IIIT DELHI ERP Project Report and Technical Documentation

1. Executive Summary

This report serves as the final documentation for the AP-Project IIIT DELHI ERP desktop application, developed using Java and Swing. The system successfully implements three distinct user roles (**Admin, Instructor, Student**) and enforces strict **Role-Based Access Control (RBAC)**. Key features include secure authentication using a dual-database architecture, dynamic course management, student enrollment flows, and a crucial **Maintenance Mode** toggle. This document details the architectural approach, key features, access enforcement mechanisms, and the final grade calculation rule implemented within the application.

2. Project Overview

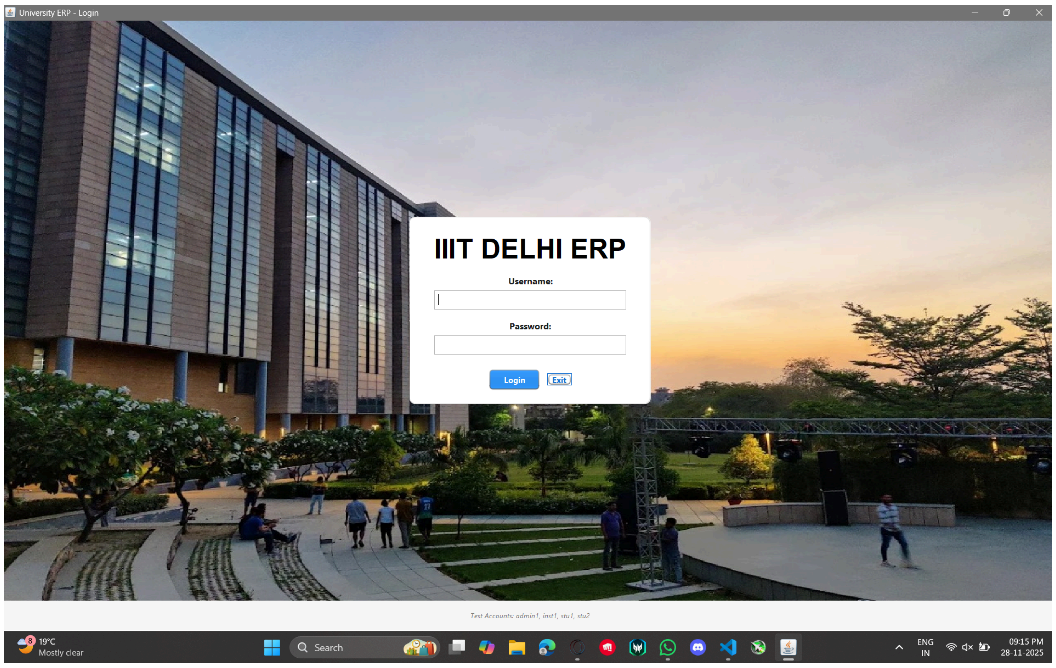
The system is a simplified **Enterprise Resource Planning (ERP)** desktop application designed to manage core academic processes.

- **Purpose:** To provide a reliable, role-specific interface for managing users, courses, sections, enrollments, and grades within a small academic environment.
- **Technology Stack:**
    - **Frontend:** Java Swing (Desktop UI).
    - **Backend:** Java 17+ core services.
    - **Persistence:** PostgreSQL Database (via JDBC driver: `postgresql-*.jar`).
    - **Configuration:** Simple file-based settings via `config/app.properties` for database connectivity.
- **Main Modules:** Authentication (`Auth`), Administration (`Admin`), Academic Management (`Instructor`, `Student`), System Settings (`Maintenance`), and Data Access Layer (`DAL`).

