/\*

\* Copyright (c) 2019. All right reserved

\* Created on 2022-08-24 ( Date ISO 2022-08-24 - Time 12:58:26 )

\* Generated by Telosys Tools Generator ( version 3.3.0 )

\*/

package com.maan.eway.master.service.impl;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Calendar;

import java.util.Comparator;

import java.util.Date;

import java.util.GregorianCalendar;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import java.util.function.Function;

import java.util.stream.Collectors;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import javax.persistence.TypedQuery;

import javax.persistence.criteria.CriteriaBuilder;

import javax.persistence.criteria.CriteriaDelete;

import javax.persistence.criteria.CriteriaQuery;

import javax.persistence.criteria.Order;

import javax.persistence.criteria.Predicate;

import javax.persistence.criteria.Root;

import javax.persistence.criteria.Subquery;

import org.apache.commons.lang3.StringUtils;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import org.dozer.DozerBeanMapper;

import org.modelmapper.ModelMapper;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.google.gson.Gson;

import com.maan.eway.master.req.CurrencyMasterGetAllReq;

import com.maan.eway.master.req.CurrencyMasterGetReq;

import com.maan.eway.master.req.CurrencyMasterSaveReq;

import com.maan.eway.master.req.OccupationChangeStatusReq;

import com.maan.eway.master.req.OccupationMasterGetAllReq;

import com.maan.eway.master.req.OccupationMasterGetReq;

import com.maan.eway.master.req.OccupationMasterSaveReq;

import com.maan.eway.master.res.CurrencyMasterRes;

import com.maan.eway.master.res.ExchangeMasterGetRes;

import com.maan.eway.master.res.OccupationMasterRes;

import com.maan.eway.master.service.CurrencyMasterService;

import com.maan.eway.master.service.OccupationMasterService;

import com.maan.eway.bean.AcExecutiveMaster;

import com.maan.eway.bean.BranchMaster;

import com.maan.eway.bean.CurrencyMaster;

import com.maan.eway.bean.ExchangeMaster;

import com.maan.eway.bean.OccupationMaster;

import com.maan.eway.bean.ProductMaster;

import com.maan.eway.bean.StateMaster;

import com.maan.eway.error.Error;

import com.maan.eway.repository.CurrencyMasterRepository;

import com.maan.eway.repository.OccupationMasterRepository;

import com.maan.eway.res.DropDownRes;

import com.maan.eway.res.SuccessRes;

import com.maan.eway.service.impl.BasicValidationService;

/\*\*

\* <h2>CurrencyMasterServiceimpl</h2>

\*/

@Service

@Transactional

public class OccupationMasterServiceImpl implements OccupationMasterService {

@PersistenceContext

private EntityManager em;

@Autowired

private OccupationMasterRepository repo;

@Autowired

private BasicValidationService basicvalidateService;

Gson json = new Gson();

private Logger log=LogManager.getLogger(OccupationMasterServiceImpl.class);

@Override

public List<Error> validateOccupation(OccupationMasterSaveReq req) {

List<Error> errorList = new ArrayList<Error>();

try {

if (StringUtils.isBlank(req.getOccupationName())) {

errorList.add(new Error("02", "OccupationName", "Please Select OccupationName"));

}else if (req.getOccupationName().length() > 100){

errorList.add(new Error("02","OccupationName", "Please Enter OccupationName 100 Characters"));

}else if (StringUtils.isBlank(req.getOccupationId()) && StringUtils.isNotBlank(req.getInsuranceId()) && StringUtils.isNotBlank(req.getBranchCode())) {

List<OccupationMaster> OccupationList = getOccupationNameExistDetails(req.getOccupationName() , req.getInsuranceId() , req.getBranchCode());

if (OccupationList.size()>0 ) {

errorList.add(new Error("01", "OccupationName", "This Occupation Name Already Exist "));

}

}else if (StringUtils.isNotBlank(req.getOccupationId()) && StringUtils.isNotBlank(req.getInsuranceId()) && StringUtils.isNotBlank(req.getBranchCode())) {

