University ERP (Java + Swing) — Project Brief (8 Weeks)

SSpecification with Testing & Evaluation Details

1. What you will build

A desktop application written in **Java** using **Swing** that helps a university manage **courses**, **class sections**, **enrollments**, and **grades**, with three kinds of users:

- **Student** — register/drop sections, view timetable, grades, download transcript.
- **Instructor** — manage assigned sections, enter scores, compute final grades, view simple stats.
- **Admin** — add users (students/instructors), create courses/sections, assign instructors, toggle **Maintenance Mode** (view-only for students/instructors).

Data is stored in **two databases**: (1) an **Auth DB** for usernames, roles, and *password hashes* (UNIX "shadow" style); and (2) an **ERP DB** for everything else (students, instructors, courses, sections, enrollments, grades, settings).

2. Access rules (who can do what)

- **Admin**: full control—manage users, courses, sections, assignments, maintenance mode.
- **Instructor**: only for their *own* sections, e.g. grades, year, course.
- **Student**: only for their *own* registrations and records.

If a user tries an action they are not allowed to do, show a clear message and do nothing.

Maintenance Mode. When ON, students and instructors can still log in and *view*, but *cannot change* anything. Show a visible banner in the UI.

3. Features and behavior

A. Common

- **Login** → open a role-matched dashboard.
- **Clear messages**: success, error, not allowed, full class, wrong password, etc.
- **Search & tables**: list courses/sections in sortable tables.

B. Student

- Browse **course catalog** (code, title, credits, capacity, instructor).
- **Register** for a section only if seats are available and not a duplicate.[1]
- **Drop** a section before your stated deadline.
- View **timetable** (their registered sections by day/time).
- See **grades**: assessment components (e.g., quiz, midterm, end-sem) and **final grade**.

[1] Optional: add a simple prerequisite rule and document it.

- **Download transcript** (CSV or PDF).

C. Instructor

- See **My Sections** for the term.
- **Enter scores** for assessments you define and **compute final** using your weighting rule (e.g., 20/30/50).
- Show simple **class stats** (averages, etc.).
- Optional: **CSV import/export** for grades.

D. Admin

- **Add users** (students/instructors/admins if needed).
- **Create/edit courses and sections** (course, day/time/room, capacity, semester/year).
- **Assign instructor** to a section.
- **Toggle Maintenance Mode** (ON/OFF) and display a banner.
- Optional: simple **backup/restore** for ERP DB.

4. Data to store

Auth DB (separate). `users_auth(user_id, username, role, password_hash, status, last_login)`. No real passwords—only secure hashes. Optional: `password_history`.

Student DB Tables such as:

- `students(user_id, roll_no, program, year)`
- `instructors(user_id, department, ...)`
- `courses(code, title, credits)`
- `sections(course_id, instructor_id, day_time, room, capacity, semester, year)`
- `enrollments(student_id, section_id, status)` (prevent duplicates)
- `grades(enrollment_id, component, score, final_grade)`
- `settings(key, value)` with `maintenance_on = true/false`

CHECK IF MORE DATA FIELDS CAN BE ADDED

Linking both DBs: `students.user_id` and `instructors.user_id` match `users_auth.user_id`.

5. Suggested package layout (folders)

These are hints to keep work readable:

- `edu.univ.erp.ui` — all windows/panels/dialogs. Optional subfolders: `ui.common`, `ui.auth`, `ui.student`, `ui.instructor`, `ui.admin`.
- `edu.univ.erp.domain` — simple data classes (Student, Instructor, Course, Section, Enrollment, Grade, Settings). No DB/UI code.

- `edu.univ.erp.service` — the "brain" methods screens call (register, drop, enter scores, compute final, add users, toggle maintenance). Changes always check access rules & maintenance flag.
- `edu.univ.erp.data` — database helpers for the ERP DB (read/write students, courses, sections, enrollments, grades, settings).
- `edu.univ.erp.access` — small helper that answers "Is this allowed?" and "Is maintenance ON?"
- `edu.univ.erp.auth` — login, password hashing/verification, current session (talks to Auth DB only). Optional subfolders: `auth.store`, `auth.hash`, `auth.session`.
- `edu.univ.erp.util` — CSV/PDF export, date/time helpers, message helpers, simple logger.

