Software Requirements Specification — Quiz Application

Version: 1.0
Prepared by: Omar Said (or the email/name you prefer)
Date: 16-Nov-2025

Table of Contents

---

1. Introduction

Project Summary:
The Quiz Application allows teachers to create multi-question exams with multiple choices and correct answers. Students can register/login and take exams via a simple terminal (CLI) interface. The system supports a timer per question or for the whole exam and shows the score along with incorrectly answered questions after completion.

1.1 Purpose and Intended Audience

This document is intended for the 8-member development team and the instructor/client. It defines all functional and non-functional requirements for the first version of the application.

1.2 Project Scope

The system will include:

- Registration/login for teachers and students.
- Teacher interface to create/edit/delete exams and questions.
- Student interface to take exams with a timer (per question or full exam).
- Local data storage using SQLite.
- Generation of result report for each student (score and incorrect questions).

Exclusions for the first phase:

- Payment system, AI grading, integration with university systems, or support for multimedia exams.

1.3 Terms, Definitions, and Acronyms

Term	Definition
Admin / Teacher	Person creating and managing exams
Student	Person taking exams
Exam	A set of questions organized with a title, time, and marks
Question	A multiple-choice question inside an exam
DBMS	SQLite
UI	User Interface (here: CLI/Terminal)

1.4 References

- Project specifications and PDF files (question interface examples)
- Internal educational standards (if any)

2. Overall Description

2.1 Product Perspective

Standalone CLI app running locally. It can be extended later to GUI or web interface, but the current version focuses on exam logic.

2.2 Key Product Features

- **Authentication:** Register/Login for teachers and students.
- **Exam CRUD:** Create/Read/Update/Delete exams.
- **Question CRUD:** For each exam (question text, choices, correct answer index, marks).
- **Student Exam Flow:** Display question, receive answer, calculate score, move to next question, finish exam.
- **Timer:** Option for per-question or full-exam timer.
- **Results:** Calculate total score, show incorrect questions, store results in DB.
- **Reports:** Simple teacher report (CSV/Console) for student performance per exam.

2.3 User Classes and Characteristics

- **Teacher:** Intermediate skill, uses terminal, can create/edit/delete exams and questions, view reports.
- **Student:** Beginner-intermediate skill, can register/login, take exams, view score.

2.4 Operating Environment

- OS: Windows / Linux / macOS
- Language: Python 3.10+
- DB: SQLite local .db file
- Execution: Terminal / Command Prompt

2.5 Design and Implementation Constraints

- Team of 8 developers (tasks divided: Auth, DB, CLI UI, Exam Logic, Timer, Reporting, Tests, CI).
- No internet or external services required in version 1.
- Use standard Python libraries where possible (sqlite3, argparse, hashlib).
- Modular files, importable.

2.6 Assumptions and Dependencies

- Python installed on all developer machines.

- Developers have write permissions.
- No integration with university systems in phase 1.

---

3. System Features (Functional Requirements)

3.1 Authentication (Register/Login)

Description: Two types of users: Teacher and Student.

Functional Requirements:

- FR1.1: Users can register with username, password, and role (student/teacher).
- FR1.2: Users can login with username/password.
- FR1.3: Passwords stored securely (hashed using hashlib.sha256 or similar).
- FR1.4: Post-login, users are directed to role-specific interface.

3.2 Exam Management (Teacher)

Description: CRUD for exams and questions.

- FR2.1: Create new exam (title, optional description, timer type: none/per_question/whole, timer duration).
- FR2.2: Add questions to exam (question text, choices list, correct answer index, marks).
- FR2.3: Edit question (text/choices/correct answer/marks).
- FR2.4: Delete question or entire exam.
- FR2.5: View list of exams with all questions.

3.3 Taking Exam (Student)

Description: Step-by-step interface for taking exams.

- FR3.1: Display list of available exams for the student.
- FR3.2: Load exam questions in order (or randomized if possible).
- FR3.3: Display one question at a time with selectable options.
- FR3.4: Support per-question or whole-exam timer. Expired time counts as wrong or moves automatically.
- FR3.5: At the end, calculate score and show detailed results (student answers, correct answers, incorrect questions).
- FR3.6: Store result in DB (exam_id, student_id, score, timestamp, answers_record).

3.4 Results

- FR4.1: Teacher can view all student results for a given exam.
- FR4.2: Export results to CSV.
- FR4.3: View details for a single student, showing incorrect questions.

3.5 Admin Utilities

- FR5.1: Reset passwords (admin or via DB).
- FR5.2: Optional import of questions/exams from CSV.

---

4. Non-Functional Requirements

- **Usability:** Simple text interface, clear instructions, teacher shortcuts (quick add question).
- **Reliability:** No data loss; commit changes immediately after each critical update.
- **Performance:** Loading/reading up to 200 questions in <1 second on normal hardware.
- **Security:** Passwords hashed, prevent code execution via input fields.
- **Maintainability:** Organized OOP code with modules and README.
- **Portability:** Runs across OS without major changes.
- **Scalability (Future):** Can extend to web UI or sync with server later.

