package com.nuclear\_risk.live.underwriting.service.impl;

import java.io.Console;

import java.io.File;

import java.io.FileOutputStream;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.time.LocalDate;

import java.time.Month;

import java.time.temporal.TemporalAdjusters;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Calendar;

import java.util.Collections;

import java.util.Comparator;

import java.util.Date;

import java.util.HashMap;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import java.util.Objects;

import java.util.Optional;

import java.util.Set;

import java.util.concurrent.TimeUnit;

import javax.persistence.EntityManager;

import javax.persistence.criteria.CriteriaBuilder;

import javax.persistence.criteria.CriteriaQuery;

import javax.persistence.criteria.Predicate;

import javax.persistence.criteria.Root;

import javax.transaction.Transactional;

import org.apache.commons.lang3.StringUtils;

import org.dozer.Mapper;

import org.hibernate.JDBCException;

import org.hibernate.exception.JDBCConnectionException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.cache.annotation.CacheEvict;

import org.springframework.cache.annotation.Cacheable;

import org.springframework.cache.annotation.Caching;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.data.domain.Sort.Direction;

import org.springframework.data.jpa.domain.Specification;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.stereotype.Service;

import org.terasoluna.plus.common.model.ContentType;

import com.nuclear\_risk.live.cache.util.CacheConstants;

import com.nuclear\_risk.live.claim.model.ClaimShortDTO;

import com.nuclear\_risk.live.claim.service.ClaimService;

import com.nuclear\_risk.live.dashboard.model.AlertDTO;

import com.nuclear\_risk.live.dashboard.model.TaskDTO;

import com.nuclear\_risk.live.dashboard.service.AlertService;

import com.nuclear\_risk.live.dashboard.service.HistoryService;

import com.nuclear\_risk.live.dashboard.service.TaskService;

import com.nuclear\_risk.live.dashboard.util.EnumUserRoles;

import com.nuclear\_risk.live.domain.entity.CountryReference;

import com.nuclear\_risk.live.domain.entity.CurrencyReference;

import com.nuclear\_risk.live.domain.entity.DocumentReference;

import com.nuclear\_risk.live.domain.entity.DocumentReferenceToView;

import com.nuclear\_risk.live.domain.entity.FavouriteReference;

import com.nuclear\_risk.live.domain.entity.LocationReference;

import com.nuclear\_risk.live.domain.entity.NriParametersEntity;

import com.nuclear\_risk.live.domain.entity.OrganisationReference;

import com.nuclear\_risk.live.domain.entity.OrganisationShortReference;

import com.nuclear\_risk.live.domain.entity.PolicyPremiumTaxesReference;

import com.nuclear\_risk.live.domain.entity.PolicyReference;

import com.nuclear\_risk.live.domain.entity.PolicyReferenceForFinance;

import com.nuclear\_risk.live.domain.entity.PolicyReferenceForTasks;

import com.nuclear\_risk.live.domain.entity.PolicySharesShortReference;

import com.nuclear\_risk.live.domain.entity.PremiumReceivedReference;

import com.nuclear\_risk.live.domain.entity.StationReference;

import com.nuclear\_risk.live.domain.entity.TaxTypeReference;

import com.nuclear\_risk.live.domain.entity.UserReference;

import com.nuclear\_risk.live.domain.entity.YearOfAccountReference;

import com.nuclear\_risk.live.domain.model.CountryReferenceDTO;

import com.nuclear\_risk.live.domain.model.CurrencyReferenceDTO;

import com.nuclear\_risk.live.domain.model.FavouriteReferenceDTO;

import com.nuclear\_risk.live.domain.model.InsuranceStructureReferenceDTO;

import com.nuclear\_risk.live.domain.model.LocationReferenceDTO;

import com.nuclear\_risk.live.domain.model.OrganisationPolicyReferenceDTO;

import com.nuclear\_risk.live.domain.model.OrganisationReferenceDTO;

import com.nuclear\_risk.live.domain.model.PolicyDatesWithClaimsReferenceDTO;

import com.nuclear\_risk.live.domain.model.PolicyMainInfoForClaim;

import com.nuclear\_risk.live.domain.model.PolicyReferenceDTO;

import com.nuclear\_risk.live.domain.model.PolicyStatusReferenceDTO;

import com.nuclear\_risk.live.domain.model.StationReferenceDTO;

import com.nuclear\_risk.live.domain.model.TaxTypeReferenceDTO;

import com.nuclear\_risk.live.domain.repository.ClaimReferenceRepository;

import com.nuclear\_risk.live.domain.repository.ContactDetailsReferenceRepository;

import com.nuclear\_risk.live.domain.repository.CurrencyReferenceRepository;

import com.nuclear\_risk.live.domain.repository.DocumentTypeReferenceRepository;

import com.nuclear\_risk.live.domain.repository.FavouriteReferenceRepository;

import com.nuclear\_risk.live.domain.repository.LocationReferenceRepository;

import com.nuclear\_risk.live.domain.repository.NRIParametersRepository;

import com.nuclear\_risk.live.domain.repository.OrganisationReferenceRepository;

import com.nuclear\_risk.live.domain.repository.OrganisationShortReferenceRepository;

import com.nuclear\_risk.live.domain.repository.PolicyPremiumTaxesReferenceRepository;

import com.nuclear\_risk.live.domain.repository.PolicyReferenceForFinanceRepository;

import com.nuclear\_risk.live.domain.repository.PolicyReferenceForTasksRepository;

import com.nuclear\_risk.live.domain.repository.PolicySharesShortReferenceRepository;

import com.nuclear\_risk.live.domain.repository.StationReferenceRepository;

import com.nuclear\_risk.live.domain.repository.TaxTypeReferenceRepository;

import com.nuclear\_risk.live.domain.repository.UserReferenceRepository;

import com.nuclear\_risk.live.domain.repository.YearOfAccountReferenceRepository;

import com.nuclear\_risk.live.domain.util.EnumCoverage;

import com.nuclear\_risk.live.domain.util.EnumDocumentType;

import com.nuclear\_risk.live.domain.util.EnumInsuranceStructure;

import com.nuclear\_risk.live.domain.util.EnumInsurerRole;

import com.nuclear\_risk.live.domain.util.EnumOrganisationType;

import com.nuclear\_risk.live.domain.util.EnumPolicyStatus;

import com.nuclear\_risk.live.domain.util.EnumRagColour;

import com.nuclear\_risk.live.domain.util.EnumTaxTypeCalculationOperation;

import com.nuclear\_risk.live.domain.util.Fetcher;

import com.nuclear\_risk.live.domain.util.PolicyConstants;

import com.nuclear\_risk.live.ecm.service.DocumentService;

import com.nuclear\_risk.live.finance.model.JournalInfoDTO;

import com.nuclear\_risk.live.finance.model.PolicyPremiumTaxesDTO;

import com.nuclear\_risk.live.finance.model.PremiumReceivedDTO;

import com.nuclear\_risk.live.finance.model.TaxTypeDTO;

import com.nuclear\_risk.live.finance.service.JournalService;

import com.nuclear\_risk.live.finance.service.PolicyPremiumTaxesService;

import com.nuclear\_risk.live.finance.service.TaxTypeService;

import com.nuclear\_risk.live.shared\_util.util.CastUtil;

import com.nuclear\_risk.live.shared\_util.util.DateUtils;

import com.nuclear\_risk.live.shared\_util.util.FilterUtil;

import com.nuclear\_risk.live.shared\_util.util.ListMapper;

import com.nuclear\_risk.live.shared\_util.util.Utils;

import com.nuclear\_risk.live.underwriting.entity.AddOn;

import com.nuclear\_risk.live.underwriting.entity.Coverage;

import com.nuclear\_risk.live.underwriting.entity.CoverageLimit;

import com.nuclear\_risk.live.underwriting.entity.Deductible;

import com.nuclear\_risk.live.underwriting.entity.Deduction;

import com.nuclear\_risk.live.underwriting.entity.DeductionName;

import com.nuclear\_risk.live.underwriting.entity.DeductionType;

import com.nuclear\_risk.live.underwriting.entity.EntryType;

import com.nuclear\_risk.live.underwriting.entity.InsuranceStructure;

import com.nuclear\_risk.live.underwriting.entity.Policy;

import com.nuclear\_risk.live.underwriting.entity.PolicyEmailTemplates;

import com.nuclear\_risk.live.underwriting.entity.PolicyForOrg;

import com.nuclear\_risk.live.underwriting.entity.PolicyForSave;

import com.nuclear\_risk.live.underwriting.entity.PolicyForStatusEvolution;

import com.nuclear\_risk.live.underwriting.entity.PolicyForView;

import com.nuclear\_risk.live.underwriting.entity.PolicyIncome;

import com.nuclear\_risk.live.underwriting.entity.PolicyLocationView;

import com.nuclear\_risk.live.underwriting.entity.PolicyShares;

import com.nuclear\_risk.live.underwriting.entity.PolicyStatus;

import com.nuclear\_risk.live.underwriting.entity.PolicyStatusEvolution;

import com.nuclear\_risk.live.underwriting.entity.PolicyYear;

import com.nuclear\_risk.live.underwriting.entity.RAGColour;

import com.nuclear\_risk.live.underwriting.entity.RelatedPolicyForView;

import com.nuclear\_risk.live.underwriting.entity.Sublimit;

import com.nuclear\_risk.live.underwriting.entity.SublimitName;

import com.nuclear\_risk.live.underwriting.entity.UnderwritingTask;

import com.nuclear\_risk.live.underwriting.entity.UnderwritingTaskConfiguration;

import com.nuclear\_risk.live.underwriting.model.AddOnDTO;

import com.nuclear\_risk.live.underwriting.model.BrokerageTypeDTO;

import com.nuclear\_risk.live.underwriting.model.ComboInfoDTO;

import com.nuclear\_risk.live.underwriting.model.CoverageDTO;

import com.nuclear\_risk.live.underwriting.model.CoverageLimitDTO;

import com.nuclear\_risk.live.underwriting.model.DeductibleDTO;

import com.nuclear\_risk.live.underwriting.model.DeductibleTypeDTO;

import com.nuclear\_risk.live.underwriting.model.DeductionCedingTypeDTO;

import com.nuclear\_risk.live.underwriting.model.DeductionDTO;

import com.nuclear\_risk.live.underwriting.model.DeductionNameDTO;

import com.nuclear\_risk.live.underwriting.model.DeductionTypeDTO;

import com.nuclear\_risk.live.underwriting.model.EntryTypeDTO;

import com.nuclear\_risk.live.underwriting.model.InsuranceStructureDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyDatesWithClaimsDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyEmailTemplateInfo;

import com.nuclear\_risk.live.underwriting.model.PolicyEmailTemplatesDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyFilterResponseDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyIncomeDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyIndicatorsChartDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyLineChartDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyMainInfoDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyNumbersAndPolicyIdDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyPieChartDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyRenewalHistoryItemDTO;

import com.nuclear\_risk.live.underwriting.model.PolicySearchDTO;

import com.nuclear\_risk.live.underwriting.model.PolicySearchResultsDTO;

import com.nuclear\_risk.live.underwriting.model.PolicySelectOptionInfo;

import com.nuclear\_risk.live.underwriting.model.PolicySharesDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyShortDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyStatusDTO;

import com.nuclear\_risk.live.underwriting.model.PolicyTimelineDTO;

import com.nuclear\_risk.live.underwriting.model.RAGColourDTO;

import com.nuclear\_risk.live.underwriting.model.SublimitDTO;

import com.nuclear\_risk.live.underwriting.model.UnderwritingProcessDTO;

import com.nuclear\_risk.live.underwriting.model.UnderwritingTaskConfigurationDTO;

import com.nuclear\_risk.live.underwriting.model.SublimitNameDTO;

import com.nuclear\_risk.live.underwriting.model.TaxTypeCalculationDTO;

import com.nuclear\_risk.live.underwriting.model.UnderwritingTaskDTO;

import com.nuclear\_risk.live.underwriting.repository.AddOnRepository;

import com.nuclear\_risk.live.underwriting.repository.CoverageLimitRepository;

import com.nuclear\_risk.live.underwriting.repository.CoverageRepository;

import com.nuclear\_risk.live.underwriting.repository.DeductibleRepository;

import com.nuclear\_risk.live.underwriting.repository.DeductionNameRepository;

import com.nuclear\_risk.live.underwriting.repository.DeductionRepository;

import com.nuclear\_risk.live.underwriting.repository.DeductionTypeRepository;

import com.nuclear\_risk.live.underwriting.repository.EntryTypeRepository;

import com.nuclear\_risk.live.underwriting.repository.InsuranceStructureRepository;

import com.nuclear\_risk.live.underwriting.repository.PolicyEmailTemplatesRepository;

import com.nuclear\_risk.live.underwriting.repository.PolicyForOrgRepository;

import com.nuclear\_risk.live.underwriting.repository.PolicyForSaveRepository;

import com.nuclear\_risk.live.underwriting.repository.PolicyForStatusEvolutionRepository;

import com.nuclear\_risk.live.underwriting.repository.PolicyForViewRepository;

import com.nuclear\_risk.live.underwriting.repository.PolicyIncomeRepository;

import com.nuclear\_risk.live.underwriting.repository.PolicyLocationViewRepository;

import com.nuclear\_risk.live.underwriting.repository.PolicyRepository;

import com.nuclear\_risk.live.underwriting.repository.PolicyStatusEvolutionRepository;

import com.nuclear\_risk.live.underwriting.repository.PolicyStatusRepository;

import com.nuclear\_risk.live.underwriting.repository.PolicyYearRepository;

import com.nuclear\_risk.live.underwriting.repository.RAGColourRepository;

import com.nuclear\_risk.live.underwriting.repository.RelatedPolicyForViewRepository;

import com.nuclear\_risk.live.underwriting.repository.SublimitNameRepository;

import com.nuclear\_risk.live.underwriting.repository.SublimitRepository;

import com.nuclear\_risk.live.underwriting.repository.UnderwritingProcessRepository;

import com.nuclear\_risk.live.underwriting.repository.UnderwritingTaskConfigurationRepository;

import com.nuclear\_risk.live.underwriting.service.BrokerageTypeService;

import com.nuclear\_risk.live.underwriting.service.DeductibleService;

import com.nuclear\_risk.live.underwriting.service.DeductionCedingTypeService;

import com.nuclear\_risk.live.underwriting.service.DeductionService;

import com.nuclear\_risk.live.underwriting.service.PolicyDocumentService;

import com.nuclear\_risk.live.underwriting.service.PolicyIncomeService;

import com.nuclear\_risk.live.underwriting.service.PolicyService;

import com.nuclear\_risk.live.underwriting.service.PolicySharesService;

import com.nuclear\_risk.live.underwriting.service.SublimitService;

import com.nuclear\_risk.live.underwriting.service.TaxTypeCalculationService;

import com.nuclear\_risk.live.underwriting.service.UnderwritingProcessService;

import com.nuclear\_risk.live.underwriting.util.EnumDeductionName;

/\*\*

\* The Class PolicyServiceImpl.

\*/

@Service

@Transactional

public class PolicyServiceImpl implements PolicyService {

private static final Logger LOGGER = LoggerFactory.getLogger(PolicyServiceImpl.class);

private static final String DEFAULT\_UNIQUE\_ID = "0001";

private static final String DEFAULT\_QUOTE\_NR = "01";

private static final String DEFAULT\_POLICY\_VERSION = "00";

private static final String INS\_STRUCTURE\_REINSURANCE\_INWARDS = "Reinsurance inwards";

private static final String RAG\_COLOUR\_GREEN = "Green";

private static final String DOCUMENT\_TYPE\_SSF = "SSF";

private static final String DOCUMENT\_TYPE\_COVER\_BOUND\_EMAIL = "Cover bound email";

private static final String DOCUMENT\_TYPE\_COVER\_NOTE = "Cover note received";

private static final String DOCUMENT\_TYPE\_REINSURANCE\_OFFER\_RESPONSE\_EMAILS = "Reinsurance offer response email";

private static final String DOCUMENT\_TYPE\_COVER\_NOTE\_RECEIVED = "Cover note received";

private static final String FAVOURITES = "Favourites";

private static final String FAV\_POLICIES = "Policy";

private static final String ENDORSEMENT = "ENDORSEMENT";

private static final String RENEWAL = "RENEWAL";

private static final String NEWBUSINESS = "New business";

private static final String QUOTE = "QUOTE";

private static final String POLICY = "POLICY";

private static final String ENQUIRY = "ENQUIRY";

private static final String SCHEDULE = "SCHEDULE";

private static final String COVERNOTE = "COVERNOTE";

private static final String DEBIT\_CREDIT\_NOTE = "DEBIT\_CREDIT\_NOTE";

private static final String DOCUMENTATION\_TO\_ISSUE = "DOCUMENTATION\_TO\_ISSUE";

private static final String STANDARD\_SUBMISSION\_FORM = "STANDARD\_SUBMISSION\_FORM";

private static final String[] CREATE\_POLICY\_TYPES = { ENQUIRY, POLICY, QUOTE, RENEWAL, ENDORSEMENT };

private static final String POLICY\_CREATED = "POLICY\_CREATED";

private static final String POLICY\_UPDATED = "POLICY\_UPDATED";

private static final String POLICY\_STATUS\_CHANGED = "POLICY\_STATUS\_CHANGED";

private static final String POLICY\_RENEWED = "POLICY\_RENEWED";

private static final String POLICY\_ENDORSED = "POLICY\_ENDORSED";

private static final String POLICY\_CANCELLED = "POLICY\_CANCELLED";

private final String DECLINED = "DECLINED";

private final String NON\_TAKEN\_UP = "NON\_TAKEN\_UP";

private final String POLICY\_IN\_FORCE = "policy in force";

@Autowired

private Mapper mapper;

@Autowired

private PolicyRepository repository;

@Autowired

private PolicyForSaveRepository repositoryForSave;

@Autowired

private PolicyForViewRepository repositoryForView;

@Autowired

private PolicyForOrgRepository policyForOrgRepository;

@Autowired

private PolicyReferenceForTasksRepository repositoryForTasks;

@Autowired

private RelatedPolicyForViewRepository relatedPolicyForViewRepository;

@Autowired

private PolicyReferenceForFinanceRepository policyForFinanceRepository;

@Autowired

private CoverageRepository coverageRepository;

@Autowired

private UserReferenceRepository userRepository;

@Autowired

private SublimitNameRepository subNameRepository;

@Autowired

private CurrencyReferenceRepository currencyRepository;

@Autowired

private EntryTypeRepository entryTypeRepository;

@Autowired

private InsuranceStructureRepository insuranceStructureRepository;

@Autowired

private OrganisationReferenceRepository organisationReferenceRepository;

@Autowired

private LocationReferenceRepository locationReferenceRepository;

@Autowired

private StationReferenceRepository stationReferenceRepository;

@Autowired

private DeductionTypeRepository deductionTypeRepository;

@Autowired

private DeductionNameRepository deductionNameRepository;

@Autowired

private PolicyYearRepository policyYearRepository;

@Autowired

private AddOnRepository addOnRepository;

@Autowired

private ContactDetailsReferenceRepository contactDetailsReferenceRepository;

@Autowired

private PolicyLocationViewRepository policyLocationViewRepository;

@Autowired

private PolicySharesService policyShareService;

@Autowired

private SublimitService sublimitService;

@Autowired

private DeductibleService deductibleService;

@Autowired

private PolicyPremiumTaxesService policyPremiumTaxesService;

@Autowired

private DeductionService deductionService;

@Autowired

private ClaimService claimService;

@Autowired

private HistoryService historyService;

@Autowired

private PolicyStatusRepository policyStatusRepository;

@Autowired

private TaxTypeReferenceRepository taxTypeReferenceRepository;

@Autowired

private RAGColourRepository rAGColourRepository;

@Autowired

private ClaimReferenceRepository claimReferenceRepository;

@Autowired

private UserReferenceRepository userReference;

@Autowired

private FavouriteReferenceRepository favouriteRepository;

@Autowired

private NRIParametersRepository parametersRepository;

@Autowired

private PolicyStatusEvolutionRepository policyStatusEvolutionRepository;

@Autowired

private SublimitRepository sublimitRepository;

@Autowired

private DeductibleRepository deductibleRepository;

@Autowired

private PolicyPremiumTaxesReferenceRepository policyPremiumTaxesReferenceRepository;

@Autowired

private DeductionRepository deductionRepository;

@Autowired

private PolicyEmailTemplatesRepository policyEmailTemplatesRepository;

@Autowired

private UnderwritingTaskConfigurationRepository uwTaskConfigurationRepository;

@Autowired

private TaskService taskService;

@Autowired

private AlertService alertService;

@Autowired

private JournalService journalService;

@Autowired

private DocumentService documentService;

@Autowired

private PolicySharesShortReferenceRepository policySharesShortReferenceRepository;

@Autowired

private YearOfAccountReferenceRepository yearOfAccountRepository;

@Autowired

private OrganisationShortReferenceRepository organisationShortReferenceRepository;

@Autowired

private CoverageLimitRepository coverageLimitRepository;

@Autowired

EntityManager em;

@Autowired

private PolicyIncomeService policyIncomeService;

@Autowired

private PolicyIncomeRepository policyIncomeRepository;

@Autowired

private PolicyDocumentService policyDocumentService;

@Autowired

private DocumentTypeReferenceRepository documentTypeReferenceRepository;

@Autowired

private PolicyForStatusEvolutionRepository policyForStatusEvolutionRepository;

@Autowired

private BrokerageTypeService brokerageTypeService;

@Autowired

private TaxTypeCalculationService taxTypeCalculationService;

@Autowired

private DeductionCedingTypeService deductionCedingTypeService;

@Autowired

private TaxTypeService taxTypeService;

@Autowired

private UnderwritingProcessService underwritingProcessService;

@Autowired

private UnderwritingProcessRepository underwritingProcessRepository;

@Override

public List<PolicyDTO> getAllPolicies() {

List<Policy> policiesAll = this.repository.findAll();

return ListMapper.mapList(policiesAll, mapper, PolicyDTO.class);

}

@Override

public List<PolicyDTO> getAllPoliciesPageable(Integer offset, Integer page) {

if (offset == 0)

offset = 30;

Pageable pageable = new PageRequest(page, offset, Sort.Direction.ASC, "policyNumber");

Page<Policy> policyShort = this.repository.findAll(pageable);

List<PolicyDTO> result = ListMapper.mapList(policyShort.getContent(), mapper, PolicyDTO.class);

return result;

}

@Override

public List<PolicyDTO> getAllPoliciesThin() {

List<PolicyDTO> result = null;

List<Policy> policiesAll = this.repository.findAll();

if (policiesAll != null && !policiesAll.isEmpty()) {

result = new ArrayList<>();

for (Policy policy : policiesAll) {

if (policy != null) {

result.add(convertPolicyEntityToDtoThin(policy));

}

}

}

return result;

}

@Override

public List<PolicyDTO> getAllPoliciesForSelectOption() {

List<PolicyDTO> result = null;

List<PolicySelectOptionInfo> policiesAll = repository.findAllPoliciesWithBasicSpecifiedQueryInfo();

if (policiesAll != null && !policiesAll.isEmpty()) {

result = new ArrayList<>();

for (PolicySelectOptionInfo item : policiesAll) {

if (item != null && item.getId() != null) {

PolicyDTO copyItem = new PolicyDTO(item.getId(), item.getPolicyNumber(), item.getStartDate(),

item.getEndDate(), item.getYearOfAccount());

if (item.getCoverageID() != null) {

copyItem.setCoverageID(new CoverageDTO(item.getCoverageID(), item.getCoverageName()));

}

if (item.getInsuredId() != null) {

CountryReferenceDTO country = null;

if (item.getInsuredCountryId() != null) {

country = new CountryReferenceDTO(item.getInsuredCountryId(), item.getInsuredCountryName());

copyItem.setInsured(new OrganisationReferenceDTO(item.getInsuredId(),

item.getInsuredfullName(), country));

} else {

copyItem.setInsured(

new OrganisationReferenceDTO(item.getInsuredId(), item.getInsuredfullName()));

}

}

if (item.getnRINetMaximumExposureGBP() != null) {

copyItem.setnRINetMaximumExposureGBP(item.getnRINetMaximumExposureGBP());

}

result.add(copyItem);

}

}

}

result.sort(Comparator.comparing(PolicyDTO::getPolicyNumber).reversed());

return result;

}

@Override

public PolicyDTO getPolicieWithBasicInfoById(Long id) {

PolicyDTO result = null;

PolicySelectOptionInfo policiesAll = repository.getPolicieWithBasicInfoById(id);

if (policiesAll != null) {

if (policiesAll != null && policiesAll.getId() != null) {

PolicyDTO copyItem = new PolicyDTO(policiesAll.getId(), policiesAll.getPolicyNumber(),

policiesAll.getStartDate(), policiesAll.getEndDate(), policiesAll.getYearOfAccount());

if (policiesAll.getCoverageID() != null) {

copyItem.setCoverageID(new CoverageDTO(policiesAll.getCoverageID(), policiesAll.getCoverageName()));