6. Passwords and login (UNIX "shadow" idea)

- Keep passwords in **Auth DB only**, never in ERP DB.
- Store only **hashes** (e.g., bcrypt), not real passwords.
- **Login flow**: lookup username in Auth DB → verify typed password against hash → if OK, remember user_id+role in session → load profile from ERP DB.
- Add **Change Password**; consider blocking login after several wrong tries (bonus).

7. Non-functional expectations

- **Clean UI**: clear labels, sensible layouts, helpful messages; sortable tables are a plus.
- **Safe inputs**: validate before saving (no negative capacity, valid dates).
- **Stable**: do not crash; handle errors with messages.
- **Separation**: screens do not talk to DB directly; they call service/brain.

8. What to submit

1. **Working app**: quick "How to run" with credentials.
2. **Sample data**: one admin, one instructor (with sections), two students (with at least one enrollment).
3. **Short report** (5–7 pages): screenshots; final-grade weighting rule; how roles & maintenance are enforced; table lists for both DBs; extras added.
4. **Testing pack**: test plan (see §) and small test dataset; one-page test summary.
5. **Diagrams**: use-case lists; "things" sketch (relations); 2–3 flow sketches (enroll, grade entry, maintenance toggle).
6. **Demo video** (5–8 minutes): student flow, instructor flow, admin flow incl. maintenance.

9. Suggested 8-week plan

W1: Decide data fields & screens. Draw simple diagrams. Prototype a window.

W2: Create both DBs and connect. Seed a few users (with hashed passwords).

W3: Build login and dashboards; role-aware menus.

W4: Student flows (catalog, register/drop, timetable).

W5: Instructor flows (gradebook, final grade).

W6: Admin flows (users, courses, sections, assign instructor).

W7: Maintenance mode + exports (CSV/PDF) + polish messages/validation.

W8: Full test pass, screenshots, final fixes, demo recording.

10. Testing and evaluation

A. Environment for checks

Use the provided sample accounts: `admin1` (Admin), `inst1` (Instructor), `stu1`, `stu2` (Students). Use your seed data.

B. Acceptance tests (must pass)

Each step should display the expected result in the UI.

Login & roles

☐ Wrong password is rejected; shows "incorrect username or password."

☐ After login, the dashboard matches the role (student/instructor/admin).

Student

☐ See catalog with code/title/credits/capacity/instructor.

☐ Register in a section with free seats → success and appears in "My Registrations" and timetable.

☐ Try to register the *same* section again → blocked with a clear message.

☐ Try to register in a *full* section → blocked with "Section full."

☐ Drop before deadline → success; appears removed.

☐ View grades for registered courses and sections.

☐ Download transcript (CSV or PDF) listing completed courses/grades.

Instructor

☐ See only their own sections.

☐ Enter scores (quiz/midterm/end-sem) → saved.

☐ Compute final grade using the rule in the report → shows final values.

☐ Try to edit a section they do not teach → blocked with "Not your section."

Admin

☐ Create a new student user (username+role in Auth DB; student profile in ERP DB).

☐ Create a course and a section; assign instructor.

☐ Toggle Maintenance ON → banner shows; student/instructor can *view* but *cannot change* (register/drop, grades blocked with message).

☐ Toggle Maintenance OFF → normal behavior returns.

Password & Auth separation

☐ Password *hash* exists in Auth DB (not the real password).

☐ ERP DB does not contain passwords.

☐ Login uses Auth DB, then loads student/instructor profile from ERP DB via the shared `user_id`.

Exports (pick at least one)

☐ Transcript export (Student) *or* grade CSV export (Instructor) works and opens.

C. Edge & negative tests

☐ Capacity cannot be negative or nonsensical.

☐ Register/drop after the stated deadline → blocked with message.

☐ Student cannot view or change another student's data.

