

**SDA PROJECT Report**

**SRS and Design Document**



**May 23, 2025**

**TALha naveed, Abdul wahab**

**01-131232-089, 01-131232-007**

****

**Software Requirements Specification (SRS)**

**Student Management System (SMS)**

**1. Introduction**

**1.1 Purpose**

This SRS document specifies the functional and non-functional requirements for the Student Management System. The system is designed to facilitate the management of academic information for students, teachers, and administrators within an academic institution.

**1.2 Scope**

The SMS provides:

* Student/teacher/admin role-based dashboards
* Academic records management (courses, grades, timetables)
* Resource and infrastructure management (rooms, floors, batches)
* Administrative functions (profile management, credit hours, GPA filters)
* Utilities (notes uploads, transcript and GPA calculation)

**1.3 Intended Audience**

* Project stakeholders
* Development team
* Quality assurance team
* End users (students, teachers, admins)

**2. Overall Description**

**2.1 User Classes & Characteristics**

* **Student:** Registers for courses, views grades, timetable, uploads/downloads notes, generates transcript, calculates GPA, edits profile.
* **Teacher:** Views assigned courses, marks/updates grades, manages timetable, edits profile.
* **Admin:** Manages users, academic structures (courses, batches, rooms, etc.), views analytics (batchwise GPA, credit hours), applies advanced filters, and manages system-wide settings.

**2.2 System Interfaces**

* Desktop GUI (JavaFX-based)
* Centralized SQL database

**2.3 Constraints**

* Only authenticated and authorized users can access respective features.
* Data stored centrally, accessible only via the official application.
* UI design supports easy modification (separation of concerns, modularization).

**3. Functional Requirements**

**🟦 EPIC: Student Portal**

**Story 1: Course Registration**

* **As a student**, I want to register for available courses in the current semester, so that I can enroll in my subjects.
* **Acceptance Criteria:**
  + Registration is allowed only for open courses in the active semester.
  + Prevents duplicate registration.
  + Only “In Progress” courses can be dropped.

**Story 2: View/Download Transcript**

* **As a student**, I want to view and download my academic transcript, so that I can keep a record.
* **Acceptance Criteria:**
  + Transcript is generated in PDF within 5 seconds.
  + Only completed courses/grades are shown.

**Story 3: GPA Calculator**

* **As a student**, I want to calculate my GPA using course grades and credit hours, so I can monitor my academic progress.
* **Acceptance Criteria:**
  + Supports multiple courses/credits.
  + Works offline; no authentication required.

**Story 4: Notes Manager**

* **As a student**, I want to upload, download, and delete my study notes (PDF/DOCX), so I can access them anywhere.
* **Acceptance Criteria:**
  + Files stored and retrievable from the database.
  + Only the owner can delete/download their notes.

**Story 5: Profile Management**

* **As a student**, I want to view and update my personal information (name, email, phone, password), so my records remain current.
* **Acceptance Criteria:**
  + Editable fields: Name, Email, Phone, Password.

**🟪 EPIC: Teacher Portal**

**Story 1: View Assigned Courses**

* **As a teacher**, I want to see all the courses assigned to me, so I can manage my workload.
* **Acceptance Criteria:**
  + List shows Course Code, Name, Section, Semester.

**Story 2: Mark/Update Student Grades**

* **As a teacher**, I want to enter or update grades for my students in my assigned courses, so their performance is recorded.
* **Acceptance Criteria:**
  + Grades can be updated only for assigned courses.
  + Supports letter grades (A, B+, etc.).
  + Once grades are entered, course status is set to “Completed”.

**Story 3: Timetable Management**

* **As a teacher**, I want to view my weekly timetable, so I can plan my schedule.
* **Acceptance Criteria:**
  + Shows all assigned periods with room/floor info.

**Story 4: Profile Management**

* **As a teacher**, I want to update my personal information (name, email, phone, password), so my records remain current.

**🟫 EPIC: Admin Dashboard**

**Story 1: Manage Students, Teachers, Courses, etc.**

* **As an admin**, I want to create, read, update, and delete (CRUD) students, teachers, batches, courses, rooms, floors, departments, and sections, so I can keep institutional records accurate.
* **Acceptance Criteria:**
  + All CRUD operations available per entity.
  + List/search/filter for each entity.

