Semester 3

Design document

LMS

sumaya gabr

Contents

[1. Introduction 3](#_Toc155698918)

[2. Architecture Constraints and Design Decisions 3](#_Toc155698919)

[2.1 Spring Boot 3](#_Toc155698920)

[2.2 React 3](#_Toc155698921)

[2.3 MySQL 3](#_Toc155698922)

[3. C4 Model Diagrams 4](#_Toc155698923)

[3.1 Level 1: System Context Diagram 4](#_Toc155698924)

[3.2 Level 2: Container Diagram 5](#_Toc155698925)

[3.3 Level 3: Component Diagram 6](#_Toc155698926)

[3.4 Level 4: code Diagram 8](#_Toc155698927)

[3.4.1 Sequence Diagram: student enrollment 8](#_Toc155698928)

[4. Explanations and Design Principles 9](#_Toc155698929)

[5. Continuous Integration (CI) Setup 9](#_Toc155698930)

[5.1 CI Overview 9](#_Toc155698931)

[5.1.1 Developer's Machine 9](#_Toc155698932)

[5.1.2 GitLab Server 10](#_Toc155698933)

[5.1.3 Pipeline Runner 10](#_Toc155698934)

[5.1.4 SonarQube Instance 10](#_Toc155698935)

[5.1.5 CI Architecture Diagram 10](#_Toc155698936)

[5.2 CI Workflow 10](#_Toc155698937)

[5.3 Configuration 11](#_Toc155698938)

[6. Security OWASP Report 11](#_Toc155698939)

[6.1 Security Risks Table 11](#_Toc155698940)

[6.2 Reasoning 12](#_Toc155698941)

# 1. Introduction

This design document outlines the architectural constraints, design decisions, and provides C4 model diagrams for the Learning Management System (LMS). It also connects the architectural choices with software design principles such as SOLID, KISS, DRY, and YAGNI.

# 2. Architecture Constraints and Design Decisions

## 2.1 Spring Boot

Spring Boot is chosen for building the backend of the LMS due to its:

* Simplicity: It simplifies the setup and development of Java applications.
* Community and Ecosystem: It has a large and active community, extensive documentation, and a rich ecosystem of libraries.
* Additionally, it aligns with the school's curriculum and best practices in enterprise Java development.

## 2.2 React

React is selected for the frontend of the LMS due to its:

* Component-Based: React's component-based architecture promotes reusability and maintainability of UI elements.
* Virtual DOM: It uses a virtual DOM to optimize rendering performance, ensuring a responsive user interface.
* Large Community: React has a strong community and numerous libraries for UI development.
* The school's curriculum also includes React as a recommended technology for frontend development.

## 2.3 MySQL

MySQL is chosen as the database for the LMS because of:

* Reliability: MySQL is known for its reliability and data integrity, making it suitable for handling student and course data.
* Scalability: It can handle large datasets and scale with the growth of the LMS.
* Compatibility: MySQL is compatible with various programming languages and frameworks.
* This choice is consistent with the school's database curriculum

# 3. C4 Model Diagrams

## 3.1 Level 1: System Context Diagram

At the top level, we have the Front-End, represented by a Single Page Application (SPA). Users interact with the application through the intuitive user interface provided by the SPA. This is the entry point for user engagement.A diagram of a diagram

Description automatically generated

## 3.2 Level 2: Container Diagram

At Level 2, we dive into the API and Data Management layer. This level consists of API controllers that handle user requests and serve as the bridge between the Front-End and the Core Application. It's also where the database is involved to store and retrieve data.

A diagram of a diagram

Description automatically generated with medium confidence

## 3.3 Level 3: Component Diagram

The core of the application is at this level. It manages the business logic and application behavior. This level interacts with data stored in the database and ensures data consistency. Data access and storage are key aspects here, making it the heart of the application's functionality.

A screenshot of a computer

Description automatically generated

## 3.4 Level 4: code Diagram

The code diagram represents the UML (Unified Modeling Language) class diagram for an educational platform application. The diagram illustrates the core entities and their relationships within the application, which include users (Admin, Student, Teacher), courses, course materials, enrollments, and user management. The application is designed to manage courses, course materials, user registrations, and user roles, allowing users to enroll in courses, and administrators to manage users and courses.

A screenshot of a computer

Description automatically generated

### 3.4.1 Sequence Diagram: student enrollment

The sequence diagram shows the interaction between a Student and the application when enrolling in a course.