}

if (policiesAll.getInsuredId() != null) {

CountryReferenceDTO country = null;

if (policiesAll.getInsuredCountryId() != null) {

country = new CountryReferenceDTO(policiesAll.getInsuredCountryId(),

policiesAll.getInsuredCountryName());

copyItem.setInsured(new OrganisationReferenceDTO(policiesAll.getInsuredId(),

policiesAll.getInsuredfullName(), country));

} else {

copyItem.setInsured(new OrganisationReferenceDTO(policiesAll.getInsuredId(),

policiesAll.getInsuredfullName()));

}

}

if (policiesAll.getnRINetMaximumExposureGBP() != null) {

copyItem.setnRINetMaximumExposureGBP(policiesAll.getnRINetMaximumExposureGBP());

}

result = copyItem;

}

}

return result;

}

@Override

@Cacheable(value = CacheConstants.Values.POLICY\_NUMBER\_ALL)

public List<PolicyDTO> getPolicyNumbersByUserInput(String Input) {

List<PolicyDTO> result = repository.getPolicyNumbersByUserInput(Input);

result.sort(Comparator.comparing(PolicyDTO::getPolicyNumber).reversed());

return result;

}

@Override

public List<PolicyDTO> getAllPoliciesForClaimPage() {

List<PolicyDTO> result = new ArrayList<>();

List<PolicyMainInfoForClaim> policiesShortInfo = repository.getPolicyShortInfoForClaim();

if (policiesShortInfo != null && !policiesShortInfo.isEmpty()) {

for (PolicyMainInfoForClaim policy : policiesShortInfo) {

if (policy != null) {

PolicyDTO policyInfo = mapper.map(policy, PolicyDTO.class);

// Get the policy shares for the policy

List<PolicySharesShortReference> pShareInfo = policySharesShortReferenceRepository

.getPolicySharesByPolicyId(policy.getId());

if (pShareInfo != null && !pShareInfo.isEmpty()) {

policyInfo.setPolicyShareID(ListMapper.mapList(pShareInfo, mapper, PolicySharesDTO.class));

}

// Get the currency for the policy

CurrencyReference currencyInfo = repository.getPolicyCurrencyInfoById(policy.getId());

if (currencyInfo != null) {

policyInfo.setCurrencyID(mapper.map(currencyInfo, CurrencyReferenceDTO.class));

}

// Get the organization for the policy

if (policy.getInsuredId() != null) {

OrganisationShortReference insuredInfo = organisationShortReferenceRepository

.getOrganizationMainInfoById((policy.getId()));

if (insuredInfo != null) {

policyInfo.setInsured(mapper.map(insuredInfo, OrganisationReferenceDTO.class));

}

}

// locationId

List<Long> locIds = policyLocationViewRepository.getLocationsIdsByPolicyId(policy.getId());

if (locIds != null && !locIds.isEmpty() && locIds.get(0) != null) {

List<LocationReference> locations = locationReferenceRepository.getLocationsByTheirIds(locIds);

policyInfo.setLocationId(ListMapper.mapList(locations, mapper, LocationReferenceDTO.class));

}

// Deductible options

List<Deductible> deduct = deductibleRepository.getDeductiblesByPolicyId(policy.getId());

if (deduct != null && !deduct.isEmpty()) {

policyInfo.setDeductibleID(ListMapper.mapList(deduct, mapper, DeductibleDTO.class));

}

result.add(policyInfo);

}

}

}

return result;

}

@Override

@Cacheable(value = CacheConstants.Values.POLICY\_NUMBER\_ALL)

public List<String> getAllPolicyNumbers() {

return this.repository.getAllPolicyNumbers();

}

@Override

public List<PolicyNumbersAndPolicyIdDTO> getAllPolicyNumbersAndPolicyId() {

return this.repository.getAllPolicyNumbersAndPolicyId();

}

@Override

public List<Integer> getAllYears() {

return this.repository.getAllYears();

}

@Override

public List<String> getAllCedingPoolReferences() {

return this.repository.getAllCedingPoolReferences();

}

@Override

public List<String> getAllPolicyNumbersForClaims() {

return this.repository.getAllPolicyNumbersForClaims();

};

@Override

public List<String> getAllPolicyNumbersWithOpenYear() {

return this.repository.getAllPolicyNumbersWithOpenYear();

}

@Override

public List<PolicyDTO> getFilteredPolicies(HashMap<String, Object> filters) {

List<Policy> vals = fetchAndFilterPolicies(filters);

List<PolicyDTO> res = null;

if (vals != null && !vals.isEmpty()) {

res = ListMapper.mapList(vals, mapper, PolicyDTO.class);

}

return res;

}

private Pageable prepareSortingPageable(HashMap<String, Object> filters, Integer offset, Integer page) {

Pageable pageable;

//sort type

Direction sort;

if (filters.get("sort") != null && filters.get("sort").toString() != "") {

if (filters.get("sort").toString().toUpperCase().equals("DESC")) {

sort = Sort.Direction.DESC;

} else {

sort = Sort.Direction.ASC;

}

} else {

sort = Sort.Direction.ASC;

}

//Preparing the ordering of the result

if (filters.get("orderBy") != null && filters.get("orderBy").toString() != "") {

switch (filters.get("orderBy").toString()) {

case "policyNumber":

pageable = new PageRequest(page, offset, sort, "policyNumber");

break;

case "insured":

pageable = new PageRequest(page, offset, sort, "insured.nRIalias");

break;

case "country":

pageable = new PageRequest(page, offset, sort, "insured.countryId.name");

break;

case "riOut":

pageable = new PageRequest(page, offset, sort, "reinsuranceOut");

break;

case "cedingPoolRef":

pageable = new PageRequest(page, offset, sort, "cedingPoolReference");

break;

default:

pageable = new PageRequest(page, offset, Sort.Direction.ASC, "coverageID.typeOfRisk", "uniqueIdentifier", "yearOfAccount");

break;

}

} else {

pageable = new PageRequest(page, offset, Sort.Direction.ASC, "coverageID.typeOfRisk", "uniqueIdentifier", "yearOfAccount");

}

return pageable;

}

@Override

public HashMap<String, Object> getFilteredPageForFinance(HashMap<String, Object> filters) {

HashMap<String, Object> result = new HashMap<String, Object>();

List<PolicyReferenceForFinance> vals;

Long total = null;

Boolean count = (Boolean) filters.get("count");

Integer pageNumber = (Integer) filters.get("pageNumber");

Integer pageSize = (Integer) filters.get("pageSize");

Pageable pageable = null;

// Order the results by ascending Coverage Type (database column COVERAGE.TYPE\_OF\_RISK), ascending 4 digit policy number

// (database column POLICY.UNIQUE\_IDENTIFIER) and ascending Year of account (database column POLICY.YEAR\_OF\_ACCOUNT)

Sort sort = new Sort(Sort.Direction.ASC, "coverageID.typeOfRisk", "uniqueIdentifier", "yearOfAccount");

if (pageNumber != null && pageSize != null) {

pageable = prepareSortingPageable(filters, pageSize, pageNumber - 1);

}

if (filters != null && !filters.isEmpty() && filters.keySet() != null && filters.values() != null) {

Specification<PolicyReferenceForFinance> specification = new Specification<PolicyReferenceForFinance>() {

public Predicate toPredicate(Root<PolicyReferenceForFinance> root, CriteriaQuery<?> query,

CriteriaBuilder builder) {

List<Predicate> predicates = new ArrayList<Predicate>();

if (filters.get("policyNumber") != null && filters.get("policyNumber").toString() != "") {

predicates

.add(builder

.or(builder

.and(builder

.isNotNull(

root.<String>get("quoteNumber")),

builder.like(

builder.concat(builder

.concat(root.<String>get("policyNumber"), "/"),

root.<String>get("quoteNumber")),

"%" + filters.get("policyNumber").toString() + "%")),

builder.like(root.<String>get("policyNumber"),

"%" + filters.get("policyNumber").toString() + "%")));

}

if (filters.get("yearOfAccount") != null && filters.get("yearOfAccount").toString() != "[]") {

predicates.add(

root.<Integer>get("yearOfAccount").in((List<Integer>) filters.get("yearOfAccount")));

}

if (filters.get("countryName") != null && filters.get("countryName").toString() != "") {

predicates.add(builder.like(root.<OrganisationReference>get("insured")

.<CountryReference>get("countryId").<String>get("name"),

(String) filters.get("countryName")));

}

if (filters.get("enquiry") != null && filters.get("enquiry").toString() != "" && Boolean.valueOf( filters.get("enquiry").toString()) == Boolean.TRUE) {

predicates.add(builder.equal(root.<Boolean>get("enquiry"),

Boolean.valueOf(filters.get("enquiry").toString())));

}

if (filters.get("broker") != null && filters.get("broker").toString() != "") {

predicates.add(builder.like(root.<OrganisationReference>get("broker").<String>get("fullName"),

"%" + Utils.prepareStringForSQLLike(filters.get("broker").toString()) + "%"));

}

if (filters.get("insured") != null && filters.get("insured").toString() != "") {

predicates.add(builder.like(root.<OrganisationReference>get("insured").<String>get("fullName"),

Utils.prepareStringForSQLLike(filters.get("insured").toString())));

}

if (filters.get("cedingPoolRef") != null && filters.get("cedingPoolRef").toString() != "") {

predicates.add(builder.like(root.<String>get("cedingPoolReference"), "%" + filters.get("cedingPoolRef").toString() + "%"));

}

if (filters.get("excludeFullyPaid") != null && filters.get("excludeFullyPaid").toString() != "" && Boolean.valueOf( filters.get("excludeFullyPaid").toString()) == Boolean.TRUE ) {

predicates.add(builder.greaterThan(root.<Double>get("outstandingAmount"), 1d));

}

return builder.and(predicates.toArray(new Predicate[predicates.size()]));

}

};

if (pageable != null) {

vals = this.policyForFinanceRepository.findAll(specification, pageable).getContent();

} else {

vals = this.policyForFinanceRepository.findAll(specification, sort);

}

if (count != null && count) {

total = this.policyForFinanceRepository.count(specification);

}

} else {

vals = this.policyForFinanceRepository.findAll(sort);

if (count != null && count) {

total = this.policyForFinanceRepository.count();

}

}

result.put("result", ListMapper.mapList(vals, mapper, PolicyDTO.class));

result.put("count", total);

return result;

}

@Override

public List<PolicyFilterResponseDTO> getFilteredPoliciesLight(HashMap<String, Object> filters) {

List<Policy> vals = fetchAndFilterPolicies(filters);

List<PolicyFilterResponseDTO> res = null;

if (vals != null && !vals.isEmpty()) {

res = ListMapper.mapList(vals, mapper, PolicyFilterResponseDTO.class);

}

Set<Long> validPolicyIds = null;

if (filters.get("locationName") != null && filters.get("locationName").toString() != "") {

validPolicyIds = new HashSet<Long>();

List<PolicyLocationView> locations = policyLocationViewRepository

.findByNameContains(filters.get("locationName").toString());

if (locations != null && locations.size() > 0) {

for (PolicyLocationView location : locations) {

validPolicyIds.add(location.getPolicyId());

}

}

}

if (filters.get("accountPolicyHolder") != null && filters.get("accountPolicyHolder").toString() != "") {

List<Policy> validPolicies = this.repository

.findPoliciesByUsers((String) filters.get("accountPolicyHolder"));

Set<Long> newValidIDs = new HashSet<Long>();

for (Policy p : validPolicies) {

newValidIDs.add(p.getId());

}

validPolicyIds.retainAll(newValidIDs);

}

if (validPolicyIds == null) {

return res;

}

return removeInvalidIds(res, validPolicyIds);

}

private List<PolicyFilterResponseDTO> removeInvalidIds(List<PolicyFilterResponseDTO> policies,

Set<Long> validPolicyIds) {

ArrayList<PolicyFilterResponseDTO> result = new ArrayList<PolicyFilterResponseDTO>();

if (validPolicyIds != null) {

if (validPolicyIds.size() == 0) {

return new ArrayList<PolicyFilterResponseDTO>();

}

for (PolicyFilterResponseDTO policy : policies) {

if (validPolicyIds.contains(policy.getId()) == true) {

result.add(policy);

}

}

}

return result;

}

@Override

public Map<String, List<PolicyDTO>> getFilteredPoliciesByType(HashMap<String, Object> filters) {

Map<String, List<PolicyDTO>> result = new HashMap<>();

try {

int invalCount = 0;

List<Policy> fetched = fetchAndFilterPolicies(filters);

if (fetched != null && !fetched.isEmpty()) {

String name = null;

List<PolicyDTO> aux = new ArrayList<>();

PolicyDTO copy = null;

for (Policy policy : fetched) {

aux.clear();

if (policy != null && policy.getPolicyStatusID() != null

&& policy.getPolicyStatusID().getPolicyStatusName() != null) {

name = policy.getPolicyStatusID().getPolicyStatusName();

copy = new PolicyDTO((policy.getInsuranceStructureID() != null

? mapper.map(policy.getInsuranceStructureID(), OrganisationReferenceDTO.class) : null),

policy.getPolicyNumber(),

policy.getEntryTypeID() != null

? mapper.map(policy.getEntryTypeID(), EntryTypeDTO.class) : null,

policy.getRenewalDate(),

policy.getrAGcolourID() != null

? mapper.map(policy.getrAGcolourID(), RAGColourDTO.class) : null,

policy.getnRInetPremium());

if (!result.isEmpty() && result.get(name) != null) {

result.get(name).add(copy);

} else {

aux.add(copy);

result.put(name, aux);

}

} else {

invalCount++;

}

}

} else {

LOGGER.warn("Warn on PolicyServiceImpl - getFilteredPoliciesByType() -> no items found for: "

+ (filters != null ? filters.toString() : " - "));

}

if (invalCount == 0) {

LOGGER.error(

"Error on PolicyServiceImpl - getFilteredPoliciesByType() -> there're items fetched that are null by the count of: "

+ invalCount);

}

} catch (Exception ex) {

LOGGER.error("Exception on PolicyServiceImpl - getFilteredPoliciesByType() -> ", ex);

return null;

}

return result;

}

@Override

public List<PolicySearchResultsDTO> processPoliciesMainSearch(HashMap<String, Object> filters) {

final PolicySearchDTO policySearchDTO = new PolicySearchDTO();

if (!Utils.isEmptyString((String) filters.get("accountPolicyHolder"))) {

policySearchDTO.setAccountPolicyHolder((String) filters.get("accountPolicyHolder"));

}

if (!Utils.isEmptyString((String) filters.get("policyNumber"))) {

policySearchDTO.setPolicyNumber("%" + ((String) filters.get("policyNumber")) + "%");

}

if (!Utils.isEmptyString((String) filters.get("insured"))) {

policySearchDTO.setInsured("%" + ((String) filters.get("insured")) + "%");

}

policySearchDTO.setEnquiry(CastUtil.toBoolean(filters.get("enquiry")));

if (!Utils.isEmptyString((String) filters.get("locationName"))) {

policySearchDTO.setLocationName("%" + ((String) filters.get("locationName")) + "%");

}

if (!Utils.isEmptyString((String) filters.get("countryName"))) {

policySearchDTO.setCountryName("%" + ((String) filters.get("countryName")) + "%");

}

if (!Utils.isEmptyString((String) filters.get("broker"))) {

policySearchDTO.setBroker("%" + ((String) filters.get("broker")) + "%");

}

if (!Utils.isEmptyString((String) filters.get("policyStatus"))) {

policySearchDTO.setPolicyStatus("%" + ((String) filters.get("policyStatus")) + "%");

}

policySearchDTO.setRenewalDateAfter(CastUtil.toDate(filters.get("renewalDateAfter")));

policySearchDTO.setRenewalDateBefore(CastUtil.toDate(filters.get("renewalDateBefore")));

policySearchDTO.setStartDateAfter(CastUtil.toDate(filters.get("startDateAfter")));

policySearchDTO.setStartDateBefore(CastUtil.toDate(filters.get("startDateBefore")));

if (!Utils.isEmptyString((String) filters.get("otherPolicyReferences"))) {

policySearchDTO.setOtherPolicyReferences("%" + ((String) filters.get("otherPolicyReferences")) + "%");

}

if (!Utils.isEmptyString((String) filters.get("colour"))) {

policySearchDTO.setColour("%" + ((String) filters.get("colour")) + "%");

}

policySearchDTO.setEntryType(CastUtil.toLong(filters.get("entryType")));

policySearchDTO.setYearOfAccount(CastUtil.toInteger(filters.get("yearOfAccount")));

policySearchDTO.setCountry(CastUtil.toLong(filters.get("country")));

policySearchDTO.setrAGcolorID(CastUtil.toLong(filters.get("rAGcolorID")));

return repository.filterPolicies(policySearchDTO);

}

/\*

\* (non-Javadoc)

\*

\* @see com.nuclear\_risk.live.underwriting.service.PolicyService#

\* getPoliciesByUser(java.lang.String)

\*/

@Override

public List<PolicyDTO> getPoliciesByUser(String id) {

List<Policy> policies = this.repository.findByCreator(new UserReference(id));

return ListMapper.mapList(policies, mapper, PolicyDTO.class);

}

@Override

@Caching(evict = { @CacheEvict(value = CacheConstants.Values.POLICY\_NUMBER\_ALL, allEntries = true) })

public PolicyMainInfoDTO createPolicy(PolicyDTO policyDTO, String policyType) {

if (!Arrays.asList(this.getCreatePolicyTypes()).contains(policyType))

throw new Error("Error on createPolicy() invalid policy type: " + policyType);

// ---------- Booleans ------------

policyDTO = this.fixPolicyBooleans(policyDTO);

PolicyForSave policy = mapper.map(policyDTO, PolicyForSave.class);

// ------------- Header ------------

policy.setUserID((policyDTO.getUserID() == null || policyDTO.getUserID().getId() == null) ? null

: userRepository.findOne(policyDTO.getUserID().getId()));

// --------- Basic Details -----------

policy.setInsured((policyDTO.getInsured() == null || policyDTO.getInsured().getId() == null) ? null

: organisationReferenceRepository.findOne(policyDTO.getInsured().getId()));

policy.setPolicyStatusID(

(policyDTO.getPolicyStatusID() == null || policyDTO.getPolicyStatusID().getPolicyStatusID() == null)

? policyStatusRepository.findByPolicyStatusNameOrderByPolicyStatusNameAsc("Enquiry")

: policyStatusRepository.findOne(policyDTO.getPolicyStatusID().getPolicyStatusID()));

policy.setEntryTypeID(

(policyDTO.getEntryTypeID() == null || policyDTO.getEntryTypeID().getEntryTypeId() == null) ? null

: entryTypeRepository.findOne(policyDTO.getEntryTypeID().getEntryTypeId()));

policy.setUsers(null);

if (policyDTO.getUsers() != null) {

ArrayList<UserReference> users = new ArrayList<UserReference>();

policyDTO.getUsers().forEach((user) -> {

if (user.getId() != null)

users.add(userRepository.findOne(user.getId()));

});

policy.setUsers(users);

}

// --------- Insurance -----------

policy.setCoverageID((policyDTO.getCoverageID() == null || policyDTO.getCoverageID().getCoverageID() == null)

? null : coverageRepository.findOne(policyDTO.getCoverageID().getCoverageID()));

policy.setAddOnID(null);

if (policyDTO.getAddOnID() != null) {

ArrayList<AddOn> addOns = new ArrayList<AddOn>();

policyDTO.getAddOnID().forEach((addOn) -> {

if (addOn.getId() != null)

addOns.add(addOnRepository.findOne(addOn.getId()));

});

policy.setAddOnID(addOns);

}

policy.setLocationId(null);

if (policyDTO.getLocationId() != null) {

ArrayList<LocationReference> locations = new ArrayList<LocationReference>();

policyDTO.getLocationId().forEach((location) -> {

if (location.getId() != null)

locations.add(locationReferenceRepository.findOne(location.getId()));

});

policy.setLocationId(locations);

}

policy.setStationId(null);

if (policyDTO.getStationId() != null) {

ArrayList<StationReference> stations = new ArrayList<StationReference>();

policyDTO.getStationId().forEach((station) -> {

if (station.getId() != null)

stations.add(stationReferenceRepository.findOne(station.getId()));

});

policy.setStationId(stations);

}

// --------- Structure -----------

policy.setInsuranceStructureID((policyDTO.getInsuranceStructureID() == null

|| policyDTO.getInsuranceStructureID().getInsuranceStructureID() == null) ? null

: insuranceStructureRepository

.findOne(policyDTO.getInsuranceStructureID().getInsuranceStructureID()));

policy.setBroker((policyDTO.getBroker() == null || policyDTO.getBroker().getId() == null) ? null

: organisationReferenceRepository.findOne(policyDTO.getBroker().getId()));

policy.setBrokerContact(

(policyDTO.getBrokerContact() == null || policyDTO.getBrokerContact().getContactDetailID() == null)

? null

: contactDetailsReferenceRepository.findOne(policyDTO.getBrokerContact().getContactDetailID()));

policy.setPoolInsurer((policyDTO.getPoolInsurer() == null || policyDTO.getPoolInsurer().getId() == null) ? null

: organisationReferenceRepository.findOne(policyDTO.getPoolInsurer().getId()));

policy.setNonPoolInsurer(

(policyDTO.getNonPoolInsurer() == null || policyDTO.getNonPoolInsurer().getId() == null) ? null

: organisationReferenceRepository.findOne(policyDTO.getNonPoolInsurer().getId()));

policy.setPolicyShareID(null);

policy.setRelatedPolicyID(null);

if (policyDTO.getRelatedPolicyID() != null) {

List<RelatedPolicyForView> policies = new ArrayList<RelatedPolicyForView>();

policyDTO.getRelatedPolicyID().forEach((pol) -> {

if (pol.getId() != null)

policies.add(relatedPolicyForViewRepository.findOne(pol.getId()));

});

policy.setRelatedPolicyID(policies);

}

// --------- Exposure -----------

policy.setCurrencyID((policyDTO.getCurrencyID() == null || policyDTO.getCurrencyID().getId() == null) ? null

: currencyRepository.findOne(policyDTO.getCurrencyID().getId()));

policy.setSublimitID(null);

if (policyDTO.getSublimitID() != null) {

ArrayList<Sublimit> sublimits = new ArrayList<Sublimit>();

policyDTO.getSublimitID().forEach((sublimit) -> {

SublimitDTO newSublimit = sublimitService.createSublimit(sublimit);

sublimits.add(sublimitRepository.findOne(newSublimit.getId()));

});

policy.setSublimitID(sublimits);

}

policy.setAdditionalLimitID(null);

if (policyDTO.getAdditionalLimitID() != null) {

ArrayList<Sublimit> additionalLimits = new ArrayList<Sublimit>();

policyDTO.getAdditionalLimitID().forEach((additionalLimit) -> {

SublimitDTO newAdditionalLimit = sublimitService.createSublimit(additionalLimit);

additionalLimits.add(sublimitRepository.findOne(newAdditionalLimit.getId()));

});

policy.setAdditionalLimitID(additionalLimits);

}

policy.setDeductibleID(null);

if (policyDTO.getDeductibleID() != null) {

ArrayList<Deductible> deductibles = new ArrayList<Deductible>();

policyDTO.getDeductibleID().forEach((deductible) -> {

DeductibleDTO newDeductible = deductibleService.createDeductible(deductible);

deductibles.add(deductibleRepository.findOne(newDeductible.getId()));

});

policy.setDeductibleID(deductibles);

}

// PolicyIncome

policy.setPolicyIncomeID(null);

if (policyDTO.getPolicyIncomeID() != null) {

ArrayList<PolicyIncome> incomes = new ArrayList<PolicyIncome>();

policyDTO.getPolicyIncomeID().forEach((dto) -> {

PolicyIncomeDTO newDTO = policyIncomeService.create(dto);

incomes.add(policyIncomeRepository.findOne(newDTO.getId()));

});

policy.setPolicyIncomeID(incomes);

}

// --------- Premium -----------

policy.setPolicyPremiumTaxesID(null);

if (policyDTO.getPolicyPremiumTaxesID() != null) {

ArrayList<PolicyPremiumTaxesReference> taxes = new ArrayList<PolicyPremiumTaxesReference>();

policyDTO.getPolicyPremiumTaxesID().forEach((tax) -> {

if (tax.getTaxTypeID() != null) {

TaxTypeReference tt = taxTypeReferenceRepository.findByCode(tax.getTaxTypeID().getCode());

if (tt != null) {

tax.setTaxTypeID(

mapper.map(taxTypeReferenceRepository.findOne(tt.getId()), TaxTypeReferenceDTO.class));

} else {

tax.setTaxTypeID(null);

}

}

PolicyPremiumTaxesDTO newTax = policyPremiumTaxesService

.createPolicyPremiumTaxes(mapper.map(tax, PolicyPremiumTaxesDTO.class));

taxes.add(policyPremiumTaxesReferenceRepository.findOne(newTax.getId()));

});

policy.setPolicyPremiumTaxesID(taxes);

