Spring Boot Microservices with JWT Authentication

A production-ready microservices architecture with API Gateway, JWT authentication, role-based access control, and LDAP/Database authentication support.

Architecture Overview

```
┌──────────┐    ┌──────────────┐    ┌──────────────┐    ┌──────────┐
│  Client  │───▶│  API Gateway │───▶│ Auth Service │───▶│ Database │
└──────────┘    │  (Port:8080) │    │ (Port:8081)  │    │  MySQL   │
                └──────────────┘    └──────────────┘    └──────────┘
                       │                   │
                       │                   ▼
                       │            ┌──────────┐
                       │            │   LDAP   │
                       │            └──────────┘
                       ▼
                ┌──────────────┐
                │ Microservices│
                │  (User/Prod) │
                └──────────────┘
```

Features

API Gateway

- ✅ JWT token validation filter
- ✅ Route-based authentication
- ✅ Circuit breaker with Resilience4j
- ✅ Service discovery with Eureka
- ✅ CORS configuration
- ✅ Request header enrichment (user info)

Auth Service

- ✅ **Dual Authentication**: Database + LDAP
- ✅ **Role-Based Access Control**: USER, ADMIN, MODERATOR
- ✅ **JWT Token Management**: Access & Refresh tokens
- ✅ **Account Security**:
  - Failed login attempt tracking

    - Account lockout mechanism (configurable)
    - Token blacklisting
    - Automatic token cleanup
- ✅ **Production Features**:
    - Global exception handling
    - Input validation
    - Comprehensive logging
    - Database connection pooling
    - Scheduled tasks

Prerequisites

- Java 21
- MySQL 8.0+
- Maven 3.8+
- (Optional) LDAP Server for LDAP authentication

Setup Instructions

1. Database Setup

sql

```sql
-- Create database
CREATE DATABASE auth_db;

-- The application will auto-create tables on startup
```

2. Environment Configuration

API Gateway (application.yml)

yaml

```yaml
jwt:
  secret: your-256-bit-secret-key-here

auth-service:
  url: http://localhost:8081
```

Auth Service (application.yml)

yaml

```yaml
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/auth_db
    username: root
    password: your-password

  ldap:
    urls: ldap://localhost:389
    base: dc=example,dc=com
    username: cn=admin,dc=example,dc=com
    password: admin-password

jwt:
  secret: your-256-bit-secret-key-here

auth:
  max-login-attempts: 5
  lockout-duration-minutes: 30
```

3. Running the Services

Start Eureka Server (Port: 8761)

bash

```bash
# Run your Eureka server first
```

Start Auth Service (Port: 8081)

bash

```bash
cd auth-service
mvn clean install
mvn spring-boot:run
```

Start API Gateway (Port: 8080)

bash

```bash
cd api-gateway
mvn clean install
mvn spring-boot:run
```

4. Default Credentials

On first startup, the system creates:

- **Username**: `admin`
- **Password**: `Admin@123`
- **Roles**: ADMIN, USER
- **Auth Type**: DATABASE

⚠️ **IMPORTANT**: Change this password immediately in production!

API Endpoints

Authentication Endpoints (via Gateway: http://localhost:8080)

1. Register New User

bash

```
POST /auth/register
Content-Type: application/json

{
  "username": "john_doe",
  "email": "john@example.com",
  "password": "SecurePass123"
}

Response:
{
  "accessToken": "eyJhbGciOiJIUzI1NiIs...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIs...",
  "tokenType": "Bearer",
  "expiresIn": 86400000,
  "username": "john_doe",
  "roles": ["ROLE_USER"]
}
```

2. Login

bash

```
POST /auth/login
Content-Type: application/json

{
  "username": "admin",
  "password": "Admin@123"
}

Response: (Same as register)
```

3. Refresh Token

bash

```
POST /auth/refresh
Content-Type: application/json

{
  "refreshToken": "eyJhbGciOiJIUzI1NiIs..."
}
```

4. Logout

bash

```
POST /auth/logout
Content-Type: application/json

{
  "token": "eyJhbGciOiJIUzI1NiIs...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIs..."
}
```

5. Validate Token (Internal - Called by Gateway)

bash

```
POST /auth/validate
Content-Type: application/json

{
  "token": "eyJhbGciOiJIUzI1NiIs..."
}

Response:
{
  "valid": true,
  "userId": "1",
  "username": "admin",
  "roles": ["ROLE_ADMIN", "ROLE_USER"],
  "message": "Token is valid"
}
```

Protected Endpoints (Example)

bash

```bash
GET /user/profile
Authorization: Bearer eyJhbGciOiJIUzI1NiIs...

# Gateway will:
# 1. Extract JWT token
# 2. Call Auth Service to validate
# 3. Add headers: X-User-Id, X-Username, X-User-Roles
# 4. Forward to User Service
```

Authentication Types

Database Authentication (Default)

1. User credentials stored in MySQL
2. Password hashed with BCrypt (strength: 12)
3. Automatic on registration

LDAP Authentication (For Admins)