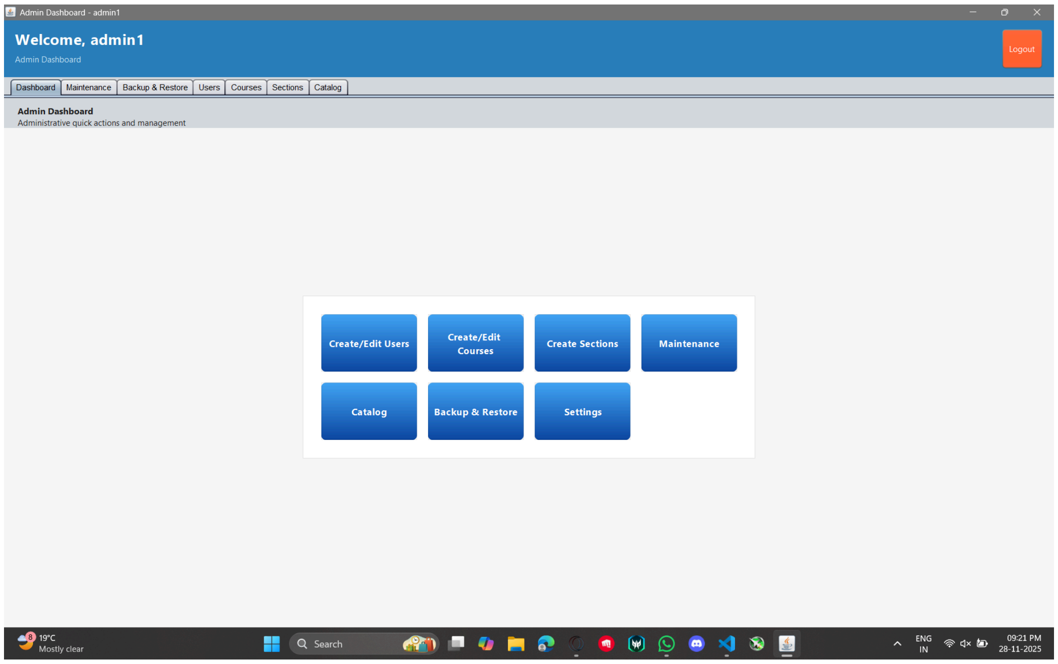
3. Screenshot Placeholders

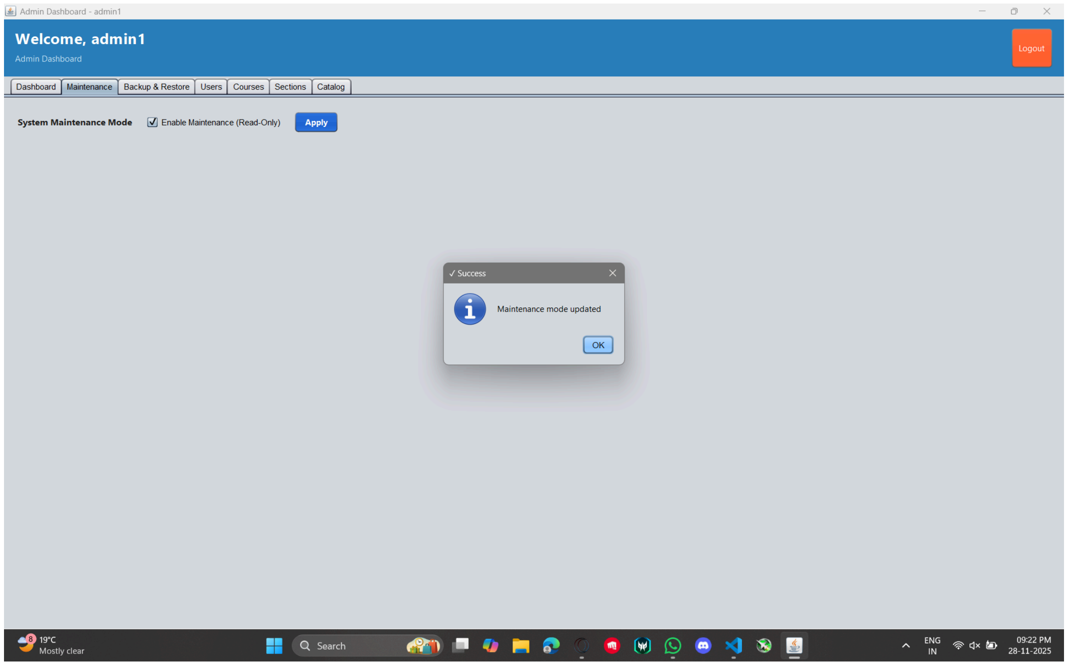
The following placeholders indicate where visual evidence of the running application and development environment should be placed in the final report.

