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.stream.Collectors;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import javax.persistence.TypedQuery;

import javax.persistence.criteria.CriteriaBuilder;

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.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.google.gson.Gson;

import com.maan.eway.bean.ExclusionMaster;

import com.maan.eway.bean.OccupationMaster;

import com.maan.eway.bean.WarrantyMaster;

import com.maan.eway.error.Error;

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

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

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

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

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

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

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

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

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

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

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

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

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

import com.maan.eway.repository.ExclusionMasterRepository;

import com.maan.eway.repository.WarrantyMasterRepository;

import com.maan.eway.res.SuccessRes;

@Service

public class WarrantyMasterServiceImpl implements WarrantyMasterService {

@PersistenceContext

private EntityManager em;

@Autowired

private WarrantyMasterRepository repo;

Gson json = new Gson();

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

@Override

public List<Error> validateWarranty(WarrantyMasterSaveReq req) {

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

try {

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

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

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

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

}else if (StringUtils.isBlank(req.getWarrantyId()) && StringUtils.isNotBlank(req.getCompanyId()) && StringUtils.isNotBlank(req.getBranchCode())) {

List<WarrantyMaster> WarrantyList = getWarrantyDescriptionExistDetails(req.getWarrantyDescription() , req.getCompanyId() , req.getBranchCode());

if (WarrantyList.size()>0 ) {

errorList.add(new Error("01", "WarrantyDescription", "This WarrantyDescription Already Exist "));

}

}else if (StringUtils.isNotBlank(req.getWarrantyId()) && StringUtils.isNotBlank(req.getCompanyId()) && StringUtils.isNotBlank(req.getBranchCode())) {

List<WarrantyMaster> WarrantyList = getWarrantyDescriptionExistDetails(req.getWarrantyDescription() , req.getCompanyId() , req.getBranchCode());

if (WarrantyList.size()>0 && (! req.getWarrantyId().equalsIgnoreCase(WarrantyList.get(0).getWarrantyId().toString())) ) {

errorList.add(new Error("01", "WarrantyDescription", "This WarrantyDescription Already Exist "));

}

}

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

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

}

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".equals(req.getStatus())||"N".equals(req.getStatus()) || "R".equals(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<WarrantyMaster> getWarrantyDescriptionExistDetails(String WarrantyDescription , String InsuranceId , String branchCode) {

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

try {

Date today = new Date();

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

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

// Find All

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

// Select

query.select(b);

// Effective Date Max Filter

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

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

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

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

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("warrantyDescription")), WarrantyDescription.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<WarrantyMaster> result = em.createQuery(query);

list = result.getResultList();

} catch (Exception e) {

e.printStackTrace();

log.info(e.getMessage());

}

return list;

}

@Override

public SuccessRes saveWarranty(WarrantyMasterSaveReq req) {

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

SuccessRes res = new SuccessRes();

WarrantyMaster saveData = new WarrantyMaster();

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

DozerBeanMapper dozerMapper = new DozerBeanMapper();

try {

Integer amendId = 0;

Date StartDate = req.getEffectiveDateStart();

String end = "31/12/2050";

Date endDate = sdf.parse(end);

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

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

Date entryDate = null;

String createdBy ="";

Integer warrantyId = 0;

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

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

warrantyId = totalCount+1;

entryDate = new Date();

createdBy = req.getCreatedBy();

res.setResponse("Saved Successfully");

res.setSuccessId(warrantyId.toString());

}

else {

warrantyId = Integer.valueOf(req.getWarrantyId());

CriteriaBuilder cb = em.getCriteriaBuilder();

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

//Findall

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

//select

query.select(b);

//Orderby

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

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

//Where

Predicate n1 = cb.equal(b.get("warrantyId"),req.getWarrantyId());

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

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

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

// Get Result

TypedQuery<WarrantyMaster> 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()- MILLS\_IN\_A\_DAY);

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

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

entryDate = new Date();

createdBy = req.getCreatedBy();

WarrantyMaster 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) {

WarrantyMaster lastRecord = list.get(1);

lastRecord.setEffectiveDateEnd(oldEndDate);

repo.saveAndFlush(lastRecord);

}

}

}

res.setResponse("Updated Successfully");

res.setSuccessId(warrantyId.toString());

