Technical Documentation – Backend

LEO-Based Assessment Tool

---

1. Introduction

This document provides the **technical documentation** for the backend of the *LEO-Based Assessment Tool*. It describes the system architecture, core components, domain model, business logic, and technical decisions.

The backend is implemented as a **Spring Boot (Java 17)** RESTful application and serves as the central logic layer between the frontend (Electron UI) and the PostgreSQL database.

---

2. Architectural Overview

2.1 High-Level Architecture

The system follows a **client–server architecture**:

- **Frontend**: Electron-based desktop application
- **Backend**: Spring Boot REST API
- **Database**: Cloud-based PostgreSQL (Neon)

The backend exposes REST endpoints that are consumed by the frontend. End users do not directly access the backend.

2.2 Layered Architecture

The backend follows a classic **layered architecture** to ensure separation of concerns, maintainability, and testability.

Layers:

1. **Controller Layer**
2. Exposes REST endpoints
3. Handles HTTP requests and responses

4. Performs request validation and authorization checks

5. **Service Layer**

6. Contains business logic
7. Implements grading rules, cascade logic, and recommendation logic

8. Coordinates transactions

9. **Persistence Layer**

10. JPA/Hibernate entities
11. Spring Data repositories
12. Database interaction abstraction

---

3. Technology Stack

- **Language**: Java 17
- **Framework**: Spring Boot
- **Build Tool**: Maven
- **Persistence**: Spring Data JPA / Hibernate
- **Database**: PostgreSQL (Neon cloud database)
- **Security**: Spring Security (role-based authorization)
- **Testing**: JUnit 5, Spring Boot Test, Testcontainers, REST-assured
- **Deployment**: Docker, Docker Compose, AWS EC2

---

4. Core Domain Model

4.1 User

Represents an authenticated system user.

Attributes: - `id` - `username` - `password` - `role` (ADMIN, TEACHER, STUDENT)

Users are authenticated and authorized based on their role.

4.2 Course

Represents a course managed by a teacher.

Attributes: - `id` - `name` - `teacher`

Relationships: - One teacher can manage multiple courses - Students are enrolled via enrollment entities

---

4.3 Learning Outcome Element (LEO)

A **LEO** represents a concrete, assessable learning outcome.

Attributes: - `id` - `title` - `description` - `course`

Relationships: - LEOs are connected via dependency relationships - Dependencies form a **directed graph**, not a simple tree

---

4.4 Assessment

Represents the assessment status of a LEO for a specific student.

Attributes: - `id` - `student` - `leo` - `status` - `assessedAt`

---

5. Assessment Status Model

Each LEO can be in one of the following states:

- **NOT_REACHED**
- **PARTIALLY_REACHED**
- **REACHED**
- **UNMARK**

The grading scale is **fixed** and not configurable.

---

6. Business Logic

6.1 Assessment Handling

Assessments are created and updated via the `AssessmentService`.

Responsibilities: - Create or update assessment entries - Validate assessment changes - Ensure transactional consistency

---

6.2 Cascade Grading Logic

The backend implements **automatic cascade grading**.

Rules: - When a higher-level LEO is marked as **REACHED**, all dependent (lower-level) LEOs are updated automatically - This reflects implied mastery of prerequisite learning outcomes

Example:

> If a student reaches "Can multiply 3-digit numbers", the system can automatically mark "Can multiply 2-digit numbers" as reached or partially reached.

The cascade logic is implemented in the service layer to guarantee consistency across all clients.

6.3 Recommendation Logic

The system generates recommendations for **next possible LEOs**.

Criteria: - Current assessment state - Dependency graph - Unlocked prerequisites

The recommendation logic helps guide students through achievable next learning outcomes.

---

7. Security & Authorization

Security is implemented using **Spring Security**.

Features: - Authentication via username and password - Role-based authorization - Protected endpoints for teachers and students

Access control examples: - Only teachers can create or modify LEOs - Only teachers can assess students - Students can only view their own progress

---

8. REST API Design

The backend exposes RESTful endpoints following standard HTTP conventions.

Main endpoint groups:

- `/api/leos` – manage LEOs and dependencies
- `/api/courses` – manage courses
- `/api/students` – manage students and enrollments
- `/api/assessments` – record and update assessment results
- `/api/recommendations` – retrieve next possible LEOs

Responses use JSON and standard HTTP status codes.

---

9. Database Design

9.1 Database Technology

- PostgreSQL (Neon cloud database)
- Secure connection via credentials and connection string

9.2 Persistence Strategy

- Entities mapped using JPA/Hibernate
- Schema managed automatically via Hibernate
- Relationships defined using entity associations

The database stores: - Users and roles - Courses and enrollments - LEOs and dependency relations - Assessments and progress data

---

10. Testing Strategy

The backend includes multiple testing levels:

Unit Tests

- Service-layer logic
- Cascade grading rules
- Recommendation logic

Integration Tests

- Controller and repository integration
- Database interaction using Testcontainers (PostgreSQL)

API Tests

- REST endpoints tested with REST-assured

---

11. Deployment

The backend is deployed using **Docker and Docker Compose**.

Deployment characteristics: - Containerized Spring Boot application - Environment-based configuration - Hosted on AWS EC2 - Connected to Neon PostgreSQL database

This setup ensures reproducible and portable deployments.

---

12. Design Decisions

Key design decisions include:

- Use of a **graph-based LEO model** instead of a simple hierarchy
- Centralized cascade grading logic in the backend
- Fixed grading scale to ensure consistency
- Separation of frontend and backend responsibilities
- Cloud-based database for scalability and reliability

---

13. Related Repositories

- Backend Repository: https://github.com/piy678/SENGPRJ_Group6
- Frontend Repository: https://github.com/piy678/SENGPRJ_Group6_FrontendPart

---

Group 6 — SENGPRJ
Supervisor: *Thomas Mandl*