}

policy.setDeductionID(null);

if (policyDTO.getDeductionID() != null) {

ArrayList<Deduction> deductions = new ArrayList<Deduction>();

policyDTO.getDeductionID().forEach((deduction) -> {

DeductionDTO newDeduction = deductionService.createDeduction(deduction);

deductions.add(deductionRepository.findOne(newDeduction.getId()));

});

policy.setDeductionID(deductions);

}

// ----- Insurance Programme ------

// --------- Indicators -----------

// --------- Compliance -----------

// --------- Documents ------------

policy.setDocuments(null);

// -------------------------------

policy.setReinstalementsID(null);

// premium received

policy.setPremiumReceivedID(null);

policy.setPremiumReceived(0d);

policy.setPremiumReceivedGBP(0d);

policy.setPremiumReceived100(0d);

policy.setUnderwriters(Fetcher.byIdList(userRepository, policyDTO.getUnderwriters(), user -> user.getId()));

policy.setEngineers(Fetcher.byIdList(userRepository, policyDTO.getEngineers(), user -> user.getId()));

policy.setOrganisationPolicyCoverageID(null);

// outstanding Amount

if (policy.getReinsuranceOut() && policy.getInsuranceStructureID().getInsuranceStructureID() == 1) {

policy.setOutstandingAmount((

(policyDTO.getnRIPremiumDebt() != null ? policyDTO.getnRIPremiumDebt() : 0d)

- (policyDTO.getiPEAndOtherFeesToNRI() != null ? policyDTO.getiPEAndOtherFeesToNRI() : 0d)) - policy.getPremiumReceived());

} else {

policy.setOutstandingAmount(

(policyDTO.getnRIPremiumDebt() != null ? policyDTO.getnRIPremiumDebt() : 0d)

- policy.getPremiumReceived());

}

policy.setFirstYearOfAttachment(policyDTO.getFirstYearOfAttachment());

String userId = (String) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

policy.setCreator(userRepository.findOne(userId));

if (policyType.equals("POLICY")) {

String newUniqueId;

policy.setYearOfAccount(null);

if (policyDTO.getStartDate() != null) {

policy.setYearOfAccount(DateUtils.getYear(policyDTO.getStartDate()));

}

policy.setFirstYearOfAttachment((policy.getYearOfAccount() == null) ? null : policy.getYearOfAccount());

String lastUniqueId = getLastPolicyUniqueIdentifier();

if (lastUniqueId != null) {

newUniqueId = String.format("%04d", new Object[] { Integer.valueOf(Integer.parseInt(lastUniqueId) + 1) });

} else {

newUniqueId = "0001";

}

policy.setUniqueIdentifier(newUniqueId);

if (policyDTO.getPolicyStatusID() != null && policyDTO.getPolicyStatusID().getPolicyStatusCode() != null) {

if (policyDTO.getPolicyStatusID().getPolicyStatusCode().equals("BOUND")) {

String coveragePrefix = policy.getCoverageID().getPreFix();

String firstYearOfAttachment = policy.getFirstYearOfAttachment().toString().substring(2, 4);

String uniqueIdentifier = policy.getUniqueIdentifier();

String yearOfAccount = policy.getYearOfAccount().toString();

String country = policy.getInsured().getCountryId().getCodeIso();

StringBuilder policyNumber = new StringBuilder();

policyNumber.append(coveragePrefix).append(firstYearOfAttachment).append(country).append(uniqueIdentifier)

.append("/").append(yearOfAccount).append("-").append("01");

policy.setPolicyNumber(policyNumber.toString());

policy.setPolicyVersion("01");

policy.setPolicyStatusID(this.policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.BOUND.name()));

policy.setrAGcolourID(null);

policy.setQuoteNumber(null);

policy.setEnquiry(Boolean.valueOf(false));

} else if (policyDTO.getPolicyStatusID().getPolicyStatusCode().equals("POLICY\_IN\_FORCE")) {

String coveragePrefix = policy.getCoverageID().getPreFix();

String firstYearOfAttachment = policy.getFirstYearOfAttachment().toString().substring(2, 4);

String uniqueIdentifier = policy.getUniqueIdentifier();

String yearOfAccount = policy.getYearOfAccount().toString();

String country = policy.getInsured().getCountryId().getCodeIso();

StringBuilder policyNumber = new StringBuilder();

policyNumber.append(coveragePrefix).append(firstYearOfAttachment).append(country).append(uniqueIdentifier)

.append("/").append(yearOfAccount).append("-").append("01");

policy.setPolicyNumber(policyNumber.toString());

policy.setPolicyVersion("01");

policy.setPolicyStatusID(this.policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.POLICY\_IN\_FORCE.name()));

policy.setrAGcolourID(null);

policy.setQuoteNumber(null);

policy.setEnquiry(Boolean.valueOf(false));

} else if (policyDTO.getPolicyStatusID().getPolicyStatusCode().equals("ENQUIRY")) {

policy.setPolicyVersion(null);

policy.setPolicyStatusID(this.policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

policy.setrAGcolourID(null);

policy.setQuoteNumber(null);

policy.setEnquiry(Boolean.valueOf(true));

policy.setPolicyNumber(policy.getUniqueIdentifier());

}

} else {

policy.setPolicyVersion(null);

policy.setPolicyStatusID(this.policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

policy.setrAGcolourID(null);

policy.setQuoteNumber(null);

policy.setEnquiry(Boolean.valueOf(true));

policy.setPolicyNumber(policy.getUniqueIdentifier());

}

}

if (policyType.equals("RENEWAL")) {

if (policyDTO.getStartDate() != null) {

policy.setYearOfAccount(DateUtils.getYear(policyDTO.getStartDate()));

}

policy.setPolicyVersion(null);

if (policy.getId() != null && policy.getPolicyStatusID() != null

&& policy.getPolicyStatusID().getPolicyStatusName().equals("Expired")) {

PolicyForSave policyForSave = repositoryForSave.findOne(policy.getId());

if (policyForSave != null) {

PolicyStatus policyStatusID = policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc("RENEWED");

policyForSave.setPolicyStatusID(policyStatusID);

policyForSave = repositoryForSave.save(policyForSave);

}

List<PolicyForSave> endorsements = repositoryForSave.findByUniqueIdentifierAndYearOfAccount(

policyForSave.getUniqueIdentifier(), policyForSave.getYearOfAccount());

if (endorsements != null && !endorsements.isEmpty()) {

for (PolicyForSave endorsement : endorsements) {

PolicyStatus expiredPolicyStatusID = policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc("EXPIRED");

if (endorsement.getPolicyStatusID() == expiredPolicyStatusID) {

PolicyStatus renewedPolicyStatusID = policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc("RENEWED");

endorsement.setPolicyStatusID(renewedPolicyStatusID);

endorsement = repositoryForSave.save(endorsement);

}

}

}

}

}

if (policyType.equals("ENDORSEMENT") && policy.getEntryTypeID() != null && policy.getId() != null

&& policy.getEntryTypeID().getEntrySubType() != null

&& policy.getEntryTypeID().getEntrySubType().toUpperCase().compareTo("MID-TERM CANCELLATION") == 0) {

PolicyStatus cancelStatus = policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc("CANCELLED");

policy.setPolicyStatusID(cancelStatus);

PolicyForSave parentPolicy = repositoryForSave.findOne(policy.getId());

if (parentPolicy != null) {

parentPolicy.setPolicyStatusID(cancelStatus);

parentPolicy = repositoryForSave.save(parentPolicy);

}

}

if (!policyType.equals("POLICY")) {

policy = setNewPolicyBasicDetails(policy, policyType);

}

policy.setCompletenessStatusCheck(this.isStatusComplete(policy));

policy = repositoryForSave.save(policy);

// Underwriting Process

policy.setUnderwritingProcess(null);

if (policyDTO.getUnderwritingProcess() != null) {

UnderwritingProcessDTO uwProcess = policyDTO.getUnderwritingProcess();

uwProcess.setIdPolicy(policy.getId());

UnderwritingProcessDTO underwritingProcess = underwritingProcessService.createUnderwritingProcess(uwProcess);

if (underwritingProcess != null && underwritingProcess.getId() != null) {

policy.setUnderwritingProcess(underwritingProcessRepository.findOne(underwritingProcess.getId()));

}

}

if (policyType.equals("ENDORSEMENT")) {

automatedDocumentProduction(policy, ENDORSEMENT);

}

if (policyDTO.getPolicyShareID() != null) {

List<PolicySharesShortReference> policyShares = new ArrayList<PolicySharesShortReference>();

PolicyDTO policyDto = new PolicyDTO();

policyDto.setId(policy.getId());

policyDTO.getPolicyShareID().forEach((policyShare) -> {

policyShare.setPolicyID(policyDto);

PolicySharesDTO policyshareDTO = policyShareService.createPolicyShares(policyShare);

policyShares.add(policySharesShortReferenceRepository.findOne(policyshareDTO.getPolicySharesID()));

});

policy.setPolicyShareID(policyShares);

}

// Create the policy Status Evolution object if necessary

if (policy.getEnquiry() != true) {

RAGColour defaultColor = rAGColourRepository.findByColourCode(EnumRagColour.GREEN.name());

if (defaultColor != null) {

PolicyForStatusEvolution policyForEvo = policyForStatusEvolutionRepository.findOne(policy.getId());

policy.setrAGcolourID(defaultColor);

PolicyStatusEvolution policyStatus = new PolicyStatusEvolution();

policyStatus.setPolicyID(policyForEvo);

policyStatus.setStartDate(new Date());

policyStatus.setrAGcolourID(defaultColor);

policyStatus.setUpdateTimesCount(0L);

policyStatusEvolutionRepository.save(policyStatus);

}

}

generateTasksAndALerts(null, policy);

final String historyType;

switch (policyType) {

case "ENDORSEMENT":

historyType = POLICY\_ENDORSED;

break;

case "RENEWAL":

historyType = POLICY\_RENEWED;

break;

default:

historyType = POLICY\_CREATED;

break;

}

if (historyType.equals(POLICY\_ENDORSED)) {

historyService.createHistoryEntry(policyDTO.getId().toString(), historyType);

} else {

historyService.createHistoryEntry(policy.getId().toString(), historyType);

}

return mapper.map(policy, PolicyMainInfoDTO.class);

}

private Boolean isStatusComplete(PolicyForSave policy) {

if (policy != null &&

policy.getStartDate() != null &&

policy.getEndDate() != null &&

policy.getInsured() != null &&

policy.getInsured().getFullName() != null &&

policy.getRenewalDate() != null &&

policy.getEntryTypeID() != null &&

policy.getEntryTypeID().getEntryTypeName() != null &&

policy.getCoverageID() != null &&

policy.getInsuranceStructureID() != null &&

policy.getInsuranceStructureID().getInsuranceStructureCode() != null &&

policy.getCoverageID().getCoverageName() != null &&

policy.getTotalPoolInsuranceShare() != null &&

policy.getNriGrossShare() != null &&

policy.getCurrencyID() != null &&

policy.getCurrencyID().getName() != null &&

policy.getrOE() != null &&

(policy.getrOEDate() != null || policy.getCurrencyID().getCode().equals("GBP")) &&

policy.getGrossPremium100Pct() != null &&

policy.getActualPrice() != null &&

policy.getTechnicalPrice() != null &&

policy.getSanctionsValidation() != null &&

policy.getSanctionsValidation() &&

policy.getNumberOfInstalments() != null &&

( (policy.getInstalmentDate() == null && (policy.getNumberOfInstalments() == null || policy.getNumberOfInstalments() == 0))

|| (policy.getNumberOfInstalments() == policy.getInstalmentDate().size()) )) {

if (policy.getInsuranceStructureID().getInsuranceStructureCode().equals("REINSURANCE\_INWARDS") && !(policy.getCedingPoolReference() != null &&

policy.getCedantShare1() != null && policy.getReinsuranceInFrom() != null)) {

return false;

}

if (policy.getInsuranceStructureID().getInsuranceStructureCode().equals("DIRECT\_INSURANCE") && !(policy.getLimitPerOcurrenceFor100() != null)) {

return false;

}

if (policy.getEntryTypeID().getEntryTypeName().toUpperCase().equals("RENEWAL") && !(policy.getRateChange() != null)) {

return false;

}

return true;

}

return false;

}

private void generateTasksAndALerts(final PolicyForSave oldPolicy, final PolicyForSave policy2) {

if (Objects.isNull(policy2.getPolicyStatusID()) || (policy2.getPolicyStatusID().getPolicyStatusID() == 1)) {

return;

}

if (policy2.getPolicyShareID() != null) {

final java.util.function.Predicate<PolicySharesShortReference> filter = ps -> {

if (!EnumInsurerRole.REOUT.getValue().equalsIgnoreCase(ps.getInsurerRole())) {

return false;

}

if (ps.getOrganisationID() == null || ps.getOrganisationID().getOrganisationType() == null) {

return false;

}

return EnumOrganisationType.POOL\_RE.name()

.equals(ps.getOrganisationID().getOrganisationType().getOrganisationTypeCode());

};

final List<PolicySharesShortReference> newPoolRe = FilterUtil.filter(policy2.getPolicyShareID(), filter);

if (newPoolRe != null && !newPoolRe.isEmpty()) {

final List<PolicySharesShortReference> oldPoolRe = FilterUtil

.filter(oldPolicy == null ? null : oldPolicy.getPolicyShareID(), filter);

if (oldPolicy == null || oldPoolRe == null || oldPoolRe.isEmpty()) {

final TaskDTO taskTemplate = new TaskDTO();

taskTemplate.setTitle("The Policy " + policy2.getPolicyNumber()

+ " with reinsurance to Pool Re has been created");

taskTemplate.setSubTitle("Please review this policy with reinsurance to Pool Re has been created");

taskTemplate.setDueDate(DateUtils.format(DateUtils.addDays(7)));

taskTemplate.setPolicyNumber(policy2.getPolicyNumber());

taskTemplate.setComplete(false);

taskService.generateTask(EnumUserRoles.FINANCE.getValue(), taskTemplate);

}

}

}

// New policies with locations trigger an alert for all engineers

if (policy2.getLocationId() != null && !policy2.getLocationId().isEmpty()

&& (oldPolicy == null || oldPolicy.getLocationId() == null || oldPolicy.getLocationId().isEmpty())) {

final AlertDTO alertTemplate = new AlertDTO();

alertTemplate.setAutomatic(false);

alertTemplate.setTitle("Policy " + policy2.getPolicyNumber() + " has been created");

if (policy2.getLocationId().size() == 1) {

alertTemplate.setDescription("Policy " + policy2.getPolicyNumber() + ", for the Location "

+ policy2.getLocationId().get(0).getName() + " has been created");

} else {

final StringBuilder builder = new StringBuilder();

builder.append("Policy ");

builder.append(policy2.getPolicyNumber());

builder.append("has been created for the following locations: ");

boolean first = true;

for (final LocationReference location : policy2.getLocationId()) {

if (first) {

first = false;

} else {

builder.append(", ");

}

builder.append(location.getName());

}

alertTemplate.setDescription(builder.toString());

}

alertService.generateAlert(EnumUserRoles.ENGINEER.getValue(), alertTemplate);

}

// When the policy's status change to CANCELLED and the policy has

// locations, we create an alert for all engineers

if (policy2.getPolicyStatusID() != null) {

if (EnumPolicyStatus.CANCELLED.name().equals(policy2.getPolicyStatusID().getPolicyStatusCode())

&& policy2.getLocationId() != null && !policy2.getLocationId().isEmpty()

&& (oldPolicy == null || oldPolicy.getPolicyStatusID() == null || !EnumPolicyStatus.CANCELLED.name()

.equals(oldPolicy.getPolicyStatusID().getPolicyStatusCode()))) {

final AlertDTO alertTemplate = new AlertDTO();

alertTemplate.setAutomatic(false);

alertTemplate.setTitle("Policy " + policy2.getPolicyNumber() + " has been cancelled");

if (policy2.getLocationId().size() == 1) {

alertTemplate.setDescription("Policy " + policy2.getPolicyNumber() + ", for the Location "

+ policy2.getLocationId().get(0).getName() + " has been cancelled");

} else {

final StringBuilder builder = new StringBuilder();

builder.append("Policy ");

builder.append(policy2.getPolicyNumber());

builder.append("has been cancelled for the following locations: ");

boolean first = true;

for (final LocationReference location : policy2.getLocationId()) {

if (first) {

first = false;

} else {

builder.append(", ");

}

builder.append(location.getName());

}

alertTemplate.setDescription(builder.toString());

}

alertService.generateAlert(EnumUserRoles.ENGINEER.getValue(), alertTemplate);

}

}

// Creation of an alert for all engineers, then a location lost a

// coverage, by changes on policy's coverage

if (policy2.getCoverageID() != null && oldPolicy != null && oldPolicy.getCoverageID() != null

&& !policy2.getCoverageID().getCoverageID().equals(oldPolicy.getCoverageID().getCoverageID())

&& oldPolicy.getLocationId() != null && !oldPolicy.getLocationId().isEmpty()) {

for (final LocationReference location : oldPolicy.getLocationId()) {

final Integer count = repository.currentCoverageCount(location.getId(),

oldPolicy.getCoverageID().getCoverageID(), PolicyConstants.POLICY\_TERMINAL\_STATUS\_CODES);

if (count == 0) {

final AlertDTO alertTemplate = new AlertDTO();

alertTemplate.setAutomatic(false);

alertTemplate

.setTitle("Cover " + oldPolicy.getCoverageID().getCoverageName() + " has been removed");

alertTemplate.setDescription("Cover " + oldPolicy.getCoverageID().getCoverageName()

+ " has been removed from the location " + location.getName());

alertService.generateAlert(EnumUserRoles.ENGINEER.getValue(), alertTemplate);

}

}

}

// Creation of an alert for all engineers, then a location lost a

// coverage, by changes on policy's locations

if (oldPolicy != null && oldPolicy.getLocationId() != null && !oldPolicy.getLocationId().isEmpty()

&& (policy2.getLocationId() == null || policy2.getLocationId().isEmpty() || Utils.areListsDifferent(

oldPolicy.getLocationId(), policy2.getLocationId(), (a, b) -> a.getId().equals(b.getId())))) {

final List<LocationReference> lostLocations = Utils.notIn(oldPolicy.getLocationId(),

policy2.getLocationId(), (a, b) -> a.getId().equals(b.getId()));

for (final LocationReference location : lostLocations) {

final Integer count = repository.currentCoverageCount(location.getId(),

oldPolicy.getCoverageID().getCoverageID(), PolicyConstants.POLICY\_TERMINAL\_STATUS\_CODES);

if (count == 0) {

final AlertDTO alertTemplate = new AlertDTO();

alertTemplate.setAutomatic(false);

alertTemplate

.setTitle("Cover " + oldPolicy.getCoverageID().getCoverageName() + " has been removed");

alertTemplate.setDescription("Cover " + oldPolicy.getCoverageID().getCoverageName()

+ " has been removed from the location " + location.getName());

alertService.generateAlert(EnumUserRoles.ENGINEER.getValue(), alertTemplate);

}

}

}

}

private PolicyForSave setNewPolicyBasicDetailsForStatus(PolicyForSave policy2, String policyType) {

if (policyType.equals("QUOTED")) {

if (policy2.getPolicyVersion() == null) {

if (policy2.getInsuranceStructureID() != null) {

policy2.setQuoteNumber(DEFAULT\_QUOTE\_NR);

}

}

policy2.setPolicyVersion(DEFAULT\_POLICY\_VERSION);

policy2.setrAGcolourID(rAGColourRepository.findByColourCode(RAG\_COLOUR\_GREEN));

policy2.setPolicyNumber((policy2.getCoverageID() == null ? "" : policy2.getCoverageID().getPreFix())

+ (!Objects.isNull(policy2.getFirstYearOfAttachment())

? policy2.getFirstYearOfAttachment().toString().substring(2, 4) : "")

+ policy2.getInsured().getCountryId().getCodeIso() + policy2.getUniqueIdentifier() + '/'

+ policy2.getYearOfAccount() + '-' + policy2.getPolicyVersion());

}

if ((policyType.equals("POLICY\_IN\_FORCE") || policyType.equals("BOUND")) && policy2.getPolicyStatusID().getPolicyStatusID() == 1) {

String coveragePrefix = policy2.getCoverageID().getPreFix();

String firstYearOfAttachment = policy2.getFirstYearOfAttachment().toString().substring(2, 4);

String country;

String uniqueIdentifier = policy2.getUniqueIdentifier();

String yearOfAccount = policy2.getYearOfAccount().toString();

country = policy2.getInsured().getCountryId().getCodeIso();

StringBuilder policyNumber = new StringBuilder();

country = (Character.isDigit(policy2.getPolicyNumber().charAt(3)))

? (Character.isDigit(policy2.getPolicyNumber().charAt(1)))

? policy2.getInsured().getCountryId().getCodeIso()

: policy2.getPolicyNumber().substring(4,7)

: policy2.getPolicyNumber().substring(3,6);

Integer lastVersion = getLatestPolicyExtension(

policy2.getUniqueIdentifier().concat("/".concat(policy2.getYearOfAccount().toString())));

String policyVersion = lastVersion != null ? (String.format("%02d", lastVersion + 1)) : policy2.getPolicyVersion();

policyNumber.append(coveragePrefix).append(firstYearOfAttachment).append(country).append(uniqueIdentifier)

.append("/").append(yearOfAccount).append("-").append(policyVersion);

policy2.setPolicyNumber(policyNumber.toString());

policy2.setPolicyVersion("01");

}

if ((policyType.equals("POLICY\_IN\_FORCE") || policyType.equals("BOUND")) && policy2.getPolicyStatusID().getPolicyStatusID() == 2) {

PolicyForSave parentPolicy = repositoryForSave.findOne(policy2.getId());

String coveragePrefix = policy2.getCoverageID().getPreFix();

String firstYearOfAttachment = policy2.getFirstYearOfAttachment().toString().substring(2, 4);

String country;

String uniqueIdentifier = policy2.getUniqueIdentifier();

String yearOfAccount = policy2.getYearOfAccount().toString();

Integer lastVersion = getLatestPolicyExtension(

policy2.getUniqueIdentifier().concat("/".concat(policy2.getYearOfAccount().toString())));

String newVersion = String.format("%02d", lastVersion + 1);

policy2.setEndorsementNumber(newVersion);

policy2.setPolicyVersion(newVersion);

country = (Character.isDigit(policy2.getPolicyNumber().charAt(3)))

? (Character.isDigit(policy2.getPolicyNumber().charAt(1)))

? policy2.getInsured().getCountryId().getCodeIso()

: policy2.getPolicyNumber().substring(4,7)

: policy2.getPolicyNumber().substring(3,6);

StringBuilder policyNumber = new StringBuilder();

policyNumber.append(coveragePrefix).append(firstYearOfAttachment).append(country).append(uniqueIdentifier)

.append("/").append(yearOfAccount).append("-").append(newVersion);

policy2.setPolicyNumber(policyNumber.toString());

}

return policy2;

}

private PolicyForSave setNewPolicyBasicDetails(PolicyForSave policy2, String policyType) {

String policyBasePolicyNumber = null;

if (!policyType.equals(POLICY)) {

policyBasePolicyNumber = policy2.getPolicyNumber().substring(0, policy2.getPolicyNumber().length() - 2);

}

;

policy2.setrAGcolourID(rAGColourRepository.findByColourCode(RAG\_COLOUR\_GREEN));

if (policy2.getPolicyVersion() == null) {

if (policy2.getInsuranceStructureID() != null && !(policyType.equals("RENEWAL"))) {

policy2.setQuoteNumber(DEFAULT\_QUOTE\_NR);

}

policy2.setPolicyVersion(DEFAULT\_POLICY\_VERSION);

}

if (policyType.equals(ENDORSEMENT)) {

Integer lastVersion = getLatestPolicyExtension(

policy2.getUniqueIdentifier().concat("/".concat(policy2.getYearOfAccount().toString())));

String newVersion = String.format("%02d", lastVersion + 1);

String lastEndorsementNumber = getLatestEndorsementNumber(policy2.getUniqueIdentifier().concat("/".concat(policy2.getYearOfAccount().toString())));

Integer newEndorsementNumber = (lastEndorsementNumber != null) ? (Integer.parseInt(lastEndorsementNumber) + 1) : 1;

policy2.setEndorsementNumber(String.format("%02d", newEndorsementNumber));

policy2.setPolicyVersion(newVersion);

} else if (policyType.equals(RENEWAL)) {

// NRIIMPLEM-2613 - allow user to pick the policy status on renewal

if (policy2.getPolicyStatusID() != null && policy2.getPolicyStatusID().getPolicyStatusCode() != null) {

if (policy2.getPolicyStatusID().getPolicyStatusCode().equals("BOUND")) {

policy2.setPolicyVersion("01");

policy2.setPolicyStatusID(this.policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.BOUND.name()));

policy2.setrAGcolourID(null);

policy2.setQuoteNumber(null);

policy2.setEnquiry(Boolean.valueOf(false));

