Architecture description template

for use with ISO/IEC/IEEE 42010:2011

Architecture Description of

Layered archiecture for

Web based platform

“Bare bones” edition version: 2.2

Template prepared by:

Rich Hilliard

r.hilliard@computer.org

Distributed under Creative Commons Attribution 3.0 Unported License.

For terms of use, see: http://creativecommons.org/licenses/by/3.0/

# Contents

Contents 2

Using the template 4

License 5

Version history 5

Editions 6

Comments or questions 6

1 Introduction 7

1.1 Identifying information 7

1.2 Supplementary information 7

1.3 Other information 7

1.3.1 Overview (optional) 7

1.3.2 Architecture evaluations 7

1.3.3 Rationale for key decisions 8

2 Stakeholders and concerns 9

2.1 Stakeholders 9

2.2 Concerns 9

2.3 Concern–Stakeholder Traceability 10

3 Viewpoints+ 11

3.1 <Viewpoint Name> 11

3.2 Overview 11

3.3 Framed concerns and typical stakeholders 11

3.3.1 Concerns 11

3.3.2 Typical stakeholders 12

3.3.3 “Anti-concerns” (optional) 13

3.4 Model kinds+ 13

3.5.2 <Model-Kind-Name> operations (optional) 14

3.5.3 <Model-Kind-Name> correspondence rules 14

3.6 Operations on views 14

3.7 Correspondence rules 15

3.8 Examples (optional) 15

3.9 Notes (optional) 15

3.10 Sources 15

4 Views+ 16

4.1 View: <View-Name> 16

4.1.1 Models+ 16

4.1.2 <Model-Name> 16

4.1.3 Known Issues with View 16

5 Consistency and correspondences 18

5.1 Known inconsistencies 18

5.2 Correspondences in the AD 18

5.3 Correspondence rules 18

A Architecture decisions and rationale 19

A.1 Decisions 19

Bibliography 20

# Using the template

ISO/IEC/IEEE 42010, *Systems and software engineering — Architecture description*, defines the contents of an architecture description (AD) [4].

Figure 1 depicts that contents in terms of a UML class diagram. The AD template in this document defines places for all required information and offers the user some additional guidance on preparing an AD.

An AD may take many forms, not prescribed by the Standard: it could be presented as a document, a set of documents, a collection of models, a model repository, or in some other form – as long as the required content is accessible in some manner. In particular, organization and ordering of required information is not defined by the Standard. Thus, headings and subheadings in this template are merely suggestive – not required.

The template uses a few conventions, as follows.

\* “Musts” are items which must be present to satisfy the Standard. Musts are marked like this.

∆ “Shoulds” are items recommended to be present, but not required by the Standard. Shoulds are marked like this.

Optional items are marked with this: (optional). Guidance that defines, explains or otherwise amplifies upon the required items, or terms used therein, looks like this.

<Items> like <this> signal names to be filled-in by a user of the template and used throughout the resulting AD.

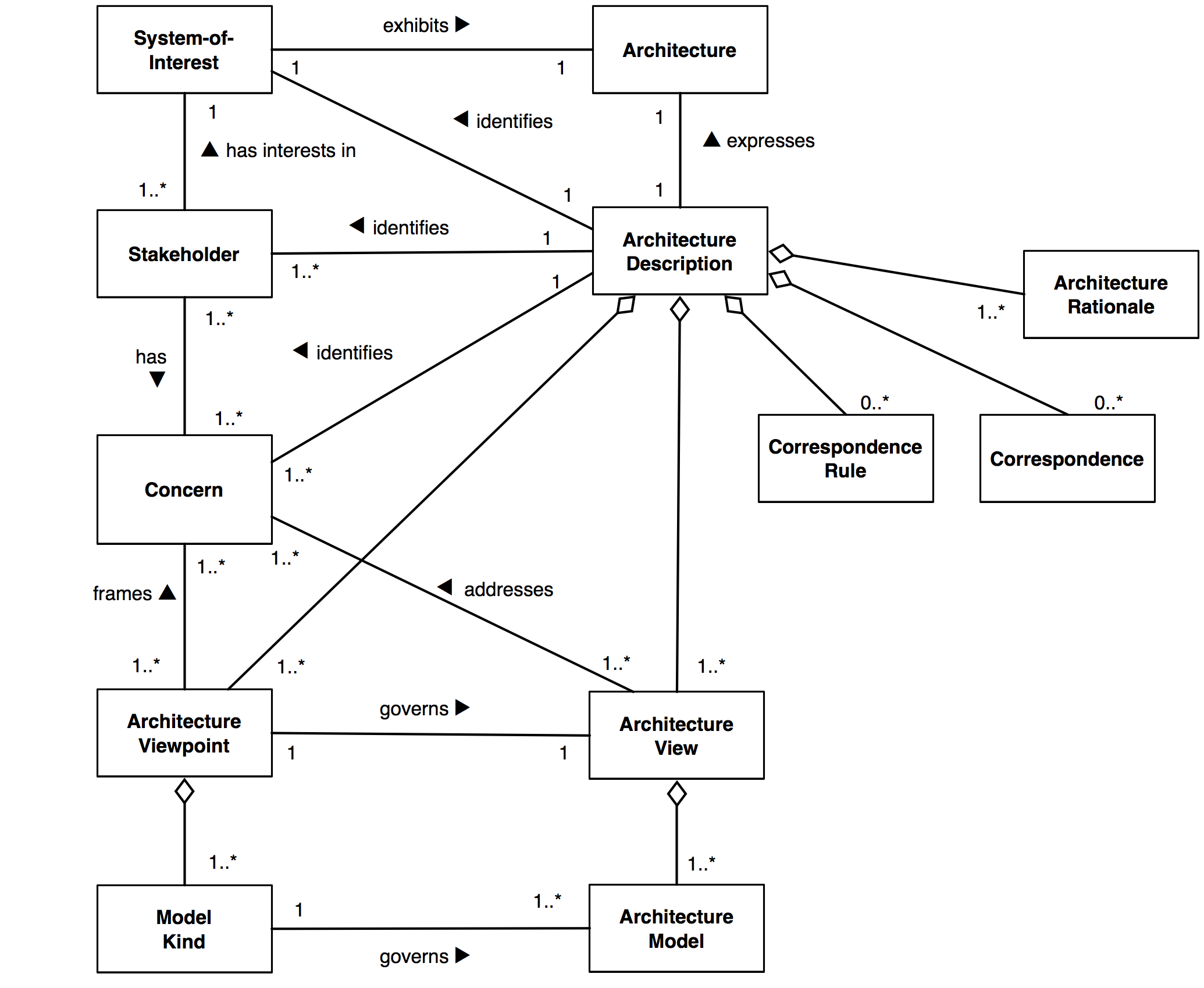


Figure 1

Figure 1: Content model of an architecture description

## License

The Architecture Description Template is copyright ©2012–2014 by Rich Hilliard.