**Story 2: Manage Timetable**

* **As an admin**, I want to auto-generate, add, edit, and delete timetable entries, so that classes are scheduled efficiently.
* **Acceptance Criteria:**
  + Conflict detection (room, teacher, section cannot overlap).
  + PDF download of current timetable.

**Story 3: Batchwise Analytics & Filters**

* **As an admin**, I want to filter students by batch, GPA ranges, completed credit hours, or registration status, so I can monitor academic health and intervene early.
* **Acceptance Criteria:**
  + Filter students with GPA below 2.0, or by GPA ranges.
  + Filter by completed credit hours.
  + List unregistered students by batch.

**Story 4: Profile Management**

* **As an admin**, I want to update my personal information (name, email, phone, password), so my records remain current.

**4. Non-Functional Requirements (Specific to User Stories)**

* **Performance:** Transcript must be generated within 5 seconds; registration within 3 seconds.
* **Reliability:** Timetable generator must avoid scheduling conflicts.
* **Usability:** All list/search/filter operations must update views instantly (<1s).
* **Data Integrity:** Only valid grades and course registrations are accepted.
* **File Handling:** Uploaded notes limited to 10MB per file; only PDF/DOCX allowed.
* **Security:** Only authorized users can CRUD respective entities; file access limited to owner.

**5. Acceptance Criteria (Examples for Key Features)**

**Course Registration**

* User can only register for courses not already enrolled.
* User cannot register for closed/inactive courses.
* Registration is confirmed with success message.

**Grade Entry**

* Only letter grades accepted (A, A-, B+, ... F).
* Cannot mark grades for unassigned courses/sections.
* Upon saving grades, course status updates to “Completed”.

**Notes Manager**

* Only logged-in student can upload/download/delete their notes.
* Notes must be retrievable by the owner at any time.

