# JWT Validation Testing Guide

## Overview

This guide demonstrates how JWT validation works in both the API Gateway and Auth Service, covering all scenarios and edge cases.

## JWT Validation Points

### 1. API Gateway Level

- **Purpose**: Validate tokens before routing to microservices
- **How**: Calls Auth Service `/validate` endpoint
- **What**: Token signature, expiry, blacklist status
- **Result**: Adds user context headers (X-User-Id, X-Username, X-User-Roles)

### 2. Auth Service Level

- **Purpose**: Protect Auth Service's own endpoints
- **How**: JwtAuthenticationFilter + Spring Security
- **What**: Token signature, expiry, blacklist status, user status
- **Result**: Sets SecurityContext for role-based access control

## Complete Testing Scenarios

### Scenario 1: Valid Token - Success Path

bash

```bash
# Login to get valid token
TOKEN_RESPONSE=$(curl -s -X POST http://localhost:8081/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "admin",
    "password": "Admin@123"
  }')

ACCESS_TOKEN=$(echo $TOKEN_RESPONSE | jq -r '.accessToken')
echo "Access Token: $ACCESS_TOKEN"

# Test 1: Access Auth Service protected endpoint directly
echo "Test 1: Direct Auth Service access"
curl -v -X GET http://localhost:8081/api/users/me \
  -H "Authorization: Bearer $ACCESS_TOKEN"

# Expected: 200 OK with user profile
# JWT validated by Auth Service's JwtAuthenticationFilter

# Test 2: Access through Gateway
echo "Test 2: Via API Gateway"
curl -v -X GET http://localhost:8080/auth/users/me \
  -H "Authorization: Bearer $ACCESS_TOKEN"

# Expected: 200 OK with user profile
# JWT validated by Gateway, then forwarded to Auth Service
```

### Scenario 2: Missing Authorization Header

bash

```
# Test 1: Direct to Auth Service
echo "Test 1: No token - Auth Service"
curl -v -X GET http://localhost:8081/api/users/me

# Expected Response:
# Status: 401 Unauthorized
# Body: {
#   "timestamp": "2025-01-15T10:30:00",
#   "status": 401,
#   "error": "Unauthorized",
#   "message": "Authentication is required to access this resource"
# }

# Test 2: Via Gateway
echo "Test 2: No token - Gateway"
curl -v -X GET http://localhost:8080/auth/users/me

# Expected Response:
# Status: 401 Unauthorized
# Gateway rejects before reaching Auth Service
```

### Scenario 3: Invalid Token Format

bash

```bash
# Test 1: Wrong format (no Bearer prefix)
echo "Test 1: Missing Bearer prefix"
curl -v -X GET http://localhost:8081/api/users/me \
  -H "Authorization: $ACCESS_TOKEN"

# Expected: 401 Unauthorized

# Test 2: Random string
echo "Test 2: Invalid token string"
curl -v -X GET http://localhost:8081/api/users/me \
  -H "Authorization: Bearer invalid.token.here"

# Expected: 401 Unauthorized
# Body: {"error": "Invalid or expired token"}

# Test 3: Malformed JWT
echo "Test 3: Malformed JWT"
curl -v -X GET http://localhost:8081/api/users/me \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ik

# Expected: 401 Unauthorized
```

### Scenario 4: Expired Token

bash

```bash
# Create a token with very short expiry (for testing)
# Or wait 24 hours for normal token to expire

# Use expired token
echo "Test: Expired token"
EXPIRED_TOKEN="eyJhbGciOiJIUzI1NiJ9..." # Use an actual expired token

curl -v -X GET http://localhost:8081/api/users/me \
  -H "Authorization: Bearer $EXPIRED_TOKEN"

# Expected: 401 Unauthorized
# Body: {"error": "Invalid or expired token"}

# JWT validation catches expired token in:
# - JwtUtil.isTokenExpired() returns true
# - JwtUtil.extractAllClaims() throws ExpiredJwtException
```