The latest version is always available at:   
http://www.iso-architecture.org/42010/templates/.

The template is licensed under a Creative Commons Attribution 3.0 Unported License. For terms of use see:   
http://creativecommons.org/licenses/by/3.0/

This license gives you the user the right to share and remix this work to create architecture descriptions. It does not require you to share the results of your usage, but if your use is non- proprietary, we encourage you to share your work with others via the WG42 website   
http://www.iso-architecture.org/42010/.

## Version history

This template is based on one originally designed for use with IEEE std 1471:2000 [3] and now updated for ISO/IEC/IEEE 42010:2011. The present document is an enhanced version of the earlier template, with additional guidance, clarifications and examples for readers.

**rev 2.2 7** October 2014, Moved bibliography from bibtex to biblatex. Released revision with minor formatting fixes.

**rev 2.1a** June 2012, Initial release on 42010 website.

## Editions

This is the “bare bones” edition – it contains exactly only information items required by the Standard. Other editions meet the requirements of the Standard and contain additional information used in various documentation approaches (such as [6, 1]).

## Comments or questions

Contact the author Rich Hilliard ⟨r.hilliard@computer.org⟩ with comments, suggestions, improvements or questions.

For more information on ISO/IEC/IEEE 42010, visit the website:   
http://www.iso-architecture.org/42010/.

The template begins here (next page) . . .

# 1 Introduction

.

## 1.1 Identifying information

**Name :** Layered Architecture for java based learning management system(LMS)

**System of Interest:**

1. **User Management:** admin , instructor and student and their roles of each user type with is different from one to another
2. **Course Management:** that have course creation , enrollment management, attendance Management and each with its specific task
3. **Assessment and Grading:** that have quiz creation, assignment submission, Grading and Feedback
4. **Performance Tracking:** have student progress tracking, monitoring progress
5. **Notification** : have to send notification to each student in enrollment confirmation , grading assignment , course related updates

## 1.2 Supplementary information

**1.2.1 Data of Issue and statues:**

The document was issued on November 25, 2024

**Statues:** the current statues of this document is Draft and still development

**1.2.2 Authors , Reviewers and Approving Authority:**

**Authors:**

Tasneem Gomaa , StudentId:20220089

Nada Mohamed , StudentId:20221175

Rana Ibrahim , StudentId: 20220130

Nour Ahmed, StudentId:20221184

Nada Assad , StudentId:20221177

Roqaia Hassan , StudentId:20220129

**Reviewers:**

Mrs.Mahetab ,teaching assistant

Mr. Omar , teaching assistant

**Approving authority:**

**This document is submitted to :**

Dr.Lamia Abo Zaid, course instructor

University Name , Cairo University

Department , Information System

**1.2.3 Issuing Organization:**

**This document is issued by :**

**University Name:** cairo University

**Department:** Information system

**Course:** Advanced Software engineering

**Project Team :** Layered Management System Development Team

**1.2.4 Change History:**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Changes | Reviewed /Approved by |
| 1.0 | November 25, 2024 | Initial creation of architecture document  Add introduction section  Identifying section | Still Not |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**1.2.5 Scope:**

This Architecture Description addresses the design and implementation of the Learning Management System for university Project and cover the following areas:

System Components: the document include the details of the main components of the LMS including the User Management system , assessment System ,Notification System

**Functional Features:** the architecture covers features like enrollment, course creation , grading and performance tracking

**Technical Design**: the architecture outline the using of java with spring boot for backend and MySQL for data storage

**Security :** the design includes role based access for different user typers and ensure of system authentication and authorization by using Spring security

**Deployment:** the architecture make on integration with notification system , email for best practices

**1.2.6 context:**

The LMS is developed for university setting system for help of instructors ,students and administrators mange educational content , assessments, and student performance also provide centralized platform where courses can be created , assignments can be submitted , and student progress can be tracked.

**System Interaction:**

Users of system are three user types : admins,instructor, student.

**Admin**: Manages overall system settings, creates users, and manages courses.

**Instructor**: Creates courses, manages course content, adds assignments and quizzes,

grades students, removes students from courses.

**Student**: Enrolls in courses, accesses course materials, take quizzes, hand in

assignments, view assignments and quiz grades.

**External systems** :the LMS will integtate with email services like in course enrollment confirmations, feedback on assignments

**Technological Environment:**

**Backend :** the backend will be developed by java with spring boot to create APIs that handle course data and user interaction

**Database:** it will use the MySQL to store data about courses ,students ,instructors , grades and attendance

**Security:** role based access control will be implemented using spring security to ensure proper authentication and authorization

**Deployment environment:**

The system will be hosted in web server and will be accessible to users via web browser and may deployed on cloud infrastructure for scalability and high availability

**Constraints:**

The project will focus on backend and leave the front end design out side the scope of this document

Integration with third party will be limited

**Security :** the design includes role based access for different user typers and ensure of system authentication and authorization by using Spring security

**Deployment:** the architecture make on integration with notification system , email for best practices

**1.2.7 Glossary:**

This glossary will define key terms of this document architecture

**Admin**: User Role that Manages overall system settings, creates users, and manages courses and students enrollments.

**Instructor**: User role that Creates courses, manages course content, adds assignments and quizzes,

grades students, removes students from courses.

**Student**: User role that Enrolls in courses, accesses course materials, take quizzes, hand in

assignments, view assignments and quiz grades.

**course** :set of education materials and lessons designed by instructors to teach students a specific subject

**Enrollment :** the process that from it the students sign up to take courses

**One time password:** unique code used by students to confirm their attendance the the lesson

**quiz:** test that check the student knowledge through multiple choice ,true or false of short answer questions

**Assignments:** task that given to students that requires then to complete it and submit work

**Grading :**

Process where instructors evaluate student performance and assign scores to quizzes and assignments

**Spring Security :** framework in spring boot that provides the security features for authentication and authorization for users

**Role based access control RBAC:** used to manage user permissions based on role assigned to each user

**1.2.8 Summary:**

This architecture is description for the design for Learning Management system intended to manage courses , assessments, student progress in university environments.

It focuses on backend architecture the includes components for user management , course management, assessment and grading and performance tracking , notifications.