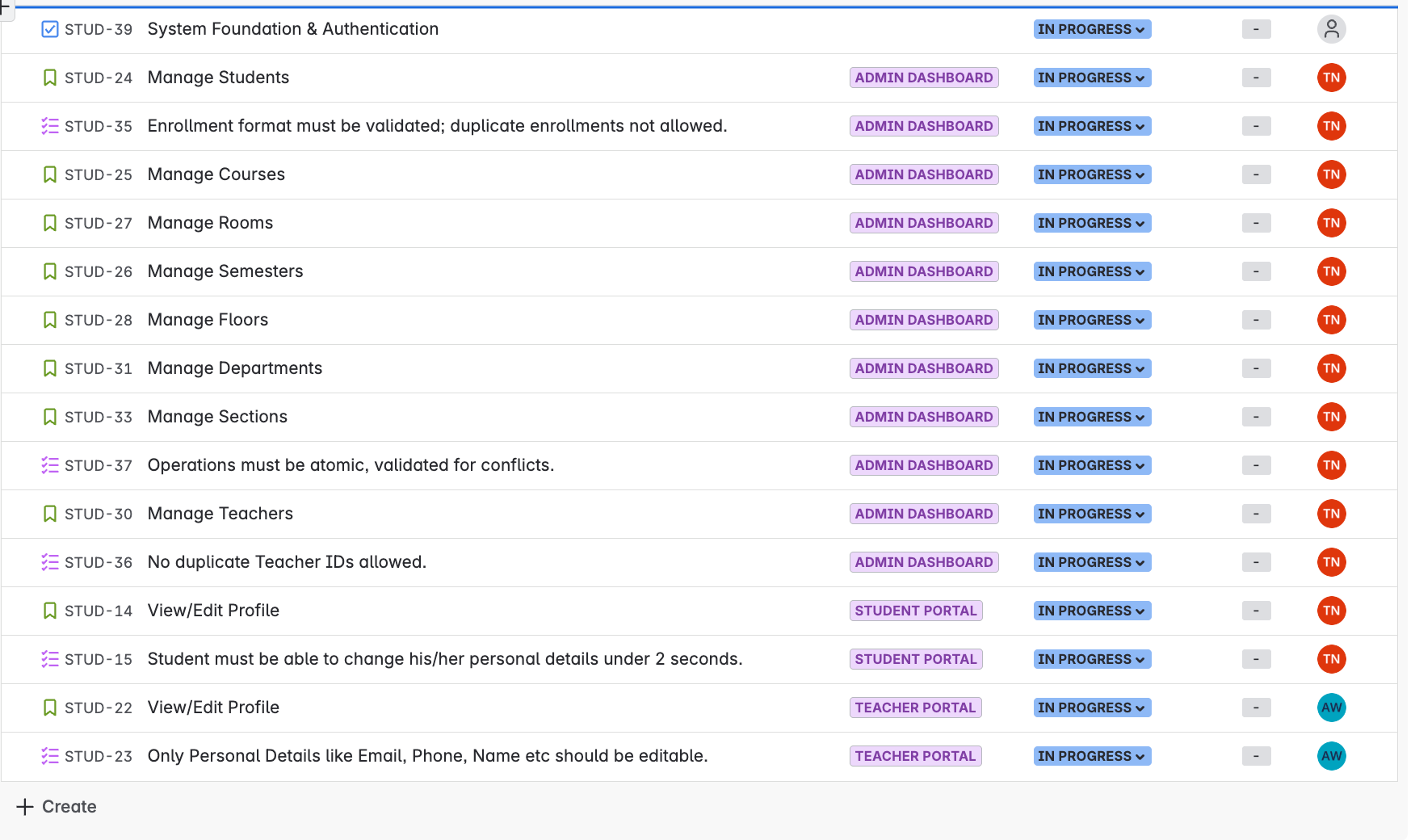
**Timetable Generation**

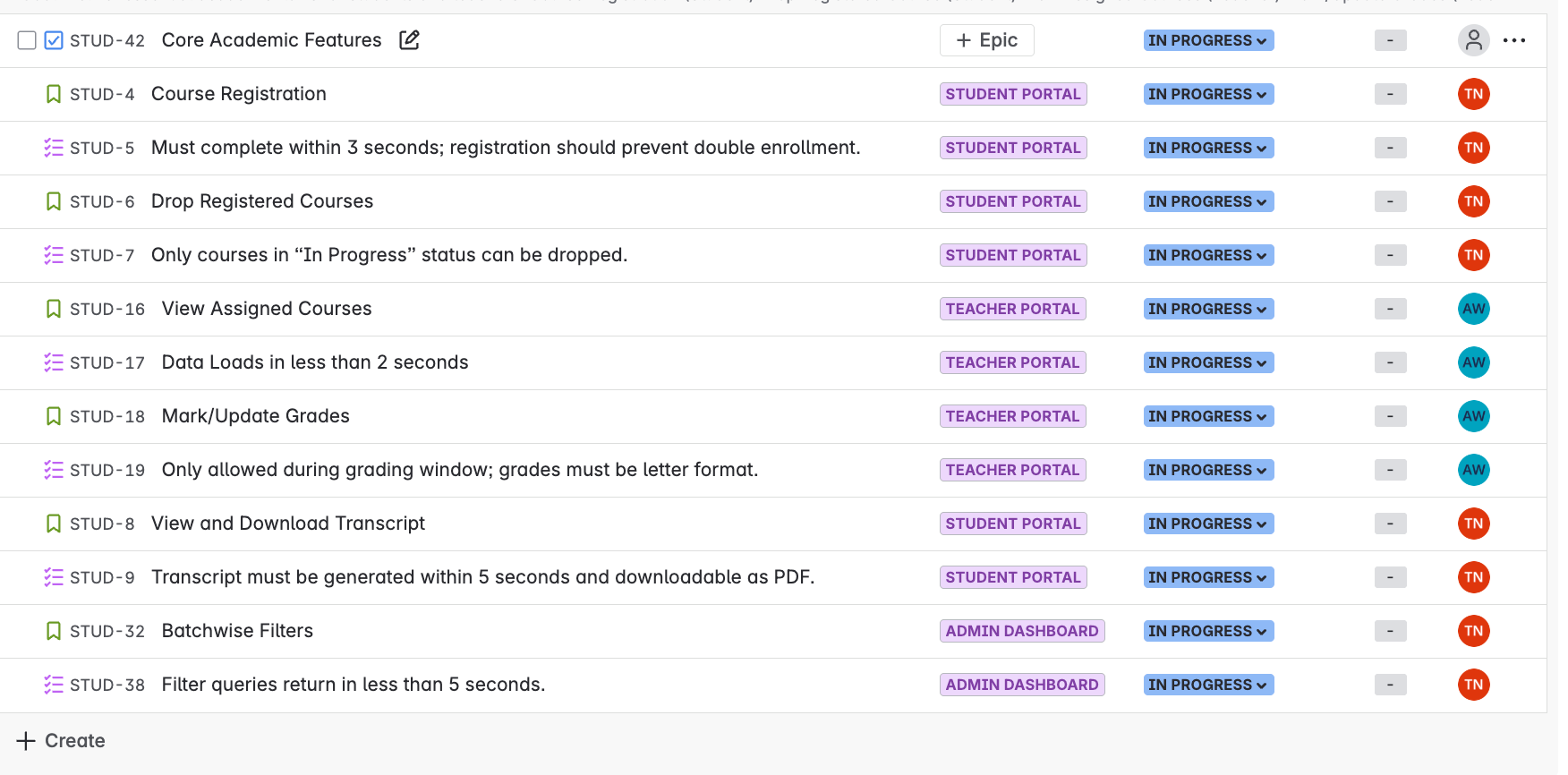
* No teacher, section, or room may have overlapping sessions.
* All periods must have correct course/teacher/room assignments.
* PDF export reflects current timetable accurately.

