```csharp
using Meditech.App.CommonService.Contanst;
using Meditech.App.CommonService.ViewModels;
using Meditech.Lib.CommonShare;
using Meditech.Lib.Share.CommonClass.Paging;
using System.ComponentModel;
using System.Reflection;
using Meditech.Core.Common.Repositories;
using Meditech.App.CommonService.Dtos;
using Meditech.Core.Common.DBModels;
using Meditech.Core.Auth.Entities;
using Meditech.Core.Common.Services;
using Meditech.Core.Common.ViewModels;
using Meditech.Lib.CommonShare.Helpers;
using Meditech.Lib.CommonShare.Extensions;
using Newtonsoft.Json.Linq;


namespace Meditech.App.CommonService.Services
{
    public interface ISignatureProcessService
    {
        List<SignatureProcessDto> getAll();
        List<SignatureProcessDto> Search(SignatureProcessSearchQuery model);
        ObjectReturnCode<SignatureProcessDto> getById(Guid id);
        SignedDocumentViewModel GetSignedDocument(SignedDocumentSearchQuery model, Guid userId);
        ReturnCode Create(SignatureProcessViewModel model, Guid userId);
        ReturnCode Update(UpdateSignatureProcessViewModel model);
        ReturnCode Delete(Guid Id);

        List<SignPositionDto> GetAllSignPositions(bool showHide = false);
        List<DocumentTypeDto> GetAllDocumentType();

        ReturnCode CreateSignPosition(SignPositionViewModel model, Guid userId);
        ReturnCode UpdateSignPosition(SignPositionViewModel model, Guid userId);
        ReturnCode DeleteSignPosition(int id);

        ReturnCode InitSignPositions();
        ReturnCode InitDocumentTypes(Guid userId);

        ReturnCode CreateSignedDocument(SignedDocumentCreateViewModel model, Guid userId);
        ReturnCode UpdateSignedDocument(SignedDocumentUpdateViewModel model, Guid userId);
        ReturnCode DeleteSignedDocument(SignedDocumentSearchQuery model, Guid userId);

        ReturnCode SignRemind(SignedDocumentSearchQuery model, Guid userId);
        ReturnCode SignReject(SignRejectViewModel model, Guid userId);

        List<SignReportViewModel> GetSignByUserId(FilterSignedDocumentViewModel filter, Guid userId);
        PagingDataModel<SignReportViewModel> SearchSignDocument(PaginationQueryModel<SearchSignDocumentViewModel> query, Guid userId);

    }

    public class SignatureProcessService : ISignatureProcessService
    {
        private readonly IEmployeeRepository _employeeRepository;
        private readonly IUserRepository _userRepository;
        private readonly ISignatureProcessRepository _signatureProcessRepository;
        private readonly ISignPositionRepository _signPositionRepository;
        private readonly IDocumentTypeRepository _documentTypeRepository;
        private readonly ISignedDocumentRepository _signedDocumentRepository;
        private readonly IDepartmentCategoryRepository _departmentCategoryRepository;
        private readonly IErpNotificationService _erpNotificationService;
        private readonly IHistoryChangeService _historyChangeService;
        private readonly CommonUnitOfWork _unitOfWork;

        public SignatureProcessService(
            IEmployeeRepository employeeRepository,
            IUserRepository userRepository,
            ISignatureProcessRepository signatureProcessRepository,
            ISignPositionRepository signPositionRepository,
            IDocumentTypeRepository documentTypeRepository,
            ISignedDocumentRepository signedDocumentRepository,
            IDepartmentCategoryRepository departmentCategoryRepository,
            IErpNotificationService erpNotificationService,
            IHistoryChangeService historyChangeService,
            CommonUnitOfWork unitOfWork)
        {
            _employeeRepository = employeeRepository;
            _userRepository = userRepository;
            _signatureProcessRepository = signatureProcessRepository;
            _signPositionRepository = signPositionRepository;
            _documentTypeRepository = documentTypeRepository;
            _signedDocumentRepository = signedDocumentRepository;
            _departmentCategoryRepository = departmentCategoryRepository;
            _erpNotificationService = erpNotificationService;
            _historyChangeService = historyChangeService;
            _unitOfWork = unitOfWork;
        }

        public List<SignatureProcessDto> getAll()
        {
            var queryData = (from a in _signatureProcessRepository.Table
                             join b in _signPositionRepository.Table on a.SignPositionId equals b.Id
                             join c in _documentTypeRepository.Table on a.DocumentTypeId equals c.Id into tpmHos
                             where a.IsDeleted == false
                             select new SignatureProcessDto
                             {
                                 Id = a.Id,
                                 DisplayName = a.DisplayName,
                                 SignPositionId = a.SignPositionId,
                                 DocumentTypeId = a.DocumentTypeId,
                                 SignPositionName = b.Name,
                                 SignOrder = a.SignOrder,
                                 Description = a.Description,
                                 Required = a.Required,
                                 SpecificSigner = a.SpecificSigner,
                                 SignDeadline = a.SignDeadline,
                                 SignerEmployee = a.SignerEmployee,
                                 CreatedById = a.CreatedById,
                                 CreatedOnUtc = a.CreatedOnUtc,
                                 UpdatedOnUtc = a.UpdatedOnUtc,
                                 HidePrint = a.HidePrint,
                                 BranchId = a.BranchId,
                                 IsDeleted = a.IsDeleted,
                             });

            return queryData.ToList();
        }

        public List<SignatureProcessDto> Search(SignatureProcessSearchQuery query)
        {
            var queryData = (from a in _signatureProcessRepository.Table
                             join b in _signPositionRepository.Table on a.SignPositionId equals b.Id
                             join c in _documentTypeRepository.Table on a.DocumentTypeId equals c.Id into tpmHos
                             where a.IsDeleted == false
                                 && a.BranchId == query.BranchId
                                 && a.DocumentTypeId == query.DocumentTypeId
                             orderby a.SignOrder ascending
                             select new SignatureProcessDto
                             {
                                 Id = a.Id,
                                 DisplayName = a.DisplayName,
                                 SignPositionId = a.SignPositionId,
                                 DocumentTypeId = a.DocumentTypeId,
                                 SignPositionName = b.Name,
                                 SignOrder = a.SignOrder,
                                 Description = a.Description,
                                 Required = a.Required,
                                 SpecificSigner = a.SpecificSigner,
                                 SignDeadline = a.SignDeadline,
                                 SignerEmployee = a.SignerEmployee,
                                 CreatedById = a.CreatedById,
                                 CreatedOnUtc = a.CreatedOnUtc,
                                 UpdatedOnUtc = a.UpdatedOnUtc,
                                 HidePrint = a.HidePrint,
                                 BranchId = a.BranchId,
                             });

            var data = queryData.ToList();

            foreach (var item in data)
            {
                item.SignerEmployeeNames = item.SignerEmployee != null ?
