Controller

package mex.walmart.archivo.central.controllers.process.authorization;  
  
import io.swagger.v3.oas.annotations.media.ArraySchema;  
import io.swagger.v3.oas.annotations.media.Content;  
import io.swagger.v3.oas.annotations.media.Schema;  
import io.swagger.v3.oas.annotations.responses.ApiResponse;  
import io.swagger.v3.oas.annotations.tags.Tag;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import mex.walmart.archivo.central.constants.SwaggerConstants;  
import mex.walmart.archivo.central.model.request.AuthorizationDeCajasRequest;  
import mex.walmart.archivo.central.model.request.CambioDeEstatusRequest;  
import mex.walmart.archivo.central.model.response.AuthorizationDeCajasResponse;  
import mex.walmart.archivo.central.model.response.AuthorizationDeCajasResponseWrapper;  
import mex.walmart.archivo.central.model.response.Response;  
import mex.walmart.archivo.central.model.response.UserResponse;  
import mex.walmart.archivo.central.service.process.authorization.AuthorizationDeCajasService;  
import org.springframework.http.MediaType;  
import org.springframework.validation.annotation.Validated;  
import org.springframework.web.bind.annotation.\*;  
  
import javax.validation.Valid;  
import java.util.List;  
  
@RestController  
@Tag(name = "Authorization De Cajas")  
@RequestMapping("/process/authorization/authorizationDeCajas")  
@RequiredArgsConstructor  
@Slf4j  
@Validated  
public class AuthorizationDeCajasController {  
 final AuthorizationDeCajasService authorizationDeCajasService;  
  
  
 */\*\*  
 \* This GET request fetch all Users from User table based on Roles.  
 \* For Admin & Supervisor, it fetches all Users and for other roles, it fetches logged in user.  
 \*  
 \** ***@return*** *Response*<*List* < *UserResponse*>*> list of UserResponse object  
 \*/* @GetMapping(value = "/users", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 @ApiResponse(responseCode = "200", description = SwaggerConstants.*DESCRIPTION\_200* , content = @Content(array = @ArraySchema(arraySchema = @Schema(implementation = UserResponse.class))))  
 public Response<List<UserResponse>> getUsers() {  
 Response<List<UserResponse>> response = new Response<>();  
 final List<UserResponse> userResponses = authorizationDeCajasService.getUsers();  
 *log*.info("{} returned", userResponses);  
 response.setData(userResponses);  
 return response;  
 }  
  
  
 */\*\*  
 \* This endpoint fetches all the boxes which are pending for authorization.  
 \** ***@param*** *userId userId  
 \** ***@return*** *List*<*AuthorizationDeCajasResponse*> *List<AuthorizationDeCajasResponse>  
 \*/* @GetMapping(value = "/search", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 @ApiResponse(responseCode = "200", description = SwaggerConstants.*DESCRIPTION\_200* , content = @Content(array = @ArraySchema(arraySchema = @Schema(implementation = AuthorizationDeCajasResponse.class))))  
 public Response<AuthorizationDeCajasResponseWrapper> getPendingBoxesForAuth(@RequestParam Integer userId ) {  
 Response<AuthorizationDeCajasResponseWrapper> response = new Response<>();  
 final AuthorizationDeCajasResponseWrapper authorizationDeCajasResponses = authorizationDeCajasService.getPendingBoxesForAuth(userId);  
 *log*.info("{} returned", authorizationDeCajasResponses);  
 response.setData(authorizationDeCajasResponses);  
 return response;  
 }  
  
 */\*\*  
 \* Updates status in Sheafs,boxes and Autobox table once Authorized  
 \*  
 \** ***@Request*** *AuthorizationDeCajasRequest  
 \*/* @PostMapping(value = "/accept", consumes = MediaType.*APPLICATION\_JSON\_VALUE*)  
 @ApiResponse(responseCode = "200", description = SwaggerConstants.*DESCRIPTION\_200*,  
 content = @Content(schema = @Schema()))  
 public Response<Void> accept(@RequestBody @Valid AuthorizationDeCajasRequest request, @RequestParam Integer autboxId) {  
 Response<Void> response = new Response<>();  
 authorizationDeCajasService.accept(request,autboxId);  
 response.setSuccess(true);  
 *log*.info("Successfully changed the status to 12 or 5");  
 return response;  
 }  
  
