Software Requirements Specification — Quiz Application

Version: 1.0
Prepared by: Omar Said (أو الايميل/الاسم اللي تحب تضيفه)
Date: 16-Nov-2025

Table of Contents

---

1. Introduction

ملخص المشروع:
نظام Quiz Application يسمح للمدرسين بإنشاء اختبارات (Exams) متعددة الأسئلة مع خيارات وإجابات صحيحة، والطلاب يسجلوا دخولهم ويأدوا الامتحان عبر واجهة بسيطة على الـ terminal (CLI). النظام يدعم تايمر لكل سؤال أو للامتحان، ويعرض نتيجة وعرض الأسئلة التي أجيبت خطأ بعد الانتهاء.

1.1 Purpose and Intended Audience

المستند ده موجه للفريق المطور (8 أفراد)، وللمحاضر/العميل، ويهدف لتحديد كل المتطلبات الوظيفية وغير الوظيفية لتطوير النسخة الأولى من التطبيق.

1.2 Project Scope

النظام سيشمل:

- تسجيل/تسجيل دخول للمدرس والطلاب.
- واجهة مدرس لإنشاء/تعديل/حذف الامتحانات والأسئلة.
- واجهة طالب لأخذ الاختبار مع تايمر (سؤال واحد أو كامل الامتحان).
- تخزين كل البيانات محليًا باستخدام SQLite.
- توليد تقرير نتائج لكل طالب (الدرجة، الأسئلة الخاطئة).
  لن يشمل المرحلة الأولى: نظام دفع، تحكيم آلي مع ذكاء اصطناعي، تكامل مع أنظمة الجامعة، أو دعم اختبارات مرئية/مسموعة.

1.3 Terms, Definitions, and Acronyms

- Admin / Teacher: شخص ينشئ ويحرر الامتحانات / مدرس.
- Student: شخص يؤدي الاختبارات / طالب.
- Exam: مجموعة أسئلة منظمة تحت اسم ووقت ووزن / اختبار / امتحان.
- Question: سؤال داخل امتحان يمكن أن يكون MCQ (اختيار من متعدد).
- DBMS: SQLite.
- UI: واجهة المستخدم (هنا CLI / Terminal).

1.4 References

- مواصفات المشروع وملفات الـ PDF (واجهة الأسئلة، أمثلة) التي سترسلها.
- معايير تعليمية داخلية (إن وُجدت).

---

2. Overall Description

2.1 Product Perspective

تطبيق مستقل (Standalone CLI app) يشتغل محليًا. يمكن لاحقًا تحويله لواجهة رسومية أو ويب، لكن النسخة الحالية تركِّز على المنطق والقواعد (logic).

2.2 Product Features (high-level)

- Authentication: تسجيل (Register) وتسجيل دخول (Login) للطلاب والمدرسين.
- Exam CRUD: Create / Read / Update / Delete للامتحانات.
- Question CRUD (نص السؤال، خيارات، علامة الإجابة الصحيحة، وزن السؤال) داخل كل امتحان.
- Student Exam Flow: عرض سؤال، استقبال إجابة، احتساب درجة، الانتقال للسؤال التالي، انتهاء الامتحان.
- Timer: خيار تايمر لكل سؤال أو للامتحان بأكمله.
- Results: حساب الدرجة الإجمالية، عرض الأسئلة الخاطئة، تخزين النتيجة في DB.
- Reports: توليد تقرير بسيط (CSV/Console) لمدرس حول أداء الطلاب في امتحان معين.

2.3 User Classes and Characteristics

- **Teacher (مدرس):** مستوى مهاري: متوسط (يعرف استخدام الـ terminal). يمكنه إنشاء/تعديل/حذف الامتحانات والأسئلة، مشاهدة التقارير.
- **Student (طالب):** مستوى مهاري: متوسط-مبتدئ. يمكنه التسجيل، تسجيل الدخول، أداء الامتحانات، رؤية النتيجة.

2.4 Operating Environment

- نظام تشغيل: Windows / Linux / macOS.
- لغة: Python 3.10+
- DB: SQLite (محلي `db`. ملف).
- تشغيل عبر Terminal/Command Prompt.

2.5 Design and Implementation Constraints

- المشروع مقسّم على 8 أفراد (يجب تقسيم المهام بوضوح: Auth, DB, CLI UI, Exam Logic, Timer, Reporting, Tests, CI).
- لا استخدام إنترنت أو خدمات خارجية في النسخة الأولى.
- قدر الإمكان Python الاعتماد على مكتبات قياسية (sqlite3، argparse أو simple input loops).
- الملفات قابلة للاستيراد والوحدة (modular).

2.6 Assumptions and Dependencies

- توفر Python مثبتة على أجهزة المطورين.
- كل مطور لديه صلاحية كتابة ملفات محلية.
- لا حاجة لتوافق مع نظام الجامعة في المرحلة 1.

---

3. System Features (Functional Requirements)

لكل ميزة أذكر وصفًا مختصرًا والمتطلبات التفصيلية.