} else if (policy2.getPolicyStatusID().getPolicyStatusCode().equals("POLICY\_IN\_FORCE")) {

policy2.setPolicyVersion("01");

policy2.setPolicyStatusID(this.policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.POLICY\_IN\_FORCE.name()));

policy2.setrAGcolourID(null);

policy2.setQuoteNumber(null);

policy2.setEnquiry(Boolean.valueOf(false));

} else if (policy2.getPolicyStatusID().getPolicyStatusCode().equals("ENQUIRY")) {

policy2.setPolicyVersion("00");

policy2.setPolicyStatusID(this.policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

policy2.setrAGcolourID(null);

policy2.setQuoteNumber(null);

policy2.setEnquiry(Boolean.valueOf(true));

}

}

} else if (policyType.equals(QUOTE)) {

int numberOfQuotes = this.repository.getNumberOfPoliciesByPolicyNumber(policyBasePolicyNumber.concat("%"));

policy2.setQuoteNumber(String.format("%02d", numberOfQuotes + 1));

}

if (policyType.equals(ENDORSEMENT)) {

PolicyForSave parentPolicy = repositoryForSave.findOne(policy2.getId());

String coveragePrefix = policy2.getCoverageID().getPreFix();

String firstYearOfAttachment = policy2.getFirstYearOfAttachment().toString().substring(2, 4);

String country;

String uniqueIdentifier = policy2.getUniqueIdentifier();

String yearOfAccount = parentPolicy.getYearOfAccount().toString();

String policyVersion = policy2.getPolicyVersion();

if (parentPolicy.getInsured().getId().equals(policy2.getInsured().getId())) {

country = parentPolicy.getPolicyNumber().substring(

parentPolicy.getCoverageID().getPreFix().length() + 2,

parentPolicy.getCoverageID().getPreFix().length() + 5);

} else {

country = policy2.getInsured().getCountryId().getCodeIso();

}

StringBuilder policyNumber = new StringBuilder();

policyNumber.append(coveragePrefix).append(firstYearOfAttachment).append(country).append(uniqueIdentifier)

.append("/").append(yearOfAccount).append("-").append(policyVersion);

policy2.setPolicyNumber(policyNumber.toString());

policy2.setYearOfAccount(parentPolicy.getYearOfAccount());

} else if (policyType.equals(RENEWAL)) {

PolicyForSave parentPolicy = repositoryForSave.findOne(policy2.getId());

String coveragePrefix = policy2.getCoverageID().getPreFix();

String firstYearOfAttachment = policy2.getFirstYearOfAttachment().toString().substring(2, 4);

String country;

String uniqueIdentifier = policy2.getUniqueIdentifier();

String yearOfAccount = policy2.getYearOfAccount().toString();

String policyVersion;

if (policy2.getPolicyStatusID() != null && policy2.getPolicyStatusID().getPolicyStatusCode() != null &&

policy2.getPolicyStatusID().getPolicyStatusCode().equals("ENQUIRY")) {

policyVersion = "00";

} else {

Integer lastVersion = getLatestPolicyExtension(

policy2.getUniqueIdentifier().concat("/".concat(policy2.getYearOfAccount().toString())));

policyVersion = lastVersion != null ? (String.format("%02d", lastVersion + 1)) : policy2.getPolicyVersion();

}

if (parentPolicy.getInsured().getId().equals(policy2.getInsured().getId())) {

country = parentPolicy.getPolicyNumber().substring(

parentPolicy.getCoverageID().getPreFix().length() + 2,

parentPolicy.getCoverageID().getPreFix().length() + 5);

} else {

country = policy2.getInsured().getCountryId().getCodeIso();

}

StringBuilder policyNumber = new StringBuilder();

policyNumber.append(coveragePrefix).append(firstYearOfAttachment).append(country).append(uniqueIdentifier)

.append("/").append(yearOfAccount).append("-").append(policyVersion);

policy2.setPolicyNumber(policyNumber.toString());

} else if (!policyType.equals(POLICY)) {

PolicyForSave parentPolicy = repositoryForSave.findOne(policy2.getId());

String coveragePrefix = policy2.getCoverageID().getPreFix();

String firstYearOfAttachment = policy2.getFirstYearOfAttachment().toString().substring(2, 4);

String country;

String uniqueIdentifier = policy2.getUniqueIdentifier();

String yearOfAccount = policy2.getYearOfAccount().toString();

String policyVersion = policy2.getPolicyVersion();

if (parentPolicy.getInsured().getId().equals(policy2.getInsured().getId())) {

country = parentPolicy.getPolicyNumber().substring(

parentPolicy.getCoverageID().getPreFix().length() + 2,

parentPolicy.getCoverageID().getPreFix().length() + 5);

} else {

country = policy2.getInsured().getCountryId().getCodeIso();

}

StringBuilder policyNumber = new StringBuilder();

policyNumber.append(coveragePrefix).append(firstYearOfAttachment).append(country).append(uniqueIdentifier)

.append("/").append(yearOfAccount).append("-").append(policyVersion);

policy2.setPolicyNumber(policyNumber.toString());

} else {

policy2.setPolicyNumber((policy2.getCoverageID() == null ? "" : policy2.getCoverageID().getPreFix())

+ (!Objects.isNull(policy2.getFirstYearOfAttachment())

? policy2.getFirstYearOfAttachment().toString().substring(2, 4) : "")

+ policy2.getInsured().getCountryId().getCodeIso() + policy2.getUniqueIdentifier() + '/'

+ policy2.getYearOfAccount() + '-' + policy2.getPolicyVersion());

}

return policy2;

}

private Integer getLatestPolicyExtension(String identifierAndYearOfAccount) {

return repository.findPolicyLastestExtensionByPolicyNumber("%".concat(identifierAndYearOfAccount.concat("%")));

}

private String getLatestEndorsementNumber(String identifierAndYearOfAccount) {

return repository.findPolicyLastestEndorsementNumberByPolicyNumber("%".concat(identifierAndYearOfAccount.concat("%")));

}

@Override

public PolicyDTO addFileToPolicy(Long idPolicy, String path, String name, String contentType, Long size) {

Policy pol = repository.findOne(idPolicy);

List<DocumentReference> documents = pol.getDocuments();

pol.setDocuments(documents);

repository.save(pol);

return mapper.map(pol, PolicyDTO.class);

}

@Override

public void changeStatusOnDocumentType(String contentType, Long policyID) {

boolean passedToDocumentationToIssue = false;

if (Objects.isNull(policyID) || StringUtils.isBlank(contentType)) {

return;

}

PolicyForSave pol = repositoryForSave.findById(policyID);

Date today = new Date();

Boolean createHistoryEntry = false;

try {

if (!pol.getInsuranceStructureID().getInsuranceStructureName()

.equalsIgnoreCase(INS\_STRUCTURE\_REINSURANCE\_INWARDS)) {

// WAITING\_FOR\_FIRM\_ORDER -> REINSURANCE\_OFFER\_TO\_SEND

/\*

\* A document with Cover bound email type is uploaded in the

\* system in Waiting for firm order status, the policy has

\* Reinsurance out field checked and sanctionsValidation

\*/

if (contentType.equalsIgnoreCase(DOCUMENT\_TYPE\_COVER\_BOUND\_EMAIL)

&& EnumPolicyStatus.ENQUIRY.name().equals(pol.getPolicyStatusID().getPolicyStatusCode())

&& pol.getReinsuranceOut() != null && pol.getReinsuranceOut()

&& pol.getSanctionsValidation() != null && pol.getSanctionsValidation()) {

pol.setPolicyStatusID(policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

pol.setStatusModificationTime(today);

pol = this.passOtherQuotesToNonTaken(pol);

this.automatedDocumentProduction(pol, DEBIT\_CREDIT\_NOTE);

this.automatedDocumentProduction(pol, STANDARD\_SUBMISSION\_FORM);

createHistoryEntry = true;

}

// REINSURANCE\_OFFER\_TO\_COLLATE -> DOCUMENTATION\_TO\_ISSUE

/\*

\* The number of document in the policy, with the type

\* 'Reinsurance offer response emails', is the same as the

\* reinsurance out organisations in the policy (rows with

\* INSURER\_ROLE at reout in Policy\_Shares table for the policy)

\* for policies in reinsurance offer to collate

\*/

if (contentType.equalsIgnoreCase(DOCUMENT\_TYPE\_REINSURANCE\_OFFER\_RESPONSE\_EMAILS)

&& EnumPolicyStatus.ENQUIRY.name().equals(pol.getPolicyStatusID().getPolicyStatusCode())) {

int totalReouts = 0;

int totalReinsuranceOfferResponseEmails = 0;

for (PolicySharesShortReference policyShares : pol.getPolicyShareID()) {

if (policyShares.getInsurerRole().equalsIgnoreCase("reout")) {

totalReouts++;

}

}

for (DocumentReferenceToView document : pol.getDocuments()) {

if (EnumDocumentType.REINSURANCE\_OFFER\_RESPONSE.name()

.equals(document.getDocumentType().getDocumentTypeCode())) {

totalReinsuranceOfferResponseEmails++;

}

}

if (totalReouts == totalReinsuranceOfferResponseEmails) {

pol.setPolicyStatusID(policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

pol.setStatusModificationTime(today);

passedToDocumentationToIssue = true;

createHistoryEntry = true;

}

}

// WAITING\_FOR\_SIGNED\_COVER\_NOTES -> POLICY\_IN\_FORCE

/\*

\* The same number of document in the policy with the type

\* 'Cover note received' created after status changed is the

\* same as the reinsurance out organisations in the policy (rows

\* with INSURER\_ROLE as reout andREOUT\_Status as 'Accepted' in

\* Policy\_Shares table for the policy)

\*/

if (contentType.equalsIgnoreCase(DOCUMENT\_TYPE\_COVER\_NOTE\_RECEIVED)

&& EnumPolicyStatus.ENQUIRY.name().equals(pol.getPolicyStatusID().getPolicyStatusCode())) {

int totalReouts = 0;

int totalReinsuranceOfferResponseEmails = 0;

for (PolicySharesShortReference policyShares : pol.getPolicyShareID()) {

if (policyShares.getInsurerRole().equalsIgnoreCase("reout")

&& policyShares.getrEOUTStatus().equalsIgnoreCase("Accepted")) {

totalReouts++;

}

}

for (DocumentReferenceToView document : pol.getDocuments()) {

if (EnumDocumentType.COVER\_NOTE\_RECEIVED.name()

.equals(document.getDocumentType().getDocumentTypeCode())

&& document.getCreationTime().after(pol.getStatusModificationTime())) {

totalReinsuranceOfferResponseEmails++;

}

}

if (totalReouts == totalReinsuranceOfferResponseEmails) {

pol.setPolicyStatusID(policyStatusRepository.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(

EnumPolicyStatus.POLICY\_IN\_FORCE.name()));

pol.setStatusModificationTime(today);

createHistoryEntry = true;

}

}

// WAITING\_FOR\_FIRM\_ORDER -> POLICY\_IN\_FORCE

/\*

\* A document with Cover bound email type is uploaded in the

\* system in Waiting for firm order status and the policy has

\* NOT Reinsurance out field checked

\*/

if (contentType.equalsIgnoreCase(DOCUMENT\_TYPE\_COVER\_BOUND\_EMAIL)

&& EnumPolicyStatus.ENQUIRY.name().equals(pol.getPolicyStatusID().getPolicyStatusCode())

&& (pol.getReinsuranceOut() == null || !pol.getReinsuranceOut())

&& pol.getSanctionsValidation() != null && pol.getSanctionsValidation()) {

pol.setPolicyStatusID(policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.POLICY\_IN\_FORCE.name()));

pol.setStatusModificationTime(today);

pol = this.passOtherQuotesToNonTaken(pol);

this.automatedDocumentProduction(pol, STANDARD\_SUBMISSION\_FORM);

this.automatedDocumentProduction(pol, DEBIT\_CREDIT\_NOTE); // NRIIMPLEM-1364

createHistoryEntry = true;

}

} else { // Policy Type: INS\_STRUCTURE\_REINSURANCE\_INWARDS

// WAITING\_REINSURANCE\_OFFER -> BINDING\_ANALYSIS

/\*

\* A document with SSF type is uploaded in the system and policy

\* is in waiting for reinsurance offer status

\*/

if (contentType.equalsIgnoreCase(DOCUMENT\_TYPE\_SSF)

&& EnumPolicyStatus.ENQUIRY.name().equals(pol.getPolicyStatusID().getPolicyStatusCode())) {

pol.setPolicyStatusID(

policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

pol.setStatusModificationTime(today);

createHistoryEntry = true;

}

// WAITING\_FOR\_COVER\_NOTE -> COVER\_NOTE\_TO\_CHECK

/\*

\* A document with Cover note type is uploaded in the system in

\* Waiting for cover note status

\*/

if (contentType.equalsIgnoreCase(DOCUMENT\_TYPE\_COVER\_NOTE)

&& EnumPolicyStatus.ENQUIRY.name().equals(pol.getPolicyStatusID().getPolicyStatusCode())) {

pol.setPolicyStatusID(

policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

pol.setStatusModificationTime(today);

createHistoryEntry = true;

}

}

repositoryForSave.saveAndFlush(pol);

if (createHistoryEntry) {

historyService.createHistoryEntry(pol.getId().toString(), POLICY\_STATUS\_CHANGED);

}

if (passedToDocumentationToIssue) {

// DOCUMENTATION\_TO\_ISSUE -> COVER\_NOTE\_TO\_ISSUE

/\*

\* The policy will be automatically moved to this status when

\* the system generate the Renewal document and the debit note

\* for reinsurance out policies. UPDATE - Andres: only check the

\* Renewal document. that debit note is not going to be

\* generated by the system

\*/

if (EnumPolicyStatus.ENQUIRY.name().equals(pol.getPolicyStatusID().getPolicyStatusCode())) {

automatedDocumentProduction(pol, DOCUMENTATION\_TO\_ISSUE);

pol.setDocuments(pol.getDocuments());

this.em.refresh(pol);

pol.setPolicyStatusID(policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

pol.setStatusModificationTime(today);

repositoryForSave.save(pol);

}

}

} catch (Exception e) {

System.out.println("ERROR ON changeStatusOnDocumentType: " + e.getStackTrace());

}

}

@Override

public PolicyDTO getPolicyById(Long idPolicy) {

if (Objects.nonNull(idPolicy)) {

PolicyForSave pol = repositoryForSave.findById(idPolicy);

PolicyDTO result = mapper.map(pol, PolicyDTO.class);

return mapForFavouriteForPolicy(result);

}

return null;

}

@Override

public List<PolicyDTO> getPoliciesById(List<Long> policyIds) {

ArrayList<PolicyDTO> result = new ArrayList<PolicyDTO>();

for (Long policyId : policyIds) {

result.add(getPolicyById(policyId));

}

return result;

}

@Override

public PolicyDTO getPolicyByIdForView(Long idPolicy) {

if (Objects.nonNull(idPolicy)) {

PolicyForView pol = repositoryForView.findById(idPolicy);

PolicyDTO result = mapper.map(pol, PolicyDTO.class);

return mapForFavouriteForPolicy(result);

}

return null;

}

private PolicyDTO mapForFavouriteForPolicy(PolicyDTO policy) {

NriParametersEntity typeOfEntity = parametersRepository.findByParameterParentAndParameterDescription(FAVOURITES,

FAV\_POLICIES);

String userId = (String) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

if (userId != null && !!userId.isEmpty()) {

FavouriteReference favouriteRef = favouriteRepository.findFavouriteByTypeOfEntityAndIdAndUserID(

typeOfEntity.getParameterValue(), policy.getId(), userId);

if (favouriteRef != null) {

FavouriteReferenceDTO favourite = mapper.map(favouriteRef, FavouriteReferenceDTO.class);

policy.setFavourite(favourite);

}

}

return policy;

}

@Override

public PolicyDTO getPolicyInfoForDocUploadByPolicyId(Long idPolicy) {

PolicyDTO result = null;

if (idPolicy != null) {

result = repository.getPolicyInfoForDocUploadByPolicyId(idPolicy);

}

return result;

}

@Override

public PolicyDTO getPolicyByPolicyNumber(String policyNumber, Boolean onlyOpenYearOfAccount) {

if (policyNumber != null) {

Policy pol = repository.findByPolicyNumber(policyNumber);

if (onlyOpenYearOfAccount) {

YearOfAccountReference year = yearOfAccountRepository.findOne(pol.getYearOfAccount());

if (year.getStatus().equalsIgnoreCase("closed")) {

return null;

}

}

PolicyDTO result = mapper.map(pol, PolicyDTO.class);

NriParametersEntity typeOfEntity = parametersRepository

.findByParameterParentAndParameterDescription(FAVOURITES, FAV\_POLICIES);

String userId = (String) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

if (userId != null && !!userId.isEmpty()) {

FavouriteReference favouriteRef = favouriteRepository.findFavouriteByTypeOfEntityAndIdAndUserID(

typeOfEntity.getParameterValue(), pol.getId(), userId);

if (favouriteRef != null) {

FavouriteReferenceDTO favourite = mapper.map(favouriteRef, FavouriteReferenceDTO.class);

result.setFavourite(favourite);

}

}

return result;

} else {

return null;

}

}

@Override

public PolicyDTO getPolicyByPolicyNumberManualMapped(String policyNumber) {

PolicyDTO result = null;

if (policyNumber != null && !policyNumber.isEmpty()) {

Policy pol = repository.findByPolicyNumber(policyNumber);

if (pol != null) {

result = convertPolicyEntityToDtoThin(pol);

NriParametersEntity typeOfEntity = parametersRepository

.findByParameterParentAndParameterDescription(FAVOURITES, FAV\_POLICIES);

String userId = (String) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

if (userId != null && !!userId.isEmpty()) {

FavouriteReference favouriteRef = favouriteRepository.findFavouriteByTypeOfEntityAndIdAndUserID(

typeOfEntity.getParameterValue(), pol.getId(), userId);

if (favouriteRef != null) {

FavouriteReferenceDTO favourite = mapper.map(favouriteRef, FavouriteReferenceDTO.class);

result.setFavourite(favourite);

}

}

}

}

return result;

}

@Override

public PolicyDTO getPolicyDetailsByNumber(String policyNumber) {

PolicyDTO result = null;

if (policyNumber != null && !policyNumber.isEmpty()) {

Policy pol = repository.findByPolicyNumber(policyNumber);

if (pol != null) {

result = convertPolicyEntityToDtoThin(pol);

}

}

return result;

}

@Override

public List<SublimitNameDTO> getSublimitNames() {

List<SublimitName> subLimNames = subNameRepository.findAllByOrderByName();

List<SublimitNameDTO> subLimNamesDTO = ListMapper.mapList(subLimNames, mapper, SublimitNameDTO.class);

return subLimNamesDTO;

}

private List<Policy> fetchAndFilterPolicies(HashMap<String, Object> filters) {

if (filters != null && !filters.isEmpty() && filters.keySet() != null && filters.values() != null

&& Collections.frequency(filters.values(), null) != filters.size()) {

Specification<Policy> specification = new Specification<Policy>() {

public Predicate toPredicate(Root<Policy> root, CriteriaQuery<?> query, CriteriaBuilder builder) {

List<Predicate> predicates = new ArrayList<Predicate>();

if (filters.get("policyNumber") != null && filters.get("policyNumber").toString() != "") {

predicates.add(builder.like(root.<String>get("policyNumber"),

"%" + filters.get("policyNumber").toString() + "%"));

}

if (filters.get("yearOfAccount") != null && filters.get("yearOfAccount").toString() != "") {

predicates.add(builder.equal(root.<Integer>get("yearOfAccount"),

(Integer) filters.get("yearOfAccount")));

}

if (filters.get("country") != null && filters.get("country").toString() != "") {

predicates.add(builder.equal(root.<OrganisationReference>get("insured")

.<CountryReference>get("countryId").<Long>get("id"),

Long.parseLong((String) filters.get("country"))));

}

if (filters.get("countryName") != null && filters.get("countryName").toString() != "") {

predicates.add(builder.like(root.<OrganisationReference>get("insured")

.<CountryReference>get("countryId").<String>get("name"),

(String) filters.get("countryName")));

}

if (filters.get("insured") != null && filters.get("insured").toString() != "") {

predicates.add(builder.like(root.<OrganisationReference>get("insured").<String>get("fullName"),

Utils.prepareStringForSQLLike(filters.get("insured").toString())));

}

if (filters.get("broker") != null && filters.get("broker").toString() != "") {

predicates.add(builder.like(root.<OrganisationReference>get("broker").<String>get("fullName"),

"%" + Utils.prepareStringForSQLLike(filters.get("broker").toString()) + "%"));

}

if (filters.get("rAGcolorID") != null && filters.get("rAGcolorID").toString() != "") {

predicates.add(builder.equal(root.<RAGColour>get("rAGcolourID").<String>get("colour"),

(String) filters.get("rAGcolorID")));

}

if (filters.get("renewalDateAfter") != null) {

predicates.add(builder.greaterThanOrEqualTo(root.<Date>get("renewalDate"),

toDate((String) filters.get("renewalDateAfter"))));

}

if (filters.get("renewalDateBefore") != null) {

predicates.add(builder.lessThanOrEqualTo(root.<Date>get("renewalDate"),

toDate((String) filters.get("renewalDateBefore"))));

}

if (filters.get("startDateAfter") != null) {

predicates.add(builder.greaterThanOrEqualTo(root.<Date>get("startDate"),

toDate((String) filters.get("startDateAfter"))));

}

if (filters.get("startDateBefore") != null) {

predicates.add(builder.lessThanOrEqualTo(root.<Date>get("startDate"),

toDate((String) filters.get("startDateBefore"))));

}

if (filters.get("colour") != null && ((String) filters.get("accountPolicyHolder")) != "") {

predicates.add(builder.like(root.<RAGColour>get("rAGcolourID").<String>get("colour"),

"%" + ((String) filters.get("colour")) + "%"));

}

if (filters.get("otherPolicyReferences") != null

&& filters.get("otherPolicyReferences").toString() != "") {

predicates.add(builder.like(root.<String>get("otherPolicyRefs"),

"%" + Utils.prepareStringForSQLLike(filters.get("otherPolicyReferences").toString())

+ "%"));

}

if (filters.get("engineer") != null && filters.get("engineer").toString() != "") {

predicates.add(builder.like(root.<PolicyReference>get("engineers").<String>get("name"),

"%" + filters.get("engineer").toString() + "%"));

}

if (filters.get("underwriter") != null && filters.get("underwriter").toString() != "") {

predicates.add(builder.like(root.<PolicyReference>get("underwriters").<String>get("name"),

"%" + filters.get("underwriter").toString() + "%"));

}

if (filters.get("policyStatus") != null && filters.get("policyStatus").toString() != "") {

predicates.add(

builder.equal(root.<PolicyStatus>get("policyStatusID").<Integer>get("policyStatusID"),

Integer.parseInt((String) filters.get("policyStatus"))));

}

if (filters.get("entryType") != null && filters.get("entryType").toString() != "") {

predicates.add(builder.equal(root.<EntryType>get("entryTypeID").<Long>get("entryTypeId"),

(Long) filters.get("entryType")));

}

// Policy or Enquiry option

if (filters.get("enquiry") != null && filters.get("enquiry").toString() != "") {

predicates.add(builder.equal(root.<Boolean>get("enquiry"),

Boolean.valueOf(filters.get("enquiry").toString())));

}

// insuranceStructure

if (filters.get("insuranceStructure") != null

&& filters.get("insuranceStructure").toString() != "") {

predicates.add(builder.like(

root.<InsuranceStructure>get("insuranceStructureID")

.<String>get("insuranceStructureName"),

"%" + filters.get("insuranceStructure").toString() + "%"));

}

return builder.and(predicates.toArray(new Predicate[predicates.size()]));

}

private Date toDate(String date) {

SimpleDateFormat format = new SimpleDateFormat("yyyy/MM/dd");

java.util.Date parsed;

try {

parsed = format.parse(date);

} catch (ParseException e) {

e.printStackTrace();

return null;

}

return new java.sql.Date(parsed.getTime());

}

};

return this.repository.findAll(specification);

} else {

return this.repository.findAll();

}

}

@Override

public List<CoverageDTO> getAllCoverages() {

List<Coverage> coverages = coverageRepository.findAllByOrderByCoverageName();

List<CoverageDTO> coveragesDTO = ListMapper.mapList(coverages, mapper, CoverageDTO.class);

for (CoverageDTO coverageDTO : coveragesDTO) {

if (coverageDTO.getAddOnID() != null) {

coverageDTO.getAddOnID().sort(Comparator.comparing(AddOnDTO::getName));

}

}

return coveragesDTO;

}

@Override

public List<EntryTypeDTO> getAllEntryTypes() {

List<EntryType> entryTypes = entryTypeRepository.findAll();

List<EntryTypeDTO> entryTypesDTO = ListMapper.mapList(entryTypes, mapper, EntryTypeDTO.class);

return entryTypesDTO;

}

@Override

@Cacheable(value = CacheConstants.Values.INSURANCE\_STRUCTURE\_ALL)

public List<InsuranceStructureDTO> getInsuranceStructures() {

List<InsuranceStructure> insuranceStructures = insuranceStructureRepository

.findAllByOrderByInsuranceStructureNameAsc();

List<InsuranceStructureDTO> insuranceStructuresDTO = ListMapper.mapList(insuranceStructures, mapper,

InsuranceStructureDTO.class);

return insuranceStructuresDTO;

}

@Override

public List<DeductionTypeDTO> getDeductionType() {

List<DeductionType> deductionTypes = deductionTypeRepository.findAllByOrderByNameAsc();

List<DeductionTypeDTO> deductionTypesDTO = ListMapper.mapList(deductionTypes, mapper, DeductionTypeDTO.class);

return deductionTypesDTO;

}

private PolicyDTO convertPolicyEntityToDtoThin(Policy policy) {

if (policy == null) {

LOGGER.error("ERROR - PocilyServiceImple - convertPolicyEntityToDtoThin() -> invalid input.");

return null;

}

// Map only needed vars!