List<OccupationMaster> OccupationList = getOccupationNameExistDetails(req.getOccupationName() , req.getInsuranceId() , req.getBranchCode());

if (OccupationList.size()>0 && (! req.getOccupationId().equalsIgnoreCase(OccupationList.get(0).getOccupationId().toString())) ) {

errorList.add(new Error("01", "OccupationName", "This Occupation Name Already Exist "));

}

}

if (StringUtils.isBlank(req.getInsuranceId())) {

errorList.add(new Error("02", "InsuranceId", "Please Enter InsuranceId"));

}

if (StringUtils.isBlank(req.getBranchCode())) {

errorList.add(new Error("02", "BranchCode", "Please Select BranchCode"));

}

/\* if (StringUtils.isBlank(req.getOccupationNameAr())) {

errorList.add(new Error("03", "OccupationNameAr", "Please Select OccupationNameAr"));

}else if (req.getOccupationNameAr().length() > 100){

errorList.add(new Error("03","OccupationNameAr", "Please Enter OccupationNameAr 100 Characters"));

} \*/

if (StringUtils.isBlank(req.getRemarks())) {

errorList.add(new Error("04", "Remarks", "Please Select Remarks "));

}else if (req.getRemarks().length() > 100){

errorList.add(new Error("04","Remarks", "Please Enter Remarks within 100 Characters"));

}

// Date Validation

Calendar cal = new GregorianCalendar();

Date today = new Date();

cal.setTime(today);cal.add(Calendar.DAY\_OF\_MONTH, -1);;

today = cal.getTime();

if (req.getEffectiveDateStart() == null || StringUtils.isBlank(req.getEffectiveDateStart().toString())) {

errorList.add(new Error("05", "EffectiveDateStart", "Please Enter Effective Date Start"));

} else if (req.getEffectiveDateStart().before(today)) {

errorList.add(new Error("05", "EffectiveDateStart", "Please Enter Effective Date Start as Future Date"));

}

//Status Validation

if (StringUtils.isBlank(req.getStatus())) {

errorList.add(new Error("06", "Status", "Please Enter Status"));

} else if (req.getStatus().length() > 1) {

errorList.add(new Error("06", "Status", "Enter Status in 1 Character Only"));

}else if(!("Y".equalsIgnoreCase(req.getStatus())||"N".equalsIgnoreCase(req.getStatus()) || "R".equalsIgnoreCase(req.getStatus()))) {

errorList.add(new Error("06", "Status", "Enter Status in Y or N or R Only"));

}

if (StringUtils.isBlank(req.getCoreAppCode())) {

errorList.add(new Error("07", "CoreAppCode", "Please Select CoreAppCode"));

}else if (req.getCoreAppCode().length() > 20){

errorList.add(new Error("07","CoreAppCode", "Please Enter CoreAppCode within 20 Characters"));

}

if (StringUtils.isBlank(req.getRegulatoryCode())) {

errorList.add(new Error("08", "RegulatoryCode", "Please Select RegulatoryCode"));

}else if (req.getRegulatoryCode().length() > 20){

errorList.add(new Error("08","RegulatoryCode", "Please Enter RegulatoryCode within 20 Characters"));

}

if (StringUtils.isBlank(req.getCreatedBy())) {

errorList.add(new Error("09", "CreatedBy", "Please Select CreatedBy"));

}else if (req.getCreatedBy().length() > 100){

errorList.add(new Error("09","CreatedBy", "Please Enter CreatedBy within 100 Characters"));

}

} catch (Exception e) {

log.error(e);

e.printStackTrace();

}

return errorList;

}

public List<OccupationMaster> getOccupationNameExistDetails(String occupationName , String InsuranceId , String branchCode) {

List<OccupationMaster> list = new ArrayList<OccupationMaster>();

try {

Date today = new Date();

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<OccupationMaster> query = cb.createQuery(OccupationMaster.class);

// Find All

Root<OccupationMaster> b = query.from(OccupationMaster.class);

// Select

query.select(b);

// Effective Date Max Filter

Subquery<Long> amendId = query.subquery(Long.class);

