**Authorization SOAP Client - Spring Boot 3 Implementation**

This project provides a complete Spring Boot 3 implementation for consuming the Authorization Shared Service SOAP web service using WebServiceTemplate.

**Features**

* ✅ **Complete SOAP Client Implementation** using WebServiceTemplate
* ✅ **Type-safe Java classes** generated from WSDL
* ✅ **REST API wrapper** for easy integration
* ✅ **Comprehensive error handling** with global exception handler
* ✅ **Request/Response logging** with configurable interceptors
* ✅ **Helper services** for request building and response parsing
* ✅ **Integration tests** and usage examples
* ✅ **Production-ready configuration** with timeouts and retries

**Project Structure**

src/

├── main/

│ ├── java/

│ │ └── com/example/

│ │ ├── config/ # SOAP configuration and interceptors

│ │ ├── controller/ # REST controllers

│ │ ├── service/ # SOAP client and helper services

│ │ └── exception/ # Global exception handling

│ └── resources/

│ ├── example-new.wsdl # WSDL file

│ └── application.yml # Configuration

└── test/

└── java/

└── com/example/

└── integration/ # Integration tests

**Setup Instructions**

**1. Prerequisites**

* Java 17 or higher
* Maven 3.6+
* Spring Boot 3.x

**2. Project Setup**

1. **Create a new Spring Boot project**:
2. mvn archetype:generate -DgroupId=com.example \
3. -DartifactId=authorization-soap-client \
4. -DarchetypeArtifactId=maven-archetype-quickstart \
5. -DinteractiveMode=false
6. **Add dependencies** to your pom.xml (see Maven Dependencies artifact)
7. **Place the WSDL file** in src/main/resources/example-new.wsdl
8. **Generate Java classes** from WSDL:
9. mvn clean generate-sources

This will create classes in target/generated-sources/xjc/com/example/generated/

1. **Copy the Java files** from the artifacts above into your project

**3. Configuration**

Update src/main/resources/application.yml:

soap:

service:

url: http://10.29.60.95:8080/easws/sharedservice/AuthorizationSharedService

timeout: 30000

logging:

level:

com.example: DEBUG

org.springframework.ws: DEBUG

**4. Available Operations**

The SOAP service provides the following operations:

| **Operation** | **Description** | **REST Endpoint** |
| --- | --- | --- |
| isHealthy | Health check | GET /api/authorization/health |
| findMatchingUserIdentity | Find user by criteria | POST /api/authorization/find-user-identity |
| findOfficesByEauthId | Find offices by EAuth ID | GET /api/authorization/find-offices |
| findUsersByCriteria | Find users by office/role | GET /api/authorization/find-users |
| getUserRoles | Get user roles | POST /api/authorization/get-user-roles |

**5. Usage Examples**

**Direct Service Usage**

@Autowired

private AuthorizationSoapClientService soapClientService;

@Autowired

private SoapRequestBuilderService requestBuilderService;

// Health check

boolean healthy = soapClientService.isHealthy();

// Find user identity

Map<String, String> criteria = Map.of("username", "john.doe");

List<MapEntry> mapEntries = requestBuilderService.createMapEntries(criteria);

UserIdentity user = soapClientService.findMatchingUserIdentity(mapEntries);

// Get user roles

GetUserRolesResponse roles = soapClientService.getUserRoles(user);

**REST API Usage**

# Health check

curl -X GET http://localhost:8080/api/authorization/health

# Find user identity

curl -X POST http://localhost:8080/api/authorization/find-user-identity \

-H "Content-Type: application/json" \

-d '{"username": "john.doe", "email": "john.doe@example.com"}'

# Find offices

curl -X GET "http://localhost:8080/api/authorization/find-offices?eauthId=12345&officeTypes=FSA&applicationId=MyApp&requestHost=localhost"

**6. Running the Application**

# Start the application

mvn spring-boot:run

# Or build and run JAR

mvn clean package

java -jar target/authorization-soap-client-0.0.1-SNAPSHOT.jar

**7. Testing**

# Run unit tests

mvn test

# Run integration tests

mvn verify

# Test with cURL (see cURL Examples artifact)

chmod +x test-api.sh

./test-api.sh

**Key Components**

**SoapConfig**

* Configures WebServiceTemplate with marshalling
* Sets up timeout and connection settings
* Adds logging interceptors

**AuthorizationSoapClientService**

* Main service for SOAP operations
* Handles all five SOAP operations
* Includes comprehensive error handling

**SoapRequestBuilderService**

* Helper service for building requests
* Type-safe request construction
* Response parsing utilities

**GlobalExceptionHandler**

* Handles SOAP faults and runtime exceptions
* Returns structured error responses
* Includes proper HTTP status codes

**Error Handling**

The implementation includes comprehensive error handling:

* **SOAP Faults**: Mapped to HTTP 400 with fault details
* **Connection Timeouts**: Configurable timeout settings
* **Invalid Arguments**: Validation with appropriate error messages
* **Runtime Exceptions**: Caught and wrapped with context

**Production Considerations**

1. **Security**: Add authentication headers if required
2. **Monitoring**: Integrate with metrics and health checks
3. **Caching**: Consider caching frequently accessed data
4. **Rate Limiting**: Implement rate limiting for external calls
5. **Circuit Breaker**: Add circuit breaker pattern for resilience

**Troubleshooting**

**Common Issues**

1. **ClassNotFoundException**: Ensure generated classes are in classpath
2. **Connection refused**: Check SOAP service URL and network connectivity
3. **SOAP Fault**: Review request parameters and service requirements
4. **Timeout**: Adjust timeout settings in configuration

**Debug Logging**

Enable debug logging to see SOAP requests/responses:

logging:

level:

org.springframework.ws.client.MessageTracing.sent: TRACE

org.springframework.ws.client.MessageTracing.received: TRACE

**Additional Resources**

* [Spring Web Services Reference](https://docs.spring.io/spring-ws/docs/current/reference/)
* [JAXB Documentation](https://docs.oracle.com/javase/tutorial/jaxb/)
* [Spring Boot Web Services Guide](https://spring.io/guides/gs/consuming-web-service/)