**Authentication Service**

package com.maan.crm.auth.service.impl;

import java.time.Duration;

import java.time.Instant;

import java.util.Arrays;

import java.util.Date;

import java.util.HashSet;

import java.util.List;

import java.util.Optional;

import java.util.Set;

import javax.servlet.http.HttpServletRequest;

import org.apache.commons.lang3.StringUtils;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

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

import org.springframework.context.annotation.Lazy;

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.stereotype.Service;

import org.springframework.util.CollectionUtils;

import com.maan.crm.auth.dto.ChangePasswordReq;

import com.maan.crm.auth.dto.ClaimLoginResponse;

import com.maan.crm.auth.dto.ClaimLogoutResponse;

import com.maan.crm.auth.dto.CommonLoginResponse;

import com.maan.crm.auth.dto.LoginRequest;

import com.maan.crm.auth.dto.LogoutRequest;

import com.maan.crm.auth.service.AuthendicationService;

import com.maan.crm.auth.token.EncryDecryService;

import com.maan.crm.auth.token.JwtTokenUtil;

import com.maan.crm.auth.token.passwordEnc;

import com.maan.crm.bean.ClaimLoginMaster;

import com.maan.crm.bean.ClaimLoginMasterId;

import com.maan.crm.bean.SessionDetails;

import com.maan.crm.repository.ClaimLoginMasterRepository;

import com.maan.crm.repository.SessionDetailsRepository;

import com.maan.crm.res.CommonCrmRes;

@Lazy

@Service

public class AuthendicationServiceImpl implements AuthendicationService, UserDetailsService {

@Autowired

private BCryptPasswordEncoder bCryptPasswordEncoder;

@Autowired

private JwtTokenUtil jwtTokenUtil;

@Autowired

private ClaimLoginMasterRepository loginRepo;

@Autowired

private SessionDetailsRepository sessionRep;

@Autowired

private EncryDecryService endecryService;

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

@Override

public CommonLoginResponse checkUserLogin(LoginRequest mslogin, HttpServletRequest http) {

CommonLoginResponse res = new CommonLoginResponse();

ClaimLoginResponse response = new ClaimLoginResponse();

try {

passwordEnc passEnc = new passwordEnc();

String epass = passEnc.crypt(mslogin.getPassword().trim());

log.info("Encrpted password "+epass);

List<ClaimLoginMaster> login =loginRepo.findByLoginIdAndPassword(mslogin.getUserId(),epass);

if (!CollectionUtils.isEmpty(login)) {

http.getSession().removeAttribute(mslogin.getUserId());

String token = jwtTokenUtil.doGenerateToken(mslogin.getUserId());

log.info("-----token------" + token);

SessionDetails session = new SessionDetails();

session.setLoginId(mslogin.getUserId());

session.setTokenId(token);

session.setCompanyId(mslogin.getCompanyId());

session.setStatus("ACTIVE");

String temptoken = bCryptPasswordEncoder.encode("CommercialClaim");

session.setTempTokenid(temptoken);

session.setEntryDate(new Date());

session =sessionRep.save(session);

response= setTokenResponse(session,login,mslogin);

res.setLoginResponse(response);

}

} catch (Exception e) {

e.printStackTrace();

}

return res;

}

private ClaimLoginResponse setTokenResponse(SessionDetails session, List<ClaimLoginMaster> login, LoginRequest mslogin) {

ClaimLoginResponse r = new ClaimLoginResponse();

try {

ClaimLoginMaster log =login.get(0);

r.setToken(session.getTempTokenid());

r.setLoginId(log.getLoginId());

r.setUserName(log.getUsername());

r.setEmail(StringUtils.isBlank(log.getUserMail())?"":log.getUserMail());

r.setMobileNo(StringUtils.isBlank(log.getMobileNumber())?"":log.getMobileNumber());

r.setCompanyId(mslogin.getCompanyId());

}catch (Exception e) {

e.printStackTrace();

}

return r;

}

@SuppressWarnings("static-access")

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

ClaimLoginMaster userList = new ClaimLoginMaster();

try {

log.info("loadUserByUsername==>" + username);

String[] split = username.split(":");

ClaimLoginMasterId id = new ClaimLoginMasterId();

id.setLoginId(split[0]);

id.setCompanyId(split[1]);

Optional<ClaimLoginMaster> userListopt = loginRepo.findById(id);

if(userListopt.isPresent()) {

userList = userListopt.get();

}

if (userList!=null) {

//user = userList.get(0);