- **Login Screen** :



- **Admin Dashboard (Users / Maintenance)**




Admin Dashboard - admin1

Welcome, admin1

Admin Dashboard

Logout

Dashboard | Maintenance | Backup & Restore | Users | Courses | Sections | Catalog

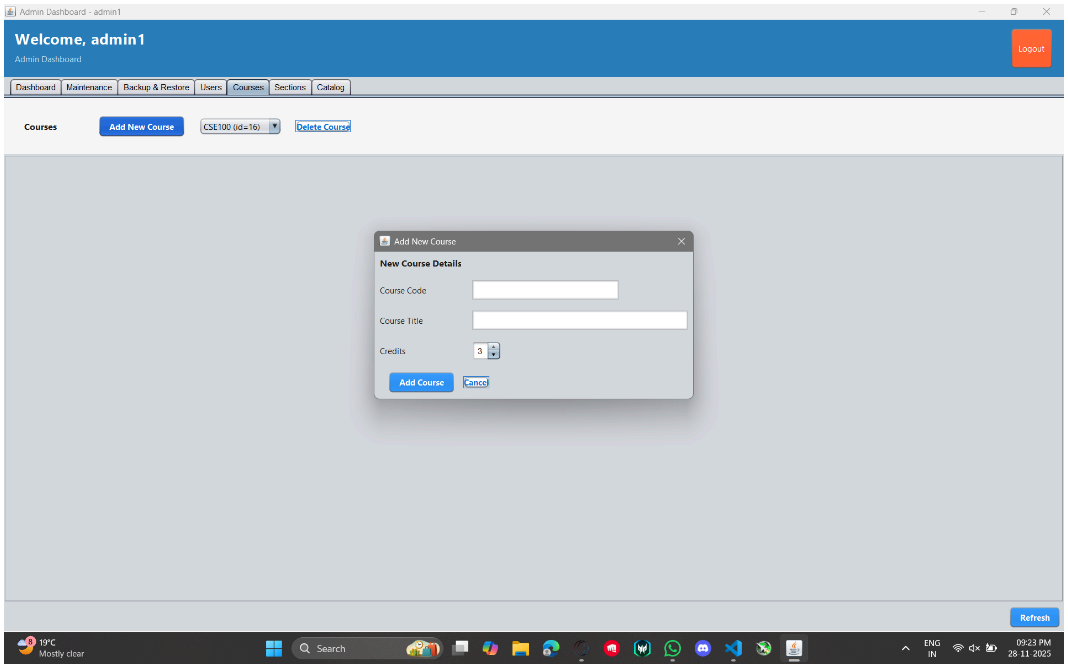
System Maintenance Mode [✓ Enable Maintenance (Read-Only)] Apply

✓ Success

Maintenance mode updated

OK

19°C Mostly clear · Search · ENG IN · 09:22 PM 28-11-2025


Admin Dashboard - admin1

Welcome, admin1

Admin Dashboard

Logout

Dashboard | Maintenance | Backup & Restore | Users | Courses | Sections | Catalog

