Software Architecture

2025-2026

Project 2

Context

In a previous sprint/project, the LMS application was developed, and configurability issues were addressed.
While this application is functional, it was observed that its centralized (monolithic or modular monolith) architecture hampers its performance, availability, scalability and elasticity.

Goal

The goal of this sprint/project is to reengineer the LMS application by adopting a decentralized/distributed architecture.

Requirements

Non-functional requirements

- The system must improve its availability.
- The system must increase the performance by 25% when in high demand (i.e. >Y requests/period).
- The system must use hardware parsimoniously, according to the runtime demanding of the system. Demanding peaks of >Y requests/period occur seldom.
- The software clients should not be affected by the changes (if any) in the API, except in extreme cases.
- The system must adhere to the company's SOA strategy of API-led connectivity.

Functional requirements

- Student A: As a librarian, I want to create a Book, Author and Genre in the same process.
- Student B: As a librarian, I want to create a Reader and the respective User in the same request.
- Student C: As a reader, upon returning a Book, I want to leave a text comment about the Book and grading it (0-10).

Conditions

- Number of students per group: 2 (exceptionally 3 if accepted by faculty)
- Submission deadline: 2026-01-04
- Mode: Individual, synchronous, face-to-face
- Weight: 100%
- Cf. Evaluation criteria in Moodle

Project 1

Context

In a previous project, the Library Management REST-oriented backend service was developed.
This application provides REST endpoints for managing:

- Books
- Genres
- Authors
- Readers
- Lendings

Problem

The Library Management service does not support:

- Extensibility
- Configurability
- Reliability

Goals

The goals of this project (P1) are:

- Documenting
  - System-as-is (reverse engineering design)
  - Requirements, especially ASR
    - Classification
  - Adoption of ADD
    - Identification and classification of requirements
    - ADD process
    - Tactics
    - Reference architectures
    - Patterns
    - Architectural design alternatives and rational (visual models)
- Make the developments required so that the project supports:
  - Persisting data in different data models (e.g. relational, document) and SGBD: (i) SQL + Redis, (ii) MongoDB+Redis and (iii) Elastic Search
  - Retrieve a book's ISBN by title using different external systems: (i) ISBNdb, (ii) Google Books API, and (iii) Open Library Search API, or (iv) two of the previous together
  - Generate IDs for other entities in different formats based on varying specifications
- The previous alternatives must be defined during configuration (setup time), which directly impacts runtime behavior
- To improve automated functional testing:
  - Functional opaque-box with SUT = classes
  - Functional transparent-box with SUT = domain classes
  - Mutation tests with SUT = classes
  - Functional opaque-box with SUT = controller+service+{domain, repository, gateways}
  - Functional opaque-box with SUT = system
  - NB: Tests provide the key evidence for evaluating development

Conditions

- Number of students per group:
  - 2
  - exceptionally 3 or 1, if accepted by faculty, and implies differences in goals
- First evaluation:
  - Mandatory
  - Submission deadline: 2025-11-02
  - Mode: Individual, synchronous, face-to-face
  - Weight: 100% (or 70% if second evaluation takes places)
- Second evaluation:
  - Optional
  - Submission deadline: (as P2 submission deadline)
  - Mode: Individual, synchronous, face-to-face
  - Weight: 30% (or 0% if it does not take place)