                    string.Join(", ",
                        _employeeRepository.Table
                            .Join(_userRepository.Table, e => e.Id, u => u.Id, (e, u) => new { e, u })
                            .Where(j => item.SignerEmployee.Contains(j.e.Id))
                            .Select(j => j.u.FullName)
                    ) : string.Empty;
            }

            return data;
        }

        public ObjectReturnCode<SignatureProcessDto> getById(Guid id)
        {
            ObjectReturnCode<SignatureProcessDto> dto = new ObjectReturnCode<SignatureProcessDto>();

            var signatureProcess = _signatureProcessRepository.GetById(id);
            if (signatureProcess == null)
            {
                dto.Code = (ErrorCode)SignatureProcessErrorCode.SignProcess_NotFound;
                dto.Description = "Quy trình ký không tại!";
                return dto;
            }

            SignatureProcessDto signatureProcessModel = new SignatureProcessDto();

            signatureProcessModel.Id = signatureProcess.Id;
            signatureProcessModel.DocumentTypeId = signatureProcess.DocumentTypeId;
            signatureProcessModel.SignPositionId = signatureProcess.SignPositionId;
            signatureProcessModel.DisplayName = signatureProcess.DisplayName;
            signatureProcessModel.SignOrder = signatureProcess.SignOrder;
            signatureProcessModel.SignDeadline = signatureProcess.SignDeadline;
            signatureProcessModel.Required = signatureProcess.Required;
            signatureProcessModel.SpecificSigner = signatureProcess.SpecificSigner;
            signatureProcessModel.SignerEmployee = signatureProcess.SignerEmployee ?? new Guid[] { };
            signatureProcessModel.HidePrint = signatureProcess.HidePrint;
            signatureProcessModel.CreatedById = signatureProcess.CreatedById;
            signatureProcessModel.CreatedOnUtc = signatureProcess.CreatedOnUtc;
            signatureProcessModel.UpdatedOnUtc = signatureProcess.UpdatedOnUtc;

            dto.Data = signatureProcessModel;

            return dto;
        }

        public ReturnCode Create(SignatureProcessViewModel model, Guid userId)
        {
            ReturnCode ret = new ReturnCode();

            var signPosition = _signatureProcessRepository.Table
                .Where(w => w.DocumentTypeId == model.DocumentTypeId &&
                            w.BranchId == model.BranchId &&
                            w.SignPositionId == model.SignPositionId &&
                            model.SignPositionId == 1)
                .FirstOrDefault();
            if (signPosition != null)
            {
                ret.Code = (ErrorCode)SignatureProcessErrorCode.SignProcess_SignPositionExists;
                ret.Description = "Vị trí ký người lập đã tồn tại!";
                return ret;
            }

            var signatureProcess = new SignatureProcess();
            signatureProcess.Id = Guid.NewGuid();
            signatureProcess.DisplayName = model.DisplayName;
            signatureProcess.Description = model.Description;
            signatureProcess.DocumentTypeId = model.DocumentTypeId;
            signatureProcess.SignPositionId = model.SignPositionId;
            signatureProcess.SignOrder = model.SignOrder;
            signatureProcess.SignerEmployee = model.SignerEmployee;
            signatureProcess.SignDeadline = model.SignDeadline;
            signatureProcess.Required = model.Required;
            signatureProcess.SpecificSigner = model.SpecificSigner;
            signatureProcess.HidePrint = model.HidePrint;
            signatureProcess.BranchId = model.BranchId;
            signatureProcess.CreatedById = userId;
            signatureProcess.CreatedOnUtc = DateTime.UtcNow;
            signatureProcess.UpdatedOnUtc = DateTime.UtcNow;

            _signatureProcessRepository.Insert(signatureProcess);
            _unitOfWork.Save();
            return ret;
        }

        public ReturnCode Update(UpdateSignatureProcessViewModel model)
        {
            ReturnCode ret = new ReturnCode();
            var signatureProcess = _signatureProcessRepository.GetById(model.Id);
            if (signatureProcess == null)
            {
                ret.Code = (ErrorCode)SignatureProcessErrorCode.SignProcess_NotFound;
                ret.Description = "Quy trình ký không tại!";
                return ret;
            }

            signatureProcess.DisplayName = model.DisplayName;
            signatureProcess.Description = model.Description;
            //signatureProcess.DocumentTypeId = model.DocumentTypeId;
            signatureProcess.SignPositionId = model.SignPositionId;
            signatureProcess.SignOrder = model.SignOrder;
            signatureProcess.SignerEmployee = model.SignerEmployee;
            signatureProcess.SignDeadline = model.SignDeadline;
            signatureProcess.Required = model.Required;
            signatureProcess.SpecificSigner = model.SpecificSigner;
            signatureProcess.HidePrint = model.HidePrint;
            signatureProcess.UpdatedOnUtc = DateTime.UtcNow;

            _signatureProcessRepository.Update(signatureProcess);
            _unitOfWork.Save();
            return ret;
        }

        public ReturnCode Delete(Guid Id)
        {
            ReturnCode ret = new ReturnCode();

            var signatureProcess = _signatureProcessRepository.GetById(Id);
            if (signatureProcess == null)
            {
                ret.Code = (ErrorCode)SignatureProcessErrorCode.SignProcess_NotFound;
                ret.Description = "Quy trình ký không tại!";
                return ret;
            }

            signatureProcess.IsDeleted = true;
            _signatureProcessRepository.Update(signatureProcess);
            _unitOfWork.Save();
            return ret;
        }

        public ReturnCode InitSignPositions()
        {
            ReturnCode ret = new ReturnCode();

            foreach (SignPositionEnum position in Enum.GetValues(typeof(SignPositionEnum)))
            {
                var id = (int)position;
                var exists = _signPositionRepository.GetById(id);
                if (exists != null)
                    continue;

                var signPosition = new SignPosition();
                signPosition.Id = id;
                signPosition.Name = GetEnumDescription(position);

                _signPositionRepository.Insert(signPosition);
            }

            _unitOfWork.Save();
            return ret;
        }

        public ReturnCode InitDocumentTypes(Guid userId)
        {
            ReturnCode ret = new ReturnCode();