PolicyDTO copy = new PolicyDTO();

copy.setId(policy.getId());

copy.setPolicyNumber(policy.getPolicyNumber());

copy.setYearOfAccount(policy.getYearOfAccount());

copy.setReinsuranceOut(policy.getReinsuranceOut());

copy.setStandardNonStandard(policy.getStandardNonStandard());

copy.setIpeIncomeAmount(policy.getIpeIncomeAmount());

copy.setStartDate(policy.getStartDate());

copy.setEndDate(policy.getEndDate());

copy.setnRILead(policy.getnRILead());

copy.setTotalReinsurerPctOf100(policy.getTotalReinsurerPctOf100());

copy.setTotalReinsurancePct(policy.getTotalReinsurancePct());

copy.setNriGrossShare(policy.getNriGrossShare());

copy.setEnquiry(policy.getEnquiry());

copy.setUniqueIdentifier(policy.getUniqueIdentifier());

copy.setOtherPolicyRefs(policy.getOtherPolicyRefs());

copy.setTotalReinsurerPctOf100(policy.getTotalReinsurerPctOf100());

copy.setTotalCedantShare(policy.getTotalCedantShare());

copy.setCoverageID(

policy.getCoverageID() != null ? mapper.map(policy.getCoverageID(), CoverageDTO.class) : null);

// Used dozer to map Insurance Structure - its simple info to copy

copy.setInsuranceStructureID(policy.getInsuranceStructureID() != null

? mapper.map(policy.getInsuranceStructureID(), InsuranceStructureDTO.class) : null);

// Used dozer to map Location - its simple info to copy

copy.setLocationId(policy.getLocationId() != null

? ListMapper.mapList(policy.getLocationId(), mapper, LocationReferenceDTO.class) : null);

copy.setStationId(policy.getStationId() != null

? ListMapper.mapList(policy.getStationId(), mapper, StationReferenceDTO.class) : null);

// Insured

if (policy.getInsured() != null) {

OrganisationReferenceDTO insured = new OrganisationReferenceDTO();

insured.setId(policy.getInsured().getId());

insured.setnRIalias(policy.getInsured().getnRIalias());

insured.setStatus(policy.getInsured().getStatus());

insured.setFullName(policy.getInsured().getFullName());

insured.setCountryId(policy.getInsured().getCountryId() != null

? mapper.map(policy.getInsured().getCountryId(), CountryReferenceDTO.class) : null);

copy.setInsured(insured);

}

// Policy Shares

if (policy.getPolicyShareID() != null && !policy.getPolicyShareID().isEmpty()) {

List<PolicySharesDTO> mappedPolicies = new ArrayList<>();

for (PolicyShares shares : policy.getPolicyShareID()) {

PolicySharesDTO shareCopy = null;

if (shares.getOrganisationID() != null) {

shareCopy = new PolicySharesDTO();

OrganisationReferenceDTO orgRefCopy = new OrganisationReferenceDTO();

orgRefCopy.setnRIalias(shares.getOrganisationID().getnRIalias());

orgRefCopy.setId(shares.getOrganisationID().getId());

shareCopy.setOrganisationID(orgRefCopy);

shareCopy.setReinsurerPct(shares.getReinsurerPct());

shareCopy.setShare2(shares.getShare2());

shareCopy.setShare100Percent(shares.getShare100Percent());

shareCopy.setInsurerRole(shares.getInsurerRole());

mappedPolicies.add(shareCopy);

}

}

copy.setPolicyShareID(mappedPolicies);

}

// Deductible

if (policy.getDeductibleID() != null && !policy.getDeductibleID().isEmpty()) {

List<DeductibleDTO> deducCopyList = new ArrayList<>();

DeductibleDTO deducCopy = null;

for (Deductible item : policy.getDeductibleID()) {

deducCopy = new DeductibleDTO();

deducCopy.setId(item.getId());

if (item.getDeductibleTypeID() != null) {

DeductibleTypeDTO decTypeCopy = new DeductibleTypeDTO();

decTypeCopy.setId(item.getDeductibleTypeID().getId());

decTypeCopy.setName(item.getDeductibleTypeID().getName());

deducCopy.setDeductibleTypeID(decTypeCopy);

deducCopy.setCurrencyID(mapper.map(item.getCurrencyID(), CurrencyReferenceDTO.class));

}

else {

deducCopy.setCurrencyID(mapper.map(item.getCurrencyID(), CurrencyReferenceDTO.class));

String deductibleComment = item.getDeductibleComment();

if(deductibleComment != null) {

deducCopy.setDeductibleComment(deductibleComment);

}

}

// Amount

Double amount = item.getAmount();

if (amount != null) {

deducCopy.setAmount(amount);

}

String deductibleComment = item.getDeductibleComment();

if(deductibleComment != null) {

deducCopy.setDeductibleComment(deductibleComment);

}

deducCopyList.add(deducCopy);

}

copy.setDeductibleID(deducCopyList);

}

// Currency

if (policy.getCurrencyID() != null) {

CurrencyReferenceDTO currency = new CurrencyReferenceDTO();

currency.setId(policy.getCurrencyID().getId());

currency.setCode(policy.getCurrencyID().getCode());

currency.setName(policy.getCurrencyID().getName());

copy.setCurrencyID(currency);

}

return copy;

}

@Override

@Cacheable(value = CacheConstants.Values.DEDUCTION\_NAME\_ALL)

public List<DeductionNameDTO> getDeductionName() {

List<DeductionName> deductionNames = deductionNameRepository.findAllByOrderByName();

List<DeductionNameDTO> deductionNameDTO = ListMapper.mapList(deductionNames, mapper, DeductionNameDTO.class);

return deductionNameDTO;

}

@Override

public List<EntryTypeDTO> getEntryTypesByName(HashMap<String, Object> filters) {

List<EntryTypeDTO> result = new ArrayList<>();

List<EntryType> fetched = null;

if (filters != null && !filters.isEmpty() && filters.keySet() != null && filters.values() != null

&& filters.get("entryTypeName") != null && !filters.get("entryTypeName").toString().isEmpty()) {

fetched = entryTypeRepository.findEntryTypesByName(filters.get("entryTypeName").toString());

} else {

fetched = entryTypeRepository.findAllByOrderByEntryTypeName();

}

if (fetched != null && !fetched.isEmpty()) {

EntryTypeDTO copy = null;

for (EntryType type : fetched) {

if (type != null) {

copy = mapper.map(type, EntryTypeDTO.class);

if (type.getEntryTypeDescription() != null && !type.getEntryTypeDescription().isEmpty()) {

copy.setEntrySubType(type.getEntrySubType() + " - " + type.getEntryTypeDescription());

}

result.add(copy);

}

}

}

return result;

}

@Override

public String getLastPolicyUniqueIdentifier() {

String lastUniqIdentf = repository.getPolicyLastUniqueIdentifier();

return lastUniqIdentf;

}

public PolicyPieChartDTO getPoliciesOverviewOld(Boolean showAll) {

PolicyPieChartDTO result = new PolicyPieChartDTO();

result.setTotal(0);

List<RAGColour> policiesColours = null;

if (showAll != null && showAll) {

policiesColours = repository.findActivePoliciesColours();

} else {

String userId = (String) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

if (userId != null && !userId.isEmpty()) {

policiesColours = repository.findActivePoliciesColoursByHolderUser(userId);

}

}

if (policiesColours != null && !policiesColours.isEmpty()) {

result.setTotal(policiesColours.size());

for (RAGColour item : policiesColours) {

if (item != null) {

if (!result.getPoliciesByColor().isEmpty()

&& result.getPoliciesByColor().get(item.getColour()) != null) {

result.getPoliciesByColor().put(item.getColour(),

(result.getPoliciesByColor().get(item.getColour()) + 1));

} else {

result.getPoliciesByColor().put(item.getColour(), 1);

}

}

}

}

return result;

}

@Override

public PolicyPieChartDTO getPoliciesOverview(Boolean showAll) {

PolicyPieChartDTO result = new PolicyPieChartDTO();

result.setTotal(0);

// List all possible colours

List<String> colours = new ArrayList<String>();

colours.add("GREEN");

colours.add("YELLOW");

colours.add("RED");

LocalDate currentDate = LocalDate.now();

Integer yearOfAccount = currentDate.getYear();

Integer totalMonthPolicies = null;

int size = 0;

LocalDate endDate = null;

LocalDate startDate = null;

for (String colour : colours) {

List<String> statusList = new ArrayList<String>();

if (colour.equals("GREEN")) {

statusList.add("POLICY\_IN\_FORCE");

statusList.add("RENEWED");

} else if (colour.equals("YELLOW")) {

statusList.add("ENQUIRY");

statusList.add("BOUND");

} else if (colour.equals("RED")) {

statusList.add("EXPIRED");

statusList.add("CANCELLED");

statusList.add("DECLINED");

statusList.add("NON\_TAKEN\_UP");

}

startDate = LocalDate.of(yearOfAccount, Month.JANUARY, 1);

endDate = LocalDate.of(yearOfAccount, Month.DECEMBER, 31);

if (showAll != null && showAll) {

totalMonthPolicies = repository

.countMonthlyPoliciesByUnderwritersYearMonthAndColour(

java.sql.Timestamp.valueOf(startDate.atStartOfDay()), java.sql.Timestamp.valueOf(endDate.atTime(23, 59, 59)), statusList);

} else {

String userId = (String) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

if (userId != null && !userId.isEmpty()) {

totalMonthPolicies = repository.countMonthlyPoliciesByUserYearMonthAndColour(

userId, java.sql.Timestamp.valueOf(startDate.atStartOfDay()), java.sql.Timestamp.valueOf(endDate.atTime(23, 59, 59)), statusList);

}

}

size += totalMonthPolicies;

if (!result.getPoliciesByColor().isEmpty() && result.getPoliciesByColor().get(colour) != null) {

result.getPoliciesByColor().put(colour, (result.getPoliciesByColor().get(colour) + totalMonthPolicies));

} else {

result.getPoliciesByColor().put(colour, totalMonthPolicies);

}

}

result.setTotal(size);

return result;

}

@Override

public PolicyLineChartDTO getPoliciesEvolution(Integer yearOfAccount, Boolean showAll) {

PolicyLineChartDTO result = new PolicyLineChartDTO();

// Fetch all possible colours

List<String> colours = rAGColourRepository.getAllColourCodes();

if (colours == null || colours.isEmpty()) {

LOGGER.error("Error on PolicyServiceImpl - getPoliciesEvolution() -> no colours found on Database");

return result;

}

LocalDate today = LocalDate.now();

int thisMonth = today.getMonthValue();

int thisYear = today.getYear();

int month = 1; // Need to start with 1 because the DB dates have a month

// range between 1 and 12

Integer totalMonthPolicies = null;

String userId = (String) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

List<Integer> monthlyValuesPerColour = null;

LocalDate endDate = null;

for (String colour : colours) {

// Clear support values

monthlyValuesPerColour = new ArrayList<>();

month = 1;

// Go through the months

while ((yearOfAccount != thisYear && month <= 12) || (yearOfAccount == thisYear && month <= thisMonth)) {

endDate = LocalDate.of(yearOfAccount, Month.of(month), 1).with(TemporalAdjusters.lastDayOfMonth());

if (showAll != null && showAll) {

totalMonthPolicies = policyStatusEvolutionRepository

.countMonthlyPoliciesByUnderwritersYearMonthAndColour(

java.sql.Timestamp.valueOf(endDate.atStartOfDay().plusDays(1)), colour);

} else {

if (userId != null && !userId.isEmpty()) {

totalMonthPolicies = policyStatusEvolutionRepository

.countMonthlyPoliciesByUserYearMonthAndColour(userId,

java.sql.Timestamp.valueOf(endDate.atStartOfDay().plusDays(1)), colour);

}

}

if (totalMonthPolicies == null) {

totalMonthPolicies = 0;

}

monthlyValuesPerColour.add(totalMonthPolicies);

month++;

}

while (month <= 12) {

monthlyValuesPerColour.add(null);

month++;

}

// Add values by colour

result.getColorBindedPolicyTotalsByMonth().put(colour, monthlyValuesPerColour);

}

result.setAvailableDataColours(colours);

return result;

}

@Override

public PolicyTimelineDTO getPoliciesTimeline() {

PolicyTimelineDTO result = new PolicyTimelineDTO();

String userId = (String) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

// We need next year here because the inception date is: Inception Date

// = Renewal Date - 1 Year

// So in this case we need to add a year to make a valid comparison

// Calendar cal = Calendar.getInstance();

// cal.add(Calendar.YEAR, 1); // to get previous year add -1

// Date nextYear = (Date) cal.getTime();

Date today = new Date();

// sort by: Red, Yellow, Green.

Sort sort = new Sort(Sort.Direction.DESC, "rAGcolourID.colour");

result.setUntilInceptionPolicies(repository.findPoliciesBeforeRenewal(userId, today));

result.setAfterInceptionPolicies(repository.findPoliciesOnAndAfterRenewal(userId, today, sort));

return result;

}

@Override

@Caching(evict = { @CacheEvict(value = CacheConstants.Values.POLICY\_NUMBER\_ALL, allEntries = true) })

public PolicyDTO updatePolicy(PolicyDTO originalPolicyDTO) {

final String userId = (String) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

if (originalPolicyDTO == null || userId == null || userId.isEmpty()) {

LOGGER.error("ERROR - PolicyServiceImpl - updatePolicy() -> Input or current user is invalid. input / user"

+ originalPolicyDTO + " / " + userId);

return null;

}

PolicyForSave modifiedPolicy = repositoryForSave.findById(originalPolicyDTO.getId());

if (modifiedPolicy == null) {

LOGGER.error("ERROR - PolicyServiceImpl - updatePolicy() - could not find Policy by id given: "

+ originalPolicyDTO.getId());

return null;

}

// get allocations for journals

List<PremiumReceivedReference> allocations = modifiedPolicy.getPremiumReceivedID();

// ---------- Booleans ------------

PolicyDTO policyDTO = fixPolicyBooleans(originalPolicyDTO);

Boolean sactionsBeforeUpdate = modifiedPolicy.getSanctionsValidation();

// Policy must be copy before any change

final PolicyForSave oldPolicy = mapper.map(modifiedPolicy, PolicyForSave.class);

modifiedPolicy.setActualPrice(policyDTO.getActualPrice());

modifiedPolicy.setBrokerageAmount(policyDTO.getBrokerageAmount());

modifiedPolicy.setAnnualAggregateLimitPerLocation(policyDTO.getAnnualAggregateLimitPerLocation());

modifiedPolicy.setAnnualAggregateLimitPerPolicy(policyDTO.getAnnualAggregateLimitPerPolicy());

modifiedPolicy.setaPTP(policyDTO.getaPTP());

modifiedPolicy.setAttachmentPoint(policyDTO.getAttachmentPoint());

modifiedPolicy.setAutomaticReinstatement(policyDTO.getAutomaticReinstatement());

modifiedPolicy.setBrokerType(policyDTO.getBrokerType());

modifiedPolicy.setCanceledPolicy(policyDTO.getCanceledPolicy());

modifiedPolicy.setCanceledPolicyDate(policyDTO.getCanceledPolicyDate());

modifiedPolicy.setCedantShare1(policyDTO.getCedantShare1());

modifiedPolicy.setCedantShare2(policyDTO.getCedantShare2());

modifiedPolicy.setEndDate(policyDTO.getEndDate());

modifiedPolicy.setEndorsementNumber(policyDTO.getEndorsementNumber());

modifiedPolicy.setEntryTypeComment(policyDTO.getEntryTypeComment());

modifiedPolicy.setExcessLayer(policyDTO.getExcessLayer());

modifiedPolicy.setExposureAboveAttachmentPoint(policyDTO.getExposureAboveAttachmentPoint());

modifiedPolicy.setBrokeragePct(policyDTO.getBrokeragePct());

modifiedPolicy.setFeePercentageOfPercentage(policyDTO.getFeePercentageOfPercentage());

modifiedPolicy.setGroundUpCover(policyDTO.getGroundUpCover());

modifiedPolicy.setIpeIncomeAmount(policyDTO.getIpeIncomeAmount());

modifiedPolicy.setIpeIncomePct(policyDTO.getIpeIncomePct());

modifiedPolicy.setiPEincomeGBP(policyDTO.getiPEincomeGBP());

modifiedPolicy.setIpeExpenseAmount(policyDTO.getIpeExpenseAmount());

modifiedPolicy.setIpeExpensePct(policyDTO.getIpeExpensePct());

modifiedPolicy.setLifetimeLimit(policyDTO.getLifetimeLimit());

modifiedPolicy.setLimitPerOcurrenceFor100(policyDTO.getLimitPerOcurrenceFor100());

modifiedPolicy.setLimitPerOcurrenceFor100GBP(policyDTO.getLimitPerOcurrenceFor100GBP());

modifiedPolicy.setMinimumDeductiblesForThePolicy(policyDTO.getMinimumDeductiblesForThePolicy());

modifiedPolicy.setMinimumDeductiblesForThePolicyGBP(policyDTO.getMinimumDeductiblesForThePolicyGBP());

modifiedPolicy.setnCBType(policyDTO.getnCBType());

modifiedPolicy.setUpfrontNcbPct(policyDTO.getUpfrontNcbPct());

modifiedPolicy.setUpfrontNcbAmount(policyDTO.getUpfrontNcbAmount());

modifiedPolicy.setDefferedNcbPct(policyDTO.getDefferedNcbPct());

modifiedPolicy.setDefferedNcbAmount(policyDTO.getDefferedNcbAmount());

modifiedPolicy.setBrokeragePct(policyDTO.getBrokeragePct());

modifiedPolicy.setBrokerageAmount(policyDTO.getBrokerageAmount());

modifiedPolicy.setNetExposure(policyDTO.getNetExposure());

modifiedPolicy.setNoAggregateLimit(policyDTO.getNoAggregateLimit());

modifiedPolicy.setNonNRIshareAmount(policyDTO.getNonNRIshareAmount());

modifiedPolicy.setNonNRIsharePercentage(policyDTO.getNonNRIsharePercentage());

modifiedPolicy.setNriGrossShare(policyDTO.getNriGrossShare());

modifiedPolicy.setnRIGrossShare100\_1(policyDTO.getnRIGrossShare100\_1());

modifiedPolicy.setTotalOverallPoolShare(policyDTO.getTotalOverallPoolShare());

modifiedPolicy.setnRILead(policyDTO.getnRILead());

modifiedPolicy.setnRINetMaximumExposure(policyDTO.getnRINetMaximumExposure());

modifiedPolicy.setnRINetMaximumExposureGBP(policyDTO.getnRINetMaximumExposureGBP());

modifiedPolicy.setnRInetPremium(policyDTO.getnRInetPremium());

modifiedPolicy.setnRIPremiumDebt(policyDTO.getnRIPremiumDebt());

modifiedPolicy.setNumberOfInstalments(policyDTO.getNumberOfInstalments());

modifiedPolicy.setReinsuranceInFrom(policyDTO.getReinsuranceInFrom());

modifiedPolicy.setOtherPolicyRefs(policyDTO.getOtherPolicyRefs());

modifiedPolicy.setPeriod(policyDTO.getPeriod());

modifiedPolicy.setPlanGrossPremiumGBP(policyDTO.getPlanGrossPremiumGBP());

modifiedPolicy.setPlanIPEIncome(policyDTO.getPlanIPEIncome());

modifiedPolicy.setPlanMonth(policyDTO.getPlanMonth());

modifiedPolicy.setPlanNetPremiumGBP(policyDTO.getPlanNetPremiumGBP());

modifiedPolicy.setPlanRateOfExchange(policyDTO.getPlanRateOfExchange());

modifiedPolicy.setPremiumAfterBrokerageBeforeDeductions(policyDTO.getPremiumAfterBrokerageBeforeDeductions());

modifiedPolicy.setPremiumAfterIPEIn(policyDTO.getPremiumAfterIPEIn());

modifiedPolicy.setPremiumAfterIPEOut(policyDTO.getPremiumAfterIPEOut());

modifiedPolicy.setPremiumAfterNCB(policyDTO.getPremiumAfterNCB());

modifiedPolicy.setPremiumAfterOtherDeductions(policyDTO.getPremiumAfterOtherDeductions());

modifiedPolicy.setGrossPremium100Pct(policyDTO.getGrossPremium100Pct());

modifiedPolicy.setPremiumAfterIPE\_1(policyDTO.getPremiumAfterIPE\_1());

modifiedPolicy.setQuoteNumber(policyDTO.getQuoteNumber());

modifiedPolicy.setRateChange(policyDTO.getRateChange());

modifiedPolicy.setReinsuranceOut(policyDTO.getReinsuranceOut());

modifiedPolicy.setRenewalDate(policyDTO.getRenewalDate());

modifiedPolicy.setrOE(policyDTO.getrOE());

modifiedPolicy.setStandardNonStandard(policyDTO.getStandardNonStandard());

modifiedPolicy.setStartDate(policyDTO.getStartDate());

modifiedPolicy.setTechnicalPrice(policyDTO.getTechnicalPrice());

modifiedPolicy.settIV(policyDTO.gettIV());

modifiedPolicy.setTotalCedantShare(policyDTO.getTotalCedantShare());

modifiedPolicy.setTotalPoolInsuranceShare(policyDTO.getTotalPoolInsuranceShare());

modifiedPolicy.setTotalReinsurerPctOf100(policyDTO.getTotalReinsurerPctOf100());

modifiedPolicy.setTotalReinsurancePct(policyDTO.getTotalReinsurancePct());

modifiedPolicy.setnCBPct(policyDTO.getnCBPct());

modifiedPolicy.setUniqueIdentifier(policyDTO.getUniqueIdentifier());

modifiedPolicy.setEnquiry(policyDTO.getEnquiry());

modifiedPolicy.setNumberOfReinstatements(policyDTO.getNumberOfReinstatements());

modifiedPolicy.setAuthoriseCapacity(policyDTO.getAuthoriseCapacity());

modifiedPolicy.setBiSeparateLimit(policyDTO.getBiSeparateLimit());

modifiedPolicy.setAdditionalInformation(policyDTO.getAdditionalInformation());

modifiedPolicy.setSpecifyForm(policyDTO.getSpecifyForm());

modifiedPolicy.setSanctionsValidation(policyDTO.getSanctionsValidation());

modifiedPolicy.setInstalmentDate(policyDTO.getInstalmentDate());

modifiedPolicy.setiPEAndOtherFeesToNRI(policyDTO.getiPEAndOtherFeesToNRI());

modifiedPolicy

.setnRIGrossSharePremiumAfterNCBBrokerageFees(policyDTO.getnRIGrossSharePremiumAfterNCBBrokerageFees());

modifiedPolicy.setnRIGSAfterDeductionPrimaryCeding(policyDTO.getnRIGSAfterDeductionPrimaryCeding());

modifiedPolicy.setnRIGShareAfterIPEAndOtherFeesToNRI(policyDTO.getnRIGShareAfterIPEAndOtherFeesToNRI());

modifiedPolicy.setnRIGSPremiumAfterUpfrontDeferredNCB(policyDTO.getnRIGSPremiumAfterUpfrontDeferredNCB());

modifiedPolicy.setnRIGSPremiumAfterUpfrontNCB(policyDTO.getnRIGSPremiumAfterUpfrontNCB());

modifiedPolicy.setNriNetPremiumBeforeIpe(policyDTO.getNriNetPremiumBeforeIpe());

modifiedPolicy.setPlanGrossPremiumGBP(policyDTO.getPlanGrossPremiumGBP());

modifiedPolicy.setAnnualAggregateLimit(policyDTO.getAnnualAggregateLimit());

modifiedPolicy.setAnnualAggregateValue(policyDTO.getAnnualAggregateValue());

modifiedPolicy.setOverallPolicyLimit(policyDTO.getOverallPolicyLimit());

modifiedPolicy.setOverallPolicyLimitGBP(policyDTO.getOverallPolicyLimitGBP());

modifiedPolicy.setrOEDate(policyDTO.getrOEDate());

modifiedPolicy.setCedingPoolReference(policyDTO.getCedingPoolReference());

modifiedPolicy.setNriNetSharePct(policyDTO.getNriNetSharePct());

modifiedPolicy.setNriGrossPremiumShare(policyDTO.getNriGrossPremiumShare());

