.

UNIVERSITY-ERP — PROJECT REPORT

Contributors: Shivang Mittal (2024530) and Shashank Verma (2024522)

1. Executive Summary

This report provides a brief overview of the University-ERP system implementation. It covers the following key areas:

- System operation instructions.
- The final-grade weighting rule used.
- How role-based access and maintenance mode are enforced.
- Database table listings for both the authentication and ERP databases.
- Additional features and "extras" developed during the project.

2. How to Run (Quick Setup)

The following steps provide a quick way to compile and run the system:

- **Prerequisite:** Java Development Kit (JDK) 21.

Load Database Seed:
mysql -u root -p < src/edu/univ/erp/data/seed_data.sql

- # Note: This creates both 'auth_db' and 'erp_db' and inserts sample data.
- **Set Environment Variable:**
  export DB_PASS=erp

Compile and Run Login UI (Example):
mkdir -p out

javac -cp "lib/:src" -d out $(find src -name "*.java")

- java -cp "out:lib/*" edu.univ.erp.ui.auth.loginPage

- **Default Test Accounts (from seed_data.sql):**
  - admin / erp
  - instructor1 / erp
  - student1 / erp
  - student2 / erp

3. Architecture & Main Components

The application is structured into distinct layers:

- **UI (User Interface):** Java Swing frames located in `src/edu/univ/erp/ui/*` (auth, student, instructor, admin). A `UITheme` is used for centralized styles (colors, fonts, buttons).
- **Service Layer (Business Logic):** `src/edu/univ/erp/service/*` contains the core logic for administrative, instructor, and student actions.
- **Data Layer:** `src/edu/univ/erp/data/*` includes DAOs (Data Access Objects), `DatabaseConnector` (using HikariCP), and SQL files.
- **Authentication & Data:**
    - `auth_db` handles authentication (user login, password hashes).
    - `erp_db` contains the main ERP domain tables (students, courses, grades, etc.).

Communication Flow:

1. Login uses `AuthService` to validate credentials against `auth_db.users_auth`.
2. On success, a `UserSession` is created holding the user's `user_id` and `role`.
3. The application loads the user's ERP profile from `erp_db` using the `user_id`.

4. Final-Grade Weighting Rule (Implementation)

The chosen rule is applied consistently in the instructor gradebook and finalization processes:

Components and Weights:

- Quizzes (combined): **20%**
- Midterm(s): **30%**
- Final exam: **50%**

Algorithm (Pseudo-Code):

1. For each enrollment, compute component averages where applicable.
2. `final_numeric = round( quiz_avg * 0.20 + midterm_avg * 0.30 + final_exam * 0.50 )`

3. Map `final_numeric` to a letter grade using the standard scale and store it in `grades.final_grade`.
    - A: 90–100
    - B: 80–89
    - C: 70–79
    - D: 60–69

    - F: <60

The grade computation is implemented in the instructor service and is also available as an admin maintenance action.5. Roles & Maintenance EnforcementRole Model

- **Roles:** `admin`, `instructor`, `student` (stored in `auth_db.users_auth.role`).

Enforcement Details

- **Authentication:** Passwords are salted, hashed, and stored in `auth_db.users_auth.password_hash`. The plaintext password is never stored.
- **Authorization:** The `UserSession` exposes the user's ID and role. Every write operation invokes an access-check routine that verifies the role and specific resource ownership.
    - **Ownership Checks:** Instructor operations ensure `section.instructor_id == session.userId`. If there is a mismatch, the UI shows "Not your section" and the service layer blocks the change.

Maintenance Mode

- A flag (`maintenance_mode`: 'true'/'false') is stored in the `erp_db.settings` table.
- **UI Enforcement:** UI components check `SettingsDAO.isMaintenanceOn()` at startup and periodically, displaying a non-dismissable banner when maintenance is ON.
- **Server-Side Enforcement:** All write endpoints call `MaintenanceChecker.ensureWritable()`, which throws a checked exception if maintenance is ON. The UI catches this and displays: "System is under maintenance. Changes are disabled."