Root<OccupationMaster> ocpm1 = amendId.from(OccupationMaster.class);

amendId.select(cb.max(ocpm1.get("amendId")));

Predicate a1 = cb.equal(ocpm1.get("occupationId"), b.get("occupationId"));

Predicate a2 = cb.equal(ocpm1.get("companyId"), b.get("companyId"));

Predicate a3 = cb.equal(ocpm1.get("branchCode"), b.get("branchCode"));

Predicate a4 = cb.lessThanOrEqualTo(ocpm1.get("effectiveDateStart"), today);

Predicate a5 = cb.greaterThanOrEqualTo(ocpm1.get("effectiveDateEnd"), today);

amendId.where(a1,a2,a3,a4,a5);

Predicate n1 = cb.equal(b.get("amendId"), amendId);

Predicate n2 = cb.equal(cb.lower( b.get("occupationName")), occupationName.toLowerCase());

Predicate n3 = cb.equal(b.get("companyId"),InsuranceId);

Predicate n4 = cb.equal(b.get("branchCode"), branchCode);

Predicate n5 = cb.equal(b.get("branchCode"), "99999");

Predicate n6 = cb.or(n4,n5);

query.where(n1,n2,n3,n6);

// Get Result

TypedQuery<OccupationMaster> result = em.createQuery(query);

list = result.getResultList();

} catch (Exception e) {

e.printStackTrace();

log.info(e.getMessage());

}

return list;

}

@Override

public SuccessRes insertOccupation(OccupationMasterSaveReq req) {

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

SuccessRes res = new SuccessRes();

OccupationMaster saveData = new OccupationMaster();

List<OccupationMaster> list = new ArrayList<OccupationMaster>();

DozerBeanMapper dozerMapper = new DozerBeanMapper();

try {

Integer amendId=0;

Date startDate = req.getEffectiveDateStart() ;

String end = "31/12/2050";

Date endDate = sdf.parse(end);

long MILLIS\_IN\_A\_DAY = 1000 \* 60 \* 60 \* 24;

Date oldEndDate = new Date(req.getEffectiveDateStart().getTime() - MILLIS\_IN\_A\_DAY);

Date entryDate = null ;

String createdBy = "" ;

Integer occupationId = 0 ;

if(StringUtils.isBlank(req.getOccupationId())) {

// Save

Integer totalCount = getMasterTableCount( req.getInsuranceId() , req.getBranchCode());

occupationId = totalCount+1 ;

entryDate = new Date();

createdBy = req.getCreatedBy();

res.setResponse("Saved Successfully");

res.setSuccessId(occupationId.toString());

}

else {

// Update

occupationId = Integer.valueOf(req.getOccupationId());

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<OccupationMaster> query = cb.createQuery(OccupationMaster.class);

//Find all

Root<OccupationMaster> b = query.from(OccupationMaster.class);

//Select

query.select(b);

// //Effective Date Max Filter

// Subquery<Long> effectiveDate = query.subquery(Long.class);

// Root<OccupationMaster> ocpm1 = effectiveDate.from(OccupationMaster.class);

// effectiveDate.select(ocpm1.get("effectiveDateStart"));

// Predicate a1 = cb.equal(ocpm1.get("occupationId"), b.get("occupationId"));

// Predicate a2 = cb.lessThanOrEqualTo(ocpm1.get("effectiveDateStart"),startDate);

//

// effectiveDate.where(a1,a2);

// Order By

List<Order> orderList = new ArrayList<Order>();

orderList.add(cb.desc(b.get("effectiveDateStart")));

// Where

// Predicate n1 = cb.equal(b.get("effectiveDateStart"), effectiveDate);

Predicate n2 = cb.equal(b.get("occupationId"), req.getOccupationId());

Predicate n3 = cb.equal(b.get("companyId"), req.getInsuranceId());

Predicate n4 = cb.equal(b.get("branchCode"), req.getBranchCode());

query.where(n2,n3,n4).orderBy(orderList);