Courses Add New Course CSE100 (id=16) Delete Course

Add New Course

New Course Details

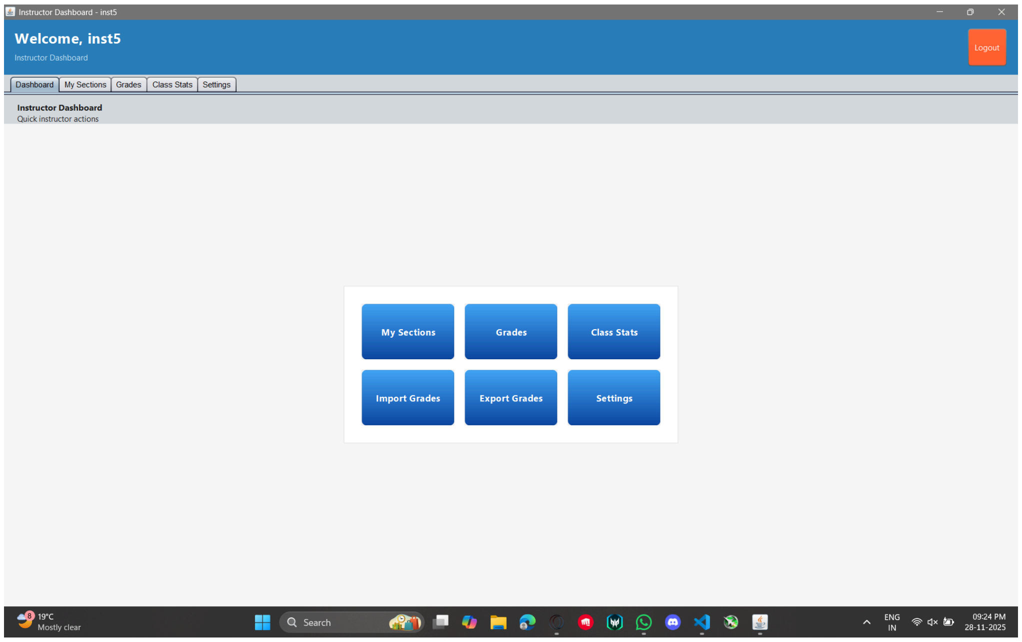
Course Code	
Course Title	
Credits	3

Add Course Cancel

Refresh

19°C Mostly clear · Search · ENG IN · 09:23 PM 28-11-2025

● **Instructor Dashboard — Set Weights / Enter Grades**