3.1 Authentication (Register / Login)

Description: مستخدمين نوعين: student و teacher.
Functional requirements:

- FR1.1: يمكن للمستخدم التسجيل باسم مستخدم وكلمة سر ودور (student/teacher).
- FR1.2: يمكن للمستخدم تسجيل الدخول بالـ username/password.
- FR1.3: يجب تخزين كلمة المرور بشكل أساسي (hashed) — على الأقل استخدام hashing بسيط مثل `hashlib.sha256`.
- FR1.4: بعد تسجيل الدخول يوجه المستخدم لواجهة خاصة بدوره.

3.2 Exam Management (Teacher)

Description: CRUD للامتحانات والأسئلة.
Functional requirements:

- FR2.1: إنشاء امتحان جديد (اسم، وصف اختياري، نوع التايمر: None / per_question / whole_exam ، مدة زمنية بالثواني إن وُجدت).
- FR2.2: إضافة سؤال إلى امتحان (نص السؤال، قائمة خيارات، رقم الإجابة الصحيحة، وزن السؤال index/marks)).
- FR2.3: تعديل سؤال (تعديل نص/خيارات/إجابة صحيحة/الوزن).
- FR2.4: حذف سؤال أو حذف الامتحان كاملاً.
- FR2.5: عرض قائمة الامتحانات وكل أسئلتها.

3.3 Taking an Exam (Student)

Description: واجهة خطوة بخطوة لأداء الامتحان.
Functional requirements:

- FR3.1: عرض قائمة الامتحانات المتاحة للطالب.
- FR3.2: عند اختيار امتحان: تحميل أسئلة الامتحان بالترتيب (أو عشوائي لو ممكن).
- FR3.3: عرض سؤال واحد في كل مرة مع خيارات قابلة للاختيار.
- FR3.4: دعم Timer لكل سؤال أو للامتحان: عند انتهاء الوقت تُسجل الإجابة كخاطئة أو يتم الانتقال تلقائيًا.
- FR3.5: في النهاية حساب الدرجة وإظهار صفحة نتيجة شاملة (الدرجة، كل سؤال وإجابة الطالب والإجابة الصحيحة، والأسئلة الخاطئة).
- FR3.6: حفظ نتيجة الطالب في DB (exam_id, student_id, score, timestamp, answers_record).

3.4 Reports & Management

Description: المدرس يستطيع مشاهدة نتائج الطلاب.
Functional requirements:

- FR4.1: عرض قائمة نتائج لكل امتحان (طالب – درجة – تاريخ/وقت).
- FR4.2: تصدير النتائج بصيغة CSV.

- FR4.3: عرض تفصيلي لنتيجة طالب محدد يظهر الأسئلة الخاطئة.

3.5 Admin Utilities

Functional requirements:

- FR5.1: إمكانية إعادة تعيين كلمة المرور (بـ admin أو عبر DB فضلًا).
- FR5.2: استيراد أسئلة/امتحانات من ملف CSV — يمكن إضافتها كمطلب لاحق (اختياري).

---

4. Non-Functional Requirements

- **Usability:** الواجهة نصية بسيطة، تعليمات واضحة عند كل خطوة، اختصارات للمدرس (مثل إضافة سؤال سريعًا).
- **Reliability:** عدم فقدان بيانات؛ بعد كل تعديل مهم يتم `commit` فورًا.
- **Performance:** فتح وقراءة امتحان حتى 200 سؤال يجب أن يتم خلال < 1 ثانية تقريباً (على جهاز عادي).
- **Security:** تخزين كلمات المرور مشفّرة (hash). منع أي تنفيذ كود من الحقول النصية.
- **Maintainability:** كود منظّم (OOP, modules) مع توثيق وملف README لتقسيم المهام.
- **Portability:** تشغيل على أنظمة تشغيل مختلفة بدون تغييرات كبيرة.
- **Scalability (مستقبلي):** التصميم يسمح لاحقًا بالتحول لواجهة ويب أو إضافة Sync مع سيرفر.

مقاييس القبول: 90% من الوظائف الوظيفية الموضحة تعمل في اختبار النظام الأساسي.

---

5. External Interface Requirements

5.1 User Interfaces

- CLI text-based menus.
- قوائم منظمة حسب الدور:
  - **Teacher Menu:** Create Exam, Edit Exam, Delete Exam, View Results, Export CSV, Logout.
  - **Student Menu:** List Exams, Take Exam, View Past Results, Logout.
- رسائل نجاح/خطأ واضحة، وتأكيد حذف عند حذف الامتحان.

5.2 Hardware Interfaces

حاسوب عادي مع لوحة مفاتيح وشاشة.

5.3 Software Interfaces

- Python 3.x standard libraries (`sqlite3`, `argparse`, `hashlib`, `csv`, `datetime`).
- لا يوجد API خارجي في النسخة الأولى.

5.4 Communication Interfaces

لا توجد — النظام محلي.

---

6. Detailed Use Cases

أعرض بضعة Use Cases تفصيليين.