// Get Result

TypedQuery<OccupationMaster> result = em.createQuery(query);

int limit = 0 , offset = 2 ;

result.setFirstResult(limit \* offset);

result.setMaxResults(offset);

list = result.getResultList();

if(list.size()>0) {

Date beforeOneDay = new Date(new Date().getTime() - MILLIS\_IN\_A\_DAY);

if ( list.get(0).getEffectiveDateStart().before(beforeOneDay) ) {

amendId = list.get(0).getAmendId() + 1 ;

entryDate = new Date() ;

createdBy = req.getCreatedBy();

OccupationMaster lastRecord = list.get(0);

lastRecord.setEffectiveDateEnd(oldEndDate);

repo.saveAndFlush(lastRecord);

} else {

amendId = list.get(0).getAmendId() ;

entryDate = list.get(0).getEntryDate() ;

createdBy = list.get(0).getCreatedBy();

saveData = list.get(0) ;

if (list.size()>1 ) {

OccupationMaster lastRecord = list.get(1);

lastRecord.setEffectiveDateEnd(oldEndDate);

repo.saveAndFlush(lastRecord);

}

}

}

res.setResponse("Updated Successfully");

res.setSuccessId(occupationId.toString());

}

dozerMapper.map(req, saveData);

saveData.setOccupationId(occupationId);

saveData.setEffectiveDateStart(startDate);

saveData.setEffectiveDateEnd(endDate);

saveData.setCreatedBy(createdBy);

saveData.setStatus(req.getStatus());

saveData.setCompanyId(req.getInsuranceId());

saveData.setEntryDate(entryDate);

saveData.setUpdatedDate(new Date());

saveData.setUpdatedBy(req.getCreatedBy());

saveData.setAmendId(amendId);

saveData.setCoreAppcode(req.getCoreAppCode());

repo.saveAndFlush(saveData);

log.info("Saved Details is --> " + json.toJson(saveData));

}

catch (Exception e) {

e.printStackTrace();

log.info("Exception is --> "+ e.getMessage());

return null;

}

return res;

}

public Integer getMasterTableCount(String companyId , String branchCode) {

Integer data =0;

try {

List<OccupationMaster> list = new ArrayList<OccupationMaster>();

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<OccupationMaster> query = cb.createQuery(OccupationMaster.class);

// Find all

Root<OccupationMaster> b = query.from(OccupationMaster.class);

//Select

query.select(b);

//Effective Date Max Filter

Subquery<Long> effectiveDate = query.subquery(Long.class);

Root<OccupationMaster> ocpm1 = effectiveDate.from(OccupationMaster.class);

effectiveDate.select(cb.max(ocpm1.get("effectiveDateStart")));

Predicate a1 = cb.equal(ocpm1.get("occupationId"), b.get("occupationId"));

Predicate a2 = cb.equal(ocpm1.get("companyId"), b.get("companyId"));

Predicate a3 = cb.equal(ocpm1.get("branchCode"), b.get("branchCode"));

effectiveDate.where(a1,a2,a3);

// Order By

List<Order> orderList = new ArrayList<Order>();

orderList.add(cb.desc(b.get("occupationId")));

Predicate n1 = cb.equal(b.get("effectiveDateStart"), effectiveDate);

Predicate n2 = cb.equal(b.get("companyId"), companyId);

Predicate n3 = cb.equal(b.get("branchCode"), branchCode);

Predicate n4 = cb.equal(b.get("branchCode"), "99999");

Predicate n5 = cb.or(n3,n4);

query.where(n1,n2,n5).orderBy(orderList);

// Get Result

TypedQuery<OccupationMaster> result = em.createQuery(query);

int limit = 0 , offset = 1 ;

result.setFirstResult(limit \* offset);

result.setMaxResults(offset);

list = result.getResultList();

data = list.size() > 0 ? list.get(0).getOccupationId() : 0 ;

}

catch(Exception e) {

e.printStackTrace();

log.info(e.getMessage());

}

return data;

}

@Override

public List<OccupationMasterRes> getallOccupation(OccupationMasterGetAllReq req) {

List<OccupationMasterRes> resList = new ArrayList<OccupationMasterRes>();