// ----------- Insured Structure -----------

modifiedPolicy.setInsuranceStructureID((policyDTO.getInsuranceStructureID() == null

|| policyDTO.getInsuranceStructureID().getInsuranceStructureID() == null) ? null

: insuranceStructureRepository

.findOne(policyDTO.getInsuranceStructureID().getInsuranceStructureID()));

// Policy Status & enquiry

if (!Objects.isNull(policyDTO.getPolicyStatusID()) && policyDTO.getPolicyStatusID().getPolicyStatusID() != 1) {

if (policyDTO.getPolicyStatusID() == null || policyDTO.getPolicyStatusID().getPolicyStatusID() == null) {

if (modifiedPolicy.getInsuranceStructureID() != null) {

if (EnumInsuranceStructure.REINSURANCE\_INWARDS.name()

.equals(modifiedPolicy.getInsuranceStructureID().getInsuranceStructureCode())) {

modifiedPolicy.setPolicyStatusID(policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

modifiedPolicy.setStatusModificationTime(new Date());

} else {

modifiedPolicy.setPolicyStatusID(policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

modifiedPolicy.setStatusModificationTime(new Date());

}

}

}

modifiedPolicy.setEnquiry(false);

} else {

modifiedPolicy.setPolicyStatusID(policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

modifiedPolicy.setEnquiry(true);

}

if (!Objects.isNull(policyDTO.getPolicyStatusID()) && policyDTO.getPolicyStatusID().getPolicyStatusID() != 1

&& modifiedPolicy.getQuoteNumber() == null) {

modifiedPolicy.setQuoteNumber(DEFAULT\_QUOTE\_NR);

}

// Colour

if (policyDTO.getrAGcolourID() != null && policyDTO.getrAGcolourID().getId() != null) {

modifiedPolicy.setrAGcolourID(rAGColourRepository.findOne(policyDTO.getrAGcolourID().getId()));

} else {

if (!Objects.isNull(policyDTO.getPolicyStatusID())

&& policyDTO.getPolicyStatusID().getPolicyStatusID() != 1) {

modifiedPolicy.setrAGcolourID(rAGColourRepository.findByColourCode(EnumRagColour.GREEN.name()));

} else {

modifiedPolicy.setrAGcolourID(null);

}

}

// User

if (policyDTO.getUserID() != null && policyDTO.getUserID().getId() != null

&& !policyDTO.getUserID().getId().isEmpty()) {

modifiedPolicy.setUserID(userRepository.findOne(policyDTO.getUserID().getId()));

}

modifiedPolicy.setModifiedByUser(userRepository.findOne(userId));

// Year of Account

if (policyDTO.getEndorsementNumber() == null || policyDTO.getEndorsementNumber().equalsIgnoreCase("0")) {

if (policyDTO.getStartDate() != null) {

modifiedPolicy.setYearOfAccount(DateUtils.getYear(policyDTO.getStartDate()));

} else {

modifiedPolicy.setYearOfAccount(null);

}

}

// --------- Basic Details -----------

// ----------- Insured -----------

modifiedPolicy.setInsured((policyDTO.getInsured() == null || policyDTO.getInsured().getId() == null) ? null

: organisationReferenceRepository.findOne(policyDTO.getInsured().getId()));

// ----------- Entry type -----------

modifiedPolicy.setEntryTypeID(

(policyDTO.getEntryTypeID() == null || policyDTO.getEntryTypeID().getEntryTypeId() == null) ? null

: entryTypeRepository.findOne(policyDTO.getEntryTypeID().getEntryTypeId()));

// First Year of Attachment

modifiedPolicy.setFirstYearOfAttachment(modifiedPolicy.getEntryTypeID() != null

&& modifiedPolicy.getEntryTypeID().getEntryTypeName().equalsIgnoreCase(NEWBUSINESS)

? (modifiedPolicy.getYearOfAccount() == null ? null : modifiedPolicy.getYearOfAccount())

: (policyDTO.getFirstYearOfAttachment() == null ? null : policyDTO.getFirstYearOfAttachment()));

// ----------- Users -----------

if (policyDTO.getUsers() != null) {

List<UserReference> users = new ArrayList<UserReference>();

policyDTO.getUsers().forEach((user) -> {

if (user.getId() != null) {

users.add(userRepository.findOne(user.getId()));

}

});

if (modifiedPolicy.getUsers() != null) {

modifiedPolicy.getUsers().clear();

modifiedPolicy.getUsers().addAll(users);

} else {

modifiedPolicy.setUsers(users);

}

} else {

if (modifiedPolicy.getUsers() != null) {

modifiedPolicy.getUsers().clear();

}

}

// --------- Addons Id -----------

if (policyDTO.getAddOnID() != null) {

List<AddOn> addOns = new ArrayList<AddOn>();

policyDTO.getAddOnID().forEach((addOn) -> {

if (addOn.getId() != null) {

addOns.add(addOnRepository.findOne(addOn.getId()));

}

});

if (modifiedPolicy.getAddOnID() != null) {

modifiedPolicy.getAddOnID().clear();

modifiedPolicy.getAddOnID().addAll(addOns);

} else {

modifiedPolicy.setAddOnID(addOns);

}

} else {

if (modifiedPolicy.getAddOnID() != null) {

modifiedPolicy.getAddOnID().clear();

}

}

// --------- Location -----------

if (policyDTO.getLocationId() != null) {

List<LocationReference> locations = new ArrayList<LocationReference>();

policyDTO.getLocationId().forEach((location) -> {

if (location.getId() != null) {

locations.add(locationReferenceRepository.findOne(location.getId()));

}

});

if (modifiedPolicy.getLocationId() != null) {

modifiedPolicy.getLocationId().clear();

modifiedPolicy.getLocationId().addAll(locations);

} else {

modifiedPolicy.setLocationId(locations);

}

} else {

if (modifiedPolicy.getLocationId() != null) {

modifiedPolicy.getLocationId().clear();

}

}

// --------- Station -----------

if (policyDTO.getStationId() != null) {

List<StationReference> stations = new ArrayList<StationReference>();

policyDTO.getStationId().forEach((station) -> {

if (station.getId() != null) {

stations.add(stationReferenceRepository.findOne(station.getId()));

}

});

if (modifiedPolicy.getStationId() != null) {

modifiedPolicy.getStationId().clear();

modifiedPolicy.getStationId().addAll(stations);

} else {

modifiedPolicy.setStationId(stations);

}

} else {

if (modifiedPolicy.getStationId() != null) {

modifiedPolicy.getStationId().clear();

}

}

// --------- Structure -----------

// ----------- Coverage -----------

modifiedPolicy

.setCoverageID((policyDTO.getCoverageID() == null || policyDTO.getCoverageID().getCoverageID() == null)

? null : coverageRepository.findOne(policyDTO.getCoverageID().getCoverageID()));

// ----------- Broker -----------

modifiedPolicy.setBroker((policyDTO.getBroker() == null || policyDTO.getBroker().getId() == null) ? null

: organisationReferenceRepository.findOne(policyDTO.getBroker().getId()));

// ----------- Broker Contact -----------

modifiedPolicy.setBrokerContact(

(policyDTO.getBrokerContact() == null || policyDTO.getBrokerContact().getContactDetailID() == null)

? null

: contactDetailsReferenceRepository.findOne(policyDTO.getBrokerContact().getContactDetailID()));

// ----------- Pool Insurer -----------

modifiedPolicy.setPoolInsurer((policyDTO.getPoolInsurer() == null || policyDTO.getPoolInsurer().getId() == null)

? null : organisationReferenceRepository.findOne(policyDTO.getPoolInsurer().getId()));

// ----------- Non Pool Insurer -----------

modifiedPolicy.setNonPoolInsurer(

(policyDTO.getNonPoolInsurer() == null || policyDTO.getNonPoolInsurer().getId() == null) ? null

: organisationReferenceRepository.findOne(policyDTO.getNonPoolInsurer().getId()));

// ----------- Policy Shares -----------

if (policyDTO.getPolicyShareID() != null) {

List<PolicySharesShortReference> policyShares = new ArrayList<>();

policyDTO.getPolicyShareID().forEach((policyShare) -> {

if (policyShare.getPolicyID() == null) {

policyShare.setPolicyID(policyDTO);

}

if (policyShare.getPolicySharesID() == null) {

// Create

PolicySharesDTO policySharesDTO = policyShareService.createPolicyShares(policyShare);

policyShares.add(policySharesShortReferenceRepository.findOne(policySharesDTO.getPolicySharesID()));

} else {

policyShareService.updatePolicyShares(policyShare);

policyShares.add(policySharesShortReferenceRepository.findOne(policyShare.getPolicySharesID()));

}

});

if (modifiedPolicy.getPolicyShareID() != null) {

modifiedPolicy.getPolicyShareID().clear();

modifiedPolicy.getPolicyShareID().addAll(policyShares);

} else {

modifiedPolicy.setPolicyShareID(policyShares);

}

} else {

if (modifiedPolicy.getPolicyShareID() != null) {

modifiedPolicy.getPolicyShareID().clear();

}

}

// ----------- Related Policy -----------

if (policyDTO.getRelatedPolicyID() != null) {

List<RelatedPolicyForView> policies = new ArrayList<RelatedPolicyForView>();

policyDTO.getRelatedPolicyID().forEach((pol) -> {

if (pol.getId() != null) {

RelatedPolicyForView relatedPolicyForView = relatedPolicyForViewRepository.findById(pol.getId());

policies.add(relatedPolicyForView);

}

});

if (modifiedPolicy.getRelatedPolicyID() != null) {

modifiedPolicy.getRelatedPolicyID().clear();

modifiedPolicy.getRelatedPolicyID().addAll(policies);

} else {

modifiedPolicy.setRelatedPolicyID(policies);

}

} else {

if (modifiedPolicy.getRelatedPolicyID() != null) {

modifiedPolicy.getRelatedPolicyID().clear();

}

}

// --------- Exposure -----------

modifiedPolicy.setCurrencyID((policyDTO.getCurrencyID() == null || policyDTO.getCurrencyID().getId() == null)

? null : currencyRepository.findOne(policyDTO.getCurrencyID().getId()));

// ----------- Sublimit -----------

if (policyDTO.getSublimitID() != null) {

List<Sublimit> sublimits = new ArrayList<>();

policyDTO.getSublimitID().forEach((sublimit) -> {

if (sublimit.getAddOnID() != null && sublimit.getAddOnID().getId() != null) {

AddOnDTO addOn = mapper.map(addOnRepository.findOne(sublimit.getAddOnID().getId()), AddOnDTO.class);

sublimit.setAddOnID(addOn);

}

if (sublimit.getCurrencyID() != null && sublimit.getCurrencyID().getId() != null) {

CurrencyReferenceDTO sublimitCurrency = mapper.map(

currencyRepository.findOne(sublimit.getCurrencyID().getId()), CurrencyReferenceDTO.class);

sublimit.setSublimitCurrency(sublimitCurrency);

}

if (sublimit.getCurrencyID() != null && sublimit.getCurrencyID().getId() != null) {

CurrencyReferenceDTO currency = mapper.map(

currencyRepository.findOne(sublimit.getCurrencyID().getId()), CurrencyReferenceDTO.class);

sublimit.setCurrencyID(currency);

}

if (sublimit.getAggregateLimitCurrencyID() != null

&& sublimit.getAggregateLimitCurrencyID().getId() != null) {

CurrencyReferenceDTO aggCurrency = mapper.map(

currencyRepository.findOne(sublimit.getAggregateLimitCurrencyID().getId()),

CurrencyReferenceDTO.class);

sublimit.setAggregateLimitCurrencyID(aggCurrency);

}

if (sublimit.getId() == null) {

SublimitDTO sublimitDTO = sublimitService.createSublimit(sublimit);

sublimits.add(sublimitRepository.findOne(sublimitDTO.getId()));

} else {

sublimitService.updateSublimit(sublimit);

sublimits.add(sublimitRepository.findOne(sublimit.getId()));

}

});

modifiedPolicy.getSublimitID().clear();

if (!sublimits.isEmpty()) {

if (modifiedPolicy.getSublimitID() != null) {

modifiedPolicy.getSublimitID().addAll(sublimits);

} else {

modifiedPolicy.setSublimitID(sublimits);

}

}

} else {

if (modifiedPolicy.getSublimitID() != null) {

modifiedPolicy.getSublimitID().clear();

}

}

// ----------- Additional Limit -----------

if (policyDTO.getAdditionalLimitID() != null) {

List<Sublimit> additionalLimits = new ArrayList<>();

policyDTO.getAdditionalLimitID().forEach((additionalLimit) -> {

if (additionalLimit.getAddOnID() != null && additionalLimit.getAddOnID().getId() != null) {

AddOnDTO addOn = mapper.map(addOnRepository.findOne(additionalLimit.getAddOnID().getId()),

AddOnDTO.class);

additionalLimit.setAddOnID(addOn);

}

if (additionalLimit.getCurrencyID() != null && additionalLimit.getCurrencyID().getId() != null) {

CurrencyReferenceDTO currency = mapper.map(

currencyRepository.findOne(additionalLimit.getCurrencyID().getId()),

CurrencyReferenceDTO.class);

additionalLimit.setCurrencyID(currency);

}

if (additionalLimit.getSublimitCurrency() != null

&& additionalLimit.getSublimitCurrency().getId() != null) {

CurrencyReferenceDTO sublimitCurrency = mapper.map(

currencyRepository.findOne(additionalLimit.getSublimitCurrency().getId()),

CurrencyReferenceDTO.class);

additionalLimit.setSublimitCurrency(sublimitCurrency);

}

if (additionalLimit.getAggregateLimitCurrencyID() != null

&& additionalLimit.getAggregateLimitCurrencyID().getId() != null) {

CurrencyReferenceDTO aggCurrency = mapper.map(

currencyRepository.findOne(additionalLimit.getAggregateLimitCurrencyID().getId()),

CurrencyReferenceDTO.class);

additionalLimit.setAggregateLimitCurrencyID(aggCurrency);

}

if (additionalLimit.getId() == null) {

SublimitDTO additionalLimitDTO = sublimitService.createSublimit(additionalLimit);

additionalLimits.add(sublimitRepository.findOne(additionalLimitDTO.getId()));

} else {

sublimitService.updateSublimit(additionalLimit);

additionalLimits.add(sublimitRepository.findOne(additionalLimit.getId()));

}

});

modifiedPolicy.getAdditionalLimitID().clear();

if (!additionalLimits.isEmpty()) {

if (modifiedPolicy.getAdditionalLimitID() != null) {

modifiedPolicy.getAdditionalLimitID().addAll(additionalLimits);

} else {

modifiedPolicy.setAdditionalLimitID(additionalLimits);

}

}

} else {

if (modifiedPolicy.getAdditionalLimitID() != null) {

modifiedPolicy.getAdditionalLimitID().clear();

}

}

// ----------- Deductible -----------

if (policyDTO.getDeductibleID() != null) {

List<Deductible> deductibles = new ArrayList<>();

policyDTO.getDeductibleID().forEach((deductible) -> {

if (deductible.getId() == null) {

DeductibleDTO deductibleDTO = deductibleService.createDeductible(deductible);

deductibles.add(deductibleRepository.findOne(deductibleDTO.getId()));

} else {

deductibleService.updateDeductible(deductible);

deductibles.add(deductibleRepository.findOne(deductible.getId()));

}

});

if (modifiedPolicy.getDeductibleID() != null) {

modifiedPolicy.getDeductibleID().clear();

modifiedPolicy.getDeductibleID().addAll(deductibles);

} else {

modifiedPolicy.setDeductibleID(deductibles);

}

} else {

if (modifiedPolicy.getDeductibleID() != null) {

modifiedPolicy.getDeductibleID().clear();

}

}

// -------------- UnderwritingProcess

if (policyDTO.getUnderwritingProcess() != null) {

UnderwritingProcessDTO uwProcess = policyDTO.getUnderwritingProcess();

uwProcess.setIdPolicy(policyDTO.getId());

UnderwritingProcessDTO underwritingProcess = underwritingProcessService.updateUnderwritingProcess(uwProcess);

modifiedPolicy.setUnderwritingProcess(underwritingProcessRepository.findOne(underwritingProcess.getId()));

}

// -------------- PolicyIncome

if (policyDTO.getPolicyIncomeID() != null) {

List<PolicyIncome> incomes = new ArrayList<>();

policyDTO.getPolicyIncomeID().forEach((dto) -> {

if (dto.getId() == null) {

PolicyIncomeDTO newDTO = policyIncomeService.create(dto);

incomes.add(policyIncomeRepository.findOne(newDTO.getId()));

} else {

policyIncomeService.update(dto);

incomes.add(policyIncomeRepository.findOne(dto.getId()));

}

});

if (modifiedPolicy.getPolicyIncomeID() != null) {

modifiedPolicy.getPolicyIncomeID().clear();

modifiedPolicy.getPolicyIncomeID().addAll(incomes);

} else {

modifiedPolicy.setPolicyIncomeID(incomes);

}

} else {

if (modifiedPolicy.getPolicyIncomeID() != null) {

modifiedPolicy.getPolicyIncomeID().clear();

}

}

// --------- Premium -----------

// ----------- Policy Premium Taxes -----------

if (policyDTO.getPolicyPremiumTaxesID() != null) {

List<PolicyPremiumTaxesReference> taxes = new ArrayList<>();

policyDTO.getPolicyPremiumTaxesID().forEach((tax) -> {

if (tax.getId() == null) {

PolicyPremiumTaxesDTO policyPremiumTaxesDTO = policyPremiumTaxesService

.createPolicyPremiumTaxes(mapper.map(tax, PolicyPremiumTaxesDTO.class));

taxes.add(policyPremiumTaxesReferenceRepository.findOne(policyPremiumTaxesDTO.getId()));

} else {

policyPremiumTaxesService.updatePolicyPremiumTaxes(mapper.map(tax, PolicyPremiumTaxesDTO.class));

taxes.add(policyPremiumTaxesReferenceRepository.findOne(tax.getId()));

}

});

if (modifiedPolicy.getPolicyPremiumTaxesID() != null) {

modifiedPolicy.getPolicyPremiumTaxesID().clear();

modifiedPolicy.getPolicyPremiumTaxesID().addAll(taxes);

} else {

modifiedPolicy.setPolicyPremiumTaxesID(taxes);

}

} else {

if (modifiedPolicy.getPolicyPremiumTaxesID() != null) {

modifiedPolicy.getPolicyPremiumTaxesID().clear();

}

}

// ----------- Deduction -----------

if (policyDTO.getDeductionID() != null) {

List<Deduction> deductions = new ArrayList<>();

policyDTO.getDeductionID().forEach((deduction) -> {

if (deduction.getId() == null) {

// Create

DeductionDTO deductionDTO = deductionService.createDeduction(deduction);

deductions.add(deductionRepository.findOne(deductionDTO.getId()));

} else {

// Updated by jpa by default

deductionService.updateDeduction(deduction);

deductions.add(deductionRepository.findOne(deduction.getId()));

}

});

if (modifiedPolicy.getDeductionID() != null) {

modifiedPolicy.getDeductionID().clear();

modifiedPolicy.getDeductionID().addAll(deductions);

} else {

modifiedPolicy.setDeductionID(deductions);

}

} else {

if (modifiedPolicy.getDeductionID() != null) {

modifiedPolicy.getDeductionID().clear();

}

}

// ------------Premium Received-----------

double newPolicyPremiumReceived = 0;

double newPolicyPremiumReceivedGBP = 0;

if (modifiedPolicy.getPremiumReceivedID() != null) {

for (PremiumReceivedReference premium : modifiedPolicy.getPremiumReceivedID()) {

newPolicyPremiumReceived += (premium.getActualNetPremiumReceived() != null

? premium.getActualNetPremiumReceived() : 0);

newPolicyPremiumReceivedGBP += (premium.getPremiumActuallyReceived() != null

? premium.getPremiumActuallyReceived() : 0);

}

}

modifiedPolicy.setPremiumReceived(newPolicyPremiumReceived);

modifiedPolicy.setPremiumReceivedGBP(newPolicyPremiumReceivedGBP);

modifiedPolicy.setPremiumReceived100(

(modifiedPolicy.getnRIPremiumDebt() == null || modifiedPolicy.getnRIPremiumDebt() == 0d) ? 0

: ((modifiedPolicy.getPremiumReceived() != null ? modifiedPolicy.getPremiumReceived() : 0) \* 100

/ modifiedPolicy.getnRIPremiumDebt()));

// ----- Insurance Programme ------

// --------- Indicators -----------

// --------- Compliance -----------

// --------- Documents ------------

// -------------------------------

// Outstanding Amount

if (modifiedPolicy.getReinsuranceOut() && modifiedPolicy.getInsuranceStructureID().getInsuranceStructureID() == 1) {

modifiedPolicy.setOutstandingAmount((

(policyDTO.getnRIPremiumDebt() != null ? policyDTO.getnRIPremiumDebt() : 0d)

- (policyDTO.getiPEAndOtherFeesToNRI() != null ? policyDTO.getiPEAndOtherFeesToNRI() : 0d)) - modifiedPolicy.getPremiumReceived());

} else {

modifiedPolicy.setOutstandingAmount(

(policyDTO.getnRIPremiumDebt() != null ? policyDTO.getnRIPremiumDebt() : 0d)

- modifiedPolicy.getPremiumReceived());

}

// Underwriters

if (policyDTO.getUnderwriters() != null) {

List<UserReference> userLst = new ArrayList<>();

policyDTO.getUnderwriters().forEach((user) -> {

userLst.add(userRepository.findOne(user.getId()));

});

if (modifiedPolicy.getUnderwriters() != null) {

modifiedPolicy.getUnderwriters().clear();

modifiedPolicy.getUnderwriters().addAll(userLst);

} else {

modifiedPolicy.setUnderwriters(userLst);

}

} else {

if (modifiedPolicy.getUnderwriters() != null) {

modifiedPolicy.getUnderwriters().clear();

}

}

// Engineers

if (policyDTO.getEngineers() != null) {

List<UserReference> userLst = new ArrayList<>();

policyDTO.getEngineers().forEach((user) -> {

userLst.add(userRepository.findOne(user.getId()));

});

if (modifiedPolicy.getEngineers() != null) {

modifiedPolicy.getEngineers().clear();

modifiedPolicy.getEngineers().addAll(userLst);

} else {

modifiedPolicy.setEngineers(userLst);

}

} else {

if (modifiedPolicy.getEngineers() != null) {

modifiedPolicy.getEngineers().clear();

}

}

// Organisation Policy Coverage

if (policyDTO.getOrganisationPolicyCoverageID() != null

&& !policyDTO.getOrganisationPolicyCoverageID().isEmpty()) {

if (modifiedPolicy.getOrganisationPolicyCoverageID() != null) {

modifiedPolicy.getOrganisationPolicyCoverageID().clear();

}

}

// Update installments dates: - as by JPA specification in this type of

// relationship there's no need to delete the records, it'll merge

// everything up

// Unique Identifier

if (modifiedPolicy.getUniqueIdentifier() == null || modifiedPolicy.getUniqueIdentifier().isEmpty()) {

String lastUniqueId = this.getLastPolicyUniqueIdentifier();

String newUniqueId;

if (lastUniqueId != null && !lastUniqueId.isEmpty()) {

newUniqueId = String.format("%04d", Integer.parseInt(lastUniqueId) + 1);

} else {

newUniqueId = DEFAULT\_UNIQUE\_ID;

}

modifiedPolicy.setUniqueIdentifier(newUniqueId);

}

// Policy Number

if (!Objects.isNull(policyDTO.getPolicyStatusID()) && policyDTO.getPolicyStatusID().getPolicyStatusID() != 1) {

if (!Objects.isNull(oldPolicy.getPolicyStatusID()) && oldPolicy.getPolicyStatusID().getPolicyStatusID() == 1

|| !oldPolicy.getInsured().getId().equals(modifiedPolicy.getInsured().getId())

|| !oldPolicy.getCoverageID().getCoverageID().equals(modifiedPolicy.getCoverageID().getCoverageID())

|| !oldPolicy.getYearOfAccount().equals(modifiedPolicy.getYearOfAccount())) {

String coveragePrefix = modifiedPolicy.getCoverageID().getPreFix();

String firstYearOfAttachment = modifiedPolicy.getFirstYearOfAttachment().toString().substring(2, 4);

String country;

String uniqueIdentifier = modifiedPolicy.getUniqueIdentifier();

String yearOfAccount = modifiedPolicy.getYearOfAccount().toString();

String policyVersion = modifiedPolicy.getPolicyVersion();

if (!Objects.isNull(oldPolicy.getPolicyStatusID())

&& oldPolicy.getPolicyStatusID().getPolicyStatusID() != 1

&& oldPolicy.getInsured().getId().equals(modifiedPolicy.getInsured().getId())) {

country = oldPolicy.getPolicyNumber().substring(oldPolicy.getCoverageID().getPreFix().length() + 2,

oldPolicy.getCoverageID().getPreFix().length() + 5);