Its use spring boot for backend framework , relational database for data storage and spring security for role based access.

**1.2.9 Version Control Information :**

**Version control system**: Git

**Repository Hosting Platform** Git Hub

**Repository URL**: <https://github.com/tasneem821/advanceds.git>

**Branching Strategy:**

**(Main Branch)** :the primary development branch used for stable release

**1.2.10 Configuration Management Information:**

**Configuration management Tools:**

**Spring Profiles** : used for managing different environments

**Configuration files**:

**Application.properties:** contains application specific configurations

**Dependency Management:**

**Version control:** ensure all dependencies are specified with version numbers to prevent issues;

**Deployment Configuration:**

**Docker:** Used to containerize the application for deployment consistency across the environment

**1.2.11 references:**

**ISO/IEC/IEEE 42010:2011** – Systems and Software Engineering – Architecture Description. International Standard, ISO, 2011. https://ieeexplore.ieee.org/document/6129467

**Spring Boot Documentation** – Official documentation for Spring

Boot. <https://spring.io/projects/spring-boot>

**MySQL Documentation** – Official guide to MySQL database.: <https://dev.mysql.com/doc/>

## 1.3 Other information

### **1.3.1 Overview (optional)**

**Purpose:**

The purpose of the LMS(Learning Management System) to provide

An integrated platform for managing university courses. Assessments, student performance . it supports three user types which are students , instructors and administrators by offering lots of tools like course creation, enrollment management , attendance tracking , grading and performance tracking .

This architecture is designed to be scalabe,secure and user-friendly to meet the wide range of stakeholders

**Scope:**

**User Management:** Admin , students and instructors roles , manage features for registration like login , profile management and role based access control

**Course Management:** Creation of courses , enrollments tracking , lesson management, attendance tracking using one time passwords

**Assessment and grading :**the creation of quizzes and assignments , grading automatically for quizzes , manual grading of assignments , performance tracking

**Notification:** real time notification for students and instructors including update of course enrollments , assignments grading

The architecture will not include front end but it will include backend

**Context:**

It will serve a core component of university’s digital learning infrastructure . the system will integrate with existing systems , like student information system and will scale to support large numbers of students and instructors . it designed to flexible and allowance for future enhancements

The stakeholders of this architecture is :

**Administrators:** its responsible for managing the system , creating students account and ensure that good operation of the platform

**instructor:** create and manage courses , grade assignments, track students performance

**students :**interact with it to enroll in courses access course material submit assignment and track their grades

**Notification:** real time notification for students and instructors including update of course enrollments , assignments grading

**Reader’s Guide :**

**1.Introduction:**This section includes essential identifying information about the system and architecture. It defines the purpose, scope, and context of the architecture description, providing the reader with a high-level overview of the LMS.

**1.1Identifying information :** Contains project-specific information, such as the system of interest and the architecture style.

**1. 2 supplementary information :** Includes details such as date of issue, version control, configuration management, and references

**1.3 other information:** Provides additional context, including the system overview, evaluations, and rationale for key decisions

**2.Archiecture Views:** This section presents different views of the architecture. Views are organized based on the stakeholder concerns

**2.1 functional view :** Describes the key features and functionality of the system

**2. 2 static view :** Illustrates the static structure of the system, including components, classes, and relationships.

**2.3 dynamic view:** Shows how components interact during runtime, covering use cases, interactions, and behaviors.

**3.supplemetary information:** This section offers detailed technical information

**3.1 version control information :** Tracks changes made to the architecture over time, ensuring consistency and traceability

**3. 2 configuration management information :** Describes how the system configurations are controlled and maintained

**3.3 references:** Lists key resources, standards, and literature that informed the design decisions

**4.Other Information:** The final section includes any additional information that supports the architecture description

**4.1** **Architecture Evaluations:** create and manage courses , grade assignments, track students performance

**4. 2 Rationale for Key Decisions :** Explains why certain architectural decisions were made, including trade-offs and alternatives considered

**5 Appendix:** Any supporting documents, diagrams, or additional clarifications can be found in the appendix, which provides a more in-depth exploration of the system’s components.

### 1.3.2 Architecture evaluations

1. **Phase 0**

* **Business driver**

**1. Usability:** Ease of use for students, instructors, and administrators.

**2. Security**: Protection of user data through robust authentication and role-based access**.**

**3. Performance:** system’s ability to handle large volumes of users and provide timely responses.

**4. Scalability:** Support for increased users and courses in the future**.**

**5. Integration:** Seamless operation with email notifications and third-party tools.

**6. Maintainability:** ease of updating and enhancing the system

1. **Phase 1**
   1. **Architectural overview**

* Style: layered architectural

1. **presentation layer:** user interface
2. **business logic layer**: functions
3. **data access layer:** database interactions

* **technology:** Java with Spring Boot for RESTful API services.
  1. **Utility tree**

|  |  |  |  |
| --- | --- | --- | --- |
| **Quality attribute** | **Scenario** | Priority | Difficulty |
| Usability | Notifications filtered and displayed in < 1 second | Medium | Medium |
| Security | Unauthorized users cannot access sensitive data | High | Low |
| Performance | Generate reports for 10000 students within 2 seconds | High | High |
| Scalability | Support 5000 concurrent users during enrollment | High | High |
| Integration | Notifications for confirmation about enrollment within 2 min | High | High |
| Maintainability | Add new features without major system changes | Medium | Medium |

* 1. **Phase 2**

1. **Use case scenario**

* **Scenario: Concurrent enrollment requests**
* **Stimulus:** 5000 students want to enroll concurrently
* **Environment:** within enrollment week
* **Response:** system process all requests successfully within average 1 sec

1. **Evolution scenarios**

* **Scenario: adding new course during peek time**
* **Stimulus:** an instructor creates a new course
* **Environment:** 2000 student use system
* **Response:** course is created, and course is available without affecting system operations

1. **Exploratory scenarios**

* **Scenario: system recovery after failure**
* **Stimulus:** some of server go offline unexpectedly
* **Environment:** within normal hours with 2000 concurrent user
* **Response:** system redistributes load between remaining servers without go down
  1. **Phase 3**
* **Risks**
* **Database contention** during high concurrency scenarios
* **Email service reliability** may affect notification timeless
* **Processing analytical in real time** might affect performance issues when it is overloaded
* **Tradeoffs**
* **Security versus usability:** role-based access control guarantees strong access control but somewhat lengthens the time needed to validate logins.
* **Accuracy vs. Performance**: Pre-calculated analytics decrease real-time precision while increasing speed.
* **Synchronization versus Scalability Overhead**: While clones increase scalability, they also necessitate more synchronization.