A screenshot of a computer

Description automatically generated

# 4. Explanations and Design Principles

In the Learning Management System (LMS) architecture, the application design adheres to the SOLID principles, which emphasize good software design practices. Here's how the SOLID principles are applied in the LMS:

**Single Responsibility Principle (SRP):**

Example: The CreateCourseUseCase, UpdateCourseUseCase, and DeleteCourseUseCase classes each have a single responsibility related to user management. They encapsulate the logic for creating, updating, and deleting user accounts, respectively.

**Open/Closed Principle (OCP):**

Example: The use cases (e.g., CreateCourseUseCaseImpl, UpdateCourseUseCaseImpl) are defined by interfaces (CreateCourseUseCase, UpdateCourseUseCase) that other developers can implement to extend the system's functionality. Existing code that uses these interfaces doesn't need to change when new use cases are added.

**Dependency Inversion Principle (DIP):**

Example: The core components of the LMS (e.g., controllers, services) depend on the use case interfaces (e.g., CreateCourseUseCase, UpdateCourseUseCase) rather than concrete implementations. This decouples the high-level components from low-level details, promoting flexibility and testability.

# 5. Continuous Integration (CI) Setup

## 5.1 CI Overview

My development process uses Git for code changes. The GitLab server is the hub, triggering a Pipeline Runner that manages tasks like fetching code, building with Gradle, testing, and analysis using SonarQube.

### 5.1.1 Developer's Machine

The development process initiates on the Developer's Machine, where the source code is created and modified. Developers use Git as the version control system to manage changes efficiently.

### 5.1.2 GitLab Server

GitLab serves as the central repository and orchestrator of the CI pipeline. It stores the source code and coordinates the CI/CD workflow. The GitLab server triggers the pipeline upon detecting changes in the repository.

### 5.1.3 Pipeline Runner

The CI pipeline is executed by a dedicated Pipeline Runner. This node is responsible for fetching the source code, building the application, running tests, and handling other defined stages in the pipeline. In our case, the pipeline runner is configured to use a Docker image with Gradle version 8.3.0 and JDK 17.

### 5.1.4 SonarQube Instance

SonarQube is integrated into the CI pipeline to perform code quality analysis. It inspects the codebase for code smells, bugs, and security vulnerabilities. The results are then reported back to the pipeline. It is integrated to check the code coverage for the business layer where it should be >80% to pass the quality gate.

### 5.1.5 CI Architecture Diagram

The diagram illustrates the flow from the Developer's Machine to GitLab, through the Pipeline Runner, and finally to the SonarQube Instance for code analysis.

A graphic of a logo and a candle

Description automatically generated with medium confidence

## 5.2 CI Workflow

**Build Stage:**

* Developers push changes on GitLab.
* The Pipeline Runner fetches and builds the code using Gradle.

**Test Stage:**

* The application is tested for correctness with Gradle.

**SonarQube Analysis Stage:**

* SonarQube checks code quality and security.

**Deployment Stage:**

* If all's well, the app can be deployed.

## 5.3 Configuration

<https://git.fhict.nl/I499796/lms-individual-track/-/blob/main/.gitlab-ci.yml#L15>

**build:** Assemble the app with Gradle.

**test:** Run tests.

**sonar:** Check code quality with SonarQube.

**deploy:** Placeholder for custom deployment scripts.

This CI setup ensures a smooth development cycle, integrating code changes through testing, quality analysis, and potential deployment.

# 6. Security OWASP Report

## 6.1 Security Risks Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Security Risk | Likelihood | Impact | Risk Assesment | Possible Actions | Planned Actions |
| A1 | Broken Access Control | High | Severe | High | implementing proper access controls and authorization mechanisms | Yes |
| A2 | Cryptographic Failure | Very Unlikely | Severe | Low | using HTTPS would be a good way to prevent it. Hashing, securing and using strong cryptographic algorithms | N/A |
| A3 | Injection | Likely | Severe | High | Implement proper input validation, sanitize input | Yes, ongoing improvement |
| A4 | Insecure Design | High | Severe | High | implementing secure design principle. Threat modeling. Miss-use cases. | Yes |
| A5 | Security Misconfiguration | Likely | Severe | High | Review and secure configurations, regular audits | Yes, ongoing improvement |
| A6 | Vulnerable and Outdated Components | Not specified | Not specified | Not specified | Regularly update components, use secure versions | Yes, ongoing improvement |
| A7 | Identification and Authentication Failures | High | Not specified | High | Implement strong authentication mechanisms | Yes, ongoing improvement |
| A8 | Software and Data Integrity Failures | Not specified | High | High | Verify software updates, secure CI/CD pipelines | Yes, ongoing improvement |
| A9 | Security Logging and Monitoring Failures | Not specified | Not specified | Not specified | Improve logging and monitoring practices | Yes, ongoing improvement |
| A10 | Server-Side Request Forgery | High | Moderate | Moderate | Improve framework implementation | No, risk accepted |

