Session Management in Spring Security

 # JWT Session Management for Secure Authentication

Objective

Our goal is to efficiently manage user sessions using JWT tokens:

1. **Login Request Handling**:
   - Generate **Access Token (AT)** and **Refresh Token (RT)**.
   - Store session details using a structured schema.
2. **Token Renewal**:
   - Issue a new AT using RT **only if RT is valid** and **session exists**.
3. **Session Limit Handling**:
   - If session limit is reached, remove the **least recently used (LRU)** session.

Schema for Session Management

A new entity SessionManagement is created to manage user sessions.

Step 1: Create SessionManagement Entity

```
@Entity
@Table(name="sessions")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class SessionManagement {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String refreshToken;

    @CreationTimestamp
    private LocalDateTime lastUsedAt;

    @ManyToOne
    private UserEntity user;

}
```

Step 2: Implement Session Management Services

Create a service to manage session storage and validation.

```
@Service
@AllArgsConstructor
public class SessionManagementServices {

    private final SessionManagementRepository sessionRepository;

    private final int SESSION_LIMIT = 2;

    public void generateSession(UserEntity user, String refreshToken){
        List<SessionManagement> userSessions =
sessionRepository.findByUser(user);

        if (userSessions.size() == SESSION_LIMIT) {
```

```
            userSessions.sort(Comparator.comparing(SessionManagement::getLastUsed
At));
            SessionManagement leastRecentlyUsedSession = userSessions.get(0);
            sessionRepository.delete(leastRecentlyUsedSession);
        }

        SessionManagement newSession = SessionManagement.builder()
                .user(user)
                .refreshToken(refreshToken)
                .lastUsedAt(LocalDateTime.now())
                .build();

        sessionRepository.save(newSession);
    }

    public void validateSession(String refreshToken){
        SessionManagement session =
sessionRepository.findByRefreshToken(refreshToken)
                .orElseThrow(() ->
                        new SessionAuthenticationException(
                                "Session not found with refreshToken: " + refreshToken));

        session.setLastUsedAt(LocalDateTime.now());

        sessionRepository.save(session);

    }
}
```

Step 3: Integrate Session Management in Authentication Services

Handle Login and Session Creation

```
public LoginResponseDTO login(LoginDTO loginDTO) {
    Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(loginDTO.getEmail(),
loginDTO.getPassword())
    );

    UserEntity userEntity = (UserEntity) authentication.getPrincipal();
    String accessToken = jwtServices.generateAccessToken(userEntity);
    String refreshToken = jwtServices.generateRefreshToken(userEntity);

    sessionManagementServices.generateSession(userEntity, refreshToken);

    return new LoginResponseDTO(userEntity.getUserid(), accessToken,
refreshToken);

}
```

Refresh Token Handling and Session Validation

```
public LoginResponseDTO refreshToken(String refreshToken) {
    Long userId = jwtServices.getUserIdFromToken(refreshToken);

    sessionManagementServices.validateSession(refreshToken);

    UserEntity userEntity = userService.findById(userId);
    String accessToken = jwtServices.generateAccessToken(userEntity);
```

```
    return new LoginResponseDTO(userEntity.getUserId(), accessToken,
 refreshToken);
}
```

Final Summary

1. **Login**: Client sends login request, and server generates AT & RT.
2. **Session Storage**: Session details are stored in SessionManagement table.
3. **Token Renewal**: New AT is issued only if RT is valid and session exists.
4. **Session Limit Handling**: Least recently used session is removed when session limit is reached.
5. **Security**: AT is short-lived, while RT ensures secure and efficient session management.

This structured approach enhances security and **prevents unauthorized access**, ensuring a seamless user experience.

By Aron