} else {

country = modifiedPolicy.getInsured().getCountryId().getCodeIso();

}

StringBuilder policyNumber = new StringBuilder();

policyNumber.append(coveragePrefix).append(firstYearOfAttachment).append(country)

.append(uniqueIdentifier).append("/").append(yearOfAccount).append("-").append(policyVersion);

modifiedPolicy.setPolicyNumber(policyNumber.toString());

}

} else {

if (!isRenew(modifiedPolicy.getUniqueIdentifier())) {

modifiedPolicy.setPolicyNumber(modifiedPolicy.getUniqueIdentifier());

}

}

// NRIIMPLEM-1795

if (this.policyHasDocumentByCode(modifiedPolicy, EnumDocumentType.COVER\_NOTE\_RECEIVED.name())

&& modifiedPolicy.getPolicyStatusID() != null

&& modifiedPolicy.getPolicyStatusID().getPolicyStatusCode() != null &&

EnumPolicyStatus.ENQUIRY.name().equals(modifiedPolicy.getPolicyStatusID().getPolicyStatusCode())) {

int totalReouts = 0;

int totalReinsuranceOfferResponseEmails = 0;

for (PolicySharesShortReference policyShares : modifiedPolicy.getPolicyShareID()) {

if (policyShares.getInsurerRole().equalsIgnoreCase("reout") && policyShares.getrEOUTStatus() != null

&& policyShares.getrEOUTStatus().equalsIgnoreCase("Accepted")) {

totalReouts++;

}

}

for (DocumentReferenceToView document : modifiedPolicy.getDocuments()) {

if (EnumDocumentType.COVER\_NOTE\_RECEIVED.name().equals(document.getDocumentType().getDocumentTypeCode())

&& document.getCreationTime().after(modifiedPolicy.getStatusModificationTime())) {

totalReinsuranceOfferResponseEmails++;

}

}

if (totalReouts == totalReinsuranceOfferResponseEmails) {

modifiedPolicy.setPolicyStatusID(policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.POLICY\_IN\_FORCE.name()));

}

}

if (modifiedPolicy.getPolicyStatusID() != null) {

if (EnumPolicyStatus.ENQUIRY.name().equals(modifiedPolicy.getPolicyStatusID().getPolicyStatusCode())

&& modifiedPolicy.getSanctionsValidation() != null && modifiedPolicy.getSanctionsValidation()

&& sactionsBeforeUpdate != null && !sactionsBeforeUpdate

&& this.policyHasDocumentByCode(modifiedPolicy, EnumDocumentType.COVER\_BOUND\_EMAIL.name())) {

modifiedPolicy.setStatusModificationTime(new Date());

modifiedPolicy = this.passOtherQuotesToNonTaken(modifiedPolicy);

if (modifiedPolicy.getReinsuranceOut() != null && modifiedPolicy.getReinsuranceOut()) {

modifiedPolicy.setPolicyStatusID(policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.ENQUIRY.name()));

} else {

modifiedPolicy.setPolicyStatusID(policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.POLICY\_IN\_FORCE.name()));

}

this.automatedDocumentProduction(modifiedPolicy, STANDARD\_SUBMISSION\_FORM);

this.automatedDocumentProduction(modifiedPolicy, DEBIT\_CREDIT\_NOTE); // NRIIMPLEM-1364

historyService.createHistoryEntry(modifiedPolicy.getId().toString(), POLICY\_STATUS\_CHANGED);

}

}

modifiedPolicy.setCompletenessStatusCheck(this.isStatusComplete(modifiedPolicy));

if (modifiedPolicy.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.BOUND.name())) {

Date todayDate = new Date();

if (DateUtils.getEndOfDay(modifiedPolicy.getStartDate()).after(DateUtils.getEndOfDay(todayDate))) {

modifiedPolicy.setPolicyStatusID(policyStatusRepository.findOne(20L));

} else {

modifiedPolicy.setPolicyStatusID(policyStatusRepository.findOne(7L));

}

}

modifiedPolicy = repositoryForSave.save(modifiedPolicy);

repository.flush();

this.em.refresh(modifiedPolicy);

historyService.createHistoryEntry(modifiedPolicy.getId().toString(), POLICY\_UPDATED);

// Generate automatic tasks

generateTasksAndALerts(oldPolicy, modifiedPolicy);

// Create the policy Status Evolution object if necessary

if (!Objects.isNull(modifiedPolicy.getPolicyStatusID())

&& modifiedPolicy.getPolicyStatusID().getPolicyStatusID() != 1) {

List<Long> policyStatusEvolutionId = policyStatusEvolutionRepository

.checkIfExistsByPolicyId(modifiedPolicy.getId());

if (policyStatusEvolutionId.isEmpty()) {

PolicyForStatusEvolution policyForEvo = policyForStatusEvolutionRepository

.findOne(modifiedPolicy.getId());

PolicyStatusEvolution policyStatus = new PolicyStatusEvolution();

policyStatus.setPolicyID(policyForEvo);

policyStatus.setStartDate(new Date());

// policyStatus.setEndDate(null);

policyStatus.setrAGcolourID(modifiedPolicy.getrAGcolourID());

policyStatus.setUpdateTimesCount(0L);

policyStatusEvolutionRepository.save(policyStatus);

}

}

PolicyDTO result = null;

this.removeDeletedDeductiblesFromClaims(modifiedPolicy, oldPolicy);

// Check if need new journals

boolean createJournals = false;

if (!Objects.isNull(policyDTO.getPolicyStatusID()) && policyDTO.getPolicyStatusID().getPolicyStatusID() != 1

&& !allocations.isEmpty() && this.checkPolicyNeedReversingJournal(modifiedPolicy, oldPolicy)) {

createJournals = true;

}

try {

result = mapper.map(modifiedPolicy, PolicyDTO.class);

} catch (Exception ex) {

LOGGER.error("EXCEPTION - PolicyServiceImpl - UpdatePolicy - converting response to DTO - ", ex);

}

// Reversing Journal if needed

if (createJournals) {

try {

PolicyReferenceDTO policyForJournal = mapper.map(result, PolicyReferenceDTO.class);

JournalInfoDTO journalInfo = new JournalInfoDTO();

journalInfo.setPolicyNumber(modifiedPolicy.getPolicyNumber());

journalInfo.setReversing(true);

List<PremiumReceivedDTO> allocationsWithPol = ListMapper.mapList(allocations, mapper,

PremiumReceivedDTO.class);

PolicyReferenceDTO oldPolicyDTO = mapper.map(oldPolicy, PolicyReferenceDTO.class);

allocationsWithPol.forEach((alloc) -> {

alloc.setPolicyID(oldPolicyDTO);

});

this.journalService.createJournal(allocationsWithPol, journalInfo, null);

journalInfo.setReversing(false);

this.journalService.createJournal(allocationsWithPol, journalInfo, policyForJournal);

this.em.refresh(modifiedPolicy);

result = mapper.map(modifiedPolicy, PolicyDTO.class);

} catch (Exception ex) {

LOGGER.error("EXCEPTION - PolicyServiceImpl - UpdatePolicy - Saving Journals - ", ex);

}

}

return result;

}

private PolicyDTO fixPolicyBooleans(PolicyDTO policyDTO) {

policyDTO.setAnnualAggregateLimitPerLocation(policyDTO.getAnnualAggregateLimitPerLocation() == null ? false

: policyDTO.getAnnualAggregateLimitPerLocation());

policyDTO.setAnnualAggregateLimitPerPolicy(policyDTO.getAnnualAggregateLimitPerPolicy() == null ? false

: policyDTO.getAnnualAggregateLimitPerPolicy());

policyDTO.setAutomaticReinstatement(

policyDTO.getAutomaticReinstatement() == null ? false : policyDTO.getAutomaticReinstatement());

policyDTO.setExcessLayer(policyDTO.getExcessLayer() == null ? false : policyDTO.getExcessLayer());

policyDTO.setGroundUpCover(policyDTO.getGroundUpCover() == null ? false : policyDTO.getGroundUpCover());

policyDTO.setLifetimeLimit(policyDTO.getLifetimeLimit() == null ? false : policyDTO.getLifetimeLimit());

policyDTO

.setNoAggregateLimit(policyDTO.getNoAggregateLimit() == null ? false : policyDTO.getNoAggregateLimit());

policyDTO.setnRILead(policyDTO.getnRILead() == null ? false : policyDTO.getnRILead());

policyDTO.setReinsuranceOut(policyDTO.getReinsuranceOut() == null ? false : policyDTO.getReinsuranceOut());

policyDTO.setStandardNonStandard(

policyDTO.getStandardNonStandard() == null ? false : policyDTO.getStandardNonStandard());

if (policyDTO.getPolicyStatusID() != null && policyDTO.getPolicyStatusID().getPolicyStatusID() == 1) {

policyDTO.setEnquiry(true);

} else {

policyDTO.setEnquiry(false);

}

policyDTO.setAuthoriseCapacity(

policyDTO.getAuthoriseCapacity() == null ? false : policyDTO.getAuthoriseCapacity());

policyDTO.setSanctionsValidation(

policyDTO.getSanctionsValidation() == null ? false : policyDTO.getSanctionsValidation());

policyDTO.setBiSeparateLimit(policyDTO.getBiSeparateLimit() == null ? false : policyDTO.getBiSeparateLimit());

policyDTO.setSpecifyForm(policyDTO.getSpecifyForm() == null ? false : policyDTO.getSpecifyForm());

policyDTO.setAnnualAggregateLimit(

policyDTO.getAnnualAggregateLimit() == null ? false : policyDTO.getAnnualAggregateLimit());

return policyDTO;

}

private boolean checkPolicyNeedReversingJournal(PolicyForSave modifiedPolicy, PolicyForSave oldPolicy) {

if (notEqualFields(modifiedPolicy.getYearOfAccount(), oldPolicy.getYearOfAccount())

|| notEqualFields(modifiedPolicy.getStandardNonStandard(), oldPolicy.getStandardNonStandard())

|| notEqualFields(modifiedPolicy.getnCBPct(), oldPolicy.getnCBPct())

|| notEqualFields(modifiedPolicy.getReinsuranceOut(), oldPolicy.getReinsuranceOut())

|| notEqualFields(modifiedPolicy.getnRInetPremium(), oldPolicy.getnRInetPremium())

|| notEqualFields(modifiedPolicy.getGrossPremium100Pct(), oldPolicy.getGrossPremium100Pct())

|| notEqualFields(modifiedPolicy.getBrokerageAmount(), oldPolicy.getBrokerageAmount())

|| notEqualFields(modifiedPolicy.getIpeExpenseAmount(), oldPolicy.getIpeExpenseAmount())

|| notEqualFields(modifiedPolicy.getnRIPremiumDebt(), oldPolicy.getnRIPremiumDebt())) {

return true;

}

if ((modifiedPolicy.getCoverageID() == null && oldPolicy.getCoverageID() != null)

|| (modifiedPolicy.getCoverageID() != null && oldPolicy.getCoverageID() == null)

|| (modifiedPolicy.getCoverageID() != null && oldPolicy.getCoverageID() != null && notEqualFields(

modifiedPolicy.getCoverageID().getCoverageID(), oldPolicy.getCoverageID().getCoverageID()))) {

return true;

}

if ((modifiedPolicy.getInsured() == null && oldPolicy.getInsured() != null)

|| (modifiedPolicy.getInsured() != null && oldPolicy.getInsured() == null)

|| (modifiedPolicy.getInsured() != null && oldPolicy.getInsured() != null

&& notEqualFields(modifiedPolicy.getInsured().getId(), oldPolicy.getInsured().getId()))) {

return true;

}

if ((modifiedPolicy.getInsuranceStructureID() == null && oldPolicy.getInsuranceStructureID() != null)

|| (modifiedPolicy.getInsuranceStructureID() != null && oldPolicy.getInsuranceStructureID() == null)

|| (modifiedPolicy.getInsuranceStructureID() != null && oldPolicy.getInsuranceStructureID() != null

&& notEqualFields(modifiedPolicy.getInsuranceStructureID().getInsuranceStructureID(),

oldPolicy.getInsuranceStructureID().getInsuranceStructureID()))) {

return true;

}

if ((modifiedPolicy.getDeductionID() == null

&& (oldPolicy.getDeductionID() != null && !oldPolicy.getDeductionID().isEmpty()))

|| ((modifiedPolicy.getDeductionID() != null && !modifiedPolicy.getDeductionID().isEmpty())

&& oldPolicy.getDeductionID() == null)

|| (modifiedPolicy.getDeductionID() != null && modifiedPolicy.getDeductionID() != null

&& modifiedPolicy.getDeductionID().size() != oldPolicy.getDeductionID().size())) {

return true;

}

if (modifiedPolicy.getDeductionID() != null) {

for (Deduction nd : modifiedPolicy.getDeductionID()) {

Deduction od = null;

for (Deduction d : oldPolicy.getDeductionID()) {

if (d.getId().equals(nd.getId())) {

od = d;

break;

}

}

if (od == null || notEqualFields(nd.getAmount(), od.getAmount())) {

return true;

}

}

}

if ((modifiedPolicy.getPolicyShareID() == null

&& (oldPolicy.getPolicyShareID() != null && !oldPolicy.getPolicyShareID().isEmpty()))

|| ((modifiedPolicy.getPolicyShareID() != null && !modifiedPolicy.getPolicyShareID().isEmpty())

&& oldPolicy.getPolicyShareID() == null)

|| (modifiedPolicy.getPolicyShareID() != null && oldPolicy.getPolicyShareID() != null

&& modifiedPolicy.getPolicyShareID().size() != oldPolicy.getPolicyShareID().size())) {

return true;

}

if (modifiedPolicy.getPolicyShareID() != null) {

for (PolicySharesShortReference nd : modifiedPolicy.getPolicyShareID()) {

PolicySharesShortReference od = null;

for (PolicySharesShortReference d : oldPolicy.getPolicyShareID()) {

if (d.getPolicySharesID().equals(nd.getPolicySharesID())) {

od = d;

break;

}

}

if (od == null || notEqualFields(nd.getPremiumReinsurerBeforeIpe(),

od.getPremiumReinsurerBeforeIpe())) {

return true;

}

}

}

if ((modifiedPolicy.getPolicyPremiumTaxesID() == null

&& (oldPolicy.getPolicyPremiumTaxesID() != null && !oldPolicy.getPolicyPremiumTaxesID().isEmpty()))

|| ((modifiedPolicy.getPolicyPremiumTaxesID() != null

&& !modifiedPolicy.getPolicyPremiumTaxesID().isEmpty())

&& oldPolicy.getPolicyPremiumTaxesID() == null)

|| (modifiedPolicy.getPolicyPremiumTaxesID() != null && oldPolicy.getPolicyPremiumTaxesID() != null

&& modifiedPolicy.getPolicyPremiumTaxesID().size() != oldPolicy.getPolicyPremiumTaxesID()

.size())) {

return true;

}

if (modifiedPolicy.getPolicyPremiumTaxesID() != null) {

for (PolicyPremiumTaxesReference nd : modifiedPolicy.getPolicyPremiumTaxesID()) {

PolicyPremiumTaxesReference od = null;

if (nd.getTaxTypeID() != null && nd.getTaxTypeID().getCode() == "DEBIT") {

for (PolicyPremiumTaxesReference d : oldPolicy.getPolicyPremiumTaxesID()) {

if (d.getId().equals(nd.getId())) {

od = d;

break;

}

}

}

if (od == null || notEqualFields(nd.getTaxAmount(), od.getTaxAmount())

|| (nd.getCountryID() == null && od.getCountryID() != null)

|| (nd.getCountryID() != null && od.getCountryID() == null)

|| (nd.getCountryID() != null && od.getCountryID() != null

&& notEqualFields(nd.getCountryID().getId(), od.getCountryID().getId()))) {

return true;

}

}

}

return false;

}

private void removeDeletedDeductiblesFromClaims(PolicyForSave modifiedPolicy, PolicyForSave oldPolicy) {

List<Deductible> removedDeductibles = new ArrayList<Deductible>();

if (oldPolicy.getDeductibleID() != null) {

oldPolicy.getDeductibleID().forEach((deductible) -> {

if (modifiedPolicy.getDeductibleID() == null || modifiedPolicy.getDeductibleID().isEmpty()

|| modifiedPolicy.getDeductibleID().stream()

.filter((modDeductible) -> modDeductible.getId() == deductible.getId()).count() == 0) {

removedDeductibles.add(deductible);

}

});

}

removedDeductibles.forEach((deductible) -> {

this.claimReferenceRepository.removeDeductibleFromClaimByDeductibleId(deductible.getId());

});

return;

}

private boolean notEqualFields(Object obj1, Object ob2) {

if ((obj1 == null && ob2 != null) || (obj1 != null && ob2 == null)

|| (obj1 != null && ob2 != null && !obj1.equals(ob2))) {

return true;

}

return false;

}

@Override

public PolicyDTO assignTo(Long policyID, String userId) {

try {

Policy policy = repository.findOne(policyID);

if (policy == null) {

return null;

}

UserReference assignedUser = userReference.findOne(userId);

if (assignedUser == null) {

return null;

}

policy.setUserID(assignedUser);

policy = repository.save(policy);

PolicyDTO response = new PolicyDTO();

response.setId(policy.getId());

return response;

} catch (JDBCConnectionException e) {

throw new RuntimeException();

} catch (JDBCException e) {

throw new RuntimeException();

} catch (Exception e) {

throw new RuntimeException(e.getMessage());

}

}

@Override

public Long updateStatus(Long policyID, Long statusID, String userID) {

try {

PolicyForSave pol = repositoryForSave.findOne(policyID);

final PolicyForSave oldPolicy = mapper.map(pol, PolicyForSave.class);

if (pol == null)

return null;

PolicyStatus policyStatus = policyStatusRepository.findOne(statusID);

if (policyStatus == null)

return null;

if (policyStatus.getPolicyStatusCode().equals(EnumPolicyStatus.QUOTED.name())

&& !pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.ENQUIRY.name())) {

StringBuilder message = new StringBuilder();

message.append(this.getClass().getName());

message.append(": Can't change policy status.");

message.append(" Policy Status can only change to Quoted if its Enquiry ");

throw new Exception(message.toString());

}

if ((pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.ENQUIRY.name())

|| pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.QUOTED.name())) &&

(policyStatus.getPolicyStatusCode().equals(EnumPolicyStatus.BOUND.name()) ||

policyStatus.getPolicyStatusCode().equals(EnumPolicyStatus.POLICY\_IN\_FORCE.name())) ) {

if (pol.getUnderwritingProcess() != null) {

for (UnderwritingTask task : pol.getUnderwritingProcess().getTasks()) {

if (task.getConfiguration().getId() == 21) {

UserReference tempUSer = this.userRepository.findById(userID);

task.setCompletedDate(new Date());

task.setCompletedBy(tempUSer);

}

}

}

}

if ( (policyStatus.getPolicyStatusCode().equals(EnumPolicyStatus.POLICY\_IN\_FORCE.name()) || policyStatus.getPolicyStatusCode().equals(EnumPolicyStatus.BOUND.name()) )

&& !(pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.ENQUIRY.name())

|| pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.QUOTED.name()))) {

StringBuilder message = new StringBuilder();

message.append(this.getClass().getName());

message.append(": Can't change policy status.");

message.append(" Policy Status can only change to Policy in Force or Bound if its Enquiry or Quoted ");

throw new Exception(message.toString());

}

if (policyStatus.getPolicyStatusCode().equals(EnumPolicyStatus.NON\_TAKEN\_UP.name())

&& !((pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.QUOTED.name())

|| pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.ENQUIRY.name()))

&& pol.getInsuranceStructureID().getInsuranceStructureName()

.equalsIgnoreCase("Direct insurance"))) {

StringBuilder message = new StringBuilder();

message.append(this.getClass().getName());

message.append(": Can't change policy status.");

message.append(

" Policy Status can only change to Non-taken if its in status Quoted or Enquiry, and the insurance structure is Direct Insurance");

throw new Exception(message.toString());

}

if (policyStatus.getPolicyStatusCode().equals(EnumPolicyStatus.DECLINED.name())

&& !((pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.QUOTED.name())

|| pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.ENQUIRY.name()))

&& pol.getInsuranceStructureID().getInsuranceStructureName()

.equalsIgnoreCase("Reinsurance inwards"))) {

StringBuilder message = new StringBuilder();

message.append(this.getClass().getName());

message.append(": Can't change policy status.");

message.append(

" Policy Status can only change to Declined if its in status Quoted or Enquiry, and the insurance structure is Reinsurance inwards");

throw new Exception(message.toString());

}

if (pol.getPolicyStatusID().getPolicyStatusID() != policyStatus.getPolicyStatusID()) {

if (!(policyStatus.getPolicyStatusCode().equals(EnumPolicyStatus.DECLINED.name())

|| policyStatus.getPolicyStatusCode().equals(EnumPolicyStatus.NON\_TAKEN\_UP.name()))

&& (pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.ENQUIRY.name())) || pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.QUOTED.name()) ) {

pol = setNewPolicyBasicDetailsForStatus(pol, policyStatus.getPolicyStatusCode());

}

if (policyStatus.getPolicyStatusCode().equals(EnumPolicyStatus.POLICY\_IN\_FORCE.name()) ||

(policyStatus.getPolicyStatusCode().equals(EnumPolicyStatus.BOUND.name())) ) {

Date todayDate = new Date();

if (DateUtils.getEndOfDay(pol.getStartDate()).after(DateUtils.getEndOfDay(todayDate))) {

policyStatus = policyStatusRepository.findOne(20L);

} else {

policyStatus = policyStatusRepository.findOne(7L);

}

}

pol.setPolicyStatusID(policyStatus);

pol.setStatusModificationTime(new Date());

if (!pol.getPolicyStatusID().getPolicyStatusCode().equals(EnumPolicyStatus.ENQUIRY.name())) {

pol.setEnquiry(false);

}

pol = repositoryForSave.save(pol);

final String historyType;

if (policyStatus.getPolicyStatusCode().equals("CANCELLED")) {

historyType = POLICY\_CANCELLED;

} else {

historyType = POLICY\_STATUS\_CHANGED;

}

historyService.createHistoryEntry(pol.getId().toString(), historyType);

}

generateTasksAndALerts(oldPolicy, pol);

return statusID;

} catch (JDBCConnectionException e) {

throw new RuntimeException();

} catch (JDBCException e) {

throw new RuntimeException();

} catch (Exception e) {

throw new RuntimeException(e.getMessage());

}

}

private PolicyForSave passOtherQuotesToNonTaken(PolicyForSave modifiedPolicy) {

List<Policy> allQuotes = this.repository.findByPolicyNumberContains(modifiedPolicy.getPolicyNumber());

allQuotes.forEach((quote) -> {

if (quote.getId().longValue() != modifiedPolicy.getId().longValue()) {

quote.setPolicyStatusID(policyStatusRepository

.findByPolicyStatusCodeOrderByPolicyStatusNameAsc(EnumPolicyStatus.NON\_TAKEN\_UP.name()));

quote.setStatusModificationTime(new Date());

this.repository.save(quote);

}

});

modifiedPolicy.setPolicyVersion("01");

modifiedPolicy.setPolicyNumber(

modifiedPolicy.getPolicyNumber().substring(0, modifiedPolicy.getPolicyNumber().length() - 2)

+ modifiedPolicy.getPolicyVersion());

return modifiedPolicy;

}

@Override

public List<PolicyStatusDTO> getAllPolicyStatus() {

List<PolicyStatus> status = policyStatusRepository.findAllByOrderByPolicyStatusNameAsc();

List<PolicyStatusDTO> statusDTO = ListMapper.mapList(status, mapper, PolicyStatusDTO.class);

return statusDTO;

}

@Override

public List<PolicyIndicatorsChartDTO> policyEvolution(String policyNumber) {

String[] policyData = policyNumber.split("/");

String policy = policyData[0];

String quoteNumber = "";

if (policyData.length > 1 && policyData[1] != null && policyData[1].contains("-")) {

quoteNumber = policyData[1].split("-")[1];

}

return repository.getPolicyEvolution(policy, quoteNumber);

}

@Override

public Double getPremiumForInsured(Long orgId) {

Double totalPremium = repository.getInsuredPremium(orgId);

return totalPremium;

}

@Override

public List<OrganisationPolicyReferenceDTO> getInsuredPolicies(Long orgId) {

List<PolicyForOrg> policies = policyForOrgRepository.findByInsuredId(orgId);

if (policies == null)

return null;

List<OrganisationPolicyReferenceDTO> result = new ArrayList<OrganisationPolicyReferenceDTO>();

for (PolicyForOrg policy : policies) {

List<PolicyDatesWithClaimsReferenceDTO> policyDatesWithClaimsDTOList = new ArrayList<PolicyDatesWithClaimsReferenceDTO>();

OrganisationPolicyReferenceDTO orgPol = new OrganisationPolicyReferenceDTO();

if (Objects.nonNull(policy.getPolicyStatusID())) {

orgPol.setPolicyStatus(mapper.map(policy.getPolicyStatusID(), PolicyStatusReferenceDTO.class));

