-2024"

};

return validKeys.Contains(apiKey);

}

}

}  
  
  
Employees Controller:  
using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MMPD.Data.Context;

using MMPD.Data.Models;

namespace MMPD.Api.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class EmployeesController : ControllerBase

{

private readonly AppDbContext \_context;

public EmployeesController(AppDbContext context)

{

\_context = context;

}

// GET: api/Employees

[HttpGet]

public async Task<ActionResult<IEnumerable<Employee>>> GetEmployees([FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

return await \_context.Employees

.Include(e => e.EmpDepartment)

.Include(e => e.EmpLocation)

.Where(e => e.Active == true) // Only active employees

.OrderBy(e => e.FirstName)

.ThenBy(e => e.LastName)

.ToListAsync();

}

// GET: api/Employees/5

[HttpGet("{id}")]

public async Task<ActionResult<Employee>> GetEmployee(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var employee = await \_context.Employees

.Include(e => e.EmpDepartment)

.Include(e => e.EmpLocation)

.FirstOrDefaultAsync(e => e.Id == id && e.Active == true);

if (employee == null)

{

return NotFound();

}

return employee;

}

// GET: api/Employees/department/5

[HttpGet("department/{departmentId}")]

public async Task<ActionResult<IEnumerable<Employee>>> GetEmployeesByDepartment(int departmentId, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

return await \_context.Employees

.Include(e => e.EmpDepartment)

.Include(e => e.EmpLocation)

.Where(e => e.Department == departmentId && e.Active == true)

.OrderBy(e => e.FirstName)

.ThenBy(e => e.LastName)

.ToListAsync();

}

// GET: api/Employees/location/5

[HttpGet("location/{locationId}")]

public async Task<ActionResult<IEnumerable<Employee>>> GetEmployeesByLocation(int locationId, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

return await \_context.Employees

.Include(e => e.EmpDepartment)

.Include(e => e.EmpLocation)

.Where(e => e.Location == locationId && e.Active == true)

.OrderBy(e => e.FirstName)

.ThenBy(e => e.LastName)

.ToListAsync();

}

// PUT: api/Employees/5

[HttpPut("{id}")]

public async Task<IActionResult> PutEmployee(int id, Employee employee, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

if (id != employee.Id)

{

return BadRequest("ID mismatch");

}

// Ensure we don't accidentally set Active to false unless intended

var existingEmployee = await \_context.Employees.AsNoTracking().FirstOrDefaultAsync(e => e.Id == id);

if (existingEmployee == null)

{

return NotFound();

}

\_context.Entry(employee).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!EmployeeExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// POST: api/Employees

[HttpPost]

public async Task<ActionResult<Employee>> PostEmployee(Employee employee, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

// Set defaults for new employee

employee.Active = true;

employee.RecordAdd = DateTime.UtcNow;

\_context.Employees.Add(employee);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetEmployee", new { id = employee.Id, apiKey }, employee);

}

// DELETE: api/Employees/5 (Soft Delete)

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteEmployee(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var employee = await \_context.Employees.FindAsync(id);

if (employee == null)

{

return NotFound();

}

// Soft delete - set Active to false instead of removing

employee.Active = false;

await \_context.SaveChangesAsync();

return NoContent();

}

// POST: api/Employees/5/restore (Restore soft deleted employee)

[HttpPost("{id}/restore")]

public async Task<IActionResult> RestoreEmployee(int id, [FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

var employee = await \_context.Employees.FindAsync(id);

if (employee == null)

{

return NotFound();

}

employee.Active = true;

await \_context.SaveChangesAsync();

return NoContent();

}

private bool EmployeeExists(int id)

{

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

}

private bool IsValidApiKey(string? apiKey)

{

if (string.IsNullOrEmpty(apiKey))

return false;

// Define your valid API keys

var validKeys = new[]

{

"maui-app-key-2024", // For MAUI sync

"crud-web-app-key-2024" // For your Blazor admin app

};

return validKeys.Contains(apiKey);

}

}

}  
  
  
Directory Controller May or may not need:  
using Microsoft.AspNetCore.Mvc;

using MMPD.Data.Context;

using MMPD.Data.Data;

namespace MMPD.Api.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class DirectoryController : ControllerBase

{

private readonly AppDbContext \_context;

private readonly ExportData \_exportData;

public DirectoryController(AppDbContext context)

{

\_context = context;

\_exportData = new ExportData(context);

}

/// <summary>

/// Full directory sync for MAUI apps - returns all location data with departments and employees

/// </summary>

[HttpGet("sync")]

public async Task<ActionResult<DirectorySyncResponse>> GetDirectorySync([FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

try

{

// Use your existing ExportData logic to get hierarchical data

var jsonData = await \_exportData.GenerateJson();

return Ok(new DirectorySyncResponse

{

Data = jsonData,

SyncTimestamp = DateTime.UtcNow,

Success = true,

Message = "Directory sync completed successfully"

});

}

catch (Exception ex)

{

return StatusCode(500, new DirectorySyncResponse

{

Success = false,

Message = $"Sync failed: {ex.Message}",

SyncTimestamp = DateTime.UtcNow

});

}

}

/// <summary>

/// Get last modified timestamp for incremental sync checking

/// </summary>

[HttpGet("lastmodified")]

public async Task<ActionResult<DateTime>> GetLastModified([FromQuery] string? apiKey = null)

{

if (!IsValidApiKey(apiKey))

return Unauthorized("Invalid or missing API key");

try

{

var lastModified = new[]

{

\_context.Employees.Where(e => e.RecordAdd.HasValue).Max(e => e.RecordAdd) ?? DateTime.MinValue,

\_context.Locations.Where(l => l.RecordAdd.HasValue).Max(l => l.RecordAdd) ?? DateTime.MinValue,

\_context.Departments.Where(d => d.RecordAdd.HasValue).Max(d => d.RecordAdd) ?? DateTime.MinValue

}.Max();

return Ok(lastModified);

}

catch (Exception ex)

{

return StatusCode(500, $"Error getting last modified date: {ex.Message}");

}

}

/// <summary>

/// Health check endpoint

/// </summary>

[HttpGet("health")]

public IActionResult HealthCheck()

{

return Ok(new

{

Status = "Healthy",

Timestamp = DateTime.UtcNow,

Database = \_context.Database.CanConnect() ? "Connected" : "Disconnected"

});

}

private bool IsValidApiKey(string? apiKey)

{

if (string.IsNullOrEmpty(apiKey))

return false;

var validKeys = new[]

{

"maui-app-key-2024",

"crud-web-app-key-2024"

};

return validKeys.Contains(apiKey);

}

}

/// <summary>

/// Response model for directory sync

/// </summary>

public class DirectorySyncResponse

{

public Dictionary<string, object> Data { get; set; } = new();

public DateTime SyncTimestamp { get; set; }

public bool Success { get; set; }

public string Message { get; set; } = string.Empty;

}

}