  
}

Service

package mex.walmart.archivo.central.service.process.authorization;  
  
import lombok.AccessLevel;  
import lombok.RequiredArgsConstructor;  
import lombok.experimental.FieldDefaults;  
import lombok.extern.slf4j.Slf4j;  
import mex.walmart.archivo.central.exception.ArchivoException;  
import mex.walmart.archivo.central.model.entity.AutBox;  
import mex.walmart.archivo.central.model.mapper.AuthorizationDeCajasMapper;  
import mex.walmart.archivo.central.model.request.AuthorizationDeCajasRequest;  
import mex.walmart.archivo.central.model.response.\*;  
import mex.walmart.archivo.central.repositorys.AutBoxRepository;  
import mex.walmart.archivo.central.repositorys.BoxesRepository;  
import mex.walmart.archivo.central.repositorys.SheafsRepository;  
import mex.walmart.archivo.central.repositorys.UsersRepository;  
import mex.walmart.archivo.central.utility.CommonMethods;  
import org.springframework.stereotype.Service;  
import org.springframework.util.CollectionUtils;  
  
import javax.transaction.Transactional;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Map;  
import java.util.Optional;  
import java.util.stream.Collectors;  
  
@Service  
@Slf4j  
@FieldDefaults(level = AccessLevel.*PRIVATE*)  
@RequiredArgsConstructor  
public class AuthorizationDeCajasService {  
 final AuthorizationDeCajasMapper authorizationDeCajasMapper;  
 final UsersRepository usersRepository;  
 final AutBoxRepository autBoxRepository;  
 final BoxesRepository boxesRepository;  
 final SheafsRepository sheafsRepository;  
  
 */\*\*  
 \* For Admin & Supervisor, it fetches all Users and for other roles, it fetches logged in user.  
 \** ***@return*** *List*<*Userresponse*> *</Userresponse>  
 \*/* public List<UserResponse> getUsers() {  
 var rolesFromToken = CommonMethods.*getRolesFromToken*();  
 var byUserLogin = usersRepository.findByUserLogin(CommonMethods.*getUpnId*());  
 //*TODO: Implement authorization code once all roles are available* //Fetch the list of all users for Admin and Supervisor. For other roles, fetch only the User  
 List<UsersResponseProjection> objects = (rolesFromToken.contains("admin") || rolesFromToken.contains("supervisor")) ?  
 usersRepository.getUsersPendingForAuthorization() : usersRepository.getUsersPendingForAuthorizationForUser(byUserLogin.getUserId());  
 if (CollectionUtils.*isEmpty*(objects))  
 throw new ArchivoException(ArchivoException.Code.*DATA\_NOT\_FOUND*);  
 return authorizationDeCajasMapper.mapUserDaoToUserResponse(objects);  
 }  
  
 */\*\*  
 \* Fetch all Boxes which are pending for Authorization (Status 4 or 5 or 12)  
 \** ***@param*** *userId userId  
 \** ***@return*** *List*<*AuthorizationDeCajasResponse*> *List<AuthorizationDeCajasResponse>  
 \*/* public AuthorizationDeCajasResponseWrapper getPendingBoxesForAuth(Integer userId){  
 //This API returns a list of list of AutboxId, cajas and status.  
 AuthorizationDeCajasResponseWrapper authorizationDeCajasResponseWrapper = new AuthorizationDeCajasResponseWrapper();  
 List<AuthorizationDeCajasResponse> authorizationDeCajasResponses = new ArrayList<>();  
 List<AutBox> autBox = autBoxRepository.getAutBox(userId);  
  
 //Fetch all boxes for the User based on Autboxid and group the boxes for each autbox  
 List<BoxesResponseProjection> boxYearBAndBoxFuc = boxesRepository.getBoxesForAuthrorization(autBox.stream().map(AutBox::getAutBoxId).collect(Collectors.*toList*()));  
 if (CollectionUtils.*isEmpty*(boxYearBAndBoxFuc))  
 throw new ArchivoException(ArchivoException.Code.*DATA\_NOT\_FOUND*);  
 Map<Integer, List<BoxesResponseProjection>> collect = boxYearBAndBoxFuc.stream().collect(Collectors.*groupingBy*(BoxesResponseProjection::getAutboxId));  
 collect.forEach((integer, boxesResponseProjections) -> {  
 List<AuthorizationDeCajasBoxesResponse> authorizationDeCajasBoxesResponses = authorizationDeCajasMapper.mapAutBoxDaoToAuthCajaResponse(boxesResponseProjections);  
 AuthorizationDeCajasResponse authorizationDeCajasResponse = new AuthorizationDeCajasResponse();  
 authorizationDeCajasResponse.setAutboxId(integer);  
 authorizationDeCajasResponse.setAuthorizationDeCajasBoxesRespnseList(authorizationDeCajasBoxesResponses);  
 authorizationDeCajasResponses.add(authorizationDeCajasResponse);  
 });  
  
  
 authorizationDeCajasResponseWrapper.setAuthorizationDeCajasResponses(authorizationDeCajasResponses);  
 authorizationDeCajasResponseWrapper.setCount(autBox.size());  
 return authorizationDeCajasResponseWrapper;  
 }  
  