### 1.3.3 Rationale for key decisions

\* An architecture description shall include rationale for each decision considered to be a key architecture decision (per ISO/IEC/IEEE 42010, 5.8.2).

See §A for further guidance about decisions and rationale.

# 2 Stakeholders and concerns

## 2.1 Stakeholders

1. **Students**:
   * Enroll in courses, access course materials, take quizzes, hand in
   * assignments, view assignments and quiz grades and also view notifications.
2. **Instructors**:
   * Create and manage courses, add assignments and quizzes, grade students, track student performance, removes students from courses and receive notifications.
3. **Administrators**:
   * Create users, monitor notifications, manages courses and oversee system operations.
4. **Developers**:
   * Build, maintain, and enhance the LMS system.
5. **University Management**:
   * Monitor system scalability, performance, and analytics for decision-making.

## 2.2 Concerns

 **Usability:** Ease of use for students, instructors, and administrators.

 **Security:** Protection of user data through robust authentication and role-based access.

 **Performance:** System's ability to handle large volumes of users and provide timely

responses.

 **Scalability**: Support for increased users and courses in the future.

 **Integration**: Seamless operation with email notifications and third-party tools.

 **Maintainability**: Ease of updating and enhancing the system.

## 2.3 Concern–Stakeholder Traceability

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Students | Instructors | Administrators | Developers | University Management |
| Usability | X | X | \_ | \_ | \_ |
| Performance | X | X | X | \_ | X |
| scalability | \_ | \_ | X | X | X |
| Security | X | \_ | X | \_ | \_ |
| Integration | \_ | X | \_ | X | \_ |
| Maintainability | \_ | \_ | \_ | X | \_ |

# 3 Viewpoints+

## 3.1 Conceptual Viewpoint

**Synonyms**:

* Domain Viewpoint
* High-Level Viewpoint

## 3.2 Overview

The Conceptual Viewpoint provides a high-level abstraction of the LMS, emphasizing the structural organization and interactions among the system's key components. It defines responsibilities, module relationships, and rules governing their use, ensuring a clear and maintainable architecture that supports the system's functional and technical requirements.

## 3.3 Concerns and stakeholders

### 3.3.1 Concerns

* **What is the primary functional responsibility assigned to each module?**

1. **User Management Module:**
   1. Handles the registration, login, and management of three user roles (Admin, Instructor, and Student).
   2. Features include profile management, role-based access, and user-specific functions:
      1. **Admin**: System settings, user management, and course creation.
      2. **Instructor**: Course content management, student grading, and attendance tracking.
      3. **Student**: Course enrollment, material access, and participation in quizzes and assignments.
2. **Course Management Module:**
   1. Manages course creation with title, description, duration, and multimedia content (e.g., videos, PDFs).
   2. Handles enrollment, enabling students to join courses and instructors/admins to view enrolled students.
   3. Supports attendance through OTP-based validation for each lesson.
3. **Assessment and Grading Module:**
   1. Facilitates quiz and assignment management:
      1. Instructors create quizzes using question banks and randomized selection.
      2. Students submit assignments as file uploads.
   2. Enables grading and feedback:
      1. Automated for quizzes, manual for assignments.
4. **Performance Tracking Module:**
   1. Allows instructors to track student progress across assignments, quizzes, and attendance.
5. **Notification Module:**
   1. Handles system and email notifications for events like enrollment confirmation, graded assignments, and course updates.

* **How is the system to be structured as a set of code units?**

1. **Presentation Layer:**
   1. User interface components (e.g., web and mobile views for students, instructors, and admins).
2. **Service Layer:**
   1. Implements business logic, such as course enrollment, quiz evaluation, and notification generation.
3. **Data Access Layer:**
   1. Manages database operations, including CRUD functionality for users, courses, and assessments.
4. **Infrastructure Layer:**
   1. Provides reusable services like email notifications, role-based access control (using Spring Security), and performance analytics.

* **What other software elements is a module allowed to use?**

1. **Role-Based Dependencies:**
   1. The **Presentation Layer** interacts only with the **Service Layer**.
   2. The **Service Layer** integrates with the **Data Access Layer** and **Infrastructure Layer** for database operations and additional functionality.
2. **Shared Utilities:**
   1. All modules can access foundational services such as logging, authentication, and notification services.
3. **Module-Specific APIs:**
   1. Notifications and email services provide specialized APIs for different user roles and events.

* **Which modules are related to other modules by relations (e.g., inheritance)?**

1. **Inheritance:**
   1. **User Management Module:**
      1. Admin, Instructor, and Student inherit from a common **User** base class, which defines shared properties (e.g., ID, name, email).
2. **Composition:**
   1. The **Course Management Module** includes components for lessons, multimedia files, and enrollment tracking.
3. **Dependency:**
   1. The **Assessment and Grading Module** depends on **User Management** to identify students and link assessments to their profiles.

### Typical stakeholders

**Developers:**

* Need clarity on module boundaries, relationships, and dependencies to implement features efficiently.

**System Architects:**

* Use the viewpoint to verify that modularization supports maintainability and scalability.

**Instructors and Students:**

* Represent end users whose workflows are directly modeled by modules (e.g., course management, assessments).

**Admins:**

* Require functionality for overall system management and performance analytics.

**Testers:**

* Rely on the viewpoint to identify test cases for individual modules and their interactions.

3.4 Execution Viewpoint

**Synonyms:**  
 • Deployment Viewpoint  
 • Runtime Viewpoint

3.5 Overview

The Execution Viewpoint captures how the LMS operates in its runtime environment, detailing the deployment of software components and their interaction during execution. It focuses on the system’s runtime behavior, addressing performance, fault tolerance, scalability, and data flow across different components. This viewpoint enables architects to ensure that the LMS functions efficiently under different load conditions, remains available even during faults, and can scale as needed.

3.6 Concerns and Stakeholders

3.6.1 Concerns

* **How is the system to be structured as a set of elements that have runtime behavior (components) and interactions (connectors)?**

The LMS system is structured into multiple runtime components, each responsible for a specific set of functionalities. These components interact through RESTful APIs provided by the backend (Spring Boot) and are connected via HTTP/HTTPS protocols.

**Major Components:**

1. **Frontend:**
   1. Web-based user interface (HTML, CSS, JavaScript frameworks).
   2. Mobile apps (if applicable).
2. **Backend Services:**
   1. Authentication Service (role-based access control).
   2. Course Management Service.
   3. Assessment and Grading Service.
   4. Notification Service (both system and email notifications).
   5. Analytics Service (for reports and performance charts).
3. **Database:**
   1. Relational database (e.g., MySQL) for user data, courses, grades, and notifications.