### Scenario 5: Blacklisted Token (After Logout)


bash

```bash
# Step 1: Login
TOKEN_RESPONSE=$(curl -s -X POST http://localhost:8081/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "test_user",
    "password": "Test@123456"
  }')

ACCESS_TOKEN=$(echo $TOKEN_RESPONSE | jq -r '.accessToken')
REFRESH_TOKEN=$(echo $TOKEN_RESPONSE | jq -r '.refreshToken')

# Step 2: Verify token works
echo "Before logout - token works:"
curl -X GET http://localhost:8081/api/users/me \
  -H "Authorization: Bearer $ACCESS_TOKEN"
# Expected: 200 OK

# Step 3: Logout (blacklists the token)
curl -X POST http://localhost:8081/api/auth/logout \
  -H "Content-Type: application/json" \
  -d "{
    \"token\": \"$ACCESS_TOKEN\",
    \"refreshToken\": \"$REFRESH_TOKEN\"
  }"

# Step 4: Try to use blacklisted token
echo "After logout - token rejected:"
curl -v -X GET http://localhost:8081/api/users/me \
  -H "Authorization: Bearer $ACCESS_TOKEN"

# Expected: 401 Unauthorized
# Body: {"error": "Token has been revoked"}
# Validation fails at: tokenBlacklistRepository.existsByToken(jwt)
```

### Scenario 6: Token for Disabled User

bash

```bash
# Step 1: Login as regular user
USER_TOKEN_RESPONSE=$(curl -s -X POST http://localhost:8081/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "test_user",
    "password": "Test@123456"
  }')

USER_TOKEN=$(echo $USER_TOKEN_RESPONSE | jq -r '.accessToken')

# Step 2: Admin disables the user
ADMIN_TOKEN_RESPONSE=$(curl -s -X POST http://localhost:8081/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "admin",
    "password": "Admin@123"
  }')

ADMIN_TOKEN=$(echo $ADMIN_TOKEN_RESPONSE | jq -r '.accessToken')

curl -X PUT "http://localhost:8081/api/users/test_user/enable?enabled=false" \
  -H "Authorization: Bearer $ADMIN_TOKEN"

# Step 3: Try to use token of disabled user
echo "Token of disabled user:"
curl -v -X GET http://localhost:8081/api/users/me \
  -H "Authorization: Bearer $USER_TOKEN"

# Expected: 401 Unauthorized
# UserDetails loaded but user.enabled = false
# Spring Security blocks access
```

### Scenario 7: Token for Locked Account


bash

```bash
# Step 1: Trigger account lockout (5 failed attempts)
for i in {1..5}; do
  curl -X POST http://localhost:8081/api/auth/login \
    -H "Content-Type: application/json" \
    -d '{
      "username": "test_user",
      "password": "WrongPassword"
    }'
done

# Step 2: Try to login (should be locked)
LOCKED_RESPONSE=$(curl -s -X POST http://localhost:8081/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "test_user",
    "password": "Test@123456"
  }')

echo $LOCKED_RESPONSE
# Expected: 423 Locked
# Body: {
#   "error": "Account Locked",
#   "message": "Account is locked due to multiple failed login attempts..."
# }

# Step 3: If user had token before lockout, try to use it
curl -v -X GET http://localhost:8081/api/users/me \
  -H "Authorization: Bearer $OLD_USER_TOKEN"

# Expected: 401 Unauthorized
# UserDetails shows accountLocked = true
```