}

dozerMapper.map(req, saveData);

saveData.setWarrantyId(warrantyId);

saveData.setEffectiveDateStart(StartDate);

saveData.setEffectiveDateEnd(endDate);

saveData.setCreatedBy(createdBy);

saveData.setEntryDate(entryDate);

saveData.setUpdatedBy(req.getCreatedBy());

saveData.setUpdatedDate(new Date());

saveData.setAmendId(amendId);

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<WarrantyMaster> list = new ArrayList<WarrantyMaster>();

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

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

//Find all

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

// Select

query.select(b);

// Effective Date Max Filter

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

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

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

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

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);

//OrderBy

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

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

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<WarrantyMaster> 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).getWarrantyId() : 0 ;

}

catch(Exception e) {

e.printStackTrace();

log.info(e.getMessage());

}

return data;

}

@Override

public List<WarrantyMasterRes> getallWarranty(WarrantyMasterGetallReq req) {

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

DozerBeanMapper mapper = new DozerBeanMapper();

try {

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

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

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

// Find All

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

// Select

query.select(b);

// Amend ID Max Filter

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

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

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

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

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.getCompanyId());

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<WarrantyMaster> result = em.createQuery(query);

list = result.getResultList();

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

list.sort(Comparator.comparing(WarrantyMaster :: getWarrantyDescription ));

// Map

for (WarrantyMaster data : list) {

WarrantyMasterRes res = new WarrantyMasterRes();

res = mapper.map(data, WarrantyMasterRes.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<WarrantyMasterRes> getActiveWarranty(WarrantyMasterGetallReq req) {

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

DozerBeanMapper mapper = new DozerBeanMapper();

try {

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

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

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

// Find All

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

// Select

query.select(b);

// Amend ID Max Filter

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

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

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

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

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.getCompanyId());

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<WarrantyMaster> result = em.createQuery(query);

list = result.getResultList();

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

list.sort(Comparator.comparing(WarrantyMaster :: getWarrantyDescription ));

// Map

for (WarrantyMaster data : list) {

WarrantyMasterRes res = new WarrantyMasterRes();

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

res.setCoreAppCode(data.getCoreAppCode());

resList.add(res);

}

} catch (Exception e) {

e.printStackTrace();

log.info(e.getMessage());

return null;

}

return resList;

}

@Override

public WarrantyMasterRes getByWarrantyId(WarrantyMasterGetReq req) {

WarrantyMasterRes res = new WarrantyMasterRes();

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<WarrantyMaster> list = new ArrayList<WarrantyMaster>();

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

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

// Find All

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

// Select

query.select(b);

// Amend ID Max Filter

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

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

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

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

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.getCompanyId());

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

Predicate n4 = cb.equal(b.get("warrantyId"), req.getWarrantyId());

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<WarrantyMaster> result = em.createQuery(query);

list = result.getResultList();

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

list.sort(Comparator.comparing(WarrantyMaster :: getWarrantyDescription ));

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

res.setWarrantyId(list.get(0).getWarrantyId().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 SuccessRes changeStatusOfWarranty(WarrantyChangeStatusReq req) {

SuccessRes res = new SuccessRes();

DozerBeanMapper dozerMapper = new DozerBeanMapper();

try {

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

// Find Latest Record

CriteriaBuilder cb = em.getCriteriaBuilder();

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

// Find all

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

//Select

query.select(b);

// Amend ID Max Filter

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

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

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

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

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.getCompanyId());

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

Predicate n4 = cb.equal(b.get("warrantyId"), req.getWarrantyId());

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<WarrantyMaster> result = em.createQuery(query);

list = result.getResultList();

WarrantyMaster updateRecord = list.get(0);

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

updateRecord.setStatus(req.getStatus());

repo.save(updateRecord);

} else {

WarrantyMaster saveNew = new WarrantyMaster();

dozerMapper.map(updateRecord,saveNew);

saveNew.setBranchCode(req.getBranchCode());

saveNew.setStatus(req.getStatus());

repo.save(saveNew);

}

// Perform Update

res.setResponse("Status Changed");

res.setSuccessId(req.getWarrantyId());

}

catch (Exception e) {

e.printStackTrace();

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

return null;

}

return res;

}

}