Software Requirements Specification (SRS)

Project: Exam Management System (C++ Console Application)

1. System Architecture & Setup

- **Goal:** Develop a C++ Console Application using Object-Oriented Programming (OOP) principles.
- **Structure:** The system must use Class Hierarchy (Inheritance, Polymorphism) and File Handling for the database.
- **Persistence:** The application must persist data (users, exams, results) to text files (e.g., .txt or .csv) or a local database file so data is not lost between sessions.
- **Technologies:** Standard C++ libraries (<iostream>, <fstream>, <vector>, <string>, <sqlite3.h> if applicable).

2. Database Design (ERD Construction)

The system requires a database structure based on the following four core entities:

1. Users

- **Attributes:** UserID (PK), Username, Password, Role (Student, Instructor, Admin).
- **Description:** Stores login credentials and privilege levels.

2. Exams

- **Attributes:** ExamID (PK), SubjectName, InstructorID (FK), Duration, Date.
- **Description:** Stores the header information of an exam created by an instructor.

3. Questions

- **Attributes:** QuestionID (PK), ExamID (FK), Content, Option1, Option2, Option3, Option4, CorrectAnswer.
- **Description:** Stores the actual content linked to a specific exam.

4. Results (Students_Exams)

- **Attributes:** ResultID (PK), StudentID (FK), ExamID (FK), Score, Grade.
- **Description:** Links students to exams and stores their performance data.

3. User Authentication Module

- **Login Functionality:** The system must accept a Username and Password.

- **Role Redirection:** Upon valid credentials, the system must redirect the user to the specific menu corresponding to their role (Student Menu, Instructor Menu, or Admin Menu).
- **Error Handling:** Invalid credentials must trigger an error message immediately.

4. Functional Requirements by Role

A. Student Actions

1. **Login:** Access the system using personal credentials.
2. **View Exams:** See a list of available exams for enrolled courses.
3. **Take Exam:**
    - Select an exam.
    - Answer a series of multiple-choice questions (A, B, C, D).
4. **View Results:**
    - **Auto-Grading:** The system must calculate the score immediately upon completion (e.g., (Correct Answers / Total) * 100).
    - **Feedback:** Display the numeric score and letter grade (e.g., "85% - B").

B. Instructor Actions

1. **Login:** Access the system with instructor privileges.
2. **Add Exam:**
    - Input Exam Title/Course Name.
    - Define the number of questions.
    - Input question text, 4 options, and the correct answer.
3. **View Specific Student Result:** Input a student's ID or Name to view their degree.
4. **View Course Results:** View the degrees of all students for the exams created by this instructor.

C. Admin Actions

1. **Default Access:** The system must have at least one pre-created Admin account.
2. **Inherited Privileges:** Admins possess all privileges of an Instructor.
3. **User Management (Add):** Ability to create new Students, Instructors, and Admins.

4. **User Management (Remove):** Ability to delete users by ID.
5. **Global Reporting:** View the degrees of a specific student across **all** exams and courses in the system.

5. Class Hierarchy (OOP Design)

To fulfill the architectural requirements, the system implements the following hierarchy:

- **Class User (Base Class)**
  - Properties: ID, Username, Password.
  - Methods: login().
- **Class Student (Inherits User)**
  - Methods: takeExam(), viewMyGrades(), showAvailableExams().
- **Class Instructor (Inherits User)**
  - Methods: addExam(), addQuestion(), viewStudentResult(), viewAllResults().
- **Class Admin (Inherits User/Instructor)**
  - Methods: addStudent(), removeStudent(), addInstructor(), removeInstructor(), viewStudentDegree(), viewAllGrades().