            foreach (DocumentTypeEnum type in Enum.GetValues(typeof(DocumentTypeEnum)))
            {
                var id = (int)type;
                var exists = _documentTypeRepository.GetById(id);
                if (exists == null)
                {
                    var documentType = new DocumentType();
                    documentType.Id = id;
                    documentType.Name = GetEnumDescription(type);
                    _documentTypeRepository.Insert(documentType);

                    exists = documentType;
                }


                var branch = _departmentCategoryRepository.GetMulti(x => x.Id == x.BranchId && x.IsDeleted == false).ToList();

                if (branch.Any())
                    foreach (var item in branch)
                    {
                        var existsPosition = _signatureProcessRepository.GetSingleByCondition(x => x.DocumentTypeId == exists.Id &&
                            x.SignPositionId == 1 && x.BranchId == item.Id);

                        if (existsPosition != null)
                            continue;

                        // default
                        var signatureProcess = new SignatureProcess();
                        signatureProcess.Id = Guid.NewGuid();
                        signatureProcess.DisplayName = "Người lập";
                        signatureProcess.DocumentTypeId = exists.Id;
                        signatureProcess.SignPositionId = 1;
                        signatureProcess.SignOrder = 0;
                        signatureProcess.Required = true;
                        signatureProcess.SpecificSigner = false;
                        signatureProcess.CreatedById = userId;
                        signatureProcess.CreatedOnUtc = DateTime.UtcNow;
                        signatureProcess.UpdatedOnUtc = DateTime.UtcNow;
                        signatureProcess.BranchId = item.Id;
                        _signatureProcessRepository.Insert(signatureProcess);
                    }
            }

            _unitOfWork.Save();
            return ret;
        }

        public List<SignPositionDto> GetAllSignPositions(bool showHide = false)
        {
            var query = from a in _signPositionRepository.Table
                        where a.IsDeleted == false
                        orderby a.DisplayOrder
                        select new SignPositionDto
                        {
                            Id = a.Id,
                            Name = a.Name,
                            Description = a.Description,
                            DisplayOrder = a.DisplayOrder,
                            InActive = a.InActive,
                        };

            if (!showHide)
            {
                query = query.Where(w => w.InActive == false);
            }

            return query.AsEnumerable().ToList();
        }

        public List<DocumentTypeDto> GetAllDocumentType()
        {
            var query = from a in _documentTypeRepository.Table
                        where a.IsDeleted == false
                        select new DocumentTypeDto
                        {
                            Id = a.Id,
                            Name = a.Name,
                        };

            return query.AsEnumerable().ToList();
        }



        public SignedDocumentViewModel GetSignedDocument(SignedDocumentSearchQuery query, Guid userId)
        {
            SignedDocumentViewModel signedDocument = new SignedDocumentViewModel();

            var signedDocumentList = (from a in _signedDocumentRepository.Table
                                      where a.DocumentTypeId == query.DocumentTypeId
                                      && a.DocumentId == query.DocumentId
                                      orderby a.SigningRound ascending
                                      select new SignedDocumentDto
                                      {
                                          Id = a.Id,
                                          SignName = a.SignName,
                                          DocumentTypeId = a.DocumentTypeId,
                                          SignOrder = a.SignOrder,
                                          SignerEmployees = a.SignerEmployees,
                                          Required = a.Required,
                                          AllowSign = a.SignerId != null ? a.SignerId == userId : false,
                                          SpecificSigner = a.SpecificSigner && a.CreatedById == userId,
                                          Signed = a.Signed,
                                          IsNextSign = a.IsNextSign,
                                          SignerId = a.SignerId,
                                          SignerName = a.SignerName,
                                          SignaturePath = a.SignaturePath,
                                          SignedOnUtc = a.SignedOnUtc,
                                          Reason = a.Reason,
                                          RejectOnUtc = a.RejectOnUtc,
                                      }).ToList();

            if (signedDocumentList.Count > 0)
            {
                var querySignOrder2 = signedDocumentList.GroupBy(p => p.SignOrder,
                    (key, g) => new SignaturePositionGroupByOrder
                    {
                        SignOrder = key,
                        signatureProcesses = g.ToList(),
                        SignedNumber = g.Where(x => x.Signed == 1).Count(),
                        TotalSign = g.Count()
                    });

                var signOrders2 = querySignOrder2.ToList();

                var signName = signedDocumentList.Where(w => w.IsNextSign && w.SignerId == userId).Select(s => s.SignName).FirstOrDefault();

                signedDocument.SignaturePositionGroupByOrder = signOrders2;
                signedDocument.TotalSign = signedDocumentList.Count();
                signedDocument.SignedNumber = signedDocumentList.Where(x => x.Signed == 1).Count();
                signedDocument.SignNameCurrent = signName;

                if (signedDocumentList.Any(t => t.IsNextSign == false))
                {
                    signedDocument.SignedStatus = SignOptionStatus.ChoKy;
                }
                if (signedDocumentList.Any(t => t.IsNextSign == true && t.SignerId == userId))
                {
                    signedDocument.SignedStatus = SignOptionStatus.ChoBanKy;
                }
                if (signedDocumentList.Any(t => t.IsNextSign == false && t.SignerId == userId && t.Signed == 1))
                {
                    signedDocument.SignedStatus = SignOptionStatus.BanDaKy;