Use Case 1: Register (Student)

Actor: Student
Preconditions: لا يوجد حساب.
Main Flow:

1. Student يختار خيار Register من القائمة الرئيسية.
2. يدخل username، password، full nameوربما رقم جامعي، .
3. النظام يتحقق أن username غير مستخدم.
4. يتم حفظ المستخدم في جدول users (role='student') مع password hash.
5. عرض رسالة: "تم التسجيل بنجاح".

Postconditions: المستخدم مسجل ويمكنه تسجيل الدخول.

Use Case 2: Teacher creates Exam

Actor: Teacher
Preconditions: Teacher مسجل ودخل بحسابه.
Main Flow:

1. Teacher يختار Create Exam.
2. يدخل اسم الامتحان، وصف (اختياري)، اختيار نوع التايمر (none/per_question/whole), والمدة إن اخترت whole.
3. النظام ينشئ سجل في جدول exams ويعطي exam_id.
4. Teacher يبدأ يضيف أسئلة: لكل سؤال يدخل النص، عدد الخيارات (مثلاً 4)، نص كل خيار، ويحدد رقم الخيار الصحيح، ودرجة السؤال.
5. بعد الانتهاء يضغط Save Exam.
6. النظام يخزن الأسئلة في جدول questions مرتبطًا بالexam_id.

Postconditions: الامتحان متاح للطلاب.

Use Case 3: Student takes Exam

Actor: Student
Preconditions: Student مسجل ومسجل دخول. الامتحان متاح.
Main Flow:

1. Student يختار Take Exam ويختار من القائمة exam.
2. إذا كان للامتحان تايمر للكل: يبدأ التايمر.
3. تايمر: يبدأ تايمر السؤال per_question يعرض السؤال الأول مع الخيارات. إذا كان.
4. Student يختار خيارًا ثم يضغط Submit.
5. النظام يسجل الإجابة مؤقتًا وينتقل للسؤال التالي.
6. عند الانتهاء أو انتهاء الوقت: النظام يحسب الدرجة (مجموع درجات الأسئلة الصحيحة)، يخزن النتيجة ويعرض صفحة النتيجة مع الأسئلة الخاطئة.

Postconditions: نتيجة الطالب محفوظة وسهلة العرض.

---

7. تصميم قاعدة البيانات (ER + جداول)

الجداول الأساسية (اقتراح Schema SQL)

```
-- 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, -- teacher id
    timer_type TEXT CHECK(timer_type IN ('none','per_question','whole'))
DEFAULT 'none',
    timer_seconds INTEGER DEFAULT 0, -- used when whole or per_question
    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 (options for each question)
CREATE TABLE choices (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    question_id INTEGER NOT NULL,
    choice_text TEXT NOT NULL,
    is_correct INTEGER DEFAULT 0, -- 0=false, 1=true
    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 (to save details of answers student gave)
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)
);
```

ملاحظات:

- `choices.is_correct` يحدد أي خيار هو الصحيح.
- عند حفظ نتيجة، تحفظ صف في `results` ثم تحفظ كل إجابة في `answers` مرتبطة بـ `result_id`.

---

8. Acceptance Criteria & Test Cases (نماذج)

Acceptance criteria نماذج

- تسجيل مستخدم جديد يتم بنجاح ويُحفظ في DB → PASS.
- مدرس ينشئ امتحان ويحفظ أسئلة → PASS.
- طالب يؤدي امتحان مكوّن من 10 أسئلة وتُحسب الدرجة صحيحة → PASS.
- تايمر يعمل (whole و per_question) → PASS.
- نتيجة محفوظة وتعرض الأسئلة الخاطئة → PASS.

Test case مثال (Student takes simple exam)

- **Preconditions:** يوجد exam مع 3 أسئلة، student مسجل.
- **Steps:** login -> select exam -> answer Q1/Q2/Q3 -> finish.
- **Expected:** حساب الدرجة = مجموع درجات الإجابات الصحيحة، نتيجة محفوظة في `results`, تفاصيل الإجابات محفوظة في `answers`.

---

9. Appendix

9.1 تقسيم العمل المقترح للفريق (8 أشخاص)

1. Auth & Users — تسجيل/تسجيل دخول (2 أفراد) + hashing.
2. DB Schema & Migrations — تنفيذ جداول (1 فرد) SQLite + seed data.
3. Exam CRUD Backend — إنشاء/تعديل/حذف (2 أفراد) exams & questions & choices.
4. Student Flow & Exam Logic — تنفيذ منطق أداء الامتحان + تايمر (1 فرد).
5. Reporting & Export CSV — نتائج وتصدير (1 فرد).
6. Testing & Documentation — كتابة حالات اختبار و README(1 فرد)وSRS النهائية.

9.2 ملفات مساعدة مقترحة

- README.md: خطوات تشغيل المشروع (create venv, install requirements, run).
- seed_data.sql: ملف يحط أمثلة امتحانات وأسئلة.
- sample_csv_import_format.csv: مثال استيراد أسئلة.