DozerBeanMapper mapper = new DozerBeanMapper();

try {

List<OccupationMaster> list = new ArrayList<OccupationMaster>();

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<OccupationMaster> query = cb.createQuery(OccupationMaster.class);

// Find All

Root<OccupationMaster> b = query.from(OccupationMaster.class);

// Select

query.select(b);

// Amend ID Max Filter

Subquery<Long> amendId = query.subquery(Long.class);

Root<OccupationMaster> ocpm1 = amendId.from(OccupationMaster.class);

amendId.select(cb.max(ocpm1.get("amendId")));

Predicate a1 = cb.equal(ocpm1.get("occupationId"), b.get("occupationId"));

Predicate a2 = cb.equal(ocpm1.get("companyId"), b.get("companyId"));

Predicate a3 = cb.equal(ocpm1.get("branchCode"),b.get("branchCode"));

amendId.where(a1, a2,a3);

// Order By

List<Order> orderList = new ArrayList<Order>();

orderList.add(cb.asc(b.get("branchCode")));

// Where

Predicate n1 = cb.equal(b.get("amendId"), amendId);

Predicate n2 = cb.equal(b.get("companyId"), req.getInsuranceId());

Predicate n3 = cb.equal(b.get("branchCode"), req.getBranchCode());

Predicate n4 = cb.equal(b.get("branchCode"), "99999");

Predicate n5 = cb.or(n3,n4);

query.where(n1,n2,n5).orderBy(orderList);

// Get Result

TypedQuery<OccupationMaster> result = em.createQuery(query);

list = result.getResultList();

list = list.stream().filter(distinctByKey(o -> Arrays.asList(o.getOccupationId()))).collect(Collectors.toList());

list.sort(Comparator.comparing(OccupationMaster :: getOccupationName ));

// Map

for (OccupationMaster data : list) {

OccupationMasterRes res = new OccupationMasterRes();

res = mapper.map(data, OccupationMasterRes.class);

res.setCoreAppCode(data.getCoreAppcode());

resList.add(res);

}

} catch (Exception e) {

e.printStackTrace();

log.info(e.getMessage());

return null;

}

return resList;

}

private static <T> java.util.function.Predicate<T> distinctByKey(java.util.function.Function<? super T, ?> keyExtractor) {

Map<Object, Boolean> seen = new ConcurrentHashMap<>();

return t -> seen.putIfAbsent(keyExtractor.apply(t), Boolean.TRUE) == null;

}

@Override

public List<OccupationMasterRes> getActiveOccupation(OccupationMasterGetAllReq req) {

List<OccupationMasterRes> resList = new ArrayList<OccupationMasterRes>();

DozerBeanMapper mapper = new DozerBeanMapper();

try {

List<OccupationMaster> list = new ArrayList<OccupationMaster>();

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<OccupationMaster> query = cb.createQuery(OccupationMaster.class);

// Find All

Root<OccupationMaster> b = query.from(OccupationMaster.class);

// Select

query.select(b);

// Amend ID Max Filter

Subquery<Long> amendId = query.subquery(Long.class);

Root<OccupationMaster> ocpm1 = amendId.from(OccupationMaster.class);

amendId.select(cb.max(ocpm1.get("amendId")));

Predicate a1 = cb.equal(ocpm1.get("occupationId"), b.get("occupationId"));

Predicate a2 = cb.equal(ocpm1.get("companyId"), b.get("companyId"));

Predicate a3 = cb.equal(ocpm1.get("branchCode"),b.get("branchCode"));

amendId.where(a1, a2,a3);

// Order By

List<Order> orderList = new ArrayList<Order>();

orderList.add(cb.asc(b.get("branchCode")));

// Where

Predicate n1 = cb.equal(b.get("amendId"), amendId);

Predicate n2 = cb.equal(b.get("companyId"), req.getInsuranceId());

Predicate n3 = cb.equal(b.get("branchCode"), req.getBranchCode());

Predicate n4 = cb.equal(b.get("status"), "Y");

Predicate n5 = cb.equal(b.get("branchCode"), "99999");