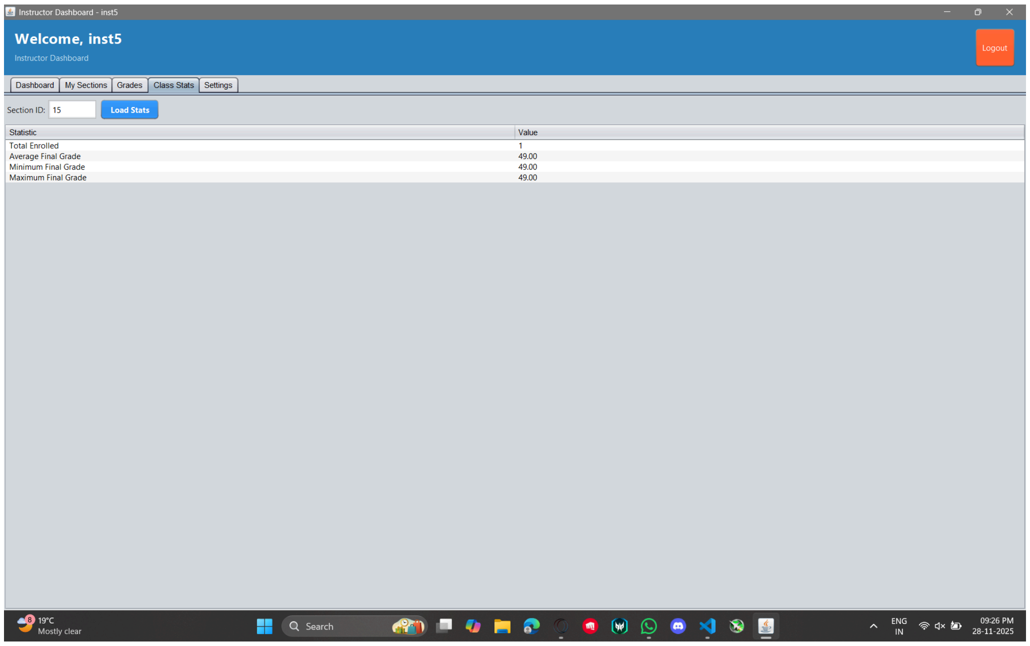
Instructor Dashboard - inst5

Welcome, inst5

Instructor Dashboard

Instructor Dashboard

Quick instructor actions



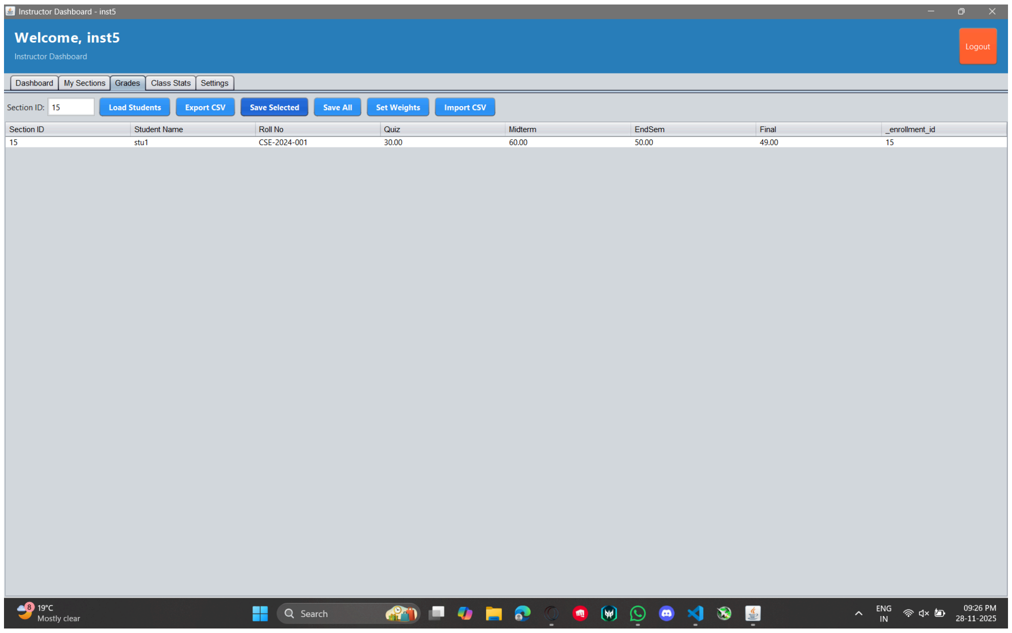
Instructor Dashboard - inst5

Welcome, inst5

Instructor Dashboard

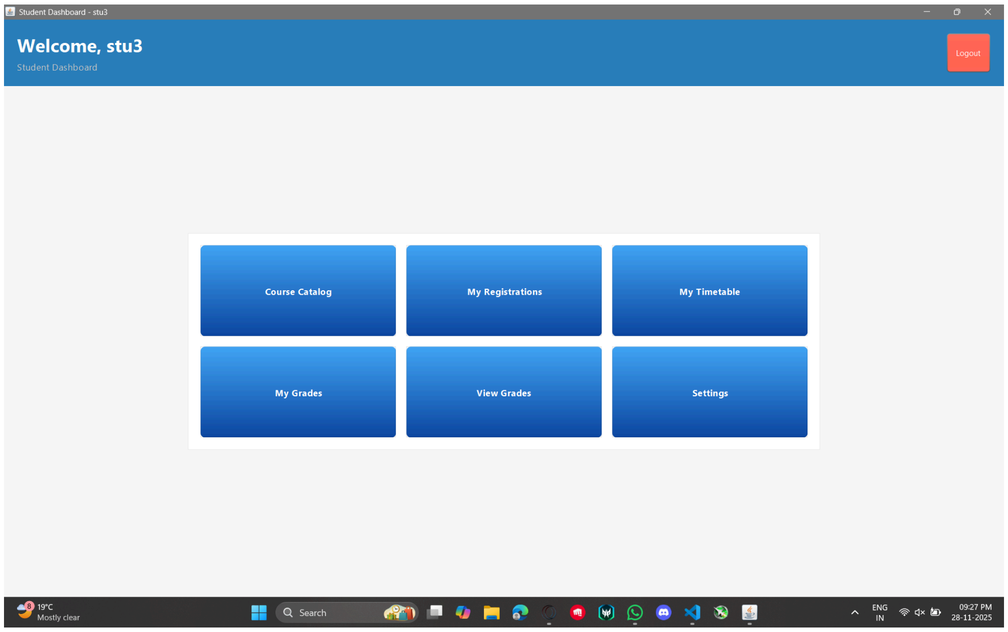
Section ID: 15