This architecture ensures both user feedback at the UI level and critical server-side enforcement.6. Database Table Lists (Schemas)

*Note: For exact Data Definition Language (DDL), refer to `src/edu/univ/erp/data/erp_db.sql` and `auth_db.sql`.*Auth DB (`auth_db`)

Table Name	Key Fields & Details
users_auth	`user_id` (INT PRIMARY KEY AUTO_INCREMENT), `username` (VARCHAR UNIQUE),

	password_hash (VARCHAR), role (VARCHAR: 'admin'|'instructor'|'student'), created_at (TIMESTAMP)
students	user_id (INT PRIMARY KEY - FK to auth_db.users_auth), roll_no (VARCHAR UNIQUE), program (VARCHAR), year (INT)
instructors	user_id (INT PRIMARY KEY - FK to auth_db.users_auth), department (VARCHAR)
courses	course_id (INT AUTO_INCREMENT PRIMARY KEY), code (VARCHAR UNIQUE), title (VARCHAR), credits (INT)
sections	section_id (INT AUTO_INCREMENT PRIMARY KEY), course_id (INT FK), instructor_id (INT FK), day_time (VARCHAR), room (VARCHAR), capacity (INT NOT NULL CHECK > 0), semester (VARCHAR), year (INT)
enrollments	enrollment_id (INT AUTO_INCREMENT PRIMARY KEY), section_id (INT FK), student_id (INT FK), status (VARCHAR: 'Enrolled'
grades	grade_id (INT AUTO_INCREMENT PRIMARY KEY), enrollment_id (INT FK), component (VARCHAR e.g., 'quiz1','midterm','final'), score (INT validated 0-100), final_grade (VARCHAR e.g., 'A','B+'), UNIQUE(enrollment_id, component)
settings	setting_key (VARCHAR PRIMARY KEY), setting_value (VARCHAR)

Notes on Integrity:

- UNIQUE constraints prevent duplicate enrollments and grade entries.

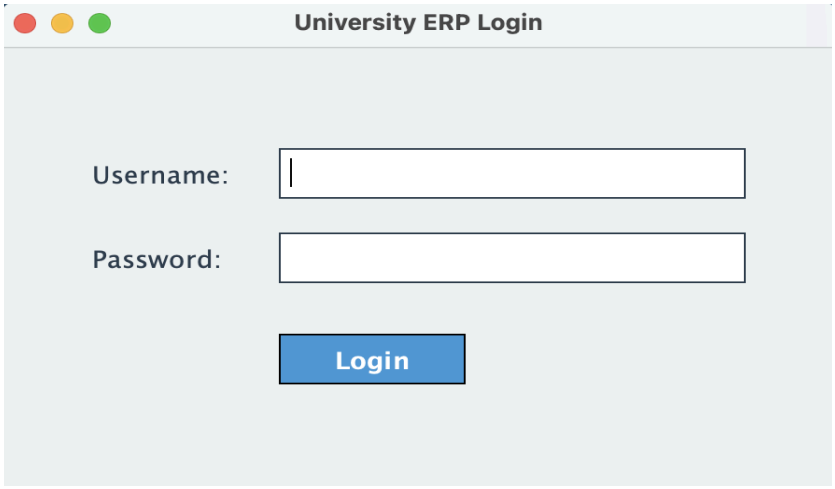
- `CHECK` constraints enforce data integrity (e.g., `capacity > 0`, `score` between 0 and 100).