}

if (Objects.nonNull(policy.getInsuranceStructureID())) {

orgPol.setInsuranceStructure(

mapper.map(policy.getInsuranceStructureID(), InsuranceStructureReferenceDTO.class));

}

orgPol.setPolicyId(policy.getId());

orgPol.setPolicyNumber(policy.getPolicyNumber());

orgPol.setPolicyPremiumBeforeDeductions(policy.getGrossPremium100Pct());

orgPol.setClaimCurrentEstimateGBP(claimService.getEstimateForPolicy(policy.getId()));

orgPol.setStartDate(policy.getStartDate());

orgPol.setEndDate(policy.getEndDate());

if (Objects.nonNull(policy.getInsured()) && Objects.nonNull(policy.getInsured().getCountryId())) {

orgPol.setCountry(mapper.map(policy.getInsured().getCountryId(), CountryReferenceDTO.class));

}

orgPol.setQuoteNumber(policy.getQuoteNumber());

orgPol.setPolicyVersion(policy.getPolicyVersion());

orgPol.setRenewalDate(policy.getRenewalDate());

List<ClaimShortDTO> claimList = claimService.getClaimByPolicyID(policy.getId());

for (ClaimShortDTO claimShortDTO : claimList) {

PolicyDatesWithClaimsReferenceDTO policyDatesWithClaimsDTO = new PolicyDatesWithClaimsReferenceDTO();

policyDatesWithClaimsDTO.setId(claimShortDTO.getClaimID());

policyDatesWithClaimsDTO.setpolicyId(policy.getId());

policyDatesWithClaimsDTO.setDateOfLoss(claimShortDTO.getDateOfLoss());

policyDatesWithClaimsDTO.setClaimRef(claimShortDTO.getClaimNumber());

policyDatesWithClaimsDTO.setPaidFigure(claimShortDTO.getTotalPayments());

policyDatesWithClaimsDTO.setPaidFigureGBP(claimShortDTO.getTotalPaymentsGBP());

policyDatesWithClaimsDTO.setEstimateFigure(claimShortDTO.getCurrentEstimate());

policyDatesWithClaimsDTO.setEstimateFigureGBP(claimShortDTO.getCurrentEstimateGBP());

policyDatesWithClaimsDTO.setCurrencySymbol(

Objects.isNull(claimShortDTO.getCurrency()) ? "" : claimShortDTO.getCurrency().getSymbol());

policyDatesWithClaimsDTOList.add(policyDatesWithClaimsDTO);

}

orgPol.setPolicyClaimsList(policyDatesWithClaimsDTOList);

result.add(orgPol);

}

return result;

}

@Override

public List<OrganisationPolicyReferenceDTO> getBrokerPolicies(Long orgId) {

List<PolicyForOrg> policies = policyForOrgRepository.findByBrokerId(orgId);

if (policies == null)

return null;

List<OrganisationPolicyReferenceDTO> result = new ArrayList<OrganisationPolicyReferenceDTO>();

for (PolicyForOrg policy : policies) {

List<PolicyDatesWithClaimsReferenceDTO> policyDatesWithClaimsDTOList = new ArrayList<PolicyDatesWithClaimsReferenceDTO>();

OrganisationPolicyReferenceDTO orgPol = new OrganisationPolicyReferenceDTO();

if (Objects.nonNull(policy.getPolicyStatusID())) {

orgPol.setPolicyStatus(mapper.map(policy.getPolicyStatusID(), PolicyStatusReferenceDTO.class));

}

if (Objects.nonNull(policy.getInsuranceStructureID())) {

orgPol.setInsuranceStructure(

mapper.map(policy.getInsuranceStructureID(), InsuranceStructureReferenceDTO.class));

}

orgPol.setPolicyId(policy.getId());

orgPol.setPolicyNumber(policy.getPolicyNumber());

orgPol.setPolicyPremiumBeforeDeductions(policy.getGrossPremium100Pct());

orgPol.setClaimCurrentEstimateGBP(claimService.getEstimateForPolicy(policy.getId()));

orgPol.setStartDate(policy.getStartDate());

orgPol.setEndDate(policy.getEndDate());

if (Objects.nonNull(policy.getBroker()) && Objects.nonNull(policy.getBroker().getCountryId())) {

orgPol.setCountry(mapper.map(policy.getBroker().getCountryId(), CountryReferenceDTO.class));

}

orgPol.setQuoteNumber(policy.getQuoteNumber());

orgPol.setPolicyVersion(policy.getPolicyVersion());

orgPol.setRenewalDate(policy.getRenewalDate());

List<ClaimShortDTO> claimList = claimService.getClaimByPolicyID(policy.getId());

for (ClaimShortDTO claimShortDTO : claimList) {

PolicyDatesWithClaimsReferenceDTO policyDatesWithClaimsDTO = new PolicyDatesWithClaimsReferenceDTO();

policyDatesWithClaimsDTO.setId(claimShortDTO.getClaimID());

policyDatesWithClaimsDTO.setpolicyId(policy.getId());

policyDatesWithClaimsDTO.setDateOfLoss(claimShortDTO.getDateOfLoss());

policyDatesWithClaimsDTO.setClaimRef(claimShortDTO.getClaimNumber());

policyDatesWithClaimsDTO.setPaidFigure(claimShortDTO.getTotalPayments());

policyDatesWithClaimsDTO.setPaidFigureGBP(claimShortDTO.getTotalPaymentsGBP());

policyDatesWithClaimsDTO.setEstimateFigure(claimShortDTO.getCurrentEstimate());

policyDatesWithClaimsDTO.setEstimateFigureGBP(claimShortDTO.getCurrentEstimateGBP());

policyDatesWithClaimsDTO.setCurrencySymbol(

Objects.isNull(claimShortDTO.getCurrency()) ? "" : claimShortDTO.getCurrency().getSymbol());

policyDatesWithClaimsDTOList.add(policyDatesWithClaimsDTO);

}

orgPol.setPolicyClaimsList(policyDatesWithClaimsDTOList);

result.add(orgPol);

}

return result;

}

@Override

public List<PolicyEmailTemplatesDTO> getEmailTemplatesByPolicyId(Long policyID) {

List<PolicyEmailTemplates> templates = policyEmailTemplatesRepository.findAllByOrderByTextAsc();

List<PolicyEmailTemplatesDTO> templatesDTO = ListMapper.mapList(templates, mapper,

PolicyEmailTemplatesDTO.class);

HashMap<String, PolicyEmailTemplatesDTO> templatesResult = new HashMap<>();

List<PolicyEmailTemplatesDTO> result = new ArrayList<>();

templatesDTO.forEach(template -> {

templatesResult.put(template.getCode(), template);

});

PolicyEmailTemplateInfo policy = repository.getPolicyEmailTemplateInfo(policyID);

if (Objects.isNull(policy)) {

return result;

}

Boolean isDirect = StringUtils.isNotBlank(policy.getInsuranceStructureName())

? policy.getInsuranceStructureName().equalsIgnoreCase("Direct insurance") : Boolean.FALSE;

Boolean isReinsuranceOut = Objects.nonNull(policy.getReinsuranceOut()) ? policy.getReinsuranceOut()

: Boolean.FALSE;

Boolean isReinsuranceIn = StringUtils.isNotBlank(policy.getInsuranceStructureName())

? policy.getInsuranceStructureName().equalsIgnoreCase("Reinsurance inwards") : Boolean.FALSE;

Boolean isRenewal = Optional.ofNullable(policy.getEntryTypeName()).filter(p -> p.equalsIgnoreCase("Renewal"))

.isPresent();

Boolean isNewBusiness = Optional.ofNullable(policy.getEntryTypeName())

.filter(p -> p.equalsIgnoreCase("New business")).isPresent();

Boolean isEndorsment = Optional.ofNullable(policy.getEntryTypeName())

.filter(p -> p.equalsIgnoreCase("Endorsement")).isPresent();

Boolean isCovPI = StringUtils.isNotBlank(policy.getCoverageName()) ? policy.getCoverageName().toLowerCase()

.contains(EnumCoverage.PROFESSIONAL\_INDEMNITY.getValue().toLowerCase()) : Boolean.FALSE;

Boolean isCovDO = StringUtils.isNotBlank(policy.getCoverageName()) ? policy.getCoverageName().toLowerCase()

.contains(EnumCoverage.DIRECTORS\_AND\_OFFICERS.getValue().toLowerCase()) : Boolean.FALSE;

Boolean isCovProdLia = StringUtils.isNotBlank(policy.getCoverageName()) ? policy.getCoverageName().toLowerCase()

.contains(EnumCoverage.PRODUCTS\_LIABILITY.getValue().toLowerCase()) : Boolean.FALSE;

Boolean isCovPubLia = StringUtils.isNotBlank(policy.getCoverageName()) ? policy.getCoverageName().toLowerCase()

.contains(EnumCoverage.PUBLIC\_LIABILITY.getValue().toLowerCase()) : Boolean.FALSE;

// More info about this conditions at NRIIMPLEM-93. Since there's some

// logic about this templates that's not related with each other

// Was decided to develop like this to ease the understanding/reading.

if (isReinsuranceOut) {

if (isRenewal || isNewBusiness) {

// 5.1

result.add(templatesResult.get("TMPL51"));

}

}

if (isReinsuranceOut || isDirect) {

if (isRenewal && !isNewBusiness && !isEndorsment) {

if (isCovPI || isCovDO) {

// 1.1

result.add(templatesResult.get("TMPL11"));

} else if (isCovProdLia) {

// 1.2

result.add(templatesResult.get("TMPL12"));

} else if (isCovPubLia) {

// 1.3

result.add(templatesResult.get("TMPL13"));

} else {

// 1.4

result.add(templatesResult.get("TMPL14"));

}

// 7.2

result.add(templatesResult.get("TMPL72"));

}

if (!isRenewal && isNewBusiness && !isEndorsment) {

// 7.1

result.add(templatesResult.get("TMPL71"));

}

if (!isRenewal && !isNewBusiness && isEndorsment) {

// 7.3

result.add(templatesResult.get("TMPL73"));

}

if (isRenewal || isNewBusiness) {

// 2.1, 6.1, 7.4

result.add(templatesResult.get("TMPL21"));

result.add(templatesResult.get("TMPL61"));

result.add(templatesResult.get("TMPL74"));

}

}

if (isReinsuranceIn) {

if (isRenewal) {

// 3.1

result.add(templatesResult.get("TMPL31"));

}

// 4.1

// 7.5

result.add(templatesResult.get("TMPL41"));

result.add(templatesResult.get("TMPL75"));

}

// 8.1

result.add(templatesResult.get("TMPL81"));

// 8.2 - not mapped (default)

result.add(templatesResult.get("TMPL82"));

return result;

}

@Override

public String[] getCreatePolicyTypes() {

return CREATE\_POLICY\_TYPES;

}

/\*\*

\*

\*/

@Override

public List<PolicyDTO> getPoliciesTwoMonthBeforeExpiredWithEngineersAndLocationsAndNoTask() {

List<PolicyReferenceForTasks> policies = repositoryForTasks

.getPoliciesTwoMonthBeforeExpiredWithEngineersAndLocationsAndNoTask();

return ListMapper.mapList(policies, mapper, PolicyDTO.class);

}

@Override

public boolean existsActiveRenewalForPolicy(String policyNumber, Integer yearOfAccount) {

if (policyNumber == null || yearOfAccount == null || policyNumber.length() < 12) {

return false;

}

String policyNumberForSearch = "%" + policyNumber.substring(0, 11).concat(Integer.toString(yearOfAccount + 1))

+ "%";

List<String> policyStatusCodes = new ArrayList<String>();

policyStatusCodes.add(this.DECLINED);

policyStatusCodes.add(this.NON\_TAKEN\_UP);

int result = this.repository.existsActiveRenewalForPolicy(policyNumberForSearch, policyStatusCodes);

if (result != 0) {

return true;

}

return false;

}

public boolean isRenewable(PolicyShortDTO policy) {

if (policy != null && policy.getPolicyStatus() != null && policy.getPolicyStatus().getPolicyStatusName() != null

&& policy.getEntryType() != null && policy.getEntryType().getEntryTypeName() != null

&& policy.getEndDate() != null) {

Date currentDate = new Date();

Calendar cal = Calendar.getInstance();

cal.setTime(policy.getEndDate());

cal.set(Calendar.HOUR, 0);

cal.set(Calendar.MINUTE, 0);

cal.set(Calendar.SECOND, 0);

cal.set(Calendar.MILLISECOND, 0);

Date comparationDate = cal.getTime();

long difference = TimeUnit.DAYS.convert((currentDate.getTime() - comparationDate.getTime()),

TimeUnit.MILLISECONDS);

if (policy.getPolicyStatus().getPolicyStatusName().toLowerCase() == this.POLICY\_IN\_FORCE && policy

.getEntryType().getEntryTypeName().toLowerCase() == PolicyServiceImpl.ENDORSEMENT.toLowerCase()

&& difference >= 0 && difference <= 90) {

return existsActiveRenewalForPolicy(policy.getPolicyNumber(), policy.getYearOfAccount());

}

}

return false;

}

private void automatedDocumentProduction(PolicyForSave pol, String docType) {

List<Map<String, Object>> documents = new ArrayList<Map<String, Object>>();

ContentType contentType = ContentType.OOXML\_WORD;

try {

switch (docType) {

case ENDORSEMENT:

Map<String, Object> endorsementdocument = policyDocumentService.generatePolicyDocument(ENDORSEMENT,pol.getId());

if (pol.getEntryTypeID() != null

&& pol.getEntryTypeID().getEntrySubType().toUpperCase().equals("EXPIRY EXTENSION")) {

endorsementdocument.put("name", "Extension\_Endorsement\_");

} else {

endorsementdocument.put("name", "Endorsement\_Addendum\_");

}

documents.add(endorsementdocument);

break;

case DOCUMENTATION\_TO\_ISSUE:

if (pol.getEntryTypeID().getEntryTypeName().toUpperCase().equals(RENEWAL)) {

Map<String, Object> renewalDoc = policyDocumentService.generatePolicyDocument(RENEWAL, pol.getId());

renewalDoc.put("name", "RENEWAL\_");

documents.add(renewalDoc);

}

Map<String, Object> schedule = policyDocumentService.generatePolicyDocument(SCHEDULE, pol.getId());

schedule.put("name", "SCHEDULE\_");

documents.add(schedule);

Map<String, Object> coverNote = policyDocumentService.generatePolicyDocument(COVERNOTE, pol.getId());

coverNote.put("name", "COVER\_NOTE\_");

documents.add(coverNote);

break;

case DEBIT\_CREDIT\_NOTE:

Map<String, Object> note = policyDocumentService.generateCreditDebitDocument(pol.getId());

if (note != null) {

note.put("name", pol.getGrossPremium100Pct() > 0 ? "DEBIT\_NOTE\_" : "CREDIT\_NOTE\_");

documents.add(note);

}

break;

case STANDARD\_SUBMISSION\_FORM:

contentType = ContentType.OOXML\_EXCEL;

Map<String, Object> form = policyDocumentService.generateStandardDocument(pol.getId());

if (form != null) {

form.put("name", "STANDARD\_SUBMISSION\_FORM\_");

documents.add(form);

}

break;

}

for (Map<String, Object> docData : documents) {

String file = CastUtil.toString(docData.get("report"));

String name = CastUtil.toString(docData.get("name")) + pol.getPolicyNumber();

// temporary file

File tempFile = File.createTempFile(name, contentType.getDefaultExtension());

FileOutputStream oos = new FileOutputStream(tempFile);

oos.write(javax.xml.bind.DatatypeConverter.parseBase64Binary(file));

oos.close();

// categories

HashMap<String, String> categories = new HashMap<String, String>();

categories.put("\_name", FAV\_POLICIES);

categories.put("Policy number", CastUtil.toString(pol.getId()));

categories.put("Type", CastUtil

.toString(documentTypeReferenceRepository.getDocumentTypeByDocumentTypeCode(POLICY).getId()));

documentService.saveDocument(tempFile.getAbsolutePath(), name + "." + contentType.getDefaultExtension(),

contentType.getContentType(), tempFile.length(), categories);

tempFile.delete();

}

} catch (Exception e) {

LOGGER.error("[POLICY\_ID: " + pol.getId() + "][DOC\_TYPE: " + docType + "]ERROR: " + e.getMessage() + "]",

e);

}

}

public boolean policyHasDocumentByCode(PolicyForSave modifiedPolicy, String docTypeCode) {

if (modifiedPolicy.getDocuments() != null) {

for (DocumentReferenceToView doc : modifiedPolicy.getDocuments()) {

if (doc != null && doc.getDocumentType() != null && doc.getDocumentType().getDocumentTypeCode() != null

&& doc.getDocumentType().getDocumentTypeCode().equals(docTypeCode)) {

return true;

}

}

}

return false;

}

@Override

@Cacheable(value = CacheConstants.Values.COVERAGE\_LIMIT\_BY\_COVERAGE\_ID\_AND\_YEAR, key = CacheConstants.Keys.COVERAGE\_ID\_DOT\_YEAR, unless = "#result == null")

public CoverageLimitDTO getCoverageLimitByCoverageAndYear(Long coverageId, Integer year) {

CoverageLimit res = coverageLimitRepository.findByCoverageIdAndYearEqualOrLower(coverageId, year);

if (res == null)

res = coverageLimitRepository.findByCoverageIdAndYearEqualOrHigher(coverageId, year);

CoverageLimitDTO resDto = mapper.map(res, CoverageLimitDTO.class);

return resDto;

}

@Override

public ComboInfoDTO getComboInfo() {

List<BrokerageTypeDTO> brokerageType = brokerageTypeService.findAll();

List<DeductionTypeDTO> iNLAFeeType = deductionService

.findByDeductionName(EnumDeductionName.INLA\_FEE.getValue());

final EnumTaxTypeCalculationOperation typeCredit = EnumTaxTypeCalculationOperation.findByName("credit");

final EnumTaxTypeCalculationOperation typeDebit = EnumTaxTypeCalculationOperation.findByName("debit");

List<TaxTypeCalculationDTO> taxTypeCalculationCredit = taxTypeCalculationService.findAll(typeCredit);

List<TaxTypeCalculationDTO> taxTypeCalculationDebit = taxTypeCalculationService.findAll(typeDebit);

List<DeductionTypeDTO> otherDeductionType = deductionService

.findByDeductionName(EnumDeductionName.OTHER\_DEDUCTION.getValue());

List<DeductionTypeDTO> escFeeType = deductionService.findByDeductionName(EnumDeductionName.ESC\_FEE.getValue());

List<DeductionCedingTypeDTO> deductionCedingType = deductionCedingTypeService.findAll();

List<TaxTypeDTO> TaxTypeConvert = taxTypeService.findAll();

List<TaxTypeReferenceDTO> TaxType = ListMapper.mapList(TaxTypeConvert, mapper, TaxTypeReferenceDTO.class);

List<DeductionNameDTO> deductionName = this.getDeductionName();

List<InsuranceStructureDTO> insuranceStructures = this.getInsuranceStructures();

List<PolicyStatusDTO> allPolicyStatus = this.getAllPolicyStatus();

ComboInfoDTO comboInfoResult = new ComboInfoDTO();

comboInfoResult.setBrokerageType(brokerageType);

comboInfoResult.setiNLAFeeType(iNLAFeeType);

comboInfoResult.setTaxTypeCalculationCredit(taxTypeCalculationCredit);

comboInfoResult.setTaxTypeCalculationDebit(taxTypeCalculationDebit);

comboInfoResult.setOtherDeductionType(otherDeductionType);

comboInfoResult.setEscFeeType(escFeeType);

comboInfoResult.setDeductionCedingType(deductionCedingType);

comboInfoResult.setTaxType(TaxType);

comboInfoResult.setDeductionName(deductionName);

comboInfoResult.setInsuranceStructures(insuranceStructures);

comboInfoResult.setAllPolicyStatus(allPolicyStatus);

return comboInfoResult;

}

@Override

public List<PolicyDatesWithClaimsDTO> getClaimsByUniquePolicyId(String id) {

List<PolicyYear> allPolicyYear = policyYearRepository.findByUniqueIdentifierOrderByYearOfAccountDesc(id);

List<PolicyDatesWithClaimsDTO> policyDatesWithClaimsDTOList = new ArrayList<PolicyDatesWithClaimsDTO>();

for (PolicyYear policy : allPolicyYear) {

List<ClaimShortDTO> claimList = claimService.getClaimByPolicyID(policy.getId());

for (ClaimShortDTO claimShortDTO : claimList) {

PolicyDatesWithClaimsDTO policyDatesWithClaimsDTO = new PolicyDatesWithClaimsDTO();

policyDatesWithClaimsDTO.setId(claimShortDTO.getClaimID());

policyDatesWithClaimsDTO.setpolicyId(policy.getId());

policyDatesWithClaimsDTO.setPolicyYear(policy.getYearOfAccount());

policyDatesWithClaimsDTO.setDateOfLoss(claimShortDTO.getDateOfLoss());

policyDatesWithClaimsDTO.setClaimRef(claimShortDTO.getClaimNumber());

policyDatesWithClaimsDTO.setPaidFigure(claimShortDTO.getTotalPayments());

policyDatesWithClaimsDTO.setPaidFigureGBP(claimShortDTO.getTotalPaymentsGBP());

policyDatesWithClaimsDTO.setEstimateFigure(claimShortDTO.getCurrentEstimate());

policyDatesWithClaimsDTO.setEstimateFigureGBP(claimShortDTO.getCurrentEstimateGBP());

policyDatesWithClaimsDTO.setCurrencySymbol(

Objects.isNull(claimShortDTO.getCurrency()) ? "" : claimShortDTO.getCurrency().getSymbol());

policyDatesWithClaimsDTOList.add(policyDatesWithClaimsDTO);

}

}

return policyDatesWithClaimsDTOList;

}

@Override

public Boolean isRenew(String id) {

Policy check = this.repository.findByUniqueIdentifierAndPolicyVersionAndQuoteNumber(id,"00",null);

if (Objects.isNull(check)) {

return false;

}

return true;

}

@Override

public Map<String, Object> getPolicyRenewalHistory(HashMap<String, Object> filters) {

if (filters != null && !filters.isEmpty() && filters.keySet() != null && filters.values() != null

&& filters.get("policyUniqueIdentifier") != null && filters.get("policyNumberToExclude") != null

&& filters.get("pageNumber") != null && filters.get("pageSize") != null) {

Map<String, Object> response = new HashMap<String, Object>();

Pageable pageable = new PageRequest((Integer) filters.get("pageNumber"), (Integer) filters.get("pageSize"));

Page<PolicyRenewalHistoryItemDTO> policyRenewalHistoryPage = this.repository.findPolicyRenewalHistory(

CastUtil.toString(filters.get("policyUniqueIdentifier")),

CastUtil.toString(filters.get("policyNumberToExclude")), pageable);

if (policyRenewalHistoryPage != null) {

List<PolicyRenewalHistoryItemDTO> renewalHistoryList = policyRenewalHistoryPage.getContent();

response.put("policyRenewalHistory", renewalHistoryList);

response.put("totalPages", policyRenewalHistoryPage.getTotalPages());

response.put("totalElements", policyRenewalHistoryPage.getTotalElements());

return response;

}

}

return null;

}

@Override

public List<UnderwritingTaskDTO> getAllUnderwritingTasks() {

List<UnderwritingTaskDTO> response = new ArrayList<UnderwritingTaskDTO>();

List<UnderwritingTaskConfiguration> tasks = uwTaskConfigurationRepository.findAllByOrderBySectionDescOrderAsc();

for (UnderwritingTaskConfiguration task : tasks) {

UnderwritingTaskDTO taskDTO = new UnderwritingTaskDTO();

taskDTO.setConfiguration( createConfiguration(task) );

taskDTO.setCode(task.getCode());

taskDTO.setSummary(task.getSummary());

taskDTO.setDeadline(task.getDeadline());

taskDTO.setDetails(task.getDetails());

taskDTO.setReviewCheckpoint(task.getReviewCheckpoint());

taskDTO.setOrder(task.getOrder());

taskDTO.setSection(task.getSection());

response.add(taskDTO);

}

return response;

}

private UnderwritingTaskConfigurationDTO createConfiguration(UnderwritingTaskConfiguration taskConfiguration) {

UnderwritingTaskConfigurationDTO result = new UnderwritingTaskConfigurationDTO();

result.setId(taskConfiguration.getId());

result.setActive(taskConfiguration.getActive());

result.setVersion(taskConfiguration.getVersion());

result.setCode(taskConfiguration.getCode());

result.setSummary(taskConfiguration.getSummary());

result.setDeadline(taskConfiguration.getDeadline());

result.setDetails(taskConfiguration.getDetails());

result.setReviewCheckpoint(taskConfiguration.getReviewCheckpoint() );

result.setOrder(taskConfiguration.getOrder());

result.setSection(taskConfiguration.getSection());

return result;

}

}