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

import com.cybercenter.core.shared.model.StaticMessages;
import com.cybercenter.core.domain.auth.entity.RefreshToken;
import com.cybercenter.core.domain.user.entity.User;
import com.cybercenter.core.infrastructure.exception.NotFoundException;
import com.cybercenter.core.infrastructure.exception.TokenRefreshException;
import com.cybercenter.core.domain.auth.repository.RefreshTokenRepository;
import com.cybercenter.core.domain.user.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.time.Instant;
import java.util.UUID;

@Component
@RequiredArgsConstructor
public class RefreshTokenService {

    private final RefreshTokenCacheService refreshTokenCacheService;
    private final RefreshTokenRepository refreshTokenRepository;
    private final UserRepository userRepository;

    @Value("${jwtRefreshExpirationMs}")
    private Long refreshTokenDurationMs;

    public RefreshToken findByToken(String token) {
        return refreshTokenCacheService.findByToken(token);
    }

    /**
     * Creates a new refresh token for a given user ID.
     * This method generates a new refresh token, sets its properties (user, expiry date, and
token value),
     * saves it to the cache, and returns the saved refresh token.
     *
     * @param userId The ID of the user for whom the refresh token is being created.
     * @return The created and saved refresh token.
     */
    public RefreshToken createRefreshToken(Long userId) {
        RefreshToken refreshToken = new RefreshToken();

        User user = userRepository.findById(userId)
```

```
                .orElseThrow(() -> new NotFoundException(StaticMessages.NOT_FOUND_USER));
        refreshToken.setUser(user);
        refreshToken.setExpiryDate(Instant.now().plusMillis(refreshTokenDurationMs));
        refreshToken.setToken(UUID.randomUUID().toString());


refreshTokenRepository.findByUser_Id(userId).ifPresent(refreshTokenCacheService::delete);
        refreshToken = refreshTokenCacheService.save(refreshToken);
        return refreshToken;
    }

    /**
     * Verifies the expiration of a given refresh token.
     * This method checks if the refresh token has expired by comparing its expiry date with the
current time.
     * If the token has expired, it deletes the token from the cache and throws a
TokenRefreshException.
     *
     * @param token The refresh token to verify.
     * @throws TokenRefreshException If the refresh token has expired.
     */
    public void verifyExpiration(RefreshToken token) {
        // Check if the token's expiry date is before the current time
        if (token.getExpiryDate().compareTo(Instant.now()) < 0) {
            // If the token has expired, delete it from the cache
            refreshTokenCacheService.delete(token);

            // Throw an exception to indicate that the token has expired
            throw new TokenRefreshException();
        }
    }

}
```