Predicate n6 = cb.or(n3,n5);

query.where(n1,n2,n4,n6).orderBy(orderList);

// Get Result

TypedQuery<OccupationMaster> result = em.createQuery(query);

list = result.getResultList();

list = list.stream().filter(distinctByKey(o -> Arrays.asList(o.getOccupationId()))).collect(Collectors.toList());

list.sort(Comparator.comparing(OccupationMaster :: getOccupationName ));

// Map

for (OccupationMaster data : list) {

OccupationMasterRes res = new OccupationMasterRes();

res = mapper.map(data, OccupationMasterRes.class);

res.setCoreAppCode(data.getCoreAppcode());

resList.add(res);

}

} catch (Exception e) {

e.printStackTrace();

log.info(e.getMessage());

return null;

}

return resList;

}

@Override

public OccupationMasterRes getByOccupationId(OccupationMasterGetReq req) {

OccupationMasterRes res = new OccupationMasterRes();

DozerBeanMapper mapper = new DozerBeanMapper();

try {

Date today = new Date();

Calendar cal = new GregorianCalendar();

cal.setTime(today);

cal.set(Calendar.HOUR\_OF\_DAY, 23);

cal.set(Calendar.MINUTE, 1);

today = cal.getTime();

List<OccupationMaster> list = new ArrayList<OccupationMaster>();

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<OccupationMaster> query = cb.createQuery(OccupationMaster.class);

// Find All

Root<OccupationMaster> b = query.from(OccupationMaster.class);

// Select

query.select(b);

// Amend ID Max Filter

Subquery<Long> amendId = query.subquery(Long.class);

Root<OccupationMaster> ocpm1 = amendId.from(OccupationMaster.class);

amendId.select(cb.max(ocpm1.get("amendId")));

Predicate a1 = cb.equal(ocpm1.get("occupationId"), b.get("occupationId"));

Predicate a2 = cb.equal(ocpm1.get("companyId"), b.get("companyId"));

Predicate a3 = cb.equal(ocpm1.get("branchCode"),b.get("branchCode"));

amendId.where(a1, a2,a3);

// Order By

List<Order> orderList = new ArrayList<Order>();

orderList.add(cb.asc(b.get("branchCode")));

// Where

Predicate n1 = cb.equal(b.get("amendId"), amendId);

Predicate n2 = cb.equal(b.get("companyId"), req.getInsuranceId());

Predicate n3 = cb.equal(b.get("branchCode"), req.getBranchCode());

Predicate n4 = cb.equal(b.get("occupationId"), req.getOccupationId());

Predicate n6 = cb.equal(b.get("branchCode"), "99999");

Predicate n7 = cb.or(n3,n6);

query.where(n1,n2,n4,n7).orderBy(orderList);

// Get Result

TypedQuery<OccupationMaster> result = em.createQuery(query);

list = result.getResultList();

list = list.stream().filter(distinctByKey(o -> Arrays.asList(o.getOccupationId()))).collect(Collectors.toList());

list.sort(Comparator.comparing(OccupationMaster :: getOccupationName ));

res = mapper.map(list.get(0), OccupationMasterRes.class);

res.setOccupationId(list.get(0).getOccupationId().toString());

res.setEntryDate(list.get(0).getEntryDate());

res.setEffectiveDateStart(list.get(0).getEffectiveDateStart());

res.setEffectiveDateEnd(list.get(0).getEffectiveDateEnd());

res.setCoreAppCode(list.get(0).getCoreAppcode());

} catch (Exception e) {

e.printStackTrace();

log.info("Exception is ---> " + e.getMessage());

return null;

}

return res;

}

/\*

@Override

public List<DropDownRes> getOccupationMasterDropdown() {

List<DropDownRes> resList = new ArrayList<DropDownRes>();

try {

Date today = new Date();

Calendar cal = new GregorianCalendar();

cal.setTime(today);

cal.set(Calendar.HOUR\_OF\_DAY, 23);;

cal.set(Calendar.MINUTE, 1);

today = cal.getTime();

cal.set(Calendar.HOUR\_OF\_DAY, 1);

cal.set(Calendar.MINUTE, 1);