**Connectors:**

1. **API Layer:** RESTful endpoints facilitate communication between the frontend and backend.
2. **Database Connector:** JPA/Hibernate for ORM to interact with the database.
3. **Notification Connectors:** Email service or push notification systems for alerts.

* **What are the major executing components, and how do they interact?**

1. **User Authentication Component:**
   1. Handles login requests, verifies credentials, and generates authentication tokens (JWT).
   2. Interacts with the Role-Based Access Control module for permissions.
2. **Course Management Component:**
   1. Processes course creation, enrollment, and multimedia file uploads.
   2. Communicates with the storage system (for media files) and the database (for course metadata).
3. **Assessment and Grading Component:**
   1. Allows instructors to create quizzes, assignments, and randomize quiz questions.
   2. Handles assignment uploads and stores them in a file system or cloud storage.
   3. Sends quiz results to the Notification Service.
4. **Notification Service:**
   1. Triggers system and email notifications based on specific events like assignment grading or new course enrollment.

* **What are the major shared data stores?**

1. **Relational Database:**
   1. Stores user profiles, courses, grades, quizzes, assignments, and notifications.
2. **File Storage (Cloud/Local):**
   1. Stores uploaded files (assignments, course materials).

* **Which parts of the system are replicated?**

1. **Backend Services:**
   1. Services like Authentication, Course Management, and Notifications can be replicated across multiple nodes for load balancing and fault tolerance.
2. **Database Replication:**
   1. Read replicas of the database for handling analytics and reporting tasks, ensuring the primary database is not overwhelmed.
3. **File Storage Replication:**
   1. Files are replicated across cloud storage or distributed file systems to ensure availability.

* **How does data progress through the system?**

1. **User Login:**
   1. Credentials are validated via the Authentication Service, which interacts with the database. A JWT token is issued.
2. **Course Enrollment:**
   1. Students submit an enrollment request via the frontend. The Course Management Service updates the database and sends notifications.
3. **Assignment Submission:**
   1. Students upload files that are processed by the backend and stored in the file storage system. Metadata is saved in the database.
4. **Grading and Feedback:**
   1. Instructors evaluate submissions and input grades, triggering notifications to students.

* **What parts of the system can run in parallel?**

1. **Parallel User Requests:**
   1. Authentication, course viewing, and enrollment can be processed concurrently by the backend.
2. **Assessment Grading:**
   1. Quizzes with automated grading execute simultaneously for multiple users.
3. **File Uploads and Notifications:**
   1. File uploads and notification processing are independent and parallelized.

* **How can the system's structure change as it executes?**

1. **Dynamic Scaling**:  
   The LMS can scale dynamically to handle increased user load by adding more backend servers or increasing database replication. Cloud-based solutions (e.g., AWS Auto Scaling) allow for automatic scaling of backend services and database nodes based on user demand, ensuring the system can accommodate fluctuations in traffic without performance degradation.
2. **Fault Tolerance and Service Recovery**:  
   If a service fails, the system can automatically switch to a backup service or retry failed operations, ensuring continued availability. Faulty nodes are detected and replaced or restarted automatically using container orchestration tools (e.g., Kubernetes), maintaining the system’s resilience and minimizing downtime.
3. **Load Balancing**:  
   The system can adjust its load balancing strategies depending on the number of concurrent users. As user demand increases, the load balancer can distribute requests more efficiently across available servers, preventing any single server from becoming overwhelmed and improving system responsiveness.

3.6.2 Typical Stakeholders

**System Architects**:

* Architects use this viewpoint to evaluate the system's runtime behavior, ensuring that components interact correctly and efficiently. They focus on scalability, fault tolerance, and performance optimization.

**Developers**:

* Developers need this viewpoint to understand how various system components (backend services, authentication, course management, etc.) interact during execution. It helps them implement functionality correctly, ensuring that services are distributed across multiple nodes and are scalable.

**Operations and DevOps Teams**:

* These teams are responsible for the system's deployment, monitoring, and maintenance. They use this viewpoint to plan for system scaling, load balancing, and fault recovery. Understanding the replication and failover strategies allows them to ensure system availability and responsiveness in production environments.

**Testers (Quality Assurance)**:

* QA teams refer to this viewpoint to ensure that the system behaves correctly under different load conditions, tests the fault tolerance mechanisms, and verifies that data flows correctly across components. They focus on system performance and reliability in real-world use cases.

**Admins**:

* System administrators and managers who monitor the health of the LMS during its runtime use this viewpoint to ensure that the system's components are functioning as expected. They may also be involved in managing user requests, scaling the infrastructure, and managing notifications or alerts.

**End Users (Students, Instructors)**:

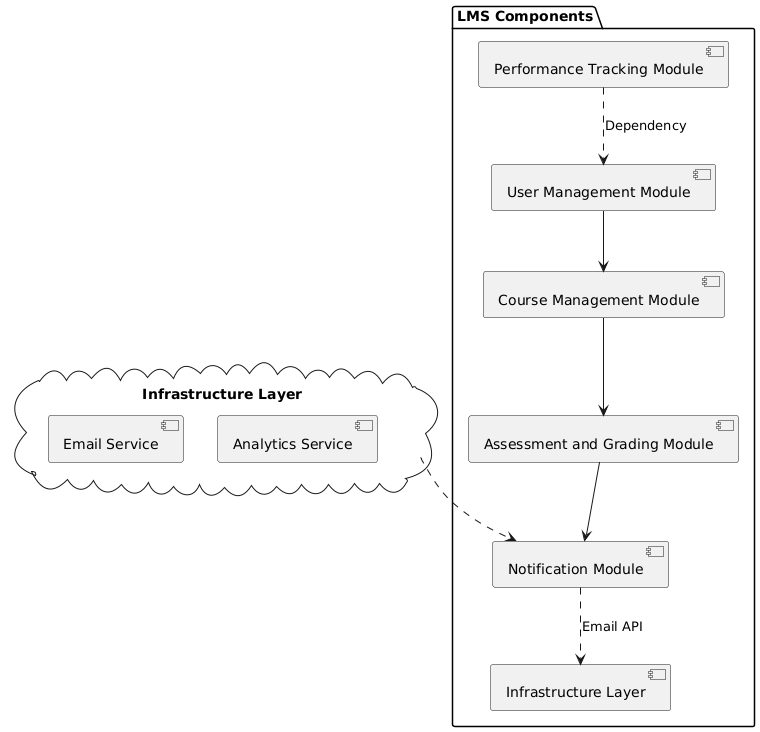
* While these stakeholders do not directly interact with the system’s runtime structure, they are the ultimate recipients of its behavior. They are affected by how efficiently the system handles user requests, whether load balancing works to prevent slowdowns, and how fault tolerance mechanisms ensure continued availability.