☐ Instructor cannot grade students from another instructor's section.

☐ With Maintenance ON, student/instructor changes are blocked everywhere (not only on one screen).

D. Data integrity checks

☐ Duplicate enrollments (same student+same section) are prevented.

☐ Removing a section: either blocked when students are enrolled or clearly explained/documented.

E. Security basics

☐ Real passwords are never stored; only hashes.

☐ A Change Password dialog exists (bonus).

☐ After 5 wrong login attempts, show a warning or temporary lock (bonus).

☐ Access rules checker is called before any change (we will try to bypass via UI flow).

F. UI/UX checks

☐ Buttons and labels are clear; errors are friendly.

☐ Tables for lists; sorting/filtering is a plus.

☐ Long actions show "please wait" (even a simple dialog).

G. Performance sanity

☐ Catalog list of ∼100 courses opens quickly (a few seconds or less).

☐ App starts without long hangs or crashes.

H. Maintenance & (optional) backup/restore

☐ Maintenance toggle flips the settings flag and the UI shows a banner immediately.

☐ All student/instructor *writes* are blocked while ON.

☐ (Bonus) Backup/restore: trigger backup, change a row, restore, and see it revert.

I. What to hand in for testing

- One-page "How to run": Java version, DB setup, two DB connection settings, default accounts.
- Seed scripts for both DBs (create tables + insert sample data).
- Test plan (2–4 pages): the acceptance tests above in your words; extra tests you added; which accounts/data to use.
- Test summary (1 page): which tests passed/failed; known issues.

11. Grading rubric (100 points + up to 10 bonus)

Category	**Points**
Functionality: student (10) + instructor (10) + admin (10)	30
Access rules & Maintenance (role enforcement 10, maintenance 5)	20
Authentication and student records separation & password safety (two DBs; hashes only)	10
Data design & integrity (sensible tables; prevent duplicates; validation)	10
UI/UX quality (clarity; messages; tables; no crashes)	10
Testing quality (plan+data 4, reproducible pass 6)	5
Documentation & Demo (report 7, video 3)	5
Code/Project organization (folders, naming)	10
Total	**100**
Bonus: CSV import/export (+3); Change Password & lockout (+3); Notifications panel (+2); Backup/restore (+2)	+10

12. Academic integrity

Work within your assigned team. You may consult public documentation, but *write your own* screens, flows, and table definitions. Cite any inspiration. Direct copy of other teams' work or online solutions is not allowed. Kindly refer to IIITD's academic honesty policy.

13. Final tip

Keep the **roles** and **maintenance** rules in mind for any button that changes data. If a click changes data, ask: (1) Is the user allowed to do this? (2) Is maintenance OFF? If both are yes $\rightarrow$ proceed; otherwise, show a friendly message.

14. Suggested API packages (UI $\leftrightarrow$ Services boundary)

To keep the app tidy, let the **UI call a small set of "API" packages**. These are just organized folders of easy-to-understand actions. No database code lives here; they simply coordinate requests and return clear results or errors.

- `edu.univ.erp.api.auth` — login, logout, change password, "who am I" (current user + role). Talks to Auth DB via the auth package.
- `edu.univ.erp.api.catalog` — list/search courses and sections for a term; show capacity & instructor.
- `edu.univ.erp.api.student` — register/drop a section (with checks), view timetable, grades, download transcript.
- `edu.univ.erp.api.instructor` — see "my sections", enter scores for assessments, compute final grade, view simple stats, export grade CSV.
- `edu.univ.erp.api.admin` — add users (via Auth DB), add students/instructors (ERP DB profiles), create/edit courses and sections, assign instructor, toggle maintenance.
- `edu.univ.erp.api.reports` — class lists (CSV), transcript export (CSV/PDF), quick summaries (averages).
- `edu.univ.erp.api.maintenance` — read/write the maintenance flag; provide a quick "isReadOnlyNow?" check for the UI.
- `edu.univ.erp.api.common` — shared things all APIs can reuse: standard success/error message shapes; simple pagination info for long lists.
- `edu.univ.erp.api.types` — simple request/response *shapes* (fields only) used by multiple APIs (e.g., a "CourseRow" or "SectionRow" listing code, title, credits, capacity, instructor name).