String pass = bCryptPasswordEncoder.encode(endecryService.decrypt("zQYgCo7GMZeX1tBQyzAi8Q=="));

userList.setPassword(pass);

Set<GrantedAuthority> grantedAuthorities = new HashSet<GrantedAuthority>();

grantedAuthorities.add(new SimpleGrantedAuthority("ADMIN"));

log.info("loadUserByTokenEncrypt==>" + userList.getPassword());

}

} catch (Exception e) {

e.printStackTrace();

}

return new org.springframework.security.core.userdetails.User(userList.getLoginId(), userList.getPassword(),getAuthority());

}

private List<SimpleGrantedAuthority> getAuthority() {

return Arrays.asList(new SimpleGrantedAuthority("ROLE\_ADMIN"));

}

@Override

public CommonCrmRes LoginChangePassword(ChangePasswordReq req) {

CommonCrmRes res = new CommonCrmRes();

try {

passwordEnc passEnc = new passwordEnc();

String epass = passEnc.crypt(req.getOldpassword().trim());

String newpass = passEnc.crypt(req.getNewPassword().trim());

ClaimLoginMaster master = new ClaimLoginMaster();

log.info("EncryptPassword-->" + epass);

ClaimLoginMasterId id = new ClaimLoginMasterId();

id.setLoginId(req.getUserId());

id.setCompanyId(req.getCompanyId());

Optional<ClaimLoginMaster> model = loginRepo.findById(id);

if (model.isPresent()) {

master = model.get();

String pass1 = master.getPassword();

String pass2 = master.getLpass1();

String pass3 = master.getLpass2();

String pass4 = master.getLpass3();

String pass5 = master.getLpass4();

master.setLpass1(pass1);

master.setLpass2(pass2);

master.setLpass3(pass3);

master.setLpass4(pass4);

master.setLpass5(pass5);

master.setPassword(newpass);

master.setPwdCount(master.getPwdCount()+1);

Instant now = Instant.now();

Instant after = now.plus(Duration.ofDays(45));

Date dateAfter = Date.from(after);

master.setPassdate(dateAfter);

ClaimLoginMaster table = loginRepo.save(master);

if (table != null) {

res.setMessage("SUCCESS");

res.setIsError(false);

} else {

res.setMessage("FAILED");

res.setIsError(true);

}

}

} catch (Exception e) {

e.printStackTrace();

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

}

return res;

}

@Override

public CommonCrmRes logout(LogoutRequest mslogin) {

CommonCrmRes res = new CommonCrmRes();

ClaimLogoutResponse r = new ClaimLogoutResponse();

try {

ClaimLoginMasterId id = new ClaimLoginMasterId();

id.setLoginId(mslogin.getUserId());

id.setCompanyId(mslogin.getCompanyId());

Optional<ClaimLoginMaster> login = loginRepo.findById(id);

if (login.isPresent()) {

SessionDetails session = sessionRep.findByTempTokenid(mslogin.getToken());

session.setLogoutDate(new Date());

session.setStatus("DE-ACTIVE");

session = sessionRep.save(session);

r.setStatus("Log Out Sucessfully");

}else {

r.setStatus("Log Out Failed");

}

} catch (Exception e) {

r.setStatus("Log Out Failed");

e.printStackTrace();

}

res.setCommonResponse(r);

return res;

}

}

**Login Validation Serive Impl**

package com.maan.crm.auth.service.impl;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import java.util.Optional;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.apache.commons.lang3.StringUtils;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

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

import org.springframework.stereotype.Component;

import org.springframework.util.CollectionUtils;

import com.maan.crm.auth.dto.ChangePasswordReq;

import com.maan.crm.auth.dto.LoginRequest;

import com.maan.crm.auth.service.LoginValidatedService;

import com.maan.crm.auth.token.passwordEnc;

import com.maan.crm.bean.ClaimLoginMaster;

import com.maan.crm.bean.ClaimLoginMasterId;

import com.maan.crm.bean.SessionDetails;

import com.maan.crm.repository.ClaimLoginMasterRepository;

import com.maan.crm.repository.SessionDetailsRepository;

import com.maan.crm.service.CriteriaQueryService;

import com.maan.crm.util.error.Error;

@Component

public class LoginValidatedServiceImpl implements LoginValidatedService {

@Autowired

private CriteriaQueryService criteriaQuery;

@Autowired

private ClaimLoginMasterRepository loginRepo;

@Autowired

private SessionDetailsRepository sessionRep;

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

public List<Error> loginInputValidation(LoginRequest req) {

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

log.info(req);

try {

List<SessionDetails> sessionlist = new ArrayList<SessionDetails>();