**Business Analysts**:

* Business analysts use this viewpoint to ensure that the system's execution aligns with business requirements, particularly in terms of handling peak loads and ensuring that the user experience remains stable even during traffic spikes or system failures.

## 3.7 Views

**3.7.1 conceptual view Model**

3.7.2 <component diagram>conventions

**Purpose**:  
The component diagram models the structural organization of the LMS backend. It shows the relationships between components and show main modules such as the **Course Management Service**, and **Notification Service**.

**Conventions**:

* **Language/Notation**: UML Component Diagram is used. It represents the system using standardized symbols:
  1. **Rectangles with the component stereotype**: Represents components.
  2. **Circles (lollipops**):for provided interfaces and half-circles (sockets) for required interfaces:
  3. **Lines and arrows**:  Visualize how components interact and depend on each other, highlighting communication paths and potential points of failure.
* **Entities**:
  1. **Components**: Modules, like Authentication Service, Course Management Service, etc.
  2. **Interfaces**: APIs or external systems with which components interact, such as REST API or Database.
  3. **Ports**: Specific points of interaction
* **Relationships**:
  1. **Dependency Arrows**: Indicates that one component relies on another.
  2. **Association (solid line):** Shows a more permanent relationship between components
* **Constraints**:
  1. Components must have **clear, single responsibilities**.
  2. Communication between components must occur through **well-defined interfaces**.
  3. Dependencies should be minimized to ensure **loose coupling**.

### 3.7.3 <Component diagram> operations (optional)

* **Visualization of System Architecture:** Component-Based Diagrams give the architecture of the system, including its dependencies, interfaces, and components, a visual representation.
* **Modularity and Reusability:** By dividing complex structures into more manageable, reusable parts, component-based diagrams encourage modularity.
* **Improved Communication:** A consistent visual language for communication between project managers, developers, architects, and testers is provided by component-based diagrams.
* **Ease of Maintenance and Evolution:** Component-Based Diagrams help in system maintenance and evolution by providing a clear documentation of system architecture.
* **Enforcement of Design Principles:** Component-Based Diagrams help enforce design principles such as encapsulation, cohesion, and loose coupling.

### 3.7.4 <Component diagram> correspondence rules

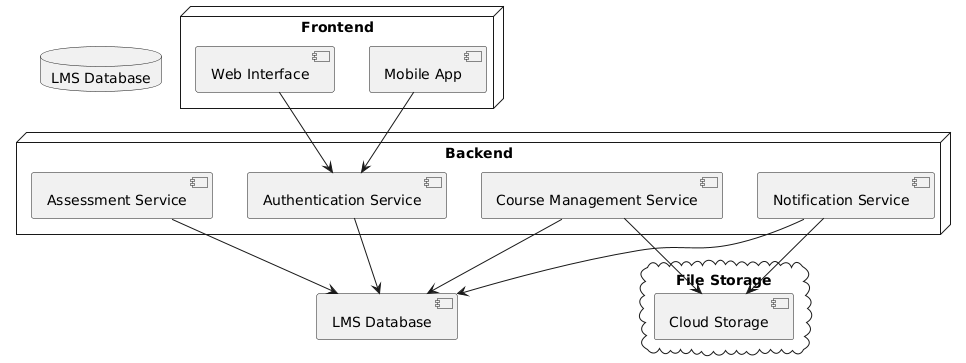
**Rule 1**: Each component in the diagram must correspond to a **logical module** in the system ( the Notification Service component corresponds to backend logic for sending emails).

**Rule 2**: Components that share data must connect through defined interfaces or ports.

**Rule 3**: All external interactions (ex: with email systems) must be represented as **dependencies** or **interfaces**.

3.8 <execution view model>

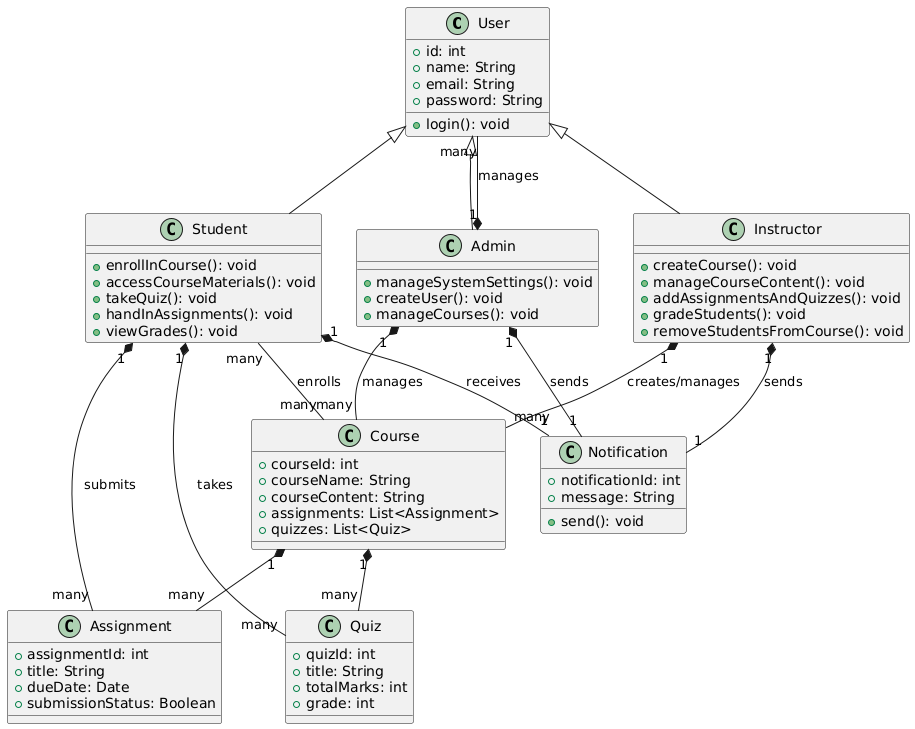
3.8.1 <deployment diagram> conventions



**Purpose:**  visualizes the architecture of systems, showcasing how software components are deployed onto hardware. It provides a clear picture of the distribution of components across various nodes, such as servers, workstations, and devices.