                }
                if (signedDocumentList.Any(t => t.Signed == (int)SignedStatus.TuChoiKy))
                {
                    signedDocument.SignedStatus = SignOptionStatus.BiTuChoi;
                }
                if (signedDocument.TotalSign == signedDocument.SignedNumber)
                {
                    signedDocument.SignedStatus = SignOptionStatus.HoanThanh;
                }

                signedDocument.SignedStatusName = signedDocument.SignedStatus.GetDescription();

                return signedDocument;
            }

            var queryNewData = (from a in _signatureProcessRepository.Table
                                join b in _signPositionRepository.Table on a.SignPositionId equals b.Id
                                join c in _documentTypeRepository.Table on a.DocumentTypeId equals c.Id into tpmHos
                                where a.IsDeleted == false
                                    && a.BranchId == query.BranchId
                                    && a.DocumentTypeId == query.DocumentTypeId
                                orderby a.SignOrder ascending
                                select new SignatureProcessDto
                                {
                                    //Id = a.Id,
                                    DisplayName = a.DisplayName,
                                    SignPositionId = a.SignPositionId,
                                    DocumentTypeId = a.DocumentTypeId,
                                    SignPositionName = b.Name,
                                    SignOrder = a.SignOrder,
                                    Description = a.Description,
                                    Required = a.Required,
                                    SpecificSigner = a.SpecificSigner,
                                    SignDeadline = a.SignDeadline,
                                    SignerEmployee = a.SignerEmployee,
                                    CreatedById = a.CreatedById,
                                    CreatedOnUtc = a.CreatedOnUtc,
                                    HidePrint = a.HidePrint,
                                    BranchId = a.BranchId,
                                });

            var querySignOrder = queryNewData.GroupBy(p => p.SignOrder,
                (key, g) => new SignaturePositionGroupByOrder
                {
                    SignOrder = key,
                    signatureProcesses = g.Select(x => new SignedDocumentDto
                    {
                        DocumentId = query.DocumentId,
                        DocumentTypeId = x.DocumentTypeId,
                        SignPositionId = x.SignPositionId,
                        SignName = x.DisplayName,
                        Required = x.Required ?? false,
                        SpecificSigner = false,
                        SignerEmployees = x.SignerEmployee,
                    }).ToList(),
                    TotalSign = g.Count()
                });

            var signOrders = querySignOrder.ToList();

            var iCount = 0;

            foreach (var item in signOrders)
            {
                foreach (var sign in item.signatureProcesses)
                {
                    sign.IsNextSign = iCount == 0;
                    sign.AllowSign = sign.SignerEmployees != null ? sign.SignerEmployees.Contains(userId) : false;
                    iCount++;
                }
            }

            //var data = queryData.ToList();

            signedDocument.SignaturePositionGroupByOrder = signOrders;
            signedDocument.TotalSign = iCount;

            return signedDocument;
        }

        public ReturnCode CreateSignedDocument(SignedDocumentCreateViewModel model, Guid userId)
        {
            ReturnCode ret = new ReturnCode();

            var query = new SignatureProcessSearchQuery();
            query.DocumentTypeId = model.DocumentTypeId;
            query.BranchId = model.BranchId;

            var data = Search(query);

            if (data == null)
            {
                ret.Code = (ErrorCode)SignatureProcessErrorCode.SignProcess_NotFound;
                ret.Description = "Quy trình ký không tồn tại!";
                return ret;
            }

            var exists = _signedDocumentRepository.Table.Where(w => w.DocumentId == model.DocumentId).FirstOrDefault();
            if (exists != null)
            {
                ret.Code = (ErrorCode)SignatureProcessErrorCode.SignProcess_SignedDocumentExists;
                ret.Description = "Chứng từ ký đã tồn tại!";
                return ret;
            }

            var iCount = 1;
            var createdOn = DateTime.UtcNow;
            foreach (var item in data)
            {
                var sign = new SignedDocument();
                sign.Id = Guid.NewGuid();
                sign.DocumentId = model.DocumentId;
                sign.DocumentCode = model.DocumentCode;
                sign.DocumentTypeId = item.DocumentTypeId;
                sign.SignPositionId = item.SignPositionId;
                sign.Description = model.Description;
                sign.SignName = item.DisplayName;
                sign.Required = item.Required ?? false;
                sign.SpecificSigner = item.SpecificSigner ?? false;

                if (iCount == 1)
                {
                    var signer = _userRepository.GetById(userId);

                    sign.IsNextSign = true;
                    sign.SignerId = userId;
                    sign.SignerName = signer.FullName;
                    sign.SignerEmployees = new Guid[] { userId };
                    sign.Signed = item.Required ?? false ? 0 : 1;
                }
                else
                {
                    sign.SignerEmployees = item.SignerEmployee;
                    if (item.SignerEmployee != null && item.SignerEmployee.Length == 1)
                    {
                        var signerId = item.SignerEmployee[0];
                        var signer = _userRepository.GetById(signerId);

                        if (signer == null)
                        {
                            ret.Code = ErrorCode.User102_NotFound;
                            ret.Description = "Nhân viên chỉ định ký không tồn tại!";
                            return ret;
                        }

                        sign.SignerId = signerId;
                        sign.SignerName = signer.FullName;
                        //sign.SpecificSigner = false;
                    }
                }

                sign.SignOrder = item.SignOrder;
                sign.CreatedById = userId;