 */\*\*  
 \* This Post endpoint updates the status to 5(Authorized) or 12 (Not Authorized) based on flag  
 \** ***@param*** *request request  
 \** ***@param*** *autboxId autboxId  
 \*/* @Transactional  
 public void accept(AuthorizationDeCajasRequest request,Integer autboxId) {  
 request.getCashRegister().forEach(s -> {  
 //Flag checked and save will change the status from pending(4) to Authorized(5)  
 if(s.isFlag()) {  
 var byBoxFucAndBoxYear = sheafsRepository.findByBoxFucAndBoxYear(s.getBoxFUC(), s.getBoxYear());  
 Optional.*of*(byBoxFucAndBoxYear).orElseThrow(() -> new ArchivoException(ArchivoException.Code.*DATA\_NOT\_FOUND*));  
 sheafsRepository.updateStatusWithBoxFucAndyear(s.getBoxFUC(),s.getBoxYear(), 5);  
  
 var box = boxesRepository.getBox(s.getBoxFUC(), s.getBoxYear());  
 Optional.*of*(box).orElseThrow(() -> new ArchivoException(ArchivoException.Code.*DATA\_NOT\_FOUND*));  
 boxesRepository.updateStatus( s.getBoxFUC(),s.getBoxYear(),5);  
  
 var count = autBoxRepository.findById(autboxId);  
 Optional.*of*(count).orElseThrow(() -> new ArchivoException(ArchivoException.Code.*DATA\_NOT\_FOUND*));  
 autBoxRepository.updateStatus(5,autboxId);  
  
 }else{  
 //Without checking the flag and save will change the status from pending(4) to NoAuthorized(12)  
 var byBoxFucAndBoxYear = sheafsRepository.findByBoxFucAndBoxYear(s.getBoxFUC(), s.getBoxYear());  
 Optional.*of*(byBoxFucAndBoxYear).orElseThrow(() -> new ArchivoException(ArchivoException.Code.*DATA\_NOT\_FOUND*));  
 sheafsRepository.updateStatusWithBoxFucAndyear(s.getBoxFUC(), s.getBoxYear(), 12);  
  
 var box = boxesRepository.getBox(s.getBoxFUC(), s.getBoxYear());  
 Optional.*of*(box).orElseThrow(() -> new ArchivoException(ArchivoException.Code.*DATA\_NOT\_FOUND*));  
 boxesRepository.updateStatus( s.getBoxFUC(),s.getBoxYear(),12);  
  
 var count = autBoxRepository.findById(autboxId);  
 Optional.*of*(count).orElseThrow(() -> new ArchivoException(ArchivoException.Code.*DATA\_NOT\_FOUND*));  
 autBoxRepository.updateStatus(12,autboxId);  
 }  
 });  
 }  
}

DAO

package mex.walmart.archivo.central.repositorys;  
  
import mex.walmart.archivo.central.model.entity.AutBox;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import org.springframework.stereotype.Repository;  
  
import javax.transaction.Transactional;  
import java.util.List;  
  
@Repository  
public interface AutBoxRepository extends JpaRepository<AutBox, Integer> {  
 @Transactional  
 @Modifying  
 @Query(value = "update AUTBOX set STA\_IDSTA=:statusId " +  
 " where AUTB\_IDAUTB=:autBoxId",nativeQuery = true)  
 void updateStatus(@Param("statusId") int statusId, @Param("autBoxId") int autBoxId);  
  
 @Transactional  
 @Modifying  
 @Query(value = "update AutBox set Sta\_IdSta=15 where AutB\_IdAutB=:autBoxId",nativeQuery = true)  
 int updateStatusId(@Param("autBoxId") Integer autBoxId);  
  
 @Query(value="Select \* from AutBox a where a.Usr\_IdUsr=:userId And " +  
 "(a.Sta\_IdSta=4 or a.Sta\_IdSta=5 or a.Sta\_IdSta=12) Order By a.AutB\_IdAutB ",nativeQuery = true)  
 List<AutBox> getAutBox(@Param("userId") Integer userId);  
}

package mex.walmart.archivo.central.repositorys;  
  
import mex.walmart.archivo.central.model.entity.Boxes;  
import mex.walmart.archivo.central.model.entity.EmbededBoxesId;  
import mex.walmart.archivo.central.model.response.BoxesResponseProjection;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.CrudRepository;  
import org.springframework.data.repository.query.Param;  
import org.springframework.stereotype.Repository;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.time.LocalDate;  
import java.util.List;  
  