Acceptance Criteria: 90% of functional requirements must pass system tests.

---

5. External Interface Requirements

5.1 User Interfaces

- CLI text-based menus.
- Organized menus by role:
    - Teacher Menu: Create Exam, Edit Exam, Delete Exam, View Results, Export CSV, Logout.
    - Student Menu: List Exams, Take Exam, View Past Results, Logout.
- Clear success/failure messages, confirmation on deletions.

5.2 Hardware Interfaces

- Standard computer with keyboard and screen.

5.3 Software Interfaces

- Python 3.x standard libraries (sqlite3, argparse, hashlib, csv, datetime).
- No external API for version 1.

5.4 Communication Interfaces

- None, system is local.

---

6. Detailed Use Cases

Use Case 1: Student Registration

- Actor: Student
- Preconditions: No existing account
- Main Flow:
  1. Student selects Register.
  2. Inputs username, password, full name, optionally university ID.
  3. System verifies username is unique.
  4. Save user in users table (role='student') with password hash.
  5. Display "Registration successful".
- Postconditions: User can login.

Use Case 2: Teacher Creates Exam

- Actor: Teacher
- Preconditions: Teacher is logged in
- Main Flow:
  1. Teacher selects Create Exam.
  2. Enters exam title, optional description, timer type, duration.
  3. System creates exam record and generates exam_id.
  4. Teacher adds questions (text, 4 choices, correct answer, marks).
  5. Press Save Exam.
  6. System stores questions in questions table linked to exam_id.
- Postconditions: Exam available to students.

Use Case 3: Student Takes Exam

- Actor: Student
- Preconditions: Student registered, exam available
- Main Flow:
  1. Student selects Take Exam and picks an exam.
  2. If whole-exam timer: start timer.
  3. Display first question, if per-question timer: start timer.

4. Student selects choice, submits, system records answer and moves to next question.
5. After finish/timeout: system calculates score, stores results, displays detailed results with incorrect questions.

- Postconditions: Student result saved and viewable.

---

7. Database Design (ER + Tables)

Users Table

```
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    full_name TEXT,
    role TEXT CHECK(role IN ('student','teacher')) NOT NULL,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
```

Exams Table

```
CREATE TABLE exams (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    description TEXT,
    author_id INTEGER,
    timer_type TEXT CHECK(timer_type IN ('none','per_question','whole'))
DEFAULT 'none',
    timer_seconds INTEGER DEFAULT 0,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (author_id) REFERENCES users(id)
);
```

Questions Table

```
CREATE TABLE questions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    exam_id INTEGER NOT NULL,
    question_text TEXT NOT NULL,
    weight INTEGER DEFAULT 1,
    FOREIGN KEY (exam_id) REFERENCES exams(id)
);
```

Choices Table

```
CREATE TABLE choices (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    question_id INTEGER NOT NULL,
    choice_text TEXT NOT NULL,
    is_correct INTEGER DEFAULT 0,
    FOREIGN KEY (question_id) REFERENCES questions(id)
);
```

Results Table

```
CREATE TABLE results (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    exam_id INTEGER NOT NULL,
    student_id INTEGER NOT NULL,
    score REAL,
    taken_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (exam_id) REFERENCES exams(id),
    FOREIGN KEY (student_id) REFERENCES users(id)
);
```

Answers Table

```
CREATE TABLE answers (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    result_id INTEGER NOT NULL,
    question_id INTEGER NOT NULL,
    chosen_choice_id INTEGER,
    is_correct INTEGER,
    FOREIGN KEY (result_id) REFERENCES results(id),
    FOREIGN KEY (question_id) REFERENCES questions(id),
    FOREIGN KEY (chosen_choice_id) REFERENCES choices(id)
);
```

Notes:

- `choices.is_correct` indicates the correct option.
- When saving results: insert into `results`, then store each answer in `answers` linked to `result_id`.

---

8. Acceptance Criteria & Test Cases

Acceptance Criteria Examples:

- User registration successfully saved → PASS
- Teacher creates exam and saves questions → PASS
- Student takes exam of 10 questions; score calculated correctly → PASS
- Timer works (per_question and whole) → PASS
- Result saved and displays incorrect questions → PASS

Test Case Example:

- Preconditions: Exam with 3 questions exists, student registered.
- Steps: login -> select exam -> answer Q1/Q2/Q3 -> finish
- Expected: score calculated correctly, result saved in `results`, answer details saved in `answers`.

---

9. Appendix

9.1 Suggested Team Task Division (8 Members)

- Auth & Users (2) — registration/login + hashing
- DB Schema & Migrations (1) — SQLite tables + seed data
- Exam CRUD Backend (2) — create/edit/delete exams, questions, choices
- Student Flow & Exam Logic (1) — exam logic + timer
- Reporting & Export CSV (1) — results and export
- Testing & Documentation (1) — test cases, README, final SRS

9.2 Suggested Helper Files

- `README.md`: project setup (create venv, install requirements, run)
- `seed_data.sql`: example exams and questions
- `sample_csv_import_format.csv`: example for importing questions
