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

import com.cybercenter.core.domain.auth.dto.ChangePasswordDTO;
import com.cybercenter.core.domain.auth.dto.VerifyCodeLoginRequestDTO;
import com.cybercenter.core.infrastructure.dto.JwtResponseDTO;
import com.cybercenter.core.domain.auth.dto.PasswordLoginRequestDTO;
import com.cybercenter.core.domain.auth.dto.RegisterRequestDTO;
import com.cybercenter.core.infrastructure.dto.CoreUserDetails;
import com.cybercenter.core.infrastructure.dto.MessageDTO;
import com.cybercenter.core.shared.model.ResponseDTO;
import com.cybercenter.core.domain.user.dto.UserInfoDTO;
import com.cybercenter.core.domain.user.service.UserCacheService;
import com.cybercenter.core.infrastructure.dto.MessageTemplate;
import com.cybercenter.core.infrastructure.dto.MessageType;
import com.cybercenter.core.shared.model.StaticMessages;
import com.cybercenter.core.domain.auth.dto.VerifyCodeType;
import com.cybercenter.core.domain.user.entity.User;
import com.cybercenter.core.infrastructure.exception.InvalidUserOrPasswordException;
import com.cybercenter.core.infrastructure.exception.PhoneNumberIsExistException;
import com.cybercenter.core.infrastructure.exception.UsernameIsExistException;
import com.cybercenter.core.infrastructure.security.JwtUtils;
import com.cybercenter.core.shared.utils.VerificationUtils;
import com.cybercenter.core.infrastructure.wrapper.MessageSenderWrapper;
import com.cybercenter.core.domain.user.service.UserService;
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.Map;

@Service
@RequiredArgsConstructor
@Slf4j
public class AuthService {

    private final AuthenticationManager authenticationManager;
    private final JwtUtils jwtUtils;
```

```
    private final VerificationUtils verificationUtils;
    private final MessageSenderWrapper messageSenderWrapper;
    private final UserCacheService userCacheService;
    private final PasswordEncoder passwordEncoder;
    private final UserService userService;
    private final RefreshTokenService refreshTokenService;

    private static final int PUBLIC_SCOPE = 0;

    /**
     * Authenticates a user by verifying their username and password, and generates a JWT token for the user.
     *
     * @param passwordLoginRequestDTO The PasswordLoginRequestDTO containing the username and password.
     * @return A JwtResponseDTO containing the JWT token and refresh token for the authenticated user.
     * @throws InvalidUserOrPasswordException If the provided username or password is incorrect.
     */
    public JwtResponseDTO loginWithPassword(PasswordLoginRequestDTO passwordLoginRequestDTO) {

        // Create an authentication token with the provided username and password
        UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(passwordLoginRequestDTO.getUsername(), passwordLoginRequestDTO.getPassword());
        try {
            // Attempt to authenticate the user with the provided credentials
            Authentication authentication = authenticationManager.authenticate(authenticationToken);

            // Generate a JWT token for the authenticated user
            JwtResponseDTO jwtResponseDTO = jwtUtils.generateToken(authentication);

            // Set the authenticated user in the security context
            SecurityContextHolder.getContext().setAuthentication(authentication);

            // Create a refresh token for the user and set it in the JwtResponseDTO
            CoreUserDetails userPrincipal = (CoreUserDetails) authentication.getPrincipal();

jwtResponseDTO.setRefreshToken(refreshTokenService.createRefreshToken(userPrincipal.getUser().getId()).getToken());
```

```
            return jwtResponseDTO;
        } catch (AuthenticationException exception) {
            throw new InvalidUserOrPasswordException();
        }
    }

    /**
     * Authenticates a user by verifying a provided verification code and generates a JWT token
for the user.
     *
     * @param verifyCodeTokenRequestDTO The VerifyCodeLoginRequestDTO containing the
username and verification code.
     * @return A JwtResponseDTO containing the JWT token and refresh token for the
authenticated user.
     */
    public JwtResponseDTO loginWithVerifyCode(VerifyCodeLoginRequestDTO
verifyCodeTokenRequestDTO) {

        User user =
userCacheService.findByUsername(verifyCodeTokenRequestDTO.getUsername());

        // Validate the provided verification code against the user's phone number
        verificationUtils.validateVerifyCode(user.getPhoneNumber(),
verifyCodeTokenRequestDTO.getVerifyCode(), VerifyCodeType.LOGIN);