@Repository  
public interface BoxesRepository extends CrudRepository<Boxes, EmbededBoxesId> {  
 @Transactional  
 @Modifying  
 @Query(value = "update Boxes set STA\_IDSTA=:statusId " +  
 " where BOX\_FUC=:boxFuc and BOX\_YEARB=:boxYear", nativeQuery = true)  
 void updateStatus(@Param("boxFuc") int boxFuc, @Param("boxYear") int boxYear, @Param("statusId") Integer statusId);  
  
 @Query(value="SELECT MAX(BOX\_FUC) FROM BOXES WHERE BOX\_YEARB=:boxYearB",nativeQuery = true)  
 Integer getLatestBoxFucId(@Param("boxYearB") Integer boxYearB);  
  
 @Transactional  
 @Modifying  
 @Query(value = "UPDATE BOXES SET BOXTOTALSHEAFS=:boxTotalSheafs WHERE BOX\_YEARB=:boxyearB AND BOX\_FUC:boxFuc", nativeQuery = true)  
 void updateBoxesTotalSheafs(@Param("boxTotalSheafs") Integer boxTotalSheafs,  
 @Param("boxyearB") Integer boxYearB,  
 @Param("boxFuc") Integer boxFuc);  
  
 @Transactional  
 @Modifying  
 @Query(value = "update Boxes set STA\_IDSTA= :statusId ,AUTB\_IDAUTB= :autBoxId" +  
 " where BOX\_FUC=:boxFuc and BOX\_YEARB=:boxYear", nativeQuery = true)  
 void updateStatuAndAutBox(@Param("boxFuc") int boxFuc, @Param("boxYear") int boxYear, @Param("statusId") Integer statusId, @Param("autBoxId") Integer autBoxId);  
  
 @Query(value = "Select \* From Boxes Where Box\_YearB=:boxYear and Box\_FUC= :boxFuc" +  
 " and AutB\_IdAutB Is Not Null", nativeQuery = true)  
 Boxes getBox(@Param("boxFuc") int boxFuc, @Param("boxYear") int boxYear);  
  
 @Query(value = "Select Count(Distinct Sta\_IdSta) From Boxes Where AutB\_IdAutB=:autBoxId", nativeQuery = true)  
 Integer getCountOfStatusId(@Param("autBoxId") Integer autBoxId);  
  
 @Transactional  
 @Modifying  
 @Query(value = "update Boxes set Sta\_IdSta=15 where AutB\_IdAutB=:autBoxId", nativeQuery = true)  
 int updateStatusId(@Param("autBoxId") Integer autBoxId);  
  
 @Query(value = "Select \* From Boxes Where AutB\_IdAutB=:autBoxId", nativeQuery = true)  
 List<Boxes> getBoxYearBAndBoxFuc(@Param("autBoxId") Integer autBoxId);  
  
 @Query(value = "Select \* From Boxes Where AutB\_IdAutB IN :autBoxId", nativeQuery = true)  
 List<Boxes> getListOfBoxYearBAndBoxFuc(@Param("autBoxId") List<Integer> autBoxId);  
  
 @Query(value = "Select BoxTotalSheafs From Boxes Where Box\_YearB=:boxYearB And Box\_FUC=:boxFuc", nativeQuery = true)  
 Integer getTotalBoxSheafs(@Param("boxYearB") Integer boxYearB, @Param("boxFuc") Integer boxFuc);  
  
 @Query(value = "select CONCAT( boxes.box\_yearb , '-',REPLICATE('0',7 - LEN(boxes.box\_fuc)) ,boxes.box\_fuc) as cashRegister, " +  
 "boxes.Sta\_IdSta as boxStatus,autbox.AutB\_IdAutB as autboxId, " +  
 "autbox.sta\_idsta as statusId,RTRIM(s.sta\_desc) as statusDesc " +  
 "from AutBox autbox,Boxes boxes,Status s \n" +  
 "where (boxes.Sta\_IdSta=4 or boxes.Sta\_IdSta=5 or boxes.Sta\_IdSta=12) and \n" +  
 "boxes.AutB\_IdAutB in (:autBoxId) \n" +  
 " and autbox.sta\_idsta=s.sta\_idsta \n" +  
 " and autbox.AutB\_IdAutB=boxes.AutB\_IdAutB \n" +  
 "Order By boxes.Box\_YearB, boxes.Box\_FUC",nativeQuery = true)  
 List<BoxesResponseProjection> getBoxesForAuthrorization(@Param("autBoxId") List<Integer> autBoxId);  
  