                sign.CreatedOnUtc = createdOn;
                sign.SigningRound = iCount++;

                _signedDocumentRepository.Insert(sign);
            }

            _unitOfWork.Save();

            var lst = _signedDocumentRepository.GetMulti(x =>
                x.DocumentTypeId == model.DocumentTypeId
                && x.DocumentId == model.DocumentId
                && x.Required);

            ret.Data = new
            {
                TotalSigns = lst.Count(),
                TotalSigned = lst.Where(x => x.Signed == 1).Count()
            };

            // Save history
            var history = new HistoryChangeViewModel();
            history.Id = Guid.NewGuid();
            history.Action = EActionType.Them;
            history.ActionName = "Thêm";
            history.ObjectId = model.DocumentId.ToString();
            history.ObjectType = ((DocumentTypeEnum)model.DocumentTypeId).ToString();
            history.Content = "Thêm chứng từ ký";
            history.CreatedById = userId;
            history.CreatedTime = createdOn;

            _historyChangeService.InsertHistoryChange(history);

            return ret;
        }

        public ReturnCode UpdateSignedDocument(SignedDocumentUpdateViewModel model, Guid userId)
        {
            ReturnCode ret = new ReturnCode();

            var sign = _signedDocumentRepository.GetById(model.SignedDocumentId);

            if (sign == null)
            {
                ret.Code = (ErrorCode)SignatureProcessErrorCode.SignProcess_NotFound;
                ret.Description = "Vị trí ký không còn tồn tại!";
                return ret;
            }

            if (sign.Signed == 1)
            {
                ret.Code = (ErrorCode)SignatureProcessErrorCode.SignProcess_PositionHasBeenSigned;
                ret.Description = "Vị trí đã được ký!";
                return ret;
            }

            // Chỉ định ký
            if (model.AuthorizedSignatoryId != null)
            {
                var user = _userRepository.GetById(model.AuthorizedSignatoryId);
                sign.SignerId = user.Id;
                sign.SignerName = user.FullName;
                if(!string.IsNullOrEmpty(model.Description))
                    sign.Description = model.Description;
                var updatedOn = DateTime.UtcNow;
                _signedDocumentRepository.Update(sign);


                DocumentTypeEnum documentTypeEnum = (DocumentTypeEnum)sign.DocumentTypeId;

                // Save history
                var history = new HistoryChangeViewModel();
                history.Id = Guid.NewGuid();
                history.Action = EActionType.ChiDinhKy;
                history.ActionName = "Chỉ định ký";
                history.ObjectId = sign.Id.ToString();
                history.ObjectType = documentTypeEnum.ToString();
                history.Content = $"Bạn được chỉ định ký vị trí {sign.SignName} cho phiếu {GetEnumDescription(documentTypeEnum)}";
                history.CreatedById = userId;
                history.CreatedTime = updatedOn;

                _historyChangeService.InsertHistoryChange(history);

                // Send Notification
                var noti = new NotificationAddViewModel();
                noti.Subject = GetEnumDescription(documentTypeEnum);
                noti.SenderUserId = userId;
                noti.SendTime = updatedOn;
                noti.Content = $"Bạn được chỉ định ký vị trí {sign.SignName} cho phiếu {GetEnumDescription(documentTypeEnum)}";
                noti.ObjectId = sign.DocumentId.ToString();
                noti.ObjectType = documentTypeEnum.ToString();
                noti.ObjectCode = sign.DocumentCode;
                noti.ReceiverEmployeeIds = new List<Guid>()
                {
                    (Guid)sign.SignerId
                };

                var rtNoti = _erpNotificationService.AddNotification(noti, ErpNotificationType.Signature);
                if (!rtNoti.IsOk)
                {
                    ret.Code = rtNoti.Code;
                    ret.Description = rtNoti.Description;
                    return ret;
                }
            }
            //Ký
            else if (model.Signed != null)
            {
                var signer = _userRepository.GetById(userId);
                if (signer == null)
                {
                    ret.Code = ErrorCode.User102_NotFound;
                    ret.Description = "Tài khoản không tồn tại!";
                    return ret;
                }

                var signedOn = DateTime.UtcNow;

                sign.Signed = 1;
                sign.SignerId = signer.Id;
                sign.SignerName = signer.FullName;
                sign.SignaturePath = signer.SignatureUrl;
                sign.IsNextSign = false;
                sign.SignedOnUtc = signedOn;
                sign.Description = model.Description;
                _signedDocumentRepository.Update(sign);

                // Save history
                var history = new HistoryChangeViewModel();
                history.Id = Guid.NewGuid();
                history.Action = EActionType.Ky;
                history.ActionName = "Ký";
                history.ObjectId = sign.Id.ToString();
                history.ObjectType = ((DocumentTypeEnum)sign.DocumentTypeId).ToString();
                history.Content = $"Bạn đã ký vị trí {sign.SignName}";
                history.CreatedById = userId;
                history.CreatedTime = signedOn;
                _historyChangeService.InsertHistoryChange(history);

                var queryData = _signedDocumentRepository.Table
                    .Where(w => w.DocumentTypeId == sign.DocumentTypeId
                        && w.DocumentId == sign.DocumentId
                        && w.Signed != 1).OrderBy(o => o.SignOrder);

                var currentOrder = queryData.Where(w => w.SignOrder == sign.SignOrder && w.Id != sign.Id).FirstOrDefault();

                if (currentOrder == null)
                {
                    var nextAndRequired = queryData.Where(w => w.SignOrder > sign.SignOrder && w.Required)
                            .OrderBy(w => w.SigningRound).Select(w => new { w.SignOrder, w.SigningRound }).FirstOrDefault();

                    var nextSignRound = nextAndRequired?.SigningRound;

                    var lstNotRequired = queryData.Where(w => w.SignOrder > sign.SignOrder)
                            .OrderBy(w => w.SigningRound).ToList();

                    if (lstNotRequired.Any())
                    {
                        var itemsToSign = nextSignRound != null
                            ? lstNotRequired.Where(w => w.SigningRound < nextSignRound).ToList()