Date todayEnd = cal.getTime();

// Criteria

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<OccupationMaster> query= cb.createQuery(OccupationMaster.class);

List<OccupationMaster> list = new ArrayList<OccupationMaster>();

// Find All

Root<OccupationMaster> c = query.from(OccupationMaster.class);

//Select

query.select(c);

// Order By

List<Order> orderList = new ArrayList<Order>();

orderList.add(cb.asc(c.get("occupationName")));

// Effective Date Start Max Filter

Subquery<Long> effectiveDate = query.subquery(Long.class);

Root<OccupationMaster> ocpm1 = effectiveDate.from(OccupationMaster.class);

effectiveDate.select(cb.max(ocpm1.get("effectiveDateStart")));

Predicate a1 = cb.equal(c.get("occupationId"),ocpm1.get("occupationId"));

Predicate a2 = cb.lessThanOrEqualTo(ocpm1.get("effectiveDateStart"), today);

effectiveDate.where(a1,a2);

// Effective Date End Max Filter

Subquery<Long> effectiveDate2 = query.subquery(Long.class);

Root<OccupationMaster> ocpm2 = effectiveDate2.from(OccupationMaster.class);

effectiveDate2.select(cb.max(ocpm2.get("effectiveDateEnd")));

Predicate a3 = cb.equal(c.get("occupationId"),ocpm2.get("occupationId"));

Predicate a4 = cb.greaterThanOrEqualTo(ocpm2.get("effectiveDateEnd"), todayEnd);

effectiveDate2.where(a3,a4);

// Where

Predicate n1 = cb.equal(c.get("status"),"Y");

Predicate n2 = cb.equal(c.get("effectiveDateStart"),effectiveDate);

Predicate n3 = cb.equal(c.get("effectiveDateEnd"),effectiveDate2);

query.where(n1,n2,n3).orderBy(orderList);

// Get Result

TypedQuery<OccupationMaster> result = em.createQuery(query);

list = result.getResultList();

for (OccupationMaster data : list) {

// Response

DropDownRes res = new DropDownRes();

res.setCode(data.getOccupationId());

res.setCodeDesc(data.getOccupationName());

resList.add(res);

}

}

catch(Exception e) {

e.printStackTrace();

log.info("Exception is --->"+e.getMessage());

return null;

}

return resList;

}

\*/

@Override

public SuccessRes changeStatusOfOccupation(OccupationChangeStatusReq req) {

SuccessRes res = new SuccessRes();

DozerBeanMapper dozerMapper = new DozerBeanMapper();

try {

List<OccupationMaster> list = new ArrayList<OccupationMaster>();

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<OccupationMaster> query = cb.createQuery(OccupationMaster.class);

// Find all

Root<OccupationMaster> b = query.from(OccupationMaster.class);

//Select

query.select(b);

// Amend ID Max Filter

Subquery<Long> amendId = query.subquery(Long.class);

Root<OccupationMaster> ocpm1 = amendId.from(OccupationMaster.class);

amendId.select(cb.max(ocpm1.get("amendId")));

Predicate a1 = cb.equal(ocpm1.get("occupationId"), b.get("occupationId"));

Predicate a2 = cb.equal(ocpm1.get("companyId"), b.get("companyId"));

Predicate a3 = cb.equal(ocpm1.get("branchCode"),b.get("branchCode"));

amendId.where(a1, a2,a3);

// Order By

List<Order> orderList = new ArrayList<Order>();

orderList.add(cb.asc(b.get("branchCode")));

// Where

Predicate n1 = cb.equal(b.get("amendId"), amendId);

Predicate n2 = cb.equal(b.get("companyId"), req.getInsuranceId());

Predicate n3 = cb.equal(b.get("branchCode"), req.getBranchCode());

Predicate n4 = cb.equal(b.get("occupationId"), req.getOccupationId());

Predicate n5 = cb.equal(b.get("branchCode"), "99999");

Predicate n6 = cb.or(n3,n5);

query.where(n1,n2,n4,n6).orderBy(orderList);

