Grade Entry System
(Design Document)

Grade Entry System: The Grade Entry system is a web-based application which enables the users to securely log in and manage student grades. Authenticated users can enter grades for students across different courses and view a structured table displaying the grades each student has received per course. The system enforces login-based access to ensure that only authorized users can enter and manage grades.

Goals:

a. Login/ logout functionality for users.
b. Grade entry and updates for courses by the user.
c. Viewing grades in a structured table showing what grade each student received in each course.

Assumptions:

a. This application follows a monolithic architecture which runs on a localhost for demonstration and evaluation purposes. This design choice is appropriate given the expected small user base (up to ~1,000 users) and limited functional scope.
b. As this system is intended for a small-scale use case, a relational database (MySQL) is used. MySQL is sufficient for structured data storage, enforces data integrity through constraints, and is easy to manage for small to medium datasets.
c. This system implements a Role Based Access Control (RBAC) with the following users roles:
    1. Admin:
        - Full CRUD access across the system.
        - Can create and manage users (teachers and students).
        - Can create and manage courses.
        - Can view, update, or delete grades.
    2. Teacher:
        - Read and Update access.
        - Can view student grades and enter or update grades for assigned courses.
    3. Student:
        - Read-only access.
        - Can view only their own grades across enrolled courses in a structured table.
d. The user interface is intentionally kept minimal and functional, with a focus on correctness and core functionality rather than advanced UI/UX design.
e. User account creation is handled by the Admin. The admin assigns usernames and passwords to students and teachers. This assumption is made because the system targets a small, controlled environment where users may not self-register. Credentials can be shared securely through trusted communication channels such as email.

Features:

a. Role-Based Access Control: Admin, Teacher, and Student roles with specific permissions
b. Grade Management: Teachers can enter and manage student grades
c. User Management: Admins can manage users and assign roles
d. Login System: Secure authentication with role verification
e. Student Dashboard: View personal grades and academic records
f. Teacher Dashboard: Manage classes and enter grades
g. Admin Dashboard: User management and system administration

Technologies Used:

- Frontend: React 19, Axios (HTTP Client), CSS
- Backend: Python/ FastAPI, SQLAlchemy (ORM), PyMySQL (MySQL driver), Pydantic (Data validation), Python-dotenv (Environment variables)
- Database: MySQL

User Roles and Permissions:

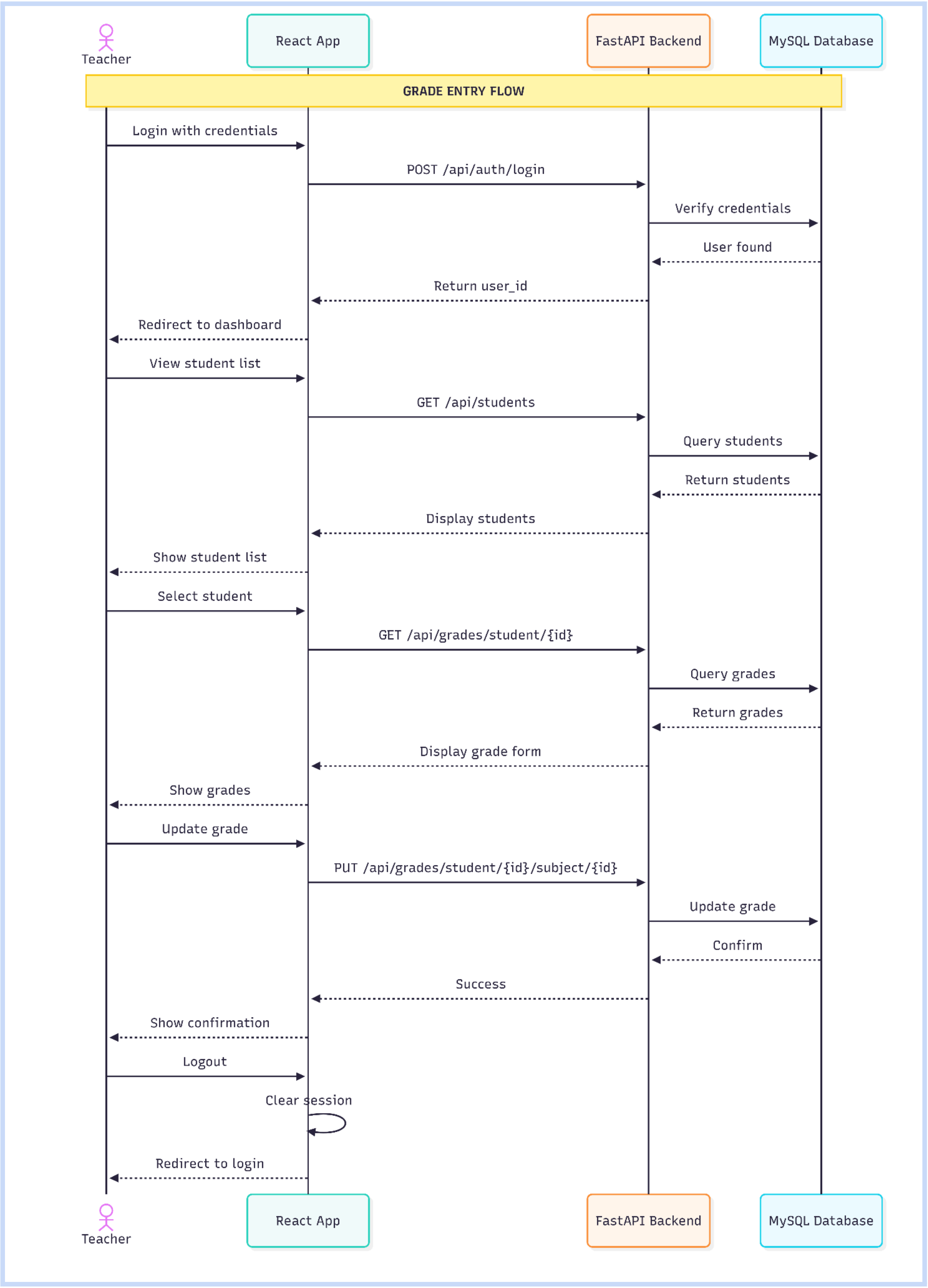
a. Admin: Manage users, subjects, view all grades
b. Teacher: Create/update grades
c. Student: View own grades only