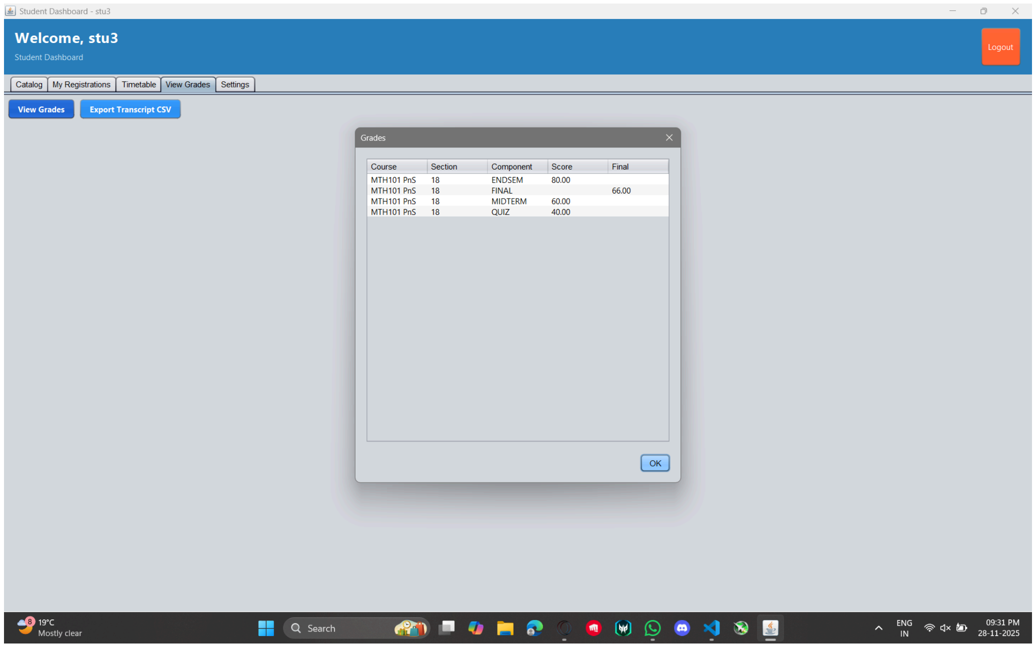
Statistic	Value
Total Enrolled	1
Average Final Grade	49.00
Minimum Final Grade	49.00
Maximum Final Grade	49.00



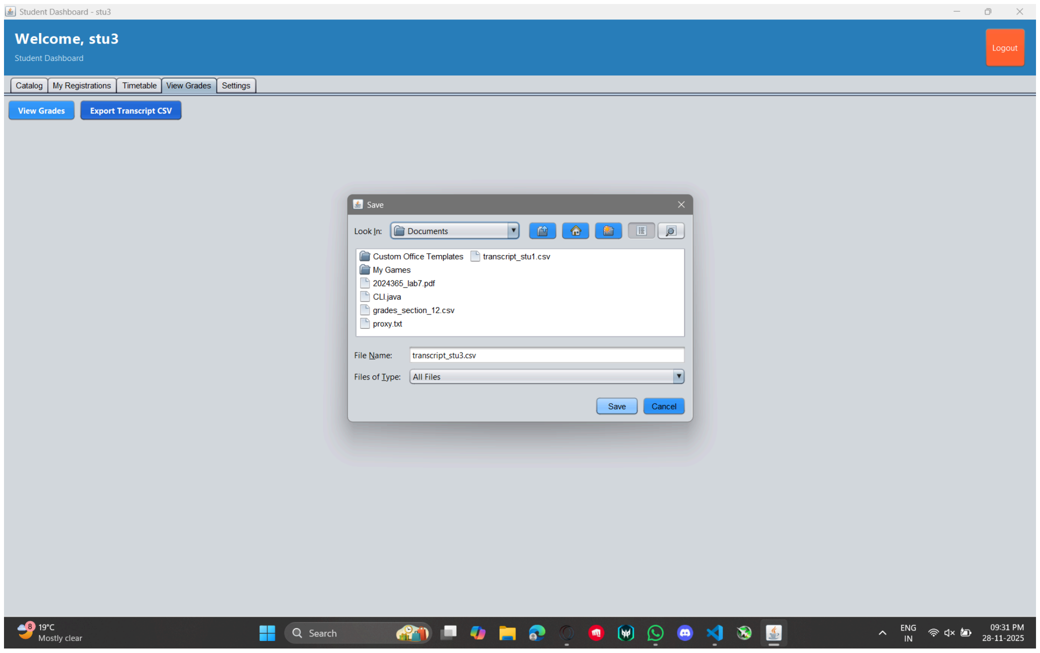
Section ID	Student Name	Roll No	Quiz	Midterm	EndSem	Final	_enrollment_id
15	stu1	CSE-2024-001	30.00	60.00	50.00	49.00	15

