Course & Grading Management System – Project Report

1. Introduction

This project is a full-stack Course & Grading Management System designed for three user roles: Admin, Professor, and Student. It streamlines course creation and enrollment, grade management with configurable policies, CSV-based imports, statistical insights, letter-grade publishing (via distribution-based z-scores), and a challenge/appeal workflow with email notifications.

The system follows a clean separation of concerns:

- Backend: Node.js/Express REST API with MongoDB/Mongoose, JWT-based authentication, role-based access control (RBAC), and Nodemailer for transactional emails.
- Frontend: React (Vite) single-page application with React Router, Tailwind CSS for UI, Axios for API integration, and PapaParse for CSV handling.

It supports LAN and localhost development, with CORS configured to whitelist known hosts and the client configured with a base API URL for easy deployment on a shared network.

2. Tools and Libraries

- Frontend
  - React (SPA) + Vite bundler
  - React Router DOM for client-side routing
  - Tailwind CSS for utility-first styling
  - Axios for HTTP requests
  - PapaParse for CSV parsing (bulk registration and grade uploads)
- Backend
  - Node.js + Express for REST API
  - MongoDB with Mongoose ODM (schemas, population, validation)
  - JSON Web Tokens (jsonwebtoken) for stateless auth
  - bcrypt for password hashing (main and temporary passwords)
  - Nodemailer for emails (Gmail transporter)
  - dotenv for environment configuration
  - cors for CORS policy enforcement
- Project & Quality

    - ESLint for linting
    - Vite/Tailwind configs for fast dev and consistent styling

Key paths for reference:

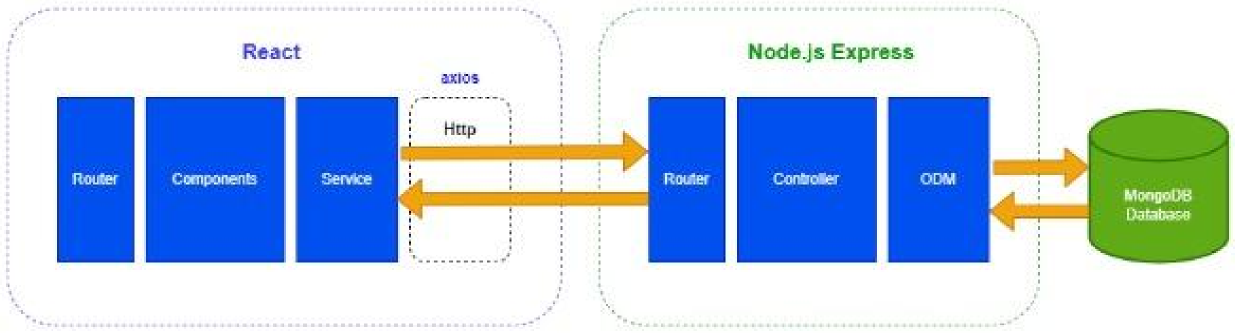
- Backend entrypoints: **`server/src/server.js`**, **`server/src/app.js`**
- Backend routes: **`server/src/routes/*`**
- Backend controllers: **`server/src/controllers/*`**
- Backend models: **`server/src/models/*`**
- Email service: **`server/src/services/emailService.js`**
- Frontend app shell: **`client/src/App.jsx`**, **`client/src/main.jsx`**
- Frontend auth context: **`client/src/context/AuthContext.jsx`**
- Frontend axios instance: **`client/src/api/axios.jsx`**

3. Features

- Authentication & Security
    - Automatic emails are sent to newly registered users with credentials
    - Temporary password generation and email delivery
    - Professors are notified when students submit challenges
    - Students are notified when professors respond to challenges
    - HTML-formatted emails with branding
    - Password logging to console when the email service is unavailable
- Admin Portal
    - Register users individually (students/professors).
    - Bulk user registration via CSV upload (PapaParse), with success/failure reporting.
    - User listing/search filters for Students and Professors.
- Professor Feature
    - Course Management:
        - Create and manage courses with custom grading policies
        - Define weightages for midterm, endterm, assignments, quizzes, attendance, and participation
        - Specify the number of quizzes for each course
    - Grade Assignment:
        - Assign and update student grades (midterm, endterm, assignments, quizzes)
        - Bulk grade entry interface
        - Individual student grade updates
        - Automatic weighted score calculation

