Testing Plan: QuickQuiz
Status: Draft
Authors: Robel Kebede (robel.kebede@bison.howard.edu), Trishma Garcon (Trishma.Garcon@bison.howard.edu), Xavier Green (Xavier.Green@bison.howard.edu), Dalvin Ticha (dalvin.ticha@bison.howard.edu), Caleb Orr (Caleb.Orr@bison.howard.edu)

Background

QuickQuiz is a lightweight web application where users create, and complete short trivia-style quizzes composed of multiple questions with defined correct answers. Logged-in users (via Google / Firebase Authentication) can create, view, and delete their own quizzes; all users can browse public quizzes, play them, submit answers, view their scores, and reveal correct answers.

This testing plan is based on the QuickQuiz 1-Pager, PRD, Design Doc, UI Design, and the Requirements Traceability Matrix. It describes how we validate that all requirements are met and verify that the implementation is correct and reliable.

Overview

Our testing strategy is structured around two core goals:

1. **Validation** – Confirm that QuickQuiz satisfies the requirements and behaves as users expect.
    - Use a **Requirements Traceability Matrix** linking PRD requirements → use cases → test cases.
    - Run **acceptance tests** that walk through real user flows.
2. **Verification** – Confirm that QuickQuiz is implemented correctly and robustly.
    - **Unit tests** for core logic and APIs.
    - **Integration tests** across React, FastAPI, DB, and Firebase Auth.
    - **End-to-end tests** for full user journeys.
    - **Regression tests** to keep behavior stable as code changes.

We use a mix of **automated testing** (Pytest for FastAPI, Jest/React Testing Library for frontend) and **manual testing** (UX checks, exploratory testing, acceptance tests).

The traceability matrix is maintained in a shared Google Sheet and is used as the primary artifact to show that all non-stretch requirements are covered by at least one executed test case.

Validation & Verification Process

Validation (Are we building the right thing?)

- For every Functional Requirement (FR) and Non-Functional Requirement (NFR) in the PRD:
    - We map it to one or more **Use Cases (UC)** and **Test Cases (TC)** in the traceability matrix.

- We perform **acceptance tests** where team members follow scripted user journeys:
    - Log in / log out
    - Create quiz
    - View “My Quizzes”
    - Browse public quizzes
    - Play a quiz (logged in and anonymous)
    - Submit answers and review results
    - Reveal correct answers
- Each acceptance test is marked **Pass/Fail** and linked back to requirements.

Verification (Are we building it correctly?)

- Automated tests:
    - Run on each major feature or change (CI or local).
- Manual tests:
    - Focus on UI alignment with mocks, accessibility basics, and unusual edge cases.
- Failures:
    - Recorded with a short description, root cause, and fix, tests re-run after fixes.

Unit Testing and Manual Testing

Testing Suite

a. **Backend (FastAPI) – Automated (Pytest)**
   - TC_API_QUIZ_CREATE_VALID
     Verify POST /api/quizzes with valid data creates quiz and questions.
   - TC_API_QUIZ_CREATE_INVALID
     Missing title/question/answer returns 400 with clear error.
   - TC_API_QUIZ_LIST_PUBLIC
     GET /api/quizzes returns list of public quizzes.
   - TC_API_QUIZ_MY_LIST
     GET /api/quizzes/my returns only quizzes for authenticated user.
   - TC_API_QUIZ_DELETE_OWNER
     Owner can delete their quiz; questions removed.
   - TC_API_QUIZ_DELETE_FORBIDDEN
     non-owner cannot delete another user’s quiz.
   - TC_API_SUBMIT_SCORE_LOGIC
     Scoring is correct (case-insensitive, trims whitespace, missing answers treated as incorrect).
   - TC_API_ATTEMPT_LOG
     For logged-in users, attempts are recorded correctly.

b. **Frontend (React) – Automated (Jest + React Testing Library)**

   - TC_UI_LANDING_RENDER – Landing shows intro + Browse button.
   - TC_UI_LOGIN_BUTTON – Login button present; triggers auth flow hook.