System architecture:

- The system follows a layered monolithic architecture suitable for small-scale deployment.
- The frontend is built using React that runs on a local development server and handles authentication, manages local states using React hooks, stores minimal session data and role-based views for Admin, Teacher, and Student users.
- The application layer is implemented using FastAPI and acts as the API gateway.
- The backend is implemented using FastAPI and exposes REST APIs for user, subject, and grade management while enforcing role-based access control.
- Business logic is handled in a dedicated CRUD layer, with request and response validation managed using Pydantic schemas.
- SQLAlchemy ORM is used for database abstraction and interaction.
- The MySQL database stores persistent data for users, subjects, and grades with normalized relational design.

Sequence Diagram

Grade Entry Flow for Teacher:



Frontend Design:
Key Components:
- LoginPage: Authenticates user
- AdminView: User & subject management
- TeacherView: Grade entry interface
- StudentView: Read-only grade table
- GradeEditor: Reusable grade form
- State Management: React 'useState' for local form state, localStorage to persist logged-in user and role, props used to pass grade/user data between components

Backend:

API Endpoints:
1. **POST** /api/auth/login: Authenticates a user and returns user role and basic session details.
2. **POST** /api/users (Admin only): Creates a new user with an assigned role.
3. **GET** /api/users (Admin only): Retrieves a list of all users in the system.
4. **DELETE** /api/users/{id} (Admin only): Deletes a user by user ID.
5. **GET** /api/subjects (Public): Retrieves the list of available subjects.
6. **POST** /api/subjects (Admin only): Creates a new subject.
7. **DELETE** /api/subjects/{id} (Admin only): Deletes a subject by subject ID.
8. **GET** /api/grades/my-grades (Student only): Retrieves the grades of the logged-in student.
9. **GET** /api/grades/student/{id} (Teacher/Admin): Retrieves grades for a specific student.
10. **PUT** /api/grades/student/{studentId}/subject/{subjectId} (Teacher/Admin): Updates a student's grade for a specific subject.
11. **GET** /api/students (Teacher/Admin): Retrieves a list of all students. GET / Health check or default API response.

Database Design

- Tables:
  - User (id, name, email, role)
  - Subject (id, name)
  - Grade (id, student_id, subject_id, grade)

- Relationships:
  - User → Grade (one-to-many)
  - Subject → Grade (one-to-many)
  -

Error Handling & Validation Purpose:
- Backend validates requests
- Frontend displays user-friendly error messages (toast message)
- API returns standard HTTP status codes

Future Scope:

a. If the system is scaled to university-level, the application can continue to follow a monolithic MVC architecture, while improving scalability by deploying the frontend, backend, and database on separate servers and applying vertical scaling (increasing server resources). A distributed microservices architecture is not required for this use case due to limited request volume and high refactoring cost.
b. For a larger user base and higher volume of data, the system can migrate to more scalable databases such as: PostgreSQL (advanced relational features, performance).
c. Additional functionalities for admin can be updated such as blocking/ flagging users violating policies.
d. Additional functionalities for teachers can be extended to create course-specific course classrooms with assignment management and course-level dashboards with grade distributions.
e. University level students can be allowed to create their own accounts, manage personal profiles and enroll in courses through approval workflows.
   AI can also be integrated for automatic grade analysis, smart recommendations for academic improvement.
f. Additional high level advancements like mobile-friendly or native application support can be done along with email and notification services for grade updates/ class announcements, integration/ unit tests can be performed.
g. Enhance authentication using secure password hashing, token-based authentication, and session expiration.