How to use these: The UI panels ask these APIs to "do a thing" or "give a list". The API validates the request and then calls the service layer. The API *never* talks to the database directly, and it returns clear results: either the data you asked for or a friendly error (e.g., "Section full", "Not your section", "Maintenance is ON"). Keep names simple and descriptive; avoid long or clever names.

15. Suggested real JAVA packages that could be used

Below are practical, widely used options students may choose from. Versions are intentionally omitted; pick current stable releases.

UI (Swing ecosystem)

- **Swing (JDK built-in)** — core UI components (`javax.swing.*`).
- **FlatLaf** (`com.formdev:flatlaf`) — modern Look & Feel for a cleaner desktop appearance.
- **MigLayout** (`com.miglayout:miglayout-swing`) — simple, flexible layout manager for tidy forms.
- **LGoodDatePicker** (`com.github.lgooddatepicker:LGoodDatePicker`) — ready-made date pickers using `java.time`.
- **JFreeChart** (`org.jfree:jfreechart`) — basic charts for class averages or score distributions (optional).

Database connectivity & pooling

- **JDBC** (JDK built-in) — standard database API (`java.sql`).
- **MySQL Connector/J** (`mysql:mysql-connector-j`) or **MariaDB** (`org.mariadb.jdbc:mariadb-java-client`).
- **HikariCP** (`com.zaxxer:HikariCP`) — lightweight connection pool.
- (Optional) **Flyway** (`org.flywaydb:flyway-core`) or **Liquibase** (`org.liquibase:liquibase-core`) — schema migrations.

Password hashing (Auth DB, "shadow" style)

- **jBCrypt** (`org.mindrot:jbcrypt`) — straightforward bcrypt hashing and verification.
- or **Password4j** (`com.password4j:password4j`) — supports bcrypt/argon2/scrypt with simple APIs.
- (Advanced) **Argon2 JVM** (`de.mkammerer:argon2-jvm`) — direct Argon2 hashing.

Tips: never store plaintext passwords; store only hashes. Use per-user salts (bcrypt/argon2 manage this). Provide a "Change Password" dialog.

CSV/PDF/Excel exports

- **OpenCSV** (`com.opencsv:opencsv`) or **Apache Commons CSV** (`org.apache.commons:commons-csv`) for CSV grade sheets/transcripts.
- **OpenPDF** (`com.github.librepdf:openpdf`) or **Apache PDFBox** (`org.apache.pdfbox:pdfbox`) for simple PDF exports.
- (Optional) **Apache POI** (`org.apache.poi:poi-ooxml`) for Excel files if desired.

Logging & configuration

- **SLF4J API** (`org.slf4j:slf4j-api`) with **Logback** (`ch.qos.logback:logback-classic`) for logging; or use JDK `java.util.logging`.
- **Typesafe Config** (`com.typesafe:config`) (optional) for external `application.conf`; otherwise, use `java.util.Properties`.

Testing

- **JUnit 5** (`org.junit.jupiter:junit-jupiter`) — unit and service tests.
- **Mockito** (`org.mockito:mockito-core`) — mocking database/services in tests (optional).
- **Hamcrest** (`org.hamcrest:hamcrest`) — readable assertions (optional).

Parsing/JSON (optional)

- **Jackson** (`com.fasterxml.jackson.core:jackson-databind`) or **Gson** (`com.google.code.gson:gson`) for simple JSON (if needed).

Backup helpers (optional)

- Use **mysqldump** / **mysql** CLI via Java's `ProcessBuilder` to implement a simple backup/restore button with a progress dialog.

Licensing note. Prefer Apache/BSD/MIT/LGPL libraries for student projects (e.g., FlatLaf, HikariCP, Commons CSV, PDFBox, OpenPDF, JUnit). If you use GPL/AGPL libraries, ensure your distribution complies with the license.