3.9 Additional diagrams

3.9.1<Class diagram>conventions



**Purpose**:  
The class diagram models the static structure of the LMS backend. It defines the main classes, their attributes, operations, and the relationships between them, such as inheritance, association, or dependency ,aggregation ,etc.

**Conventions**:

* **Language/Notation**: UML Class Diagram ,The key symbols include:
  1. **Rectangles divided into three sections**: Represent classes (name, attributes, methods).
  2. **Lines with arrowheads**: Represent relationships like inheritance, association, and dependency.
* **Entities**:
  1. **Classes**: Represent logical data structures, such as Student, Instructor, Course, Assignment, etc.
  2. **Attributes**: Represent the properties of the classes (e.g., Student.name, student id).
  3. **Methods**: Define the operations/classes perform (register new course)
* **Relationships**:
  1. **Inheritance**: A solid line with a triangle pointing to the parent class.
  2. **Association**: A solid line connecting two classes, possibly with multiplicity ( 1..\* for one-to-many relationships , 1..1 for one to one relationships).
  3. **Dependency**: A dotted arrow indicating that one class depends on another class
* **Constraints**:
  1. Each class must have a **clear role** in the system.(do not add unnecessary classes)
  2. Attributes and methods must align with the Single Responsibility Principle.(solid)
  3. Multiplicities must be defined for associations (one-to-many, many-to-many).(more clear associations)

### 3.9.2 <Class diagram> operations (optional)

**Operations on Class Diagrams**:

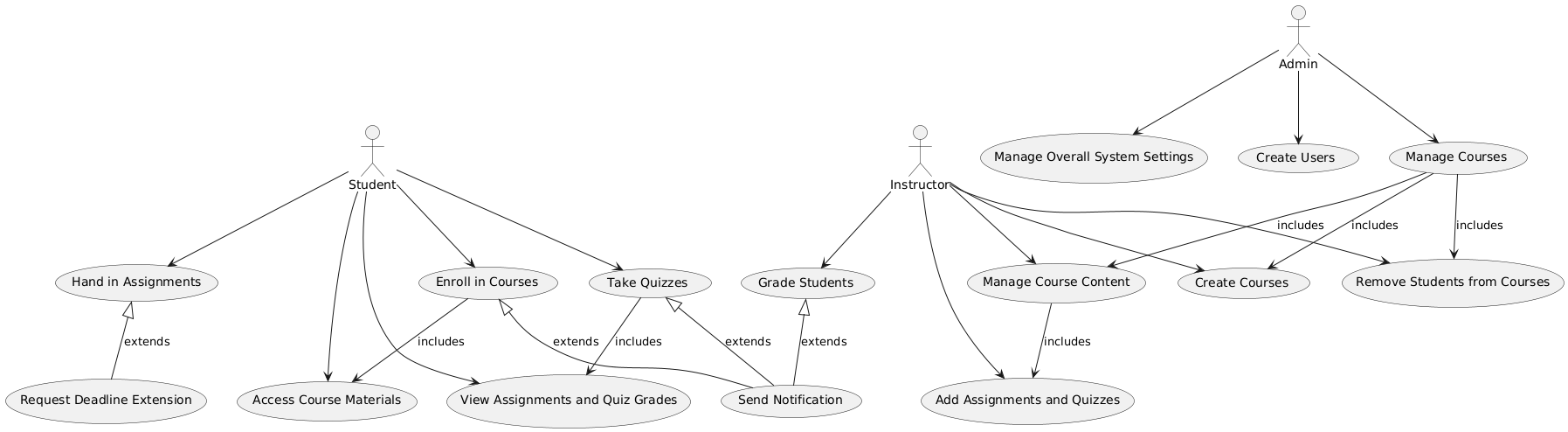
* **System Design**: Use the diagram to define class responsibilities, attributes, and methods.
* **Code Generation**: Implement code from the class diagram.
* **Refactoring**: Helps identify opportunities to refactor code into smaller, reusable components.

### 3.9.3 <Class diagram> correspondence rules

**Rule 1**: Each class must map directly to a software module or database entity.

**Rule 2**: Attributes in the diagram must correspond to fields or columns in the database (if applicable).

**Rule 3**: Relationships like associations must represent valid connections (Instructor is assigned to one or more Courses).

3.10 <Use case diagram>

3.10.1 <use case diagram> conventions

**Purpose**:  
The use case diagram models the interactions between **actors** (users or external systems) and the LMS backend. It highlights the system’s functionalities from a user perspective (functions to be implemented based on user stories)

**Conventions**:

* **Language/Notation**: UML Use Case Diagram
  1. **Ovals**: Represent use cases (e.g., Enroll in Course, Submit Assignment, Grade Assignment).
  2. **Stick Figures**: Represent actors (e.g., Student, Instructor, Administrator).
  3. **Lines**: Connect actors to their use cases.
* **Entities**:
  1. **Actors**: External entities interacting with the system (e.g., Student, Instructor, Email Service).
  2. **Use Cases**: Specific tasks or actions that the system should do.
* **Relationships**:
  1. **Association**: A straight line connecting actors to their use cases.
  2. **Extend**: A dotted line with "<<extend>>", used when one use case optionally extends another.
  3. **Include**: A dotted line with "<<include>>", used when one use case always involves another.
* **Constraints**:
  1. Each actor must have at least one association with a use case.(or he will be with no value)
  2. Each use case should represent a complete and meaningful system function.(to facilitate implementation)

### 3.10.2 <use case diagram> operations (optional)

**Operations on Use Case Diagrams**:

* **Requirement Analysis**: Identify and verify system functionality based on user needs.
* **Gap Analysis**: Check if all required functionalities are captured.(we can monitor the needed functionalities )
* **Communication**: Serve as a visual aid to explain system functionality to non-technical stakeholders (easy understanding the system functionalities)

### 3.10.3 <use case diagram> correspondence rules

**Rule 1**: Each use case must map to a system function or feature.

**Rule 2**: Actors must correspond to real-world users or systems interacting with the LMS.

**Rule 3**: All relationships must reflect logical interactions between actors and use cases (Student submits an Assignment)

## 3.7 Operations on views

Operations define the methods to be applied to views and their models. Types of operations include:

* **construction methods** are the means by which views are constructed under this viewpoint. These operations could be in the form of process guidance (how to start, what to do next); or work product guidance (templates for views of this type). Construction techniques may also be heuristic: identifying styles, patterns, or other idioms to apply in the synthesis of the view.
* **interpretation methods** which guide readers to understanding and interpreting architecture views and their models.
* **analysis methods** are used to check, reason about, transform, predict, and evaluate architectural results from this view, including operations which refer to model correspondence rules.
* **implementation methods** are the means by which to design and build systems using this view.