    - Challenge Review
        - View and respond to student grade challenges
        - Review student submissions with attachments
        - Respond with detailed feedback and optional attachments
        - Track challenge resolution status
    - Automatic notifications for new challenges and responses
- Student Features
    - Course Management: View enrolled courses and join available courses
    - Grade Tracking: View detailed grade breakdowns with visual analytics
    - Password Management: Change password and recover forgotten passwords
    - Challenge Tracking: Monitor challenge status and professor responses
        - Download the professor's response attachments
        - View detailed challenge history
        - Real-time status updates (pending, reviewed, resolved)
    - Grade Challenges: Submit multiple challenges per course (max 5 challenges)
        - Upload attachments up to 25MB (PDF, images)
        - Track challenge count and remaining submissions
        - View challenge status with colour-coded indicators
- CSV Workflows
    - Bulk user registration: parse and validate user lists for admin import.
    - Grade imports: map CSV headers to grade fields; robust matching using name/email within a course.

4. Methodology



4.1 Backend

- Architecture
    - Express app (`server/src/app.js`) mounts versioned routes under `/api`:
        - `/api/auth` (auth routes)
        - `/api/users` (admin + RBAC)

        - `/api/courses` (course management)
        - `/api/grades` (grade CRUD, CSV, statistics)
        - `/api/challenges` (student/professor challenge workflows)
    - CORS is configured to allow localhost and specific LAN hosts; JSON body parsing is enabled; a central error handler returns structured errors.
- Data Models (Mongoose)
    - `User` (`server/src/models/userModel.js`): name, email (unique), password hash, role, enrolledCourses; supports temporary password hash + expiry and methods to match main or temporary password.
    - `Course` (`server/src/models/Course.js`): name, code (unique), description; professor, owner, and enrolled student refs; configurable policy weights (midsem, endsem, quizzes, project, assignment, attendance, participation); `maxMarks` map; `quizCount` and `assignmentCount`; `letterGradesPublished` flag.
    - `Grade` (`server/src/models/Grade.js`): one per student/course; `marks` object holds per-component and per-quiz/assignment fields; upserted on updates and CSV imports.
    - `Challenge` (`server/src/models/Challenge.js`): links student/course/grade with description, optional attachment, status (pending/reviewed/resolved), and professor response; auto-populates refs on queries.
- Controllers & Core Logic
    - Auth (`server/src/controllers/authController.js`):
        - Register user (admin only) with welcome email.
        - Log in with main or temporary password (temporary passwords are time-boxed); returns a JWT and a `usedTempPassword` hint for UX.
        - Forgot password sets a temporary password and its expiry, emails it, and provides a developer-friendly fallback log for the temporary password in case the email fails.
        - Change Password accepts the current (main or temporary) password and updates the main password; it then clears the temporary credentials.
    - Users (`server/src/controllers/userController.js`): list by role and bulk registration with per-record success/failure details.
    - Courses (`server/src/controllers/courseController.js`): create, list (my courses/all courses), join, get by id, and update operations, including:
        - Policy weights are updated with validation to ensure the total equals 100.
        - Update quiz/assignment counts within safe bounds and ownership checks.
        - Update per-field `maxMarks` with positive values and merging.

        - Toggle letter grade publishing to release computed grades to students.
    - Grades (`server/src/controllers/gradeController.js`):
        - Per-course grade retrieval for professors with merged marks defaults.
        - Grade updates per student via grid-style payloads (upsert).
        - Student self-view for a course with letter grade computation when published.
        - Course statistics, including overall and per-assessment aggregates, provide derived weighted scores for each student.
        - CSV ingest that maps CSV fields to `marks`, matches by name/email, validates enrollment, and returns a detailed summary.
    - Challenges (`server/src/controllers/challengeController.js`): end-to-end lifecycle to create, view, and respond to challenges with role-aware access and notification emails.
- Authentication & RBAC
    - Middleware in `server/src/middleware/authMiddleware.js` verifies JWTs, populates `the req.user property`, and exposes guards: `adminOnly`, `professorOnly`, `studentOnly`, and `verifyToken` for shared protected endpoints.
- Grading Computation
    - The weighted score combines each active component using the course's policy weights and configured max marks. Let `s_i` be the student's score and `m_i` the max for component i; the normalised component score is s_i / m_i
    - Weighted score formula:
        - Weighted component: $c_i = (s_i / m_i) \times w_i$ where wi is the weight percentage for component i.
        - Total weighted score: $W = \Sigma c_i$ with the constraint $\Sigma w_i = 100$.
    - Letter grades (when published) use distribution-based z-scores across the class:
        - $z = x - \mu / \sigma$ where x is a student's overall weighted score, and μ, σ are the class mean and standard deviation.
        - Thresholds are derived from z-score buckets and are only revealed to students once the professor toggles publishing for the course.
- Email Service
    - Implemented in `server/src/services/emailService.js` using a Gmail transporter; supports welcome, password reset, challenge notifications, and challenge response templates with HTML content.
- Error Handling
    - Centralised error middleware (`errorHandler.js`) logs errors and returns `{ message, stack? }` JSON; controllers use try/catch with meaningful messages for client UX.