                            : lstNotRequired;

                        foreach (var item in itemsToSign)
                        {
                            item.Signed = 1;
                            _signedDocumentRepository.Update(item);
                        }
                    }

                    if (nextAndRequired != null)
                    {
                        var lst = queryData.Where(w => w.SignOrder == nextAndRequired.SignOrder).ToList();

                        foreach (var signNext in lst)
                        {
                            signNext.IsNextSign = true;
                            _signedDocumentRepository.Update(signNext);

                            if (signNext.SignerId == null)
                                continue;

                            DocumentTypeEnum documentTypeEnum = (DocumentTypeEnum)signNext.DocumentTypeId;

                            // Send Notification
                            var notiSignNext = new NotificationAddViewModel();
                            notiSignNext.Subject = GetEnumDescription(documentTypeEnum);
                            notiSignNext.SenderUserId = userId;
                            notiSignNext.SendTime = DateTime.UtcNow;
                            notiSignNext.Content = $"Đến lượt bạn ký tại vị trí {signNext.SignName}";
                            if (!string.IsNullOrEmpty(signNext.Description))
                            {
                                notiSignNext.Content += $" cho phiếu {GetEnumDescription(documentTypeEnum)} có nội dung {signNext.Description}";
                            }
                            notiSignNext.ObjectId = signNext.DocumentId.ToString();
                            notiSignNext.ObjectType = documentTypeEnum.ToString();
                            notiSignNext.ObjectCode = signNext.DocumentCode;
                            notiSignNext.ReceiverEmployeeIds = new List<Guid>()
                            {
                                (Guid)signNext.SignerId
                            };

                            var rtNoti = _erpNotificationService.AddNotification(notiSignNext, ErpNotificationType.Signature);
                            if (!rtNoti.IsOk)
                            {
                                ret.Code = rtNoti.Code;
                                ret.Description = rtNoti.Description;
                                return ret;
                            }
                        }
                    }
                }

                var signList = _signedDocumentRepository.Table
                    .Where(w => w.DocumentTypeId == sign.DocumentTypeId
                        && w.DocumentId == sign.DocumentId && w.Required).ToList();

                ret.Data = new
                {
                    TotalSigns = signList.Count(),
                    TotalSigned = signList.Where(w => w.Signed == 1).Count()
                };

            }

            _unitOfWork.Save();
            return ret;
        }

        public ReturnCode DeleteSignedDocument(SignedDocumentSearchQuery model, Guid userId)
        {
            ReturnCode ret = new ReturnCode();

            var signedDocument = _signedDocumentRepository.GetMulti(x => model.DocumentId.HasValue
                    && x.DocumentId == model.DocumentId).ToList();

            if (signedDocument.Count < 1)
            {
                ret.Code = ErrorCode.HasError;
                ret.Description = "Chứng từ không tồn tại!";
                return ret;
            }

            foreach (var sign in signedDocument)
            {
                sign.IsDeleted = true;
                _signedDocumentRepository.Update(sign);
            }

            _unitOfWork.Save();
            return ret;
        }

        public ReturnCode SignRemind(SignedDocumentSearchQuery model, Guid userId)
        {
            ReturnCode ret = new ReturnCode();
            var signList = _signedDocumentRepository.GetMulti(x => x.DocumentTypeId == model.DocumentTypeId
                    && x.DocumentId == model.DocumentId && x.Signed == 0 && x.IsNextSign).ToList();

            if (signList.Count < 1)
            {
                ret.Code = ErrorCode.HasError;
                ret.Description = "Không còn vị trí nào cần ký!";
                return ret;
            }

            var remindOn = DateTime.UtcNow;
            var signNames = new List<string>();

            DocumentTypeEnum documentTypeEnum = (DocumentTypeEnum)model.DocumentTypeId;

            // Send Notification
            foreach (var sign in signList)
            {
                if (sign.SignerId != null)
                {
                    signNames.Add(sign.SignName);


                    var noti = new NotificationAddViewModel();
                    noti.Subject = $"[Trình ký] {GetEnumDescription(documentTypeEnum)}";
                    noti.SenderUserId = userId;
                    noti.SendTime = remindOn;
                    noti.Content = $"Bạn cần ký phiếu {GetEnumDescription(documentTypeEnum)}";
                    if (!string.IsNullOrEmpty(sign.Description))
                    {
                        noti.Content += $" có nội dung: {sign.Description}";
                    }
                    noti.ObjectId = sign.DocumentId.ToString();
                    noti.ObjectType = documentTypeEnum.ToString();
                    noti.ObjectCode = sign.DocumentCode;
                    noti.Description = sign.Description;
                    noti.ReceiverEmployeeIds = new List<Guid>()
                    {
                        (Guid)sign.SignerId
                    };

                    var rtNoti = _erpNotificationService.AddNotification(noti, ErpNotificationType.Signature);
                    if (!rtNoti.IsOk)
                    {
                        ret.Code = rtNoti.Code;
                        ret.Description = rtNoti.Description;
                        return ret;
                    }
                }
            }

            // Save history
            var history = new HistoryChangeViewModel();
            history.Id = Guid.NewGuid();
            history.Action = EActionType.TrinhKy;
            history.ActionName = "Trình ký";
            history.ObjectId = model.DocumentId.ToString();
            history.ObjectType = documentTypeEnum.ToString();
            history.Content = $"Bạn đã gửi yêu cầu ký tới vị trí: {string.Join(", ", signNames)}";

            history.CreatedById = userId;
            history.CreatedTime = remindOn;
            _historyChangeService.InsertHistoryChange(history);

            return ret;