 List<Boxes> findByEmbededBoxesIdInAndStatusStatusIdNot(List<EmbededBoxesId> ids, Integer statusId);  
  
 @Transactional  
 @Modifying  
 void deleteByEmbededBoxesIdIn(List<EmbededBoxesId> ids);  
  
 */\*\*  
 \* Update Boxes table to set 'status' by 'boxFuc' and 'boxYear' input param  
 \*  
 \** ***@param*** *boxFuc Integer type  
 \** ***@param*** *boxYearB Integer type  
 \*/* @Transactional  
 @Modifying  
 @Query(value = "update Boxes set sta\_idsta=:status where Box\_Fuc=:boxFuc and Box\_YearB=:boxYearB", nativeQuery = true)  
 void updateBoxes(@Param("status") Integer status, @Param("boxFuc") Integer boxFuc, @Param("boxYearB") Integer boxYearB);  
  
 */\*\*  
 \* Get total count of 'boxes' records filtered by 'boxyear' & 'boxFuc'  
 \*  
 \** ***@param*** *boxFuc Integer type  
 \** ***@param*** *boxYear Integer type  
 \** ***@return*** *Integer count  
 \*/* @Query(value = "select count(\*) from boxes where box\_yearb=:boxyear and box\_fuc=:boxFuc", nativeQuery = true)  
 Integer getCount(@Param("boxyear") Integer boxYear, @Param("boxFuc") Integer boxFuc);  
  
 */\*\*  
 \* Update Boxes table to set 'status' by 'boxFuc' and 'boxYear' input param  
 \*  
 \** ***@param*** *boxFuc Integer type  
 \** ***@param*** *boxYearB Integer type  
 \*/* @Transactional  
 @Modifying  
 @Query(value = "update Boxes set sta\_idsta=:status ,box\_whsdate=:boxWhsDate where Box\_Fuc=:boxFuc and Box\_YearB=:boxYearB", nativeQuery = true)  
 void updateBoxesForArchvoEntrada(@Param("status") Integer status,  
 @Param("boxWhsDate") LocalDate boxWhsDate,  
 @Param("boxFuc") Integer boxFuc,  
 @Param("boxYearB") Integer boxYearB);  
  
 @Query(value = "Select \* From Boxes Where AutB\_IdAutB = :autBoxId", nativeQuery = true)  
 List<Boxes> getListOfBoxesByAutId(@Param("autBoxId") Integer autBoxId);  
  
 @Query(value = "SELECT \* FROM BOXES WHERE DATEADD(DAY, :days,BOX\_DATEDESTRUC) <= GETDATE()",nativeQuery = true)  
 List<Boxes> getDestructionBoxes(@Param("days") Integer days);  
  
 @Query(value = "select isNull(max(box\_fuc),0) from boxes where box\_yearb=:currentYear", nativeQuery = true)  
 Integer getMaxBoxFuc(@Param("currentYear")int year);  
}

Mapper

package mex.walmart.archivo.central.model.mapper;  
  
import mex.walmart.archivo.central.model.entity.AutBox;  
import mex.walmart.archivo.central.model.response.\*;  
import org.apache.commons.lang3.StringUtils;  
import org.mapstruct.Mapper;  
import org.mapstruct.Mapping;  
import org.springframework.stereotype.Component;  
  
import java.util.List;  
  
@Component  
@Mapper(componentModel = "spring",imports = StringUtils.class)  
public interface AuthorizationDeCajasMapper {  
  
 List<UserResponse> mapUserDaoToUserResponse(List<UsersResponseProjection> usersList);  
  
 AuthorizationDeCajasBoxesResponse mapAutBoxDaoToAuthCajaResponse(BoxesResponseProjection objects);  
 List<AuthorizationDeCajasBoxesResponse> mapAutBoxDaoToAuthCajaResponse(List<BoxesResponseProjection> objects);  
}