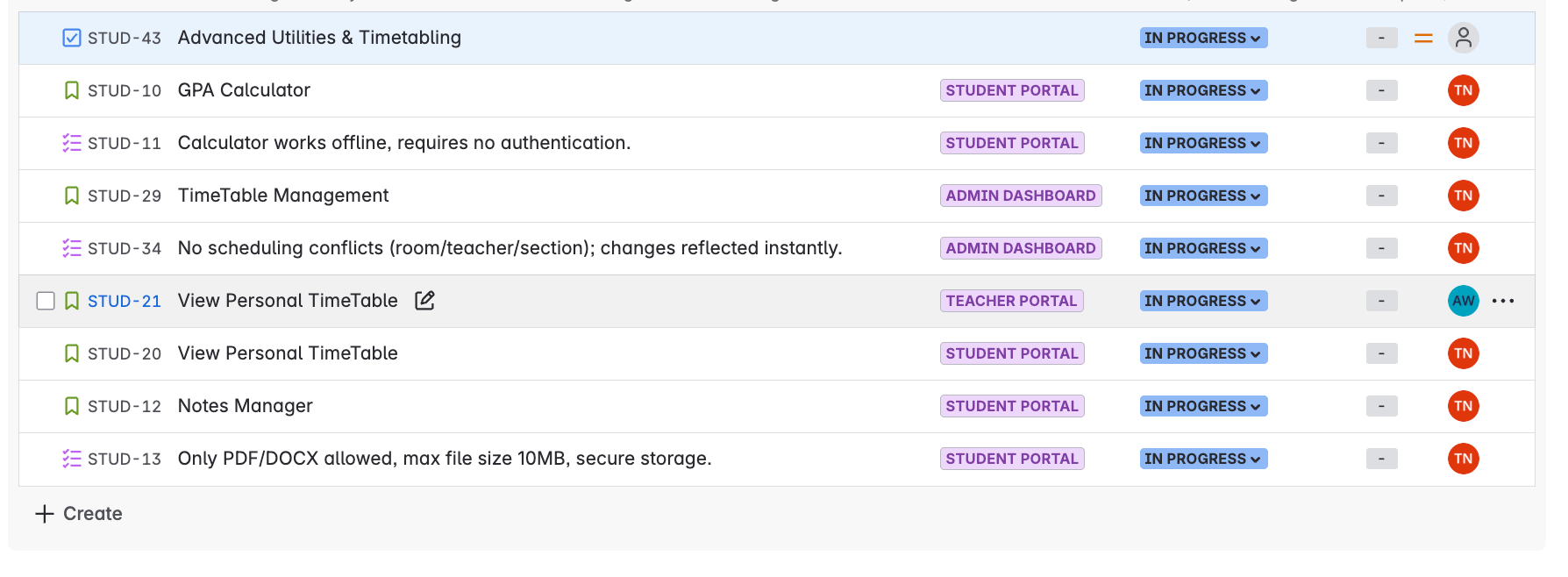
**6. Assumptions**

* Enrollment number format is enforced as 01-131232-XXX.
* Each student belongs to a batch, section, and department.
* Central database is accessible only through this application.

**7. JIRA Screenshots**







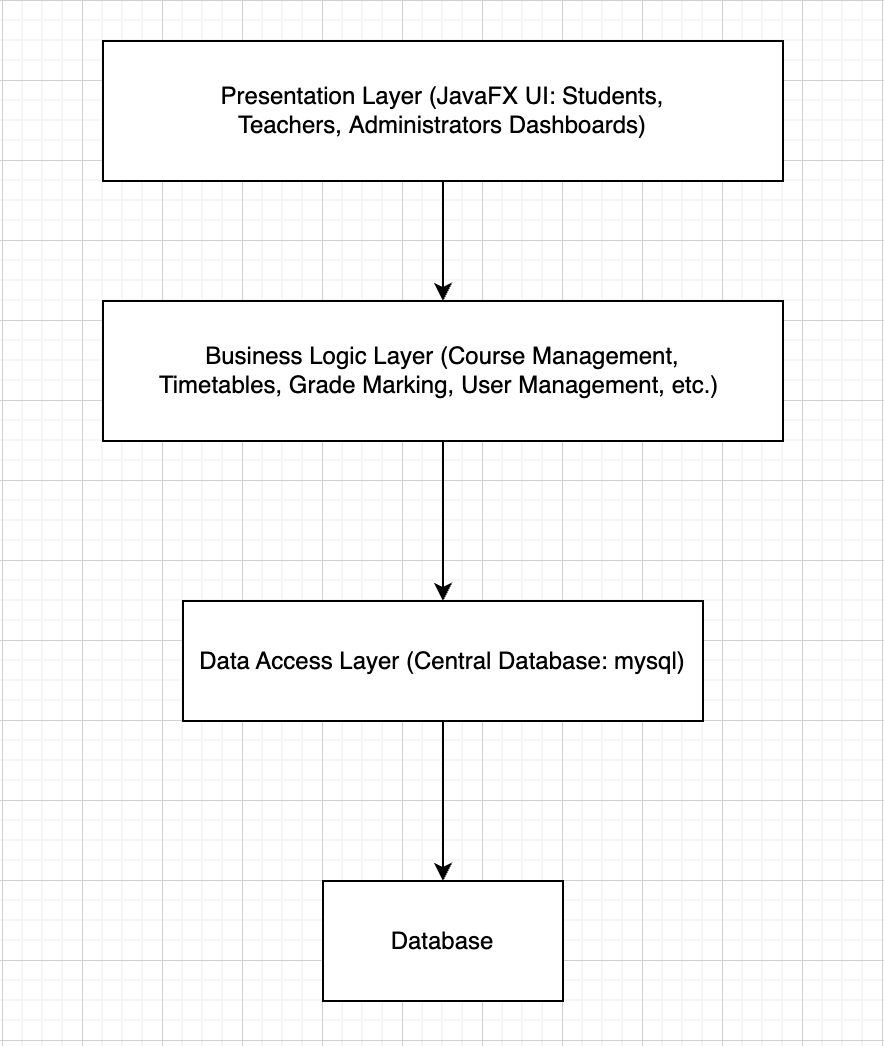
## ****System Architecture and Application of Design Patterns****

### ****Architectural Style: Three-Tier (Layered) Architecture****

The architectural pattern chosen for the **Student Management System** is a **Three-Tier (Layered) Architecture**. This pattern clearly divides the system into three distinct layers:

1. **Presentation Layer**
2. **Business Logic Layer**
3. **Data Access Layer**

This architectural approach supports modularization, separation of concerns, scalability, maintainability, and flexibility in modifying the UI and business logic independently.