### Scenario 8: Role-Based Access Control

bash

```bash
# Login as regular user (ROLE_USER only)
USER_TOKEN=$(curl -s -X POST http://localhost:8081/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "test_user",
    "password": "Test@123456"
  }' | jq -r '.accessToken')

# Test 1: Access allowed endpoint
echo "Test 1: User accessing own profile (allowed)"
curl -v -X GET http://localhost:8081/api/users/me \
  -H "Authorization: Bearer $USER_TOKEN"
# Expected: 200 OK (has ROLE_USER or ROLE_ADMIN)

# Test 2: Access admin-only endpoint
echo "Test 2: User accessing admin endpoint (forbidden)"
curl -v -X GET http://localhost:8081/api/users \
  -H "Authorization: Bearer $USER_TOKEN"
# Expected: 403 Forbidden
# Token is valid but user lacks ROLE_ADMIN
# @PreAuthorize("hasRole('ADMIN')") blocks access

# Login as admin
ADMIN_TOKEN=$(curl -s -X POST http://localhost:8081/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "admin",
    "password": "Admin@123"
  }' | jq -r '.accessToken')

# Test 3: Admin accessing admin endpoint
echo "Test 3: Admin accessing admin endpoint (allowed)"
curl -v -X GET http://localhost:8081/api/users \
  -H "Authorization: Bearer $ADMIN_TOKEN"
# Expected: 200 OK with list of all users
```

### Scenario 9: Token Refresh with Validation

bash

```bash
# Step 1: Login
TOKEN_RESPONSE=$(curl -s -X POST http://localhost:8081/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "test_user",
    "password": "Test@123456"
  }')

ACCESS_TOKEN=$(echo $TOKEN_RESPONSE | jq -r '.accessToken')
REFRESH_TOKEN=$(echo $TOKEN_RESPONSE | jq -r '.refreshToken')

# Step 2: Use refresh token to get new access token
NEW_TOKEN_RESPONSE=$(curl -s -X POST http://localhost:8081/api/auth/refresh \
  -H "Content-Type: application/json" \
  -d "{
    \"refreshToken\": \"$REFRESH_TOKEN\"
  }")

NEW_ACCESS_TOKEN=$(echo $NEW_TOKEN_RESPONSE | jq -r '.accessToken')

# Step 3: Verify new token works
curl -v -X GET http://localhost:8081/api/users/me \
  -H "Authorization: Bearer $NEW_ACCESS_TOKEN"
# Expected: 200 OK

# Step 4: Try to use revoked refresh token again
curl -v -X POST http://localhost:8081/api/auth/refresh \
  -H "Content-Type: application/json" \
  -d "{
    \"refreshToken\": \"$REFRESH_TOKEN\"
  }"
# Expected: 401 Unauthorized (if refresh token set to single-use)
```

### Scenario 10: Password Change Invalidates Tokens

bash

```
# Step 1: Login
TOKEN_RESPONSE=$(curl -s -X POST http://localhost:8081/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "test_user",
    "password": "Test@123456"
  }')

ACCESS_TOKEN=$(echo $TOKEN_RESPONSE | jq -r '.accessToken')
REFRESH_TOKEN=$(echo $TOKEN_RESPONSE | jq -r '.refreshToken')

# Step 2: Change password
curl -X PUT http://localhost:8081/api/users/me/password \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "oldPassword": "Test@123456",
    "newPassword": "NewTest@789"
  }'

# Step 3: Try to use old access token
echo "Using old access token after password change:"
curl -v -X GET http://localhost:8081/api/users/me \
  -H "Authorization: Bearer $ACCESS_TOKEN"
# Expected: Could still work if token not expired
# But refresh token is revoked

# Step 4: Try to refresh with old refresh token
echo "Trying to refresh with old refresh token:"
curl -v -X POST http://localhost:8081/api/auth/refresh \
  -H "Content-Type: application/json" \
  -d "{
    \"refreshToken\": \"$REFRESH_TOKEN\"
  }"
# Expected: 401 Unauthorized
# Refresh tokens deleted on password change

# Step 5: Login with new password
curl -X POST http://localhost:8081/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "username": "test_user",
    "password": "NewTest@789"
  }'
# Expected: 200 OK with new tokens
```

# JWT Validation Flow Details

## In Auth Service (JwtAuthenticationFilter)

java

1. Extract JWT from Authorization header
   - Check "Bearer " prefix
   - Extract token string

2. Check token blacklist
   - Query: tokenBlacklistRepository.existsByToken(jwt)
   - If blacklisted → 401 with "Token has been revoked"

3. Validate token signature and expiry
   - JwtUtil.extractUsername(jwt)
   - May throw ExpiredJwtException
   - May throw SignatureException
   - May throw MalformedJwtException