        }

        public ReturnCode SignReject(SignRejectViewModel model, Guid userId)
        {
            ReturnCode ret = new ReturnCode();

            var signReject = _signedDocumentRepository.Table.Where(x =>
                x.DocumentTypeId == model.DocumentTypeId
                && x.DocumentId == model.DocumentId
                && x.Signed != 1
                && x.SignerId == userId)
                .OrderBy(x => x.SigningRound)
                .FirstOrDefault();

            if (signReject == null)
            {
                ret.Code = ErrorCode.HasError;
                ret.Description = "Bạn không có vị trí nào cần ký!";
                return ret;
            }
            else if (signReject.IsNextSign == false)
            {
                ret.Code = ErrorCode.HasError;
                ret.Description = "Chưa đến lượt bạn ký!";
                return ret;
            }

            var rejectDate = DateTime.UtcNow;
            signReject.Signed = 2;
            signReject.IsNextSign = signReject.SigningRound == 1;
            signReject.RejectOnUtc = rejectDate;
            signReject.Reason = $"{signReject.SignerName} đã từ chối ký vào {rejectDate.ToString("dd/MM/yyyy HH:mm")}. Lý do: {model.Reason}";
            _signedDocumentRepository.Update(signReject);

            var signList = _signedDocumentRepository.GetMulti(x =>
                x.DocumentTypeId == model.DocumentTypeId
                && x.DocumentId == model.DocumentId
                && (x.Signed == 1 || x.IsNextSign) && x.Id != signReject.Id);

            foreach (var sign in signList)
            {
                sign.Signed = 0;
                sign.IsNextSign = sign.SigningRound == 1;
                sign.SignedOnUtc = null;
                _signedDocumentRepository.Update(sign);
            }

            _unitOfWork.Save();

            var lst = _signedDocumentRepository.GetMulti(x =>
                x.DocumentTypeId == model.DocumentTypeId
                && x.DocumentId == model.DocumentId
                && x.Required);

            ret.Data = new
            {
                TotalSigns = lst.Count(),
                TotalSigned = lst.Where(x => x.Signed == 1).Count()
            };

            // Save history
            var history = new HistoryChangeViewModel();
            history.Id = Guid.NewGuid();
            history.Action = EActionType.TuChoiKy;
            history.ActionName = "Từ chối ký";
            history.ObjectId = signReject.Id.ToString();
            history.ObjectType = ((DocumentTypeEnum)model.DocumentTypeId).ToString();
            history.Content = $"Bạn đã từ chối ký vị tí {signReject.SignName}";
            history.CreatedById = userId;
            history.CreatedTime = rejectDate;
            _historyChangeService.InsertHistoryChange(history);

            return ret;
        }

        public List<SignReportViewModel> GetSignByUserId(FilterSignedDocumentViewModel filter, Guid userId)
        {
            Guid employeeId = filter.EmployeeId.HasValue ? (Guid)filter.EmployeeId : userId;
            var query = from a in _signedDocumentRepository.Table
                        join b in _documentTypeRepository.Table on a.DocumentTypeId equals b.Id
                        where a.IsDeleted == false
                        select new
                        {
                            a.Id,
                            a.CreatedOnUtc,
                            a.DocumentId,
                            a.DocumentCode,
                            a.CreatedById,
                            DocumentName = b.Name,
                            a.SignName,
                            a.Description,
                            a.SignPositionId,
                            a.SignerId,
                            a.Signed,
                            a.SignedOnUtc,
                            a.IsNextSign,
                            a.SignerEmployees,
                            a.DocumentTypeId

                        };

            var groupedQuery = query.GroupBy(r => r.DocumentId).AsEnumerable()
                .Select(g => new SignReportViewModel
                {
                    Id = g.First().Id,
                    CreatedAt = g.First().CreatedOnUtc,
                    DocumentId = g.Key,
                    DocumentTypeId = g.First().DocumentTypeId,
                    DocumentCode = g.First().DocumentCode,
                    DocumentName = g.First().DocumentName,
                    // CreatedName = g.First().CreatedName,
                    CreatedById = g.First().CreatedById,
                    Description = g.First().Description,
                    MySignaturesList = g.Select(r => new Signature
                    {
                        SignPositionId = r.SignPositionId,
                        SignPositionName = r.SignName,
                        SignedStatus = r.Signed,
                        SignedOnUtc = r.SignedOnUtc,
                        IsNextSign = r.IsNextSign,
                        SignerId = r.SignerId,
                        SignerEmployees = r.SignerEmployees
                    }).ToList(),
                    AllSignaturesList = g.Select(r => new Signature
                    {
                        SignPositionId = r.SignPositionId,
                        SignPositionName = r.SignName,
                        SignedStatus = r.Signed
                    }).ToList(),

                });
            //Filter by EMPLOYEE ID OR USER ID
            groupedQuery = groupedQuery.Select(item =>
            {
                if (item.MySignaturesList != null)
                {
                    //item.MySignaturesList = item.MySignaturesList
                    //    .Where(s => s.SignerId == employeeId ||
                    //                 (s.SignerEmployees != null && s.SignerEmployees.Any(e => e == filter.EmployeeId)))
                    //    .ToList();
                    item.MySignaturesList = item.MySignaturesList.Where(s => s.SignerId == employeeId).ToList();
                }
                return item;
            }).Where(item => item.MySignaturesList != null && item.MySignaturesList.Any());

            //var x = groupedQuery.ToList();

            if (!string.IsNullOrEmpty(filter.SearchText))
            {
                var searchValue = filter.SearchText.ToLower().Trim();
                string unsign = TextHelper.ConvertToUnsign(searchValue);
                groupedQuery = groupedQuery.Where(o =>
                o.Description != null && o.Description.ToLower().Contains(searchValue) ||
                 o.DocumentName != null && o.DocumentName.ToLower().Contains(searchValue) ||