Another approach to categorizing operations is from Finkelstein et al. [2]. The work plan for a viewpoint defines 4 kinds of actions (on the view representations): *assembly actions* which contains the actions available to the developer to build a specification; *check actions* which contains the actions available to the developer to check the consistency of the specification; *viewpoint actions* which create new viewpoints as development proceeds; *guide actions* which provide the developer with guidance on what to do and when.

## 3.7 Correspondence rules

\* Document any correspondence rules defined by this viewpoint or its model kinds.

Usually, these rules will be across models or across views since, constraints within a model kind will have been specified as part of the conventions of that model kind. See: ISO/IEC/IEEE 42010 4.2.6 and 5.7.

## 3.8 Examples (optional)

Provide helpful examples of use of the viewpoint for the reader (architects and other stakeholders).

## 3.9 Notes (optional)

Provide any additional information that users of the viewpoint may need or find helpful.

## 3.10 Sources

\* Identify sources for this architecture viewpoint, if any, including author, history, bibliographic references, prior art, per ISO/IEC/IEEE 42010, 7e.

# 4 Views+

Much of the material in an AD is presented through its architecture views. Each view follows the conventions of its governing viewpoint. A view is made up of architecture models.

\* Include an architecture view for each viewpoint selected in §3.

Repeat and complete the following section for each architecture view in the AD.

## 4.1 View: <View Name>

\* Give the architecture view a <View Name>.

\* Provide any identifying and supplementary information about <View Name>.

The details of this information will be as specified by the organization and/or project. See §1 for examples of identifying and supplementary information.

Views have their own identifying and supplementary information distinct from ADs because they may be developed and evolve separately over the lifetime of a project.

\* Identify the viewpoint governing this view from among those identified in §3.

See also: ISO/IEC/IEEE 42010, 5.5

### 4.1.1 Models+

An architecture view is composed of one or more architecture models.

\* Provide one or more architecture models adhering to the governing viewpoint.

\* The models must address all of the concerns framed by the view’s governing viewpoint and cover the whole system from that viewpoint.

Repeat the section below for each model.

### 4.1.2 <Model Name>

\* Each architecture model shall include version identification as specified by the organization and/or project.

\* Each architecture model shall identify its governing model kind and adhere to the conventions of that model kind from §3.5.

See ISO/IEC/IEEE 42010, 5.4.

An architecture model may be a part of more than one architecture view. This enables sharing of details and addressing distinct but related concerns without redundancy. Other uses of multiple models: aspect-oriented style of architecture description: architecture models shared across architecture views can be used to express architectural perspectives [6] and architecture textures [5]. Architecture models can be used as containers for applying architecture patterns or architecture styles to express fundamental schemes (such as layers, three-tier, peer-to-peer, model-view-controller) within architecture views.

### 4.1.3 Known Issues with View

\* Document any discrepancies between the view and its viewpoint conventions.

Each architecture view must adhere to the conventions of its governing architecture viewpoint.

Known issues could include: inconsistencies, items to be completed, open or unresolved issues, exceptions and deviations from the conventions established by the viewpoint. Open issues can lead to decisions to be made. Exceptions and deviations can be documented as decision outcomes and rationale.

# 5 Consistency and correspondences

This chapter describes consistency requirements, recording of known inconsistencies in an AD, and the use and documentation of correspondences and correspondence rules.

## 5.1 Known inconsistencies

\* Record any known inconsistencies in the AD.

Although consistent ADs obviously are to be preferred, it is sometimes infeasible or impractical to resolve all inconsistencies for reasons of time, effort, or insufficient information.

∆ An architecture description should include an analysis of consistency of its architecture models and its views.

## 5.2 Correspondences in the AD

\* Identify each correspondence in the AD and its participating AD elements. Identify any correspondence rules governing

Correspondences are used to express, record, enforce and analyze consistency between models, views and other AD elements within an architecture description, between ADs, or between an AD and other forms of documentation.

AD elements include instances of stakeholders, concerns, viewpoints and views, model kinds and models, decisions and rationales. Constructs introduced by viewpoints and model kinds are also AD elements.

Correspondences are n-ary mathematical relations. Correspondences can be depicted via tables, via links, or via other forms of association (such as in UML).

## 5.3 Correspondence rules

\* Identify each correspondence rule applying to the AD.

Correspondence rules can be introduced by the AD, by one of its viewpoints, or from an architecture framework or architecture description language being used.

\* For each identified correspondence rule, record whether the rule holds (is satisfied) or otherwise record all known violations.

# A Architecture decisions and rationale

It is not required by the Standard to capture architecture decisions. This section describes recommendations (“shoulds”) for their recording.

## A.1 Decisions

∆ Provide evidence of consideration of alternatives and the rationale for the choices made.

∆ Record architecture decisions considered to be key to the architecture of <System of Interest>.

Areas to consider to selecting key decisions include those:

* affecting key stakeholders or many stakeholders
* essential to project planning and management
* expensive to enforce or implement
* highly sensitive to changes or costly to change
* involving intricate or non-obvious reasoning
* pertaining to architecturally significant requirements
* requiring major expenditures of time or effort to make
* resulting in capital expenditures or indirect costs

∆ When recording decisions, the following information items should be considered:

* unique identifier for the decision
* statement of the decision
* correspondences or linkages concerns to which it pertains
* owner of the decision
* correspondences or linkages to affected AD elements
* rationale linked to the decision
* forces and constraints on the decision
* assumptions influencing the decision
* considered alternatives and their potential consequences

See [7] and references there for various approaches to documenting decisions compatible with the Standard.

The template ends here!

# Bibliography

[1] Paul C. Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. Documenting Software Architectures: views and beyond. Addison Wesley, 2nd edition, 2010.

[2] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in system development. International Journal of Software Engineering and Knowledge Engineering, 2(1):31–57, March 1992.

[3] IEEE Std 1471, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, October 2000.

[4] ISO/IEC/IEEE 42010, Systems and software engineering — Architecture description, December 2011.

[5] Alexander Ran. Ares conceptual framework for software architecture. In M. Jazayeri, A. Ran, and F. van der Linden, editors, Software Architecture for Product Families Principles and Practice, pages 1–29. Addison-Wesley, 2000.

[6] Nick Rozanski and Eo ́in Woods. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison Wesley, 2nd edition, 2011.

[7] Uwe van Heesch, Paris Avgeriou, and Rich Hilliard. A documentation framework for architecture decisions. The Journal of Systems & Software, 85(4):795–820, April 2012.