TEST PLAN

for

Kaiju Academy

Version 0.1

Prepared by
Group A2

YU Ching Hei	**1155193237**	**chyu@link.cuhk.edu.hk**
Lei Hei Tung	**1155194969**	**1155194969@link.cuhk.edu.hk**
Ankhbayar Enkhtaivan	**1155185142**	**1155185142@link.cuhk.edu.hk**
Yum Ho Kan	**1155195234**	**1155195234@link.cuhk.edu.hk**
Leung Chung Wang	**1155194650**	**1155194650@link.cuhk.edu.hk**

The Chinese University of Hong Kong
Department of Computer Science and Engineering

CSCI3100: Software Engineering

May 8, 2025

Contents

Contents

Document Revision History

Version	Revised By	Revision Date	Comments
0.1	C. W. Leung	2025-05-07	Initial draft for Kaiju Academy Test Plan

1 Introduction

1.1 Purpose

This document defines the test plan for the Kaiju Academy online learning platform. It describes the scope, objectives, test cases, resources, approach, schedule, risks, and reporting methods for the project.

1.2 References Acknowledgments

- Kaiju Academy Software Requirements Specification v1.1
- Kaiju Academy Design and Implementation v1.1
- This document was prepared with the assistance of AI tools (e.g., ChatGPT 4.1) for drafting and review.

2 Scope and Objectives

2.1 Scope

Testing will cover the following key areas of Kaiju Academy:
- User registration, login, authentication (including MFA)
- Course browsing, enrollment, and access
- Interactive code assessment and automated grading
- Profile and progress tracking (student and educator)
- Educator course and material management
- Error handling, security, and data integrity
- Responsive UI/UX across devices (desktop, tablet, mobile)
- Licence management: entering and validating a licence key before accessing course content.

2.2 Out of Scope

- Community features (forum, dashboard, notifications, calendar)
- Credit/Payment purchasing and related flows
- Modding/hacking scenarios and hardware-specific edge cases
- Third-party payment gateway/internal payment logic

2.3 Objectives

- Verify that all main user stories and requirements are implemented and work as intended
- Ensure the platform meets performance, usability, and security expectations
- Confirm the system is stable and ready for release
- Validate integration with external modules (SurrealDB, AWS)

3 Test Cases and Scenarios

3.1 Functional Test Cases

1. **User Registration and Login**
    - Steps: Register with email, verify email, login with password (with/without MFA)
    - Expected Result: Account created, verification email sent, login succeeds or fails as appropriate
    - Pass/Fail: Account appears in database, valid JWT issued, protected endpoints accessible after login

2. **Course Browsing and Enrollment**
    - Steps: Browse course catalog, enroll in available courses
    - Expected Result: Enrollment successful, course appears in user's profile
    - Pass/Fail: Enrolled courses listed, correct access to course content

3. **Interactive Code Assessment**
    - Steps: Access code editor, submit code, receive auto-grading and result
    - Expected Result: Code executes, output and feedback displayed, grade recorded
    - Pass/Fail: Output matches expected, grade visible in profile

4. **Educator Course Management**
    - Steps: Create/update/delete course, upload materials
    - Expected Result: Changes reflected for students, materials accessible
    - Pass/Fail: CRUD operations function, permissions enforced

5. **Profile and Progress Tracking**
    - Steps: Complete modules, view progress page
    - Expected Result: Progress updates in real time
    - Pass/Fail: Progress bar/percentage accurate, completed modules listed

6. **Licence Management**
    - Steps: Attempt to access system features without a licence key; enter invalid key; enter valid key.
    - Expected Result: Access denied until a valid key is provided; valid key grants access.
    - Pass/Fail: System restricts access appropriately; error messages are clear.

