QUIZICALLY

UNIT TESTING TEST CASE

Version 1.0
11/07/2025

VERSION HISTORY

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	Jake	11/06/2025	Khadija	11/07/2025	
2.0	Rana Abdellatif	11/21/2025	Khadija	11/21/2025	PM Approval - Revision

UP Template Version: 12/31/07

TABLE OF CONTENTS

1 INTRODUCTION

1.1 PURPOSE OF THE UNIT TESTING TEST CASE DOCUMENT

This document outlines the **Unit Testing Test Case** for Quizically. The objective is to verify that individual software components — such as scoring logic, authentication, quiz generation, and API data parsing — function as intended. Unit testing ensures code reliability, supports early bug detection, and confirms compliance with security and performance standards.

2 TEST CASE SPECIFICATION

This test case verifies that all primary functions of the Quizically backend and frontend logic behave correctly when tested in isolation using mock data and automated test frameworks.

2.1 DESCRIPTION

This test case verifies that:

- Each function (score calculation, login validation, CRUD operations) operates as expected in isolation.
- All modules meet coverage and accuracy requirements.
- Mocked dependencies simulate real system behavior accurately.

2.2 RESOURCES

Role	Responsibility
QA Reviewer	Verify code coverage and results
Developer	Write and execute unit test scripts
DevOps Engineer	Maintain CI/CD test automation pipeline
Project Lead	Review summary report

2.3 PRECONDITIONS

1. Source code complete and compiled successfully.
2. Test framework configured (PyTest/Jest).
3. Mock datasets and test accounts created.
4. CI/CD pipeline active for automatic runs.

2.4 POST CONDITIONS

1. Unit test report generated and reviewed by QA.
2. Failed tests logged with defect IDs in tracking tool.

2.5 FLOW OF EVENTS

Unit tests are organized by component. Each test verifies a single function or class in isolation using mocks.

Test Case UT-01 | Login Validation

Related Requirements:

- FS-001 - System shall enforce authentication
- UR-002 - System shall allow social/email login
- BR-017 - UI must be intuitive and maintainable

Component: validateLogin(email, password)

Step	Action	Expected Result	Requirement
1	Call validateLogin()	Returns error: “Email and password required”	BR-017
2	Call validateLogin(“user@”,”pass1”)	Returns error: “Invalid email format”	UR-002
3	Call validateLogin(“test@email.com”, ”tiny”)	Returns error: “Password too short”	FS-001
4	Call validateLogin(“wrong@email.com”, “wrongpass”)	Returns error: “Incorrect login credentials”	FS-001
5	Call validateLogin(“correct@email.com”, “correct123”)	Function returns token/session object (success)	UR-002

Test Case UT- 02 | Create New Quiz Session Component

Related Requirements:

- BR-015 — Host can create trivia games
- FIR-001 — Host dashboard supports start, pause, end, view leaderboard
- HR-004 — Questions must be displayed to users

Step	Action	Expected Result	Requirement
1	Call function with empty quizData object	Returns error: “quiz must have a title”	BR-017

2	Call function with quiz title only	Returns error: “quiz must contain at least one question”	HR-004
3	Call function with missing answer keys	Returns “invalid question format”	HR-004
4	Mock DB failure	Returns “unable to save quiz, try again”	FIR-001
5	Call function with valid structure: {title: “Animals”, questions: [...]}	Returns a new quiz ID and success flag	BR-015, FIR-001

Test Case UT-03 | Score Calculation Component

Related Requirements:

- UR-001 — System stores and displays game history
- FIR-002 — Player dashboard supports view score
- HR-004 — Leaderboard must display updated results

Component: calculateScore(userAnswers, correctAnswers)

Step	Action	Expected Result	Requirement
1	Call function with partially correct answers	Returns correct total	FIR-002
2	Call function where all answers incorrect	Returns score = 0	FIR-002
3	Call function where all answers correct	Returns max score	FIR-002
4	Mock DB history save	Verifies score is written to mock DB	UR-001
5	Mock leaderboard update trigger	Function emits “scoreUpdated” event	HR-004

Alternate Flow (errors/exceptions):

1. Test script failure - review code and mock data.
2. Coverage < 80% - add missing test cases.

2.6 INCLUSION/EXCLUSION POINTS

N/A

2.7 SPECIAL REQUIREMENTS

- Python and JavaScript test environments.
- Access to CI/CD system (GitHub Actions/Jenkins).
- Mock database (SQLite or in-memory).

Appendix A: References

The following table summarizes the documents referenced in this document.

Document Name and Version	Description	Location
Quizically Requirements Definition v1.0	Defines functional requirements for quiz access and scoring	Requirements Document
Quizically Requirements Traceability Matrix v1.0	Requirement IDs for mapping	Traceability Matrix

Appendix B: Key Terms

The following table provides definitions for terms relevant to this document.

Term	Definition
Unit Test	Verifies individual functions or classes in isolation
Mock Data	Simulated data used to test without live systems
Coverage	Percentage of source code lines tested