                o.DocumentCode != null && o.DocumentCode.ToLower().Contains(searchValue));
            }
            if (filter.StartDate != DateTime.MinValue && filter.StartDate != null)
            {
                groupedQuery = groupedQuery.Where(item => item.CreatedAt >= filter.StartDate);
            }
            if (filter.DocumentType != null)
            {
                groupedQuery = groupedQuery.Where(item => item.DocumentTypeId == filter.DocumentType);
            }
            if (filter.EndDate != DateTime.MinValue && filter.EndDate != null)
            {
                groupedQuery = groupedQuery.Where(item => item.CreatedAt <= filter.EndDate);
            }
            var data = groupedQuery.AsEnumerable().ToList();

            data.ForEach(item =>
            {
                var user = _userRepository.GetById(item.CreatedById);
                if (user != null)
                {
                    item.CreatedName = user.FullName;
                }

                SignOptionStatus optionStatus = SignOptionStatus.TatCa;

                if (item.AllSignaturesList != null && item.AllSignaturesList.All(t => t.SignedStatus == (int)SignedStatus.DaKy))
                {
                    optionStatus = SignOptionStatus.HoanThanh;
                }
                else if (item.MySignaturesList != null && item.MySignaturesList.Count > 0 &&
                    item.AllSignaturesList != null && item.AllSignaturesList.Any(t => t.SignedStatus == (int)SignedStatus.TuChoiKy))
                {
                    optionStatus = SignOptionStatus.BiTuChoi;
                }
                else if (item.MySignaturesList != null && item.MySignaturesList.All(t => t.SignedStatus == (int)SignedStatus.DaKy))
                {
                    optionStatus = SignOptionStatus.BanDaKy;
                }
                else if (item.MySignaturesList != null && item.MySignaturesList.Any(t => t.IsNextSign == true))
                {
                    optionStatus = SignOptionStatus.ChoBanKy;
                }
                else if (item.MySignaturesList != null && item.MySignaturesList.Any(t => t.IsNextSign == false && t.SignedStatus != (int)SignedStatus.DaKy))
                {
                    optionStatus = SignOptionStatus.ChoKy;
                }

                item.SignStatus = optionStatus;
                item.SignStatusName = optionStatus.GetDescription();
            });
            if (filter.SignStatus != SignOptionStatus.TatCa)
            {
                data = data.Where(t => t.SignStatus == filter.SignStatus).ToList();
            }
            data = data.OrderByDescending(item => item.CreatedAt).ToList();
            return data;
        }
        public PagingDataModel<SignReportViewModel> SearchSignDocument(PaginationQueryModel<SearchSignDocumentViewModel> query, Guid userId)
        {
            FilterSignedDocumentViewModel model = new FilterSignedDocumentViewModel();
            model.EmployeeId = userId;
            model.SignStatus = query.Query.Type;
            model.SearchText = query.Query.SearchText;
            model.DocumentType = query.Query.DocumentId;
            var data = GetSignByUserId(model, userId).OrderByDescending(x => x.CreatedAt);

            PagingDataModel<SignReportViewModel> ret = new PagingDataModel<SignReportViewModel>(data, query.Skip, query.Take);

            return ret;
        }

        public ReturnCode CreateSignPosition(SignPositionViewModel model, Guid userId)
        {
            ReturnCode ret = new ReturnCode();

            var existed = _signPositionRepository.GetSingleByCondition(x => x.Name == model.Name);
            if (existed != null)
            {
                ret.Code = ErrorCode.Common_Existed;
                ret.Description = "Vị trí ký đã tồn tại!";
                return ret;
            }

            var signPosition = new SignPosition();
            var id = _signPositionRepository.Table.Max(x => x.Id);
            signPosition.Id = id + 1;
            signPosition.Description = model.Description;
            signPosition.DisplayOrder = model.DisplayOrder;
            signPosition.InActive = model.InActive ?? false;
            signPosition.Name = model.Name;
            signPosition.CreatedById = userId;
            signPosition.CreatedOnUtc = DateTime.UtcNow;

            _signPositionRepository.Insert(signPosition);

            _unitOfWork.Save();
            ret.Data = signPosition;

            return ret;
        }

        public ReturnCode UpdateSignPosition(SignPositionViewModel model, Guid userId)
        {
            ReturnCode ret = new ReturnCode();

            var nameExisted = _signPositionRepository.GetSingleByCondition(x => x.Name == model.Name && x.Id != model.Id);
            if (nameExisted != null)
            {
                ret.Code = ErrorCode.Common_Existed;
                ret.Description = "Vị trí ký đã tồn tại!";
                return ret;
            }

            var updatePosition = _signPositionRepository.GetById(model.Id ?? 1);
            if (updatePosition == null)
            {
                ret.Code = ErrorCode.Common_NotFound;
                ret.Description = "Vị trí ký không tồn tại!";
                return ret;
            }

            updatePosition.Name = model.Name;
            updatePosition.Description = model.Description;
            updatePosition.DisplayOrder = model.DisplayOrder;
            updatePosition.InActive = model.InActive ?? false;

            _signPositionRepository.Update(updatePosition);

            _unitOfWork.Save();
            return ret;
        }

        public ReturnCode DeleteSignPosition(int id)
        {
            ReturnCode ret = new ReturnCode();
            var deletePosition = _signPositionRepository.GetById(id);
            if (deletePosition == null)
            {
                ret.Code = ErrorCode.Common_NotFound;
                ret.Description = "Vị trí ký không tồn tại!";
                return ret;
            }

            deletePosition.IsDeleted = true;

            _signPositionRepository.Update(deletePosition);
            _unitOfWork.Save();

            return ret;
        }

        private string GetEnumDescription(Enum value)
        {
            FieldInfo fi = value.GetType().GetField(value.ToString());
            DescriptionAttribute[] attributes = (DescriptionAttribute[])fi.GetCustomAttributes(typeof(DescriptionAttribute), false);

            if (attributes != null && attributes.Length > 0)
                return attributes[0].Description;
            else
                return value.ToString();
        }

    }
}
```