3.2 Non-Functional Test Cases

3.2.1 Performance Testing

- Home page, login, and course page load within 2 seconds for 95% of users
- Code execution returns result within 5 seconds for 99% of cases

3.2.2 Security Testing

- Only authenticated users can access protected endpoints
- Role-based access control enforced (e.g., only educators can create courses)
- No sensitive data stored in plain text; JWT securely signed

3.2.3 Usability Testing

- Navigation is clear and intuitive for both students and educators
- UI is responsive on mobile and desktop devices

3.2.4 Reliability Testing

- System recovers from AWS or DB failure without data loss
- Error messages are meaningful for invalid operations

3.2.5 Compatibility Testing

- Supported browsers (Chrome, Firefox, Safari, Edge) render all pages correctly
- All main functions work on Windows, macOS, Android, iOS

4 Resource Allocation

4.1 Team Roles and Responsibilities

Role	Name	Responsibilities
Backend Developer	C. H. Yu	Assist with unit/integration test, bug fixing
Cloud Engineer	Ankhbayar	Deploy project to AWS, configure hosting and cloud resources
UI/UX Designer	H. T. Lee	Validate user interface and accessibility
Frontend Developer	H. K. Yum	Develop and maintain user interface; assist with unit/integration testing and bug fixing
Documentation Specialist	C. W. Leung	Maintain documentation, reports
Product Owner	Group A2	Validate requirements and acceptance criteria

4.2 Tools and Software

- Test Management: GitHub issues, test cases in code repository
- Automation: Cypress (UI), Jest (unit), custom scripts
- Bug Tracking: GitHub, Jira
- Performance: Browser dev tools, Lighthouse
- Compatibility: BrowserStack, manual device testing
- CI/CD: GitHub Actions

4.3 Testing Environments

Environment	Purpose	Owner
Development	Unit testing, feature development	Developers
Staging	System/integration testing, regression	QA Team
Production (UAT)	Final acceptance, release candidate	Product Owner

4.4 Time and Budget Estimation

- Test Planning: 10%
- Test Case Development: 15%
- Test Execution: 50%
- Bug Fixing/Retesting: 20%
- Regression: 5%
- Budget: Team time, device access, third-party tools (if any)

5 Testing Approach

5.1 Types of Testing

- **Unit Testing**: Rust/TypeScript unit tests for backend/frontend logic
- **Integration Testing**: End-to-end user flows (registration, course enrollment, code submission)
- **System Testing**: Full workflow from user registration to course completion
- **Regression Testing**: After each major code change
- **User Acceptance Testing**: Final validation by product owner/instructors

5.2 Methodologies

- Manual testing for UI/UX and major user stories
- Automated testing for regression and repetitive flows
- Exploratory testing for edge and negative cases

6 Timeline and Schedule

6.1 Waterfall Model Example

Phase	Duration	Activities
Test Planning & Preparation	Week 1	Test plan, environment setup
Unit Testing	Week 2	Backend/frontend unit tests
Integration Testing	Week 3	End-to-end flow testing
System Testing	Week 4	Full platform testing
UAT & Regression	Week 5	User acceptance, bug fixing, regression

6.2 Agile/Sprint Model (If Used)

- Each sprint: Write user-story based tests, run regression, demo to stakeholders

7 Risk Assessment and Mitigation

7.1 Risk Analysis

- **Delays in Development**: Frequent communication, adjust test schedule
- **High Bug Volume**: Prioritize critical bugs, allocate more QA time
- **Integration Failures**: Early integration testing, CI/CD monitoring
- **Compatibility Issues**: Early device/browser testing, use BrowserStack
- **Untestable Features**: Add debug/logging where possible, clarify requirements

8 Success Criteria

8.1 Acceptance Criteria

- All high/critical bugs are resolved
- All functional and non-functional requirements pass
- Positive feedback from UAT/instructors
- System is ready for deployment

9 Reporting Requirements

9.1 Reporting Methods

- Test results and bug status tracked via GitHub issues
- Test execution summary and coverage report before release
- Weekly status meetings with stakeholders
- Final test summary and sign-off document