4. Load user details
   - userDetailsService.loadUserByUsername(username)
   - Checks user.enabled
   - Checks user.accountNonLocked
   - Loads user roles

5. Validate token against user
   - jwtUtil.validateToken(jwt, userDetails)
   - Checks username matches
   - Checks token not expired

6. Set authentication
   - Create UsernamePasswordAuthenticationToken
   - Set in SecurityContextHolder
   - Include user authorities (roles)

7. Continue filter chain
   - Request reaches controller
   - @PreAuthorize checks roles

### In API Gateway (JwtAuthenticationFilter)

java

1. Extract JWT from Authorization header
   - Same as Auth Service

2. Call Auth Service validation endpoint
   - POST http://auth-service/api/auth/validate
   - Body: { "token": "jwt_token" }
   - Uses Circuit Breaker pattern

3. Process validation response
   - Check response.valid == true
   - Extract userId, username, roles

4. Enrich request headers
   - Add X-User-Id: "123"
   - Add X-Username: "john_doe"
   - Add X-User-Roles: "ROLE_USER,ROLE_ADMIN"

5. Forward to target microservice
   - Microservice receives user context
   - No need to validate JWT again

# Common Error Responses

### 401 Unauthorized - Missing Token

json

```
{
  "timestamp": "2025-01-15T10:30:00",
  "status": 401,
  "error": "Unauthorized",
  "message": "Authentication is required to access this resource",
  "path": "/api/users/me"
}
```

### 401 Unauthorized - Invalid Token

json

```
{
 "error": "Invalid or expired token"
}
```

### 401 Unauthorized - Blacklisted Token

json

```
{
 "error": "Token has been revoked"
}
```

### 403 Forbidden - Insufficient Permissions

json

```
{
 "timestamp": "2025-01-15T10:30:00",
 "status": 403,
 "error": "Forbidden",
 "message": "Access Denied"
}
```

### 423 Locked - Account Locked

json

```
{
  "timestamp": "2025-01-15T10:30:00",
  "status": 423,
  "error": "Account Locked",
  "message": "Account is locked due to multiple failed login attempts. Please try again after 2025-01-15T11:00:00"
}
```


# Debugging JWT Issues

## Enable Debug Logging

yaml

```
logging:
  level:
    com.microservices.auth.security: DEBUG
    com.microservices.auth.util: DEBUG
    org.springframework.security: DEBUG
```

## Check JWT Token Content

bash

```
# Decode JWT (without verification)
echo $ACCESS_TOKEN | cut -d'.' -f2 | base64 -d | jq

# Expected output:
{
  "sub": "john_doe",
  "userId": 2,
  "email": "john@example.com",
  "authType": "DATABASE",
  "roles": ["ROLE_USER"],
  "iat": 1705316400,
  "exp": 1705402800
}
```

### Common Issues and Solutions

| Issue | Cause | Solution |
|---|---|---|
| "Invalid JWT signature" | Wrong secret key | Verify JWT_SECRET matches in both services |
| "Token has expired" | Token older than 24h | Use refresh token or login again |
| "User not found" | User deleted after token issued | Login again |
| "Access Denied" (403) | Missing role | Check user roles and @PreAuthorize |
| "Token has been revoked" | Token blacklisted | Login again to get new token |

# Performance Considerations

### Token Validation Performance

- **Auth Service**: ~5-10ms (DB query + JWT parsing)
- **Gateway**: ~15-25ms (HTTP call + circuit breaker)
- **Optimization**: Consider caching validation results (with short TTL)

### Database Queries per Request

1. Check token blacklist: 1 query
2. Load user details: 1-2 queries (user + roles)
3. Total: 2-3 queries per authenticated request

### Recommended Optimizations

1. Add database indexes (already included)
2. Use connection pooling (HikariCP configured)
3. Consider Redis for token blacklist
4. Cache user details for short periods
5. Use async validation where possible

---

**This testing guide covers all JWT validation scenarios in the microservices architecture. Use it to verify security and troubleshoot issues.**