4.2 Frontend

- App Shell & Routing
    - React app (`client/src/App.jsx`) defines role-protected routes via `ProtectedRoute`. Public: `/login`. Student, Professor, and Admin sections are isolated by required roles. A shared Change Password route exists for all authenticated users.
- State & Auth
    - `AuthContext` persists user and token in `localStorage`, exposes `login`/`logout`, and appends the JWT to the Axios instance for authorised API calls.
- API Integration
    - `axios` instance (`client/src/api/axios.jsx`) sets the base URL (e.g., `http://10.7.45.10:5000/api`) and Authorisation header from stored token; all pages use this instance.
- UI & Pages
    - Admin: `AdminPanel`, `AdminRegister` (individual + CSV bulk), `StudentsList`, `ProfessorList`.
    - Professor: `ProfessorCourses`, `CreateCourse`, `EditCourse`, `GradeManagement` (editable grid, CSV import, set max marks, toggle letter grades), `GradeStatistics` (per-assessment stats and distribution), `ProfessorChallenges`.
    - Student: `StudentCourses` (enrol/join), `StudentGrades` (detailed breakdown + letter grade when published), `ChallengeGrade`, `StudentChallenges`.
    - Shared: `Navbar` with role-aware nav and `ChangePassword` page with strength meter and guidance.
- CSV Handling
    - Admin bulk registration and professor grade uploads use PapaParse to parse, validate, and map data prior to POSTing to the API; success/failure summaries are surfaced in the UI.
- Styling
    - Tailwind CSS powers consistent, responsive layouts with a focus on usability for data-dense forms and tables.

5. Challenges Faced

- Handling incomplete/partial marks while computing means, standard deviations, and z-scores without skewing the distribution

- Marks of one student are getting allocated to all.
- Keeping grading policy totals at exactly 100% while allowing dynamic quiz/assignment counts and per-component max marks.
- Developing on a shared network with specific IPs required careful CORS whitelisting.

6. API Endpoints

- Authentication
  - POST /api/auth/register` - Register new user (Admin only)
  - POST /api/auth/login` - User login
  - POST /api/auth/forgot-password` - Request password reset
  - PUT /api/auth/change-password` - Change password (Authenticated)
- Courses
  - GET /api/courses` - Get all courses
  - POST /api/courses` - Create course (Professor)
  - GET /api/courses/:id` - Get course by ID
  - PUT /api/courses/:id` - Update course (Professor)
  - DELETE /api/courses/:id` - Delete course (Professor)
- Grades
  - GET /api/grades/:courseId` - Get grades for a course
  - POST /api/grades/:courseId` - Create/Update grades (Professor)
  - PUT /api/grades/:gradeId` - Update specific grade (Professor)
- Challenges
  - POST /api/challenges` - Create grade challenge (Student, max 5 per course)
  - GET /api/challenges/student` - Get student's challenges
  - GET /api/challenges/professor` - Get professor's challenges
  - GET /api/challenges/course/:courseId` - Get course challenges
  - GET /api/challenges/count/:courseId` - Get challenge count for a course
  - PUT /api/challenges/:id/respond` - Respond to challenge (Professor)
  - GET /api/challenges/:id` - Get challenge details
- Users
  - GET /api/users` - Get all users (Admin)
  - GET /api/users/students` - Get all students
  - GET /api/users/professors` - Get all professors
  - POST /api/users/bulk-register` - Bulk register users via CSV (Admin)

7. Future Work

- Development
  - Create a new TA role/privilege that can modify the same course.
- Security & Compliance
  - Enforce stronger password policies on the server-side (length, entropy) and consider implementing two-factor authentication (2FA) or multi-factor authentication (MFA).
- Data & Performance
  - Cache expensive statistics and add background jobs for heavy CSV imports.
  - Add an Unauthorised page for cleaner redirects; improve empty/error states across tables and forms.