Response

package mex.walmart.archivo.central.model.response;  
  
  
import com.fasterxml.jackson.annotation.JsonInclude;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.util.List;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@JsonInclude(JsonInclude.Include.*NON\_NULL*)  
public class AuthorizationDeCajasBoxesResponse {  
 String cashRegister;  
 Integer boxStatus;  
 Integer statusId;  
 String statusDesc;  
}

package mex.walmart.archivo.central.model.response;  
  
  
import com.fasterxml.jackson.annotation.JsonInclude;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.util.List;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@JsonInclude(JsonInclude.Include.*NON\_NULL*)  
public class AuthorizationDeCajasResponseWrapper {  
 Integer count;  
 List<AuthorizationDeCajasResponse> authorizationDeCajasResponses;  
}

package mex.walmart.archivo.central.model.response;  
  
  
import com.fasterxml.jackson.annotation.JsonInclude;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.util.List;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@JsonInclude(JsonInclude.Include.*NON\_NULL*)  
public class AuthorizationDeCajasResponse {  
 Integer autboxId;  
 List<AuthorizationDeCajasBoxesResponse> authorizationDeCajasBoxesRespnseList;  
}

Projection

package mex.walmart.archivo.central.model.response;  
  
public interface BoxesResponseProjection {  
 String getCashRegister();  
  
 Integer getBoxStatus();  
  
 Integer getAutboxId();  
  
 Integer getStatusId();  
  
 String getStatusDesc();  
}

Testcase

package mex.walmart.archivo.central.controllers;  
  
import mex.walmart.archivo.central.controllers.process.authorization.AuthorizationDeCajasController;  
import mex.walmart.archivo.central.model.request.ArchivoRequest;  
import mex.walmart.archivo.central.model.request.AuthorizationDeCajasRequest;  
import mex.walmart.archivo.central.model.request.EmbeddedCajasRequest;  
import mex.walmart.archivo.central.model.response.\*;  
import mex.walmart.archivo.central.service.process.authorization.AuthorizationDeCajasService;  
import org.junit.Test;  
import org.junit.jupiter.api.Assertions;  
import org.junit.runner.RunWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.Mockito;  
import org.springframework.test.context.junit4.SpringRunner;  
  
import java.util.Collections;  
import java.util.List;  
import java.util.function.Supplier;  
  
@RunWith(SpringRunner.class)  
public class AuthorizationDeCajasControllerTest {  
  
 @InjectMocks  
 AuthorizationDeCajasController authorizationDeCajasController;  
  
 @Mock  
 AuthorizationDeCajasService authorizationDeCajasService;  
  
 Supplier<List<UserResponse>> userResponse = () -> Collections.*singletonList*(new UserResponse(1, "desc"));  
 Supplier<AuthorizationDeCajasRequest> cajasRequestSupplier = () -> new AuthorizationDeCajasRequest(Collections.*singletonList*(new EmbeddedCajasRequest(1,1,true)));  
 Supplier<AuthorizationDeCajasResponseWrapper> authResponse = () -> new AuthorizationDeCajasResponseWrapper(2,Collections.*singletonList*(new AuthorizationDeCajasResponse(1,Collections.*singletonList*(new AuthorizationDeCajasBoxesResponse("2022-000129",1,2,"name")))));  
  
 @Test  
 public void testGetUsers() {  
 Mockito.*when*(authorizationDeCajasService.getUsers()).thenReturn(userResponse.get());  
 var response = authorizationDeCajasController.getUsers();  
 Assertions.*assertNotNull*(response);  
 Assertions.*assertTrue*(response.isSuccess());  
 }  
  
  
 @Test  
 public void testGetPendingBoxes() {  
 Mockito.*when*(authorizationDeCajasService.getPendingBoxesForAuth(Mockito.*anyInt*())).thenReturn(authResponse.get());  
 var response = authorizationDeCajasController.getPendingBoxesForAuth(Mockito.*anyInt*());  
 Assertions.*assertNotNull*(response);  
 Assertions.*assertTrue*(response.isSuccess());  
 }  
  
 @Test  
 public void testPostAccept(){  
 Mockito.*doNothing*().when(authorizationDeCajasService).accept(cajasRequestSupplier.get(),12);  
 var response = authorizationDeCajasController.accept(Mockito.*any*(), Mockito.*anyInt*());  
 Assertions.*assertNotNull*(response);  
 Assertions.*assertTrue*(response.isSuccess());  
 }  
  