**Justification for Choosing Three-Tier Architecture:**

The Three-Tier Architecture aligns perfectly with the requirements and design constraints specified in your project:

1. **Separation of Concerns:**
   * It separates UI logic, business rules, and data management clearly, enhancing readability, maintainability, and scalability.
   * Allows independent development and testing of each tier.
2. **Modularity & Maintainability:**
   * Easier to identify, isolate, and fix bugs since each layer handles distinct functionalities.
   * Supports incremental development and easy future modifications without impacting other layers.
3. **Flexibility in UI and Business Logic Changes:**
   * User Interface changes do not affect business logic or database design.
   * Business logic changes (e.g., new rules for grading or registration) do not require extensive UI redesign.
4. **Security & Data Integrity:**
   * Direct database access from the presentation layer is prohibited, ensuring data integrity and secure transactions.
   * Implements role-based access control effectively.

**Applied Design Patterns**

Below are the chosen design patterns that meet the specific design constraints and enhance system architecture:

**1. MVC (Model-View-Controller) Pattern**

**Applied at:** Presentation and Business Logic Layers.

**Reason:**

* Clean separation between User Interface (View), Application Logic (Controller), and Data (Model).
* Meets the constraint of allowing UI modifications without affecting data or business logic.
* Enhances testability and maintainability.

**Example in SMS:**

* JavaFX interfaces act as **View**.
* Controllers (AssignedCoursesController, ManageStudentsController, etc.) act as **Controller**.
* Database entities (Student, Course, etc.) act as **Model**.

**2. Data Access Object (DAO) Pattern**

**Applied at:** Data Access Layer.

**Reason:**

* Encapsulates all database interaction logic.
* Decouples data access logic from business logic.
* Ensures easy modification to database without affecting business logic.

**Example in SMS:**

* Separate Java classes (like DBConnection, StudentDAO) handle database operations, isolating database logic from business logic.

**3. Singleton Pattern**

**Applied at:** DB Connection Management.

**Reason:**

* Ensures a single, consistent database connection instance throughout the application.
* Efficient resource utilization and connection management.

**Example in SMS:**

* The DBConnection class uses Singleton pattern, providing a single reusable connection.

**4. Facade Pattern**

**Applied at:** Business Logic Layer.

**Reason:**

* Simplifies complex subsystem interactions into simpler interfaces for higher-level components.
* Meets constraints of easily modifiable system components without major redesign.

**Example in SMS:**

* Simple interfaces like CourseRegistrationFacade, GradesManagementFacade simplify interactions for controllers and UI, hiding complexity behind single calls.

| **Constraint** | **How Met** |
| --- | --- |
| Authenticated user access only | Presentation & Business Layers enforce security |
| Authorized operations only | MVC & DAO Patterns |
| UI modification independent from data/business logic | MVC Pattern |
| Central data repository accessed via application only | Three-Tier & DAO Patterns |
| Easy modification of business logic/UI without extensive redesign | MVC, DAO, and Facade Patterns |

1. **Detailed Design [20] [CLO2]**

Produce detailed system design specification in the form of the following diagrams:

1. Use Case Diagram showing important system features and actors interacting with the system. [2]
2. UI Design of your application [2]
3. Component Diagram presenting all important system components [2]
4. Deployment Diagram presenting all important system components and their deployment in various computing nodes [2]
5. Presentation of static view using Class Diagram [3]
6. Data Model using appropriate modeling notation [2]
7. Presentation of Dynamic View using
   1. Sequence Diagram for all important use cases [3]
   2. Activity Diagram for all important processes [2]
   3. State transition diagram of the most important aspect (object, system, or subsystem) [2]

**Design Constraints to be Considered:**

Your project shall meet the following design constraints.