- **Student — View Grades / Transcript**





Course	Section	Component	Score	Final
MTH101 PnS	18	ENDSEM	80.00	
MTH101 PnS	18	FINAL		66.00
MTH101 PnS	18	MIDTERM	60.00	
MTH101 PnS	18	QUIZ	40.00	



Look In: Documents

Custom Office Templates
My Games
2024365_lab7.pdf
CLI.java
grades_section_12.csv
proxy.txt
transcript_stu1.csv

File Name: transcript_stu3.csv

Files of Type: All Files

4. Final-Grade Weighting Rule Implementation

The application is responsible for computing a final grade for each student enrolled in a section. This is achieved by combining normalized component scores (typically **QUIZ**, **MIDTERM**, **ENDSEM**) based on weights defined by the instructor for that specific section.

Calculation Formula:

Final Grade=(wq×Quiz Score)+(wm×Midterm Score)+(we×EndSem Score)

Where wq,wm,we are the weights (decimals summing to 1.0) for Quiz, Midterm, and EndSem, respectively.

Instructor Workflow for Weighting:

1. The Instructor accesses the **Grades** tab for an assigned section.
2. They click **Set Weights** and enter three percentages (e.g., 20, 30, 50). The UI validates that these values sum to 100%.
3. The system stores these weights in the `settings` table using a unique key format: `weights_section_<sectionId>`.
4. Upon saving weights, the server automatically triggers a recalculation, updating the stored `FINAL` grade component for all existing enrollments in that section.

Example Configuration & Computation:

- **Weights:** Quiz (20%), Midterm (30%), EndSem (50%).
- **Scores:** Quiz = 90, Midterm = 80, EndSem = 90.
- **Calculation:** (0.20×90)+(0.30×80)+(0.50×90)=18+24+45=**87.00**

Notes:

- All stored final grades are rounded to two decimal places.
- The `InstructorService` handles a **fallback mechanism**: if no specific weights are found for a section, it uses a predefined default to avoid calculation errors.

5. Role-Based Access Control (RBAC) & Maintenance Enforcement

5.1. Role Enforcement

- **Role Model:** The `users_auth` table in the Authentication DB is the single source of truth for user roles (**ADMIN, INSTRUCTOR, STUDENT**).

- **Enforcement Layer:** The application uses a dedicated utility class, `edu.univ.erp.access.AccessControl`, to gate all state-changing or privileged methods in the Service layer.
- **Flow:** When a user initiates an action (e.g., enrolling), the service layer retrieves the user's `Session` (containing their role) and checks it against the required permissions via `AccessControl.checkPermission(Session, Action)`. Unauthorized attempts result in an immediate exception or a clear permission error displayed to the user.