}

service

package mex.walmart.archivo.central.service;  
  
  
import mex.walmart.archivo.central.exception.ArchivoException;  
import mex.walmart.archivo.central.model.entity.\*;  
import mex.walmart.archivo.central.model.mapper.AuthorizationDeCajasMapper;  
import mex.walmart.archivo.central.model.request.AuthorizationDeCajasRequest;  
import mex.walmart.archivo.central.model.request.EmbeddedCajasRequest;  
import mex.walmart.archivo.central.model.response.\*;  
import mex.walmart.archivo.central.repositorys.AutBoxRepository;  
import mex.walmart.archivo.central.repositorys.BoxesRepository;  
import mex.walmart.archivo.central.repositorys.SheafsRepository;  
import mex.walmart.archivo.central.repositorys.UsersRepository;  
import mex.walmart.archivo.central.service.process.authorization.AuthorizationDeCajasService;  
import mex.walmart.archivo.central.utility.CommonMethods;  
import org.junit.Before;  
import org.junit.Ignore;  
import org.junit.Test;  
import org.junit.jupiter.api.Assertions;  
import org.junit.runner.RunWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.Mockito;  
import org.powermock.api.mockito.PowerMockito;  
import org.powermock.core.classloader.annotations.PowerMockIgnore;  
import org.powermock.core.classloader.annotations.PrepareForTest;  
import org.powermock.modules.junit4.PowerMockRunner;  
  
import java.util.Collections;  
import java.util.List;  
import java.util.Map;  
import java.util.Optional;  
import java.util.function.Supplier;  
  
@RunWith(PowerMockRunner.class)  
@PrepareForTest({CommonMethods.class})  
@PowerMockIgnore({"javax.management.\*", "com.sun.org.apache.xerces.jaxp.\*", "javax.xml.parsers.\*", "com.sun.org.apache.xerces.\*", "org.xml.\*"})  
public class AuthorizationDeCajasServiceTest {  
  
 @InjectMocks  
 AuthorizationDeCajasService authorizationDeCajasService;  
  
 @Mock  
 AuthorizationDeCajasMapper authorizationDeCajasMapper;  
  
 @Mock  
 BoxesRepository boxesRepository;  
  
 @Mock  
 SheafsRepository sheafsRepository;  
  
 @Mock  
 AutBoxRepository autBoxRepository;  
  
 @Mock  
 UsersRepository usersRepository;  
  
 UsersResponseProjection usersResponseProjection;  
 BoxesResponseProjection boxesResponseProjection;  
  
 @Before  
 public void setup() {  
 List<String> roles = List.*of*("admin");  
 PowerMockito.*mockStatic*(CommonMethods.class);  
 PowerMockito.*when*(CommonMethods.*getUserName*()).thenReturn("SystemAdmin");  
  
 PowerMockito.*when*(CommonMethods.*getRolesFromToken*()).thenReturn(roles);  
 PowerMockito.*when*(CommonMethods.*getRoleNameFromToken*(Mockito.*any*())).thenReturn("SystemAdmin");  
  
 usersResponseProjection = new UsersResponseProjection() {  
 @Override  
 public Integer getUserId() {  
 return 1;  
 }  
  
 @Override  
 public String getUserName() {  
 return "Hello";  
 }  
 };  
  
  
 }  
  
 Supplier<List<UserResponse>> userResponse = () -> Collections.*singletonList*(new UserResponse(1, "desc"));  
 Supplier<AuthorizationDeCajasResponseWrapper> authResponse = () -> new AuthorizationDeCajasResponseWrapper(2, Collections.*singletonList*(new AuthorizationDeCajasResponse(1, Collections.*singletonList*(new AuthorizationDeCajasBoxesResponse("2022-000129",1, 2, "name")))));  
 Supplier<AuthorizationDeCajasRequest> cajasRequestSupplier = () -> new AuthorizationDeCajasRequest(Collections.*singletonList*(new EmbeddedCajasRequest(1, 1, true)));  
 Supplier<List<AutBox>> autbox = () -> Collections.*singletonList*(new AutBox(1, new Status(), new Users()));  
  
 Supplier<List<Sheafs>> listSupplierSheafs = () -> Collections.*singletonList*(new Sheafs());  
  
 @Test  
 public void testGetUsers() {  
  
 Mockito.*when*(usersRepository.getUsersPendingForAuthorization()).thenReturn(Collections.*singletonList*(usersResponseProjection));  
 Mockito.*when*(authorizationDeCajasMapper.mapUserDaoToUserResponse(Mockito.*anyList*())).thenReturn(userResponse.get());  
 List<UserResponse> response = authorizationDeCajasService.getUsers();  
 Mockito.*verify*(usersRepository, Mockito.*atLeastOnce*()).getUsersPendingForAuthorization();  
 Mockito.*verify*(authorizationDeCajasMapper, Mockito.*atLeastOnce*()).mapUserDaoToUserResponse(Mockito.*anyList*());  
 Assertions.*assertNotNull*(response);  
 Assertions.*assertTrue*(response.size() > 0);  
 }  
  