1. Configure user in database with `authType: LDAP`
2. Login credentials validated against LDAP server

3. User roles managed in database

Creating LDAP User in Database:

sql

```sql
INSERT INTO users (username, email, password, auth_type, enabled, account_non_locked)
VALUES ('ldap_admin', 'ldap_admin@company.com', '', 'LDAP', true, true);

-- Assign role
INSERT INTO user_roles (user_id, role_id)
SELECT u.id, r.id FROM users u, roles r
WHERE u.username = 'ldap_admin' AND r.name = 'ROLE_ADMIN';
```

Security Features

Account Lockout

- After 5 failed login attempts (configurable)
- Account locked for 30 minutes (configurable)
- Automatic unlock after duration
- Tracks attempts per user

Token Management

- **Access Token**: 24 hours validity
- **Refresh Token**: 7 days validity
- Token blacklisting on logout
- Automatic cleanup of expired tokens (hourly)

Request Flow with JWT

```
1. Client → Gateway: Request with JWT
2. Gateway → Auth Service: Validate token
3. Auth Service: Check blacklist, validate signature, check expiry
4. Auth Service → Gateway: Validation response with user info
5. Gateway → Microservice: Forward request + user headers
6. Microservice: Process with user context (X-User-Id, X-Username, X-User-Roles)
```

Testing Scenarios

1. Successful Authentication Flow

bash

```bash
# Register
curl -X POST http://localhost:8080/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "test_user",
    "email": "test@example.com",
    "password": "Test@123456"
  }'

# Login
curl -X POST http://localhost:8080/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "test_user",
    "password": "Test@123456"
  }'

# Use access token for protected endpoints
curl -X GET http://localhost:8080/user/profile \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN"
```

2. Token Refresh

bash

```bash
curl -X POST http://localhost:8080/auth/refresh \
  -H "Content-Type: application/json" \
  -d '{
    "refreshToken": "YOUR_REFRESH_TOKEN"
  }'
```

3. Failed Login & Lockout

bash

```bash
# Try 5 times with wrong password
for i in {1..5}; do
  curl -X POST http://localhost:8080/auth/login \
    -H "Content-Type: application/json" \
    -d '{
      "username": "test_user",
      "password": "WrongPassword"
    }'
done

# 6th attempt will return 423 Locked
```

4. Logout

bash

```bash
curl -X POST http://localhost:8080/auth/logout \
  -H "Content-Type: application/json" \
  -d '{
    "token": "YOUR_ACCESS_TOKEN",
    "refreshToken": "YOUR_REFRESH_TOKEN"
  }'

# Token is now blacklisted
```

Error Handling

The service handles all edge cases with appropriate HTTP status codes:

Status Code	Scenario
400	Invalid input / Validation errors
401	Invalid credentials / Invalid token
409	User already exists
423	Account locked
500	Server error

Example Error Response:

json

```json
{
  "timestamp": "2025-01-15T10:30:00",
  "status": 401,
  "error": "Unauthorized",
  "message": "Invalid username or password"
}
```

Production Considerations

Security Checklist

- ✅ Change default admin password
- ✅ Use environment variables for secrets
- ✅ Configure JWT secret (256-bit minimum)
- ✅ Enable HTTPS in production
- ✅ Set appropriate CORS origins
- ✅ Configure rate limiting (external)
- ✅ Enable audit logging
- ✅ Regular security updates

Database Optimization

- ✅ Indexes on frequently queried columns
- ✅ Connection pooling (HikariCP)
- ✅ Query optimization
- ✅ Regular backup strategy

Monitoring

yaml

```yaml
management:
  endpoints:
    web:
      exposure:
        include: health,info,metrics,prometheus
```

Access metrics at: `http://localhost:8081/actuator/metrics`

Project Structure

```
auth-service/
├── entity/          # JPA entities
├── repository/      # Spring Data repositories
├── service/         # Business logic
├── security/        # Security configurations
├── controller/      # REST controllers
├── dto/             # Data Transfer Objects
├── exception/       # Custom exceptions
├── util/            # Utility classes (JWT)
└── config/          # Configuration classes

api-gateway/
├── filter/          # Gateway filters
├── dto/             # DTOs for communication
└── exception/       # Exception classes
```

Troubleshooting

Issue: Gateway can't reach Auth Service

Solution: Ensure Eureka server is running and both services are registered.

Issue: LDAP authentication fails

Solution:

1. Verify LDAP server is running
2. Check LDAP configuration in application.yml
3. Ensure user DN pattern is correct

Issue: Token validation fails

Solution:

1. Verify JWT secret matches in both services
2. Check token hasn't expired
3. Ensure token isn't blacklisted

Issue: Account lockout not working

Solution: Check `auth.max-login-attempts` configuration

Advanced Configuration

Custom JWT Expiration

yaml

```yaml
jwt:
  expiration: 3600000      # 1 hour
  refresh-expiration: 86400000  # 24 hours
```

Custom Lockout Settings

yaml

```yaml
auth:
  max-login-attempts: 3
  lockout-duration-minutes: 60
```

Circuit Breaker Tuning

yaml

```
resilience4j:
  circuitbreaker:
    instances:
      authService:
        failure-rate-threshold: 50
        wait-duration-in-open-state: 10s
```

License

This project is licensed under the MIT License.

Support

For issues and questions:

- Check logs in both services
- Verify configuration settings
- Ensure all dependencies are properly installed
- Check Eureka dashboard for service registration

Built with ❤️ using Spring Boot 3.2, Spring Security 6, and JWT