// Get Result

TypedQuery<OccupationMaster> result = em.createQuery(query);

list = result.getResultList();

OccupationMaster updateRecord = list.get(0);

if( req.getBranchCode().equalsIgnoreCase(updateRecord.getBranchCode())) {

updateRecord.setStatus(req.getStatus());

repo.save(updateRecord);

} else {

OccupationMaster saveNew = new OccupationMaster();

dozerMapper.map(updateRecord,saveNew);

saveNew.setBranchCode(req.getBranchCode());

saveNew.setStatus(req.getStatus());

repo.save(saveNew);

}

// Perform Update

res.setResponse("Status Changed");

res.setSuccessId(req.getOccupationId());

}

catch (Exception e) {

e.printStackTrace();

log.info("Exception is --> " + e.getMessage());

return null;

}

return res;

}

@Override

public List<DropDownRes> getAcExecutivesDropdown(AcExecutiveDropDownReq req) {

List<DropDownRes> resList = new ArrayList<DropDownRes>();

try {

Date today = new Date();

Calendar cal = new GregorianCalendar();

cal.setTime(today);

cal.set(Calendar.HOUR\_OF\_DAY, 23);;

cal.set(Calendar.MINUTE, 1);

today = cal.getTime();

cal.set(Calendar.HOUR\_OF\_DAY, 1);

cal.set(Calendar.MINUTE, 1);

Date todayEnd = cal.getTime();

// Criteria

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<AcExecutiveMaster> query= cb.createQuery(AcExecutiveMaster.class);

List<AcExecutiveMaster> list = new ArrayList<AcExecutiveMaster>();

// Find All

Root<AcExecutiveMaster> c = query.from(AcExecutiveMaster.class);

//Select

query.select(c);

// Order By

List<Order> orderList = new ArrayList<Order>();

orderList.add(cb.desc(c.get("effectiveDateStart")));

// Effective Date Start Max Filter

Subquery<Long> effectiveDate = query.subquery(Long.class);

Root<AcExecutiveMaster> ocpm1 = effectiveDate.from(AcExecutiveMaster.class);

effectiveDate.select(cb.max(ocpm1.get("effectiveDateStart")));

Predicate a1 = cb.equal(c.get("acExecutiveId"),ocpm1.get("acExecutiveId"));

Predicate a2 = cb.lessThanOrEqualTo(ocpm1.get("effectiveDateStart"), today);

effectiveDate.where(a1,a2);

// Effective Date End Max Filter

Subquery<Long> effectiveDate2 = query.subquery(Long.class);

Root<AcExecutiveMaster> ocpm2 = effectiveDate2.from(AcExecutiveMaster.class);

effectiveDate2.select(cb.max(ocpm2.get("effectiveDateEnd")));

Predicate a3 = cb.equal(c.get("acExecutiveId"),ocpm2.get("acExecutiveId"));

Predicate a4 = cb.greaterThanOrEqualTo(ocpm2.get("effectiveDateEnd"), todayEnd);

effectiveDate2.where(a3,a4);

// Where

Predicate n1 = cb.equal(c.get("status"),"Y");

Predicate n2 = cb.equal(c.get("effectiveDateStart"),effectiveDate);

Predicate n3 = cb.equal(c.get("effectiveDateEnd"),effectiveDate2);

Predicate n4 = cb.equal(c.get("oaCode"),req.getOaCode());

Predicate n5 = cb.notEqual(c.get("acExecutiveId"),"1");

query.where(n1,n2,n3,n4,n5).orderBy(orderList);

// Get Result

TypedQuery<AcExecutiveMaster> result = em.createQuery(query);

list = result.getResultList();

for (AcExecutiveMaster data : list) {

// Response

DropDownRes res = new DropDownRes();

res.setCode(data.getAcExecutiveId().toString());

res.setCodeDesc(data.getAcExecutiveName());

resList.add(res);

}

}

catch(Exception e) {

e.printStackTrace();

log.info("Exception is --->"+e.getMessage());

return null;

}

return resList;

}

}