1. Application shall only allow access to authenticated users only.
2. Authenticated users shall only be able to perform authorized operations on the system.
3. Your system design shall support easy modification to the User interface during design and development phase by applying the principles of Separation of Concerns & Modularization. For this you will need to apply appropriate architectural patterns. Justify your choice.
4. All data needs to be stored in a central repository which can only be accessed through an appropriate application (Web, desktop, mobile etc.) to access and modify data. If modification to data or the way data is handled (business logic) needs to be modified, it shall have no effect on the User Interface (please refer to Constraint 3 discussed earlier).
5. The system shall be easily modifiable if some variation in the existing features has to be implemented without a major redesign.

**Evaluation Criteria:**

Evaluation criteria is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A(85 & above)** | **B(75-85)** | **C(60-74)** | **D(50-59)** | **F( less than 50)** |
| * Comprehensive & Correct Requirements Specification * Correct Design Specification meeting all required & desired functional & Non-functional requirements. * OOD concepts are correctly implemented. * Correct architectural patterns/styles are applied and justification is given. | * Requirements specification meets essential requirements but has fewer mistakes and missing functional & non-functional requirements. * Design specification has some mistakes in UML models, system architecture but fundamental concepts have been correctly implemented. * No justification for architectural/design decision is given. * OOD concepts have been applied but correctness is not consistent. | * Requirements specification is incomplete, NFRs are not measurable and vague. * Design specification has mistakes with various views of UML are either missing or are not mapped with each other. * Wrong architectural & design pattern applied. * Fundamental UML & OOD concepts are applied incorrectly most of the times. | * Requirement specification document is incomplete and demonstrate poor understanding. * Distinction in Functional & non-functional requirements is made but lack consistency and correctness. * UML design is incomplete. * incorrect with few major mistakes in fundamental concept implementation. * OOD concepts have not been applied correctly. | * -Incomplete and wrong specification both at functional and non-functional level. * UML diagrams are poorly drawn with no demonstration of usage of the correct diagrammatic notations and concepts. * OOD concepts are incorrectly applied demonstrating poor understanding. |
| * Engaging, Well focused, presentation | * Presentation covers all aspects but lacks focus and able to finish hardly in time. * Lack bit of focus and communication skills are somewhat weak. | * Presentation is missing the required elements. * Communication skills are average. * Presentation is cluttered with too much of information, diagrams are not readable | * Missing information * Poor time management * Not engaging * Presentation/slide show has problems of content, readability | * Not presented * Missing information * Poor time management |
| * Developed application meets the specifications. * All design diagrams can be mapped correctly to the code. * Design patterns are applied correctly. * Project progress is depicted on Jira regularly. | * Developed application meets the specifications to considerable extent. * UML diagrams can be mapped to implementation artifacts most of the time with moderate level of violations. * Some of the Design patterns have been implemented to meet the constraints * Project progress is depicted on Jira. | * Developed application meets the specifications to considerable extent. * Some mapping of the UML diagrams to the code can be found. * Very few of the design patterns have been implemented * Project progress is depicted on Jira | * Developed application hardly meets the specifications. * Very little mapping of the UML diagrams to the code can be found. * No design patterns is implemented * The continuous project is depicted on Jira. * Project progress is depicted on Jira irregularly | * No working application. * No link of UML design with the implementation. * No pattern is applied. * Jira is not updated /all activities done at the end. |
| * Answers most of the questions correctly all the time demonstrating the knowledge about:   + Design patterns   + Architectural patterns   + Quality tactics   + UML & OOD * Can strongly & correctly justify about the design choices made. | * Answers most of the questions correctly most of the time demonstrating the knowledge about:   + Design patterns   + Architectural patterns   + Quality tactics   + UML & OOD * Can justify about the design choices made to some extent with some confidence. | * Answers most of the questions correctly most of the time demonstrating the knowledge about:   + Design patterns   + Architectural patterns   + Quality tactics   + UML & OOD * Can justify about the design choices made to some extent with little confidence. | * Answers some of the questions correctly a shallow knowledge about:   + Design patterns   + Architectural patterns   + Quality tactics   + UML & OOD * Can not justify about the design choices made to some extent. * Developed application hardly meets the specifications | * Unable to answer the fundamental questions demonstrating very weak or knowledge about:   + Design patterns   + Architectural patterns   + Quality tactics   + UML & OOD * Can not justify about the design choices made to some extent. |