- TC_UI_CREATE_VALIDATION – Form prevents save without required fields.
- TC_UI_MYQUIZZES_RENDER – My Quizzes page lists user's quizzes from mock API.
- TC_UI_PLAY_RENDER – Play page shows questions and inputs.
- TC_UI_RESULTS_RENDER – Results show score and per-question status.
- TC_UI_REVEAL_BEHAVIOR – Reveal button shows correct answer after submission.

Manual Unit-Level Checks

- All primary buttons and links are clickable and labeled clearly.
- Pages render without visual or console errors in a modern browser.

Test Results

- All critical backend unit tests executed and **passed** after minor fixes.
- All core frontend tests for create/play/results/reveal flows **passed**.
- Notable defects found and resolved:
  - Required fields not enforced on quiz creation → added validation on client and server.
  - Initial scoring was case-sensitive → updated to normalize case and whitespace.
- No open critical unit-level defects at the time of submission.

Integration Testing

Testing Suite

Focus on communication between components and services:

- TC_INT_AUTH_PROTECTED_ROUTES
  - Valid Firebase token → can create/delete quizzes.
  - Missing/invalid token → access denied.
- TC_INT_CREATE_THEN_LIST_MY
  - Create quiz via API, then confirm it appears in "My Quizzes".
- TC_INT_DELETE_CASCADE
  - Delete quiz; ensure related questions and My Quizzes entry are removed.
- TC_INT_SUBMIT_E2E_SCORE
  - From UI, submit answers; backend score matches expected, UI displays correctly.

Test Results

- Auth + protected route behavior verified:
  - Authorized users can perform create/delete; unauthorized access blocked.
- Data consistency confirmed:
  - Created quizzes appear correctly, deleted quizzes no longer accessible.
- End-to-end score pipeline (UI → API → UI) behaves as expected.

- No unresolved integration issues at submission time.

End-to-End (System) Testing

Testing Suite

End-to-end tests simulate real user flows in a browser:

- E2E_01_FULL_FLOW_AUTHED
  - Log in → Create quiz → See it in My Quizzes → Play it → Submit → See correct score → Reveal answers → Navigate back.
- E2E_02_ANON_PLAY
  - Without logging in → Browse public quizzes → Play quiz → Submit → See score and reveals.
- E2E_03_DELETE_FLOW
  - Log in → Create quiz → Delete quiz → Confirm it disappears and direct link fails gracefully.
- E2E_04_ERROR_HANDLING
  - Simulated failure (e.g., backend down or invalid ID) → User sees friendly error, app does not crash.

Test Results

- All primary E2E flows **passed**:
  - No blockers in main user journeys.
- Confirmed behavior:
  - Anonymous users can play but cannot create/delete.
  - Navigation is consistent; users can always return to quizzes list.
- Minor UX polish items noted (non-blocking), such as wording tweaks.

Acceptance Testing

Testing Suite

Acceptance tests directly validate against PRD and user expectations:

- AT_01_CORE_REQUIREMENTS
  - Verify all PRD MVP requirements:
    - Create challenge
    - View my challenges
    - Delete my challenges
    - Complete challenges made by others
    - Reveal answers
- AT_02_UI_MATCHES_MOCKS
  - Screens and flows match agreed wireframes and PRD descriptions.
- AT_03_NEW_USER_DISCOVERABILITY

    - New user (not on dev team) can:
        - Log in
        - Create a quiz
        - Find and play a quiz
        - Understand scores and reveals
        - Without reading internal docs.

Test Results

- All core acceptance tests **passed**:
    - Observers were able to complete tasks without guidance after a short explanation of the app's purpose.
- Feedback incorporated:
    - Improved error messaging on invalid quiz creation.
    - Ensured "Back to Quizzes" button present on Results screen for clarity.

Regression Testing

Testing Suite

Regression tests ensure existing features continue to work after changes:

- Re-run:
    - Key backend unit tests (quiz create/list/delete, scoring).
    - Key frontend tests (create form, play flow, results).
    - Core integration tests.
    - Short E2E smoke test:
        - Login → Create quiz → Play → Submit → Reveal.

Test Results

- A full regression run was completed after final bug fixes:
    - All core paths remained functional.
    - No new critical defects introduced.
- Plan for future:
    - Run this regression set before each major demo or deployment.