        // Generate a JWT token for the authenticated user based on their username
        JwtResponseDTO jwtResponseDTO =
jwtUtils.generateTokenByUsername(verifyCodeTokenRequestDTO.getUsername());

        // Create a refresh token for the user and set it in the JwtResponseDTO

jwtResponseDTO.setRefreshToken(refreshTokenService.createRefreshToken(user.getId()).getT
oken());

        return jwtResponseDTO;
    }

    /**
     * Sends a verification code to the phone number associated with a specified username.
     *
     * @param username The username associated with the user whose phone number will
receive the verification code.
     * @param verifyCodeType The type of verification code being sent (e.g., for registration,
password reset, etc.).
```

```
     * @return A ResponseDTO indicating the success of sending the verification code.
     */
    public ResponseDTO sendUserVerifyCode(String username, VerifyCodeType
verifyCodeType) {
        User user = userCacheService.findByUsername(username);
        return sendPhoneNumberVerifyCode(user.getPhoneNumber(), verifyCodeType);
    }

    /**
     * Sends a verification code to a specified phone number for authentication purposes.
     *
     * @param phoneNumber The phone number to which the verification code will be sent.
     * @param type The type of verification code being sent (e.g., for registration, password
reset, etc.).
     * @return A ResponseDTO indicating the success of sending the verification code.
     */
    public ResponseDTO sendPhoneNumberVerifyCode(String phoneNumber, VerifyCodeType
type) {
        // Generate a verification code for the specified phone number and type
        Integer verifyCode = verificationUtils.generateVerifyCode(phoneNumber, type);

        // Construct a message to be sent, including the phone number, message type (SMS),
template, and tokens (including the verification code)
        messageSenderWrapper.sendMessage(MessageDTO.builder()
                .to(phoneNumber)
                .messageType(MessageType.SMS.getId())
                .template(MessageTemplate.AUTH.VERIFY)
                .tokens(Map.of("verifyCode", String.valueOf(verifyCode)))
                .build());

        return ResponseDTO.success(StaticMessages.SEND_OTP_TO_PHONE_NUMBER);
    }

    /**
     * Registers a new user by validating the provided information and creating a new user
account.
     *
     * @param registerRequestDTO The RegisterRequestDTO containing the user's registration
details.
     * @return A JwtResponseDTO containing the JWT token and refresh token for the newly
registered user.
     * @throws UsernameIsExistException If the provided username already exists in the system.
     * @throws PhoneNumberIsExistException If the provided phone already exists in the
system.
```

```
     */
    @Transactional
    public JwtResponseDTO registerUser(RegisterRequestDTO registerRequestDTO) {
        // Check if the username already exists in the system
        if (userService.existUsername(registerRequestDTO.getUsername())) {
            throw new UsernameIsExistException();
        }

        // Check if the mobile number already exists in the system
        if (userService.existPhoneNumber(registerRequestDTO.getPhoneNumber())) {
            throw new PhoneNumberIsExistException();
        }

        // Validate the verification code provided by the user to ensure it matches the one sent to
their phone number
        verificationUtils.validateVerifyCode(registerRequestDTO.getPhoneNumber(),
registerRequestDTO.getVerifyCode(), VerifyCodeType.REGISTER);

        //encode password

registerRequestDTO.setPassword(passwordEncoder.encode(registerRequestDTO.getPassword
()));

        UserInfoDTO userInfoDTO = userService.save(registerRequestDTO);
        userService.addDefaultAuthority(userInfoDTO.getId(), PUBLIC_SCOPE);

        // Generate a JWT token for the newly registered user based on their username
        JwtResponseDTO jwtResponseDTO =
jwtUtils.generateTokenByUsername(registerRequestDTO.getUsername());

        // Create a refresh token for the user and set it in the JwtResponseDTO

jwtResponseDTO.setRefreshToken(refreshTokenService.createRefreshToken(userInfoDTO.getI
d()).getToken());

        return jwtResponseDTO;
    }


    public void setNewPassword(ChangePasswordDTO dto) {
        User user = userCacheService.findByUsername(dto.getUsername());

        // Validate the verification code for the user's phone number
```

```
        verificationUtils.validateVerifyCode(user.getPhoneNumber(), dto.getVerifyCode(),
VerifyCodeType.FORGET_PASSWORD);

        // Encode the new password and set it for the user
        user.setPassword(passwordEncoder.encode(dto.getPassword()));
        userCacheService.save(user);
    }
}
```