## 6.2 Reasoning

**1. Broken Access Control (A1): (granting or denying access to a specific resource or functionality)**

- High likelihood: Unauthorized access can lead to unauthorized creation or modification of learning content or student enrollments.

- Severe impact: Compromised access control can undermine the integrity of educational materials.

- High-risk assessment: Recognizing the sensitivity of data.

- Planned actions involve implementing proper access controls and authorization mechanisms.

**2. Cryptographic Failure (A2): (when attackers target sensitive information like password, user credentials, credit card credentials etc.)**

- Very Unlikely likelihood: The absence of passwords or user data reduces the likelihood of cryptographic failure.

- Severe impact: Cryptographic failures could compromise the confidentiality of user information.

- Low-risk assessment: Acknowledging the reduced likelihood in an LMS context.

- No planned actions, but using HTTPS would be a good way to prevent it. Hashing, securing and using strong cryptographic algorithms

**3. Injection (A3): (when user supplied data are not filtered or validated by the application)**

- Likely likelihood: Injection vulnerabilities pose a risk to the integrity of learning content.

- Severe impact: Exploitation could lead to the manipulation of course content.

- High-risk assessment: Considering the centrality of content integrity in an LMS.

- Planned actions involve implementing input validation to safeguard against injection attacks.

**4. Insecure Design (A4): (attackers target design flaws and take advantage of that)**

- High likelihood: Insecure design could result in systemic vulnerabilities affecting the entire learning platform.

- Severe impact: Systemic vulnerabilities may compromise the reliability of the learning environment.

- High-risk assessment: Acknowledging the foundational importance of secure design in an LMS.

- Planned actions involve implementing secure design principle. Threat modeling. Miss-use cases.

**5. Security Misconfiguration (A5):**

- Likely likelihood: Misconfigurations may expose sensitive learning materials.

- Severe impact: Misconfigurations can lead to unauthorized access to educational resources.

- High-risk assessment: Recognizing the potential impact on the confidentiality of educational content.

- Planned actions involve regular reviews (checking and changing default passwords, updating security packages, too much details in error messages) and secure configurations to maintain a secure learning environment.

**6. Vulnerable and Outdated Components (A6): (users look at the app as a whole but attackers look at the app’s components and libraries and scan for vulnerabilities, outdatedness, and if components are unsupported)**

- Not specified: Lack of specificity in likelihood and impact.

- Planned actions involve proactive measures like regular updates to safeguard against vulnerabilities.

**7. Identification and Authentication Failures (A7): (Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks)**

- High likelihood: Authentication failures can lead to unauthorized access to student, teacher, or admin accounts.

- Not specified impact: Specific impact is not provided.

- High-risk assessment: Recognizing the importance of secure identification in an LMS.

- Planned actions involve implementing robust authentication mechanisms.

**8. Software and Data Integrity Failures (A8):**

- Not specified likelihood: Lack of specificity in likelihood.

- High impact: Data integrity failures can compromise the accuracy of assessment results.

- High-risk assessment: Recognizing the potential impact on the educational process.

- Planned actions involve verifying software updates to ensure data integrity.

**9. Security Logging and Monitoring Failures (A9): (to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. Many organizations don't log application activities properly, and even when they do, they don't monitor those logs to see what happened (or what is currently happening))**

- Not specified: Lack of specificity in likelihood and impact.

- Planned actions emphasize improving logging and monitoring practices to enhance incident detection.

**10. Server-Side Request Forgery (A10): (when a web application is fetching a remote resource without validating the user-supplied URL)**

- High likelihood: Indicated by the community survey, posing a risk to server-side operations.

- Moderate impact: The impact is moderate but could affect server reliability.

- Moderate risk assessment: Balancing likelihood and impact in the context of an LMS.

- Planned actions involve improving the framework, with a measured acceptance of risk.