```
package com.cybercenter.core.domain.membership.service;

import com.cybercenter.core.domain.membership.constant.RequestStatus;
import com.cybercenter.core.domain.user.entity.User;
import com.cybercenter.core.infrastructure.dto.MessageDTO;
import com.cybercenter.core.infrastructure.dto.MessageTemplate;
import com.cybercenter.core.infrastructure.dto.MessageType;
import com.cybercenter.core.infrastructure.wrapper.MessageSenderWrapper;
import com.cybercenter.core.shared.model.StaticMessages;
import com.cybercenter.core.domain.membership.dto.ActMembershipRequestDTO;
import com.cybercenter.core.domain.membership.dto.MembershipRequestDTO;
import com.cybercenter.core.domain.membership.dto.MembershipResponseDTO;
import com.cybercenter.core.domain.membership.dto.MembershipSearchDTO;
import com.cybercenter.core.domain.membership.entity.MembershipRequest;
import com.cybercenter.core.infrastructure.exception.HaveOpenRequestException;
import com.cybercenter.core.infrastructure.exception.NotAllowedAccessException;
import com.cybercenter.core.infrastructure.exception.NotFoundException;
import com.cybercenter.core.domain.membership.mapper.MembershipRequestMapper;
import com.cybercenter.core.domain.membership.repository.MembershipRequestRepository;
import com.cybercenter.core.domain.membership.spe.MembershipRequestSpecification;
import com.cybercenter.core.domain.user.service.UserService;
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Map;
import java.util.Optional;

@Service
@RequiredArgsConstructor
public class MembershipRequestService {

    private final MembershipRequestRepository membershipRequestRepository;
    private final MembershipRequestMapper membershipRequestMapper;
    private final UserService userService;
    private final MessageSenderWrapper messageSenderWrapper;

    public MembershipRequest save(MembershipRequestDTO membershipRequestDTO) {
```

```
        Optional<MembershipRequest> openRequest =
membershipRequestRepository.findOpenRequest(membershipRequestDTO.getUserId(),
membershipRequestDTO.getScopeId());
        if (openRequest.isPresent()) {
            throw new HaveOpenRequestException();
        }
        return
membershipRequestRepository.save(membershipRequestMapper.toEntity(membershipRequest
DTO));
    }

    public MembershipResponseDTO getUserRequest(long userId, long id) {
        MembershipRequest membershipRequest = membershipRequestRepository.findById(id)
                .orElseThrow(() -> new
NotFoundException(StaticMessages.NOT_FOUND_MEMBERSHIP_REQUEST));
        if (membershipRequest.getUser().getId() != userId) {
            throw new
NotFoundException(StaticMessages.NOT_FOUND_MEMBERSHIP_REQUEST);
        }
        return membershipRequestMapper.toDTO(membershipRequest);
    }

    public List<MembershipResponseDTO> getUserRequests(long userId,
MembershipSearchDTO searchDTO) {
        searchDTO.setUserId(userId);
        Specification<MembershipRequest> specification = new
MembershipRequestSpecification(searchDTO);
        return
membershipRequestRepository.findAll(specification).stream().map(membershipRequestMapper:
:toDTO).toList();
    }

    public MembershipResponseDTO getRequest(long id, Integer scopeId) {
        MembershipRequest membershipRequest =
membershipRequestRepository.findById(id).orElseThrow(NotFoundException::new);
        if (scopeId != null && membershipRequest.getScope().getId() != scopeId) {
            throw new NotAllowedAccessException();
        }
        return membershipRequestMapper.toDTOWithUserInfo(membershipRequest);
    }

    public Page<MembershipResponseDTO> getRequests(MembershipSearchDTO searchDTO,
Pageable pageable) {
```

```
        Specification<MembershipRequest> specification = new
MembershipRequestSpecification(searchDTO);
        return membershipRequestRepository.findAll(specification,
pageable).map(membershipRequestMapper::toDTOWithUserInfo);
    }

    @Transactional
    public void resolve(ActMembershipRequestDTO dto, Integer scopeId) {
        MembershipRequest membershipRequest =
membershipRequestRepository.findById(dto.getId()).orElseThrow(() -> new
NotFoundException(StaticMessages.NOT_FOUND_MEMBERSHIP_REQUEST));
        if (scopeId != null && membershipRequest.getScope().getId() != scopeId) {
            throw new NotAllowedAccessException();
        }
        if (membershipRequest.getStatus() != RequestStatus.PENDING ) {
            throw new NotAllowedAccessException(StaticMessages.INVALID_ACCEPT_STATE);
        }
        membershipRequest.setStatus(dto.isAccepted() ? RequestStatus.ACCEPTED :
RequestStatus.REJECTED);
        membershipRequest.setReason(dto.getReason());
        membershipRequestRepository.save(membershipRequest);
        User user = membershipRequest.getUser();
        userService.addDefaultAuthority(user, scopeId);
        if (dto.isAccepted()) {
            messageSenderWrapper.sendMessage(MessageDTO.builder()
                    .to(user.getPhoneNumber())
                    .messageType(MessageType.SMS.getId())
                    .template(MessageTemplate.MEMBERSHIP.ACCEPT_REQUEST)
                    .tokens(Map.of("scope",
String.valueOf(membershipRequest.getScope().getTitle())))
                    .build());
        }
    }
}
```