List<ClaimLoginMaster> data = new ArrayList<ClaimLoginMaster>();

if (req.getUserId() == null || StringUtils.isBlank(req.getUserId())) {

list.add(new Error("", "UserId", "Please enter username"));

}else if (req.getCompanyId() == null || StringUtils.isBlank(req.getCompanyId())) {

list.add(new Error("", "CompanyId", "Please enter CompanyId"));

} else {

sessionlist = sessionRep.findByLoginIdAndCompanyIdOrderByEntryDateDesc(req.getUserId(),req.getCompanyId());

}

if (req.getPassword() == null || StringUtils.isBlank(req.getPassword())) {

list.add(new Error("", "Password", "Please enter password"));

}

if (req.getUserType() == null || StringUtils.isBlank(req.getUserType())) {

list.add(new Error("", "UserType", "Please enter UserType"));

}

if (StringUtils.isNotBlank(req.getUserId()) && StringUtils.isNotBlank(req.getPassword())) {

data = criteriaQuery.isvalidUser(req);

if (CollectionUtils.isEmpty(data)) {

list.add(new Error("", "User", "Please enter valid username/password"));

}else if(isExpired(data.get(0).getPassdate())) {

list.add(new Error("", "User", "Password Expired Please Change Your Password"));

}

}

if(req.getReLoginKey()!=null) {

if(req.getReLoginKey().equals("Y")) {

SessionDetails updatelogout = sessionlist.get(0);

updatelogout.setLogoutDate(new Date());

updatelogout.setStatus("DE-ACTIVE");

sessionRep.save(updatelogout);

}

}else if(sessionlist.size()!=0) {

if(sessionlist.get(0).getLogoutDate()==null) {

list.add(new Error("", "SessionError", "You already have an active logged in session on another device or window Do you want to start new session and terminate that session?"));

list.add(new Error("", "SessionError", "User :" + data.get(0).getUsername() + " : logged in at " +sessionlist.get(0).getEntryDate().toString()));

}

}

} catch (Exception e) {

e.printStackTrace();

}

return list;

}

private boolean isExpired(Date date) {

Date d = date;

Date d1 = new Date();

// not expired

if (d1.compareTo(d) < 0) {

return false;

}

// both date are same

else if (d.compareTo(d1) == 0) {

if (d.getTime() < d1.getTime()) {// not expired

return false;

} else if (d.getTime() == d1.getTime()) {// expired

return true;

} else {// expired

return true;

}

}

// expired

else {

return true;

}

}

public static String getToday(String format){

Date date = new Date();

return new SimpleDateFormat(format).format(date);

}

@Override

public List<Error> LoginChangePwdValidation(ChangePasswordReq req) {

log.info(req);

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

if (req.getUserId() == null || StringUtils.isBlank(req.getUserId())) {

list.add(new Error("", "UserId", "Please enter username"));

list.add(new Error("", "ChangePassword", "You are not authorized user..!"));

}else {

ClaimLoginMasterId id = new ClaimLoginMasterId();

id.setLoginId(req.getUserId());

id.setCompanyId(req.getCompanyId());

Optional<ClaimLoginMaster> model = loginRepo.findById(id);

if (model.isPresent()) {

String epass = "";

passwordEnc passEnc = new passwordEnc();

if (req.getNewPassword() == null || StringUtils.isBlank(req.getNewPassword())) {

list.add(new Error("", "New password", "Please enter New password"));

}else if (!validPassword(req.getNewPassword())) {

list.add(new Error("", "NewPassword", "Please enter the valid password"));

}

else {

epass = passEnc.crypt(req.getNewPassword().trim());

}

if (req.getOldpassword() == null || StringUtils.isBlank(req.getOldpassword())) {

list.add(new Error("", "Old password", "Please enter Oldpassword"));

} else if (model.get().getPassword().equals(epass)) {

list.add(new Error("", "ChangePassword", "Oldpassword and Newpassword should not match"));

} else if(model.get().getLpass1().equals(epass) || model.get().getLpass2().equals(epass) || model.get().getLpass3().equals(epass) || model.get().getLpass4().equals(epass) || model.get().getLpass5().equals(epass)) {

list.add(new Error("", "ChangePassword", "Newpassword should not be last 5 Password"));

}

} else {

list.add(new Error("", "ChangePassword", "You are not authorized user..!"));

}

}

return list;

}

private boolean validPassword(String newPassword) {

Pattern pattern=Pattern.compile("(?=\\S+$).{7,20}");

Matcher matcher = pattern.matcher(newPassword);

return matcher.matches();

}

}