Validation  
  
using System.ComponentModel.DataAnnotations;

namespace ReportClaim.Validation

{

public class PolicyNumberValidator:ValidationAttribute

{

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

{

if (value == null)

{

return new ValidationResult("Policy Number cannot be empty");

}

if (value.ToString().Length < 6)

{

return new ValidationResult("Policy Number should be length of 6");

}

return ValidationResult.Success;

}

}

}

PolicyModel  
using System.ComponentModel.DataAnnotations;

using ReportClaim.Validation;

namespace ReportClaim.Models

{

public class Policy

{

public int Id { get; set; }

[PolicyNumberValidator]

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

public IEnumerable<Report> Reports { get; set; }

public Policy()

{

Reports=new List<Report>();

}

}

}

ReportModel  
  
using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace ReportClaim.Models

{

public class Report

{

public int Id { get; set; }

[Required(ErrorMessage = "IncidentDate can not be blank")]

public DateTime IncidentDate { get; set; }

[Required(ErrorMessage = "ClaimName can not be blank")]

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

[Required(ErrorMessage = "ClaimPhone can not be blank")]

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

[Required(ErrorMessage = "ClaimEmail can not be blank")]

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

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

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

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

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

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

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

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

// Foreign Keys

public int ClaimId { get; set; } // Foreign key for Claim

public Claim Claim { get; set; } // Navigation property

public int PolicyId { get; set; } // Foreign key for Policy

public Policy Policy { get; set; } // Navigation property

public Report()

{

Policy = new Policy();

Claim = new Claim();

}

}

}

Filer

using Microsoft.AspNetCore.Mvc.Filters;

using Microsoft.AspNetCore.Mvc;

using ReportClaim.Models.DTO;

namespace ReportClaim.Filter

{

public class ReportExceptionFilter:ExceptionFilterAttribute

{

public override void OnException(ExceptionContext context)

{

context.Result = new BadRequestObjectResult(new ErrorReponseDTO

{

ErrorCode = 500,

ErrorMessage = context.Exception.Message

});

}

}

}

Controllers

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using ReportClaim.Interfaces;

using ReportClaim.Models;

using ReportClaim.Models.DTO;

using System.Net;

using System.Security.Claims;

namespace ReportClaim.Controllers

{

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

[ApiController]

public class ReportController : ControllerBase

{

private readonly IPolicyService \_policyService;

private readonly IClaimService \_claimService;

private readonly IReportService \_reportService;

public ReportController(IPolicyService policyService, IClaimService claimService, IReportService reportService)

{

\_policyService = policyService;

\_claimService = claimService;

\_reportService = reportService;

}

[HttpGet]

[Route("getPolicy")]

public async Task<ActionResult<IEnumerable<PolicyDTO>>> GetPolicies()

{

List<string> policyNumbers = new List<string>();

try

{

var policies = await \_policyService.GetAllPolicies();

foreach (var policy in policies) { policyNumbers.Add(policy.PolicyNumber); }

if (policies == null || !policies.Any())

{

return NotFound("No policies found.");

}

return Ok(policyNumbers);

}

catch (Exception ex)

{

return StatusCode((int)HttpStatusCode.InternalServerError, $"Error retrieving policies: {ex.Message}");

}

}

[HttpGet]

[Route("getClaim")]

public async Task<ActionResult<IEnumerable<ClaimDTO>>> GetClaims()

{

List<string> claimTypes = new List<string>();

try

{

var claims= await \_claimService.GetAllClaims();

foreach (var claimType in claims) { claimTypes.Add(claimType.ClaimType); }

if (claimTypes == null || !claimTypes.Any())

{

return NotFound("No claims found.");

}

return Ok(claimTypes);

}

catch (Exception ex)

{

return StatusCode((int)HttpStatusCode.InternalServerError, $"Error retrieving claims: {ex.Message}");

}

}

[HttpPost]

public async Task<ActionResult> CreateReport([FromForm] ReportDTO reportDTO)

{

try

{

if (ModelState.IsValid)

{

var report = await \_reportService.CreateReport(reportDTO);

return Ok(report.Id);

}

else

{

return BadRequest(new ErrorReponseDTO

{

ErrorCode = 400,

ErrorMessage = "one or more fields validate error"

});

}

}

catch (Exception ex)

{

return StatusCode(500,ex.Message);

}

}

}

}

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using ReportClaim.Interfaces;

using ReportClaim.Models;

using ReportClaim.Models.DTO;

namespace ReportClaim.Controllers

{

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

[ApiController]

public class PolicyController : ControllerBase

{

private readonly IPolicyService \_policyService;

public PolicyController(IPolicyService policyService)

{

\_policyService = policyService;

}

[HttpPost]

public async Task<ActionResult> InputPolicy(PolicyDTO policyDTO)

{

try

{

if (ModelState.IsValid)

{

var policy = await \_policyService.CreatePolicy(policyDTO);

return Ok(policy);

}

else

{

return BadRequest(new ErrorReponseDTO

{

ErrorCode = 400,

ErrorMessage = "one or more fields validate error"

});

}

}

catch (Exception ex)

{

return StatusCode(500, ex);

}

}

}

}

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using ReportClaim.Interfaces;

using ReportClaim.Models.DTO;

using ReportClaim.Services;

namespace ReportClaim.Controllers

{

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

[ApiController]

public class ClaimController : ControllerBase

{

private readonly IClaimService \_claimService;

public ClaimController(IClaimService claimService)

{

\_claimService = claimService;

}

[HttpPost]

public async Task<ActionResult> CreateClaim(ClaimDTO claimDTO)

{

try

{

if (ModelState.IsValid)

{

var claim = await \_claimService.CreateClaim(claimDTO);

return Ok(claim);

}

else

{

return BadRequest(new ErrorResponseDTO

{

ErrorCode = 400,

ErrorMessage = "one or more fields validate error"

});

}

}

catch (Exception ex)

{

return StatusCode(500, ex);

}

}

}

}