5.2. Maintenance Mode

- **State Storage:** The maintenance status is centrally managed and persisted in the **ERP DB's** `settings` table under the key `maintenance_on` (`'true'` or `'false'`).
- **Toggle:** Only **Admin** users can modify this setting via the dedicated Admin Dashboard control.
- **Impact:** When the mode is **ON**, the `MaintenanceService.isMaintenanceOn()` flag is checked by the `AccessControl` layer. All write/change operations (e.g., Student registration, Instructor grade entry) for **non-Admin roles** are blocked, restricting them to **read-only access**. A prominent banner is displayed on the UI to communicate the status.

6. Database Table Lists

The project utilizes a **two-database architecture** to separate secure authentication data from general academic data. This enhances security by ensuring the main ERP database never holds sensitive password hashes.

6.1. Authentication DB — `users_auth`

(Schema: `sql/auth_schema.sql`)

Column	Data Type	Constraint/Notes
user_id	INTEGER	**PRIMARY KEY** (Foreign key link to ERP DB)

username	VARCHAR(50)	**UNIQUE**, NOT NULL
role	VARCHAR(20)	NOT NULL CHECK ('ADMIN','INSTRUCTOR','STUDENT')
password_hash	TEXT	**NOT NULL** (Stores bcrypt/argon2 hash)
status	VARCHAR(20)	DEFAULT 'ACTIVE'
last_login	TIMESTAMP	

6.2. ERP DB — Main Tables

(Schema: `sql/erp_schema.sql`)

Table	**Primary Key**	**Key Columns and Relationships**
students	user_id	**PRIMARY KEY** (References users_auth.user_id), roll_no (UNIQUE)
instructors	user_id	**PRIMARY KEY** (References users_auth.user_id), department

courses	course_id	code (UNIQUE), title, credits
sections	section_id	**FK** to courses.course_id, **FK** to instructors.user_id (instructor_user_id), capacity
enrollments	enrollment_id	**FK** to students.user_id, **FK** to sections.section_id **(ON DELETE CASCADE)**, **UNIQUE** (student_user_id, section_id)
grades	grade_id	**FK** to enrollments.enrollment_id **(ON DELETE CASCADE)**, component, score, final_grade
settings	key	Stores system variables like maintenance_on and per-section weights.

7. Development Quality and Architectural Choices

This section highlights key decisions made during implementation to ensure code quality, maintainability, and security.

A. Dependency Injection (Optional/Mocking)

To facilitate testing and adherence to the single-responsibility principle, the application was structured to use **Service Classes** (e.g., `InstructorService`) that take **Data Access Objects (DAOs)** as dependencies. This separation allowed for easier **JUnit testing** by mocking the database layer, verifying business logic (like grade calculation) without requiring a live database connection.

B. Input Validation Strategy

Beyond basic UI form validation (e.g., checking for empty strings or non-negative integers for capacity), the application implements **server-side validation** within the Service layer. This protects the database from bad data (e.g., negative credits, duplicate enrollment attempts) even if the UI were bypassed, thus enhancing data integrity.

C. Security Hardening (Hashing)

All user passwords are not only stored in a separate database but are secured using a strong, iterative hashing function (e.g., **bcrypt** or similar, as suggested by the project brief). The ERP database (`univ_erp`) holds **zero** password or hash information, minimizing the damage risk from an ERP-only data breach.

8. Conclusion & Next Steps

The AP-Project ERP application successfully meets all core requirements outlined in the project brief. The implementation demonstrates a solid understanding of object-oriented principles, database architecture separation, and robust access control.

Next Steps (Optional):

- **Documentation Finalization:** Replace all screenshot placeholders with high-resolution images.
- **Code Review:** Perform a final review to ensure adherence to Java naming conventions and to remove any deprecated code or commented-out sections.
- **Deployment Scripting:** Develop a final distribution package (e.g., a self-contained executable JAR with bundled dependencies) to simplify deployment and launch.

Appendix: Files Referenced

- `src/edu/univ/erp/service/InstructorService.java` (Weights computation and storage)
- `src/edu/univ.erp.service/MaintenanceService.java` (Maintenance flag storage)
- `src/edu/univ.erp.access/AccessControl.java` (Permission enforcement logic)
- `sql/auth_schema.sql`, `sql/erp_schema.sql` (Database schemas)