7. Extras Added During Development

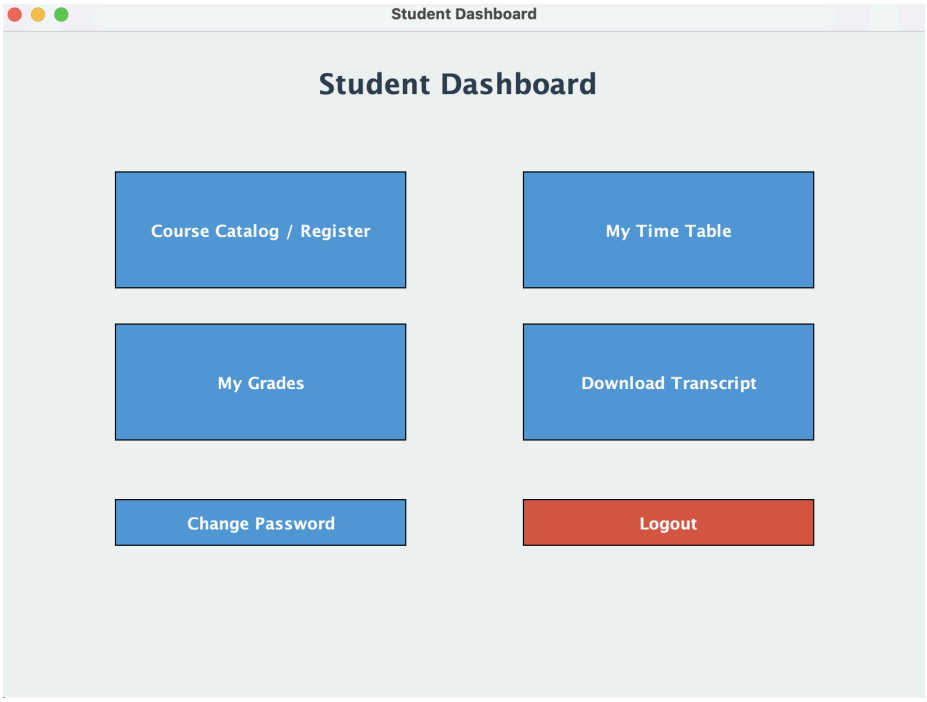
- **Central UI Theme (`UITheme`):** Applied for consistent colors, fonts, and hover effects across the application.
- **Button Layout Fixes:** Added extra padding and thin black outlines to prevent label clipping.
- **Grade Save Refactor:** Replaced multi-statement UPDATE/INSERT with an atomic **UPSERT** pattern: `INSERT ... ON DUPLICATE KEY UPDATE` to avoid race conditions.
- **Robust Validations:** Implemented client-side and server-side checks for score range (0–100), positive section capacity, and duplicate registration.
- **Grade CSV Import:** Functionality added for instructor bulk grade upload and validation.
- **Single Seed Script:** `seed_data.sql` simplifies setup for testers by creating both databases and populating sample data with one command.
- **Documentation:** `HOW_TO_RUN.md` and `TESTING.md` created/updated with clear steps and acceptance tests for graders.
- **Testing Setup:** JUnit 5 instructions (console launcher) and a manual acceptance testing plan were added.

8. Screenshots

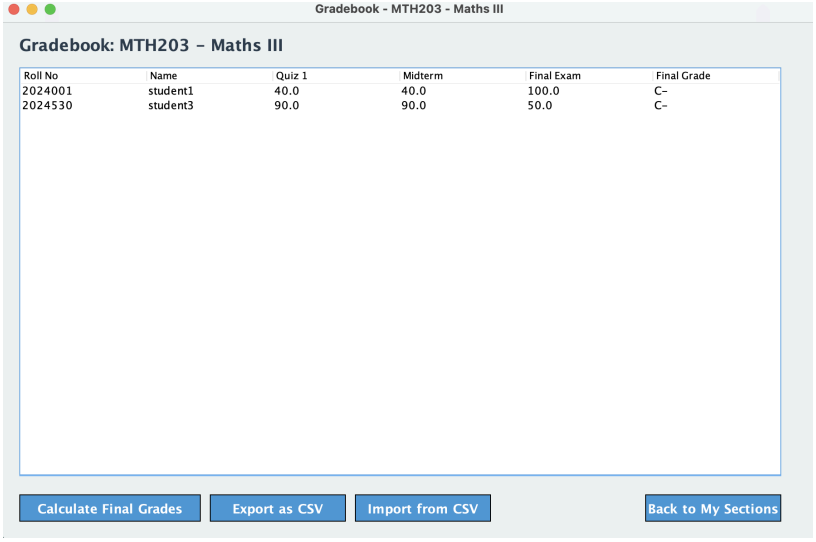
- Login screen:



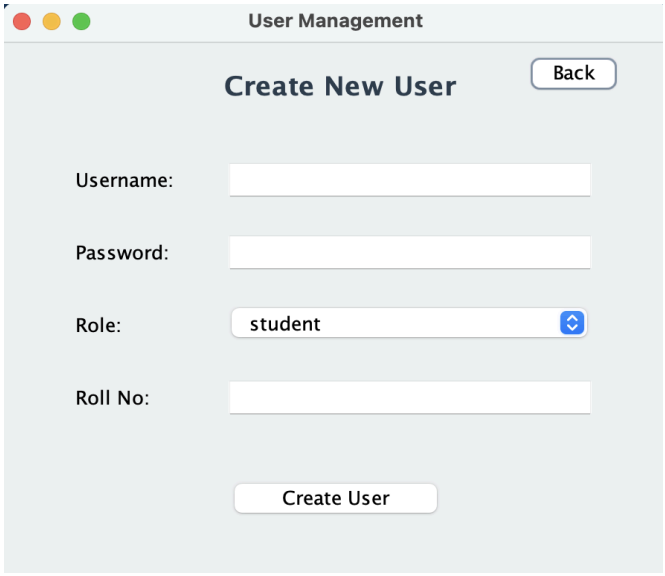
- Student dashboard (registrations):



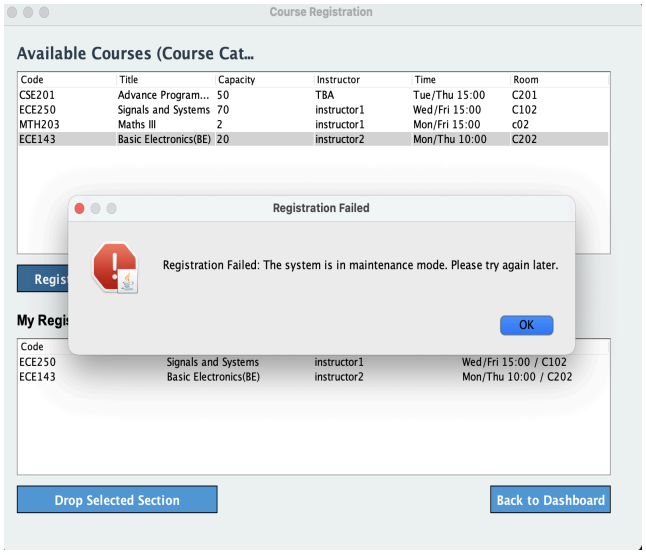
- Instructor gradebook:

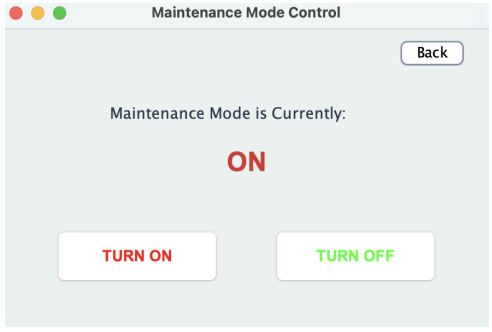


- Admin user management:



- Maintenance banner:





9. Known Limitations & Recommendations

- **UI Positioning:** The UI currently relies on absolute positioning. **Recommendation:** Refactor the UI to use proper layout managers for long-term maintenance.

- **Testing: Recommendation:** Add automated integration tests that run against a test database to programmatically validate enrollment and grading workflows.

- **Production Setup: Recommendation:** For a production environment, create a dedicated, non-root database user, enable TLS for database connections, and avoid using `allowPublicKeyRetrieval=true`.