 @Test(expected = ArchivoException.class)  
 public void testGetUsersUserId() {  
  
 Mockito.*when*(usersRepository.getUsersPendingForAuthorizationForUser(Mockito.*anyInt*())).thenReturn(Collections.*singletonList*(usersResponseProjection));  
 Mockito.*when*(authorizationDeCajasMapper.mapUserDaoToUserResponse(Mockito.*anyList*())).thenReturn(userResponse.get());  
 List<UserResponse> response = authorizationDeCajasService.getUsers();  
 Mockito.*verify*(usersRepository, Mockito.*atLeastOnce*()).getUsersPendingForAuthorizationForUser(Mockito.*anyInt*());  
 Mockito.*verify*(authorizationDeCajasMapper, Mockito.*atLeastOnce*()).mapUserDaoToUserResponse(Mockito.*anyList*());  
 Assertions.*assertNotNull*(response);  
 Assertions.*assertTrue*(response.size() > 0);  
 }  
  
 @Test  
 public void testGetPendingRecords() {  
  
 Supplier<Map<Integer, List<BoxesResponseProjection>>> collect = () -> Collections.*singletonMap*(1,Collections.*singletonList*(new BoxesResponseProjection() {  
 @Override  
 public String getCashRegister() {  
 return "abc";  
 }  
  
 @Override  
 public Integer getBoxStatus() {  
 return 1;  
 }  
  
 @Override  
 public Integer getAutboxId() {  
 return 1;  
 }  
  
 @Override  
 public Integer getStatusId() {  
 return null;  
 }  
  
 @Override  
 public String getStatusDesc() {  
 return "acd";  
 }  
 }));  
  
 boxesResponseProjection = new BoxesResponseProjection() {  
 @Override  
 public String getCashRegister() {  
 return "abcd";  
 }  
  
 @Override  
 public Integer getBoxStatus() {  
 return 1;  
 }  
  
 @Override  
 public Integer getAutboxId() {  
 return 2;  
 }  
  
 @Override  
 public Integer getStatusId() {  
 return 3;  
 }  
  
 @Override  
 public String getStatusDesc() {  
 return "abcd";  
 }  
 };  
 Mockito.*when*(autBoxRepository.getAutBox(Mockito.*anyInt*())).thenReturn(autbox.get());  
 Mockito.*when*(boxesRepository.getBoxesForAuthrorization(Mockito.*anyList*())).thenReturn(Collections.*singletonList*(boxesResponseProjection));  
 Mockito.*when*(authorizationDeCajasMapper.mapAutBoxDaoToAuthCajaResponse(Mockito.*anyList*())).thenReturn(Collections.*singletonList*(new AuthorizationDeCajasBoxesResponse("abcd",1,1,"abcd")));  
 Mockito.*when*(boxesRepository.getBoxesForAuthrorization(Mockito.*anyList*())).thenReturn(collect.get().get(0));  
 AuthorizationDeCajasResponseWrapper response = authorizationDeCajasService.getPendingBoxesForAuth(Mockito.*anyInt*());  
 Mockito.*verify*(autBoxRepository, Mockito.*atLeastOnce*()).getAutBox(Mockito.*anyInt*());  
 Mockito.*verify*(boxesRepository, Mockito.*atLeastOnce*()).getBoxesForAuthrorization(Mockito.*anyList*());  
 Mockito.*verify*(authorizationDeCajasMapper, Mockito.*atLeastOnce*()).mapAutBoxDaoToAuthCajaResponse(Mockito.*anyList*());  
 Assertions.*assertNotNull*(response);  
 // Assertions.assertTrue(response != null);  
 }  
  
  
 @Test  
 public void testAccept() {  
 Mockito.*when*(sheafsRepository.findByBoxFucAndBoxYear(Mockito.*anyInt*(), Mockito.*anyInt*())).thenReturn(listSupplierSheafs.get());  
 Mockito.*when*(boxesRepository.getBox(Mockito.*anyInt*(), Mockito.*anyInt*())).thenReturn(new Boxes());  
 Mockito.*when*(autBoxRepository.findById(Mockito.*anyInt*())).thenReturn(Optional.*of*(new AutBox(1, new Status(), new Users())));  
  
 Mockito.*doNothing*().when(sheafsRepository).updateStatusWithBoxFucAndyear(Mockito.*anyInt*(), Mockito.*anyInt*(), Mockito.*anyInt*());  
 Mockito.*doNothing*().when(boxesRepository).updateStatus(Mockito.*anyInt*(), Mockito.*anyInt*(), Mockito.*anyInt*());  
 Mockito.*doNothing*().when(autBoxRepository).updateStatus(Mockito.*anyInt*(), Mockito.*anyInt*());  
  
 }  
  
}