CSCI3100: Software Engineering

Test Plan of AIO - Course Planner

April 23, 2025

Table of contents

0.1 Document Revision History

Version	Revised By	Revision Date	Comments
1.0	Anson	23 Apr 2025	Initial draft

1 Test Plan for AIO - Course Planner

This is a test plan for **AIO - Course Planner**. The most important sections include:

1. Scope and Objectives — This part states what will be (and what not will be) included in the testing.
2. Test Cases and Scenarios — This part explains each test case and scenario, by providing a detailed breakdown of the scope, the testing procedures and pass/fail criteria.
3. Team Roles and Responsibilities — This part shows the assignment of specific duties and accountabilities to each team member.
4. Testing Approach & Timeline and Schedule — This part outlines the approach and schedule of the tests.

1.1 1. Scope and Objectives

This part defines the scope of test cases. By having a clear definition, it avoids multiple test cases overlapping on certain areas.

1.1.1 Scope:

The test plan focuses on ensuring the functionality, performance, and user experience of the **AIO - Course Planner**. The following areas will be tested:

- Core mechanics (e.g., Timetable planning, Study Plan planner, Viewing degree roadmap, Course commenting).
- User interface (UI) and user experience (UX).
- Account system (Login, Logout, Registration).
- Email token verification module.
- Save/load functionality (for Timetable and Study Plan).
- Third-party integration to Google Calendar.

1.1.2 Out of Scope:

- Some hard-to-test aspects of the software (please see Risk assessment and mitigation).

1.1.3 Objectives:

- Verify that the system meets functional and non-functional requirements, and provides a bug-free experience.
- Ensure the system is intuitive, and free of critical bugs.
- Confirm that the system performs well under various conditions (e.g., high user load, low-end devices). (TODO: FIX ????) [draft: The system can support around 100 concurrent users, which can be verified with Selenium testing.]
- Validate that the system is ready for release and meets quality standards.

1.2 2. Test Cases and Scenarios

The section outlines comprehensive testing strategies for the **AIO - Course Planner** web application. It has test cases for both functional and non-functional test scenarios designed to ensure the application meets quality standards and user requirements. Each test case includes procedures, expected results, and criteria to guide manual testing efforts into the testing process.

The test cases below use manual testing steps currently, which can serve as the foundation for future test automation efforts.

1.2.1 Test Cases for Functional Requirements:

1. User Authentication:
    - Steps: Register a new account, verify email, and login with credentials.
    - Expected Result: User is registered successfully and can log in without email verification (it is required for course commenting only).
    - Pass/Fail Criteria: Account is not created in database or users cannot log in.
    - Test cases: (TODO: DELETE THIS? NO AUTOMATED TESTING IS AVAILABLE)
        - Test Case ID: TC001
            - Repository Link: GitHub - Test Code for TC001
        - Test Case ID: TC002
            - Repository Link: GitHub - Test Code for TC002
2. Course Search and Selection:
    - Steps: Search for courses by keyword, filter results, and view course details.
    - Expected Result: Relevant courses are displayed and details are accessible.
    - Pass/Fail Criteria: Course search returns accurate results and course information is complete.

3. Study Plan Creation:
    - Steps: Add courses to study plan, modify sequence, and save changes.
    - Expected Result: Study plan is updated and persisted correctly.
    - Pass/Fail Criteria: Changes to study plan are saved and retrieved accurately on reload.
4. Timetable Management:
    - Steps: Add courses to timetable, detect conflicts, and optimize schedule.
    - Expected Result: Courses appear in timetable at correct times with conflict warnings.
    - Pass/Fail Criteria: Timetable visually displays all courses without UI errors.
5. Comment System:
    - Steps: Add comments to courses, edit/delete own comments, and view others' comments.
    - Expected Result: Comments are saved and displayed correctly for all users.
    - Pass/Fail Criteria: Comment operations complete without errors and update in real-time.
6. Degree Roadmap Visualization:
    - Steps: View degree roadmap, interact with course nodes, and check prerequisites.
    - Expected Result: Roadmap displays correct course sequence with prerequisites highlighted.
    - Pass/Fail Criteria: All course relationships are accurately represented in the visualization.

1.2.2 Test Cases for Non-Functional Requirements

This part focus on test cases that assess the non-functional requirements focus on the quality attributes of the performance, scalability, usability, reliability, and security.

1.2.2.1 1. Performance Testing

1. Page Load Time
    - Objective: Verify that pages load within acceptable time limits.
    - Steps:
        1. Navigate to each main page (index, timetable, study planner, etc.)
        2. Measure the time taken to fully load content.
    - Expected Result: Pages load within 2 seconds on standard connections.
    - Pass/Fail Criteria: Pass if load time meets expectations; fail otherwise.

2. API Response Time

   - Objective: Ensure API endpoints respond quickly to client requests.
   - Steps:
     1. Make requests to key API endpoints (course search, authentication, etc.)
     2. Measure response times under different load conditions.
   - Expected Result: API responses return within 500ms on average.
   - Pass/Fail Criteria: Pass if response times remain within acceptable thresholds.

3. Simultaneous User Testing

   - Objective: Test application behavior with many concurrent users.
   - Steps:
     1. Simulate multiple users accessing the application simultaneously.
     2. Monitor server resource usage and response times.
   - Expected Result: Application remains responsive with up to 100 simultaneous users.
   - Pass/Fail Criteria: Pass if performance degradation is minimal under load.

1.2.2.2 2. Usability Testing

1. Navigation Flow

   - Objective: Verify that users can navigate the application intuitively.
   - Steps:
     1. Ask new users to complete common tasks without instructions.
     2. Observe navigation patterns and points of confusion.
   - Expected Result: Users can complete basic tasks without assistance.
   - Pass/Fail Criteria: Pass if users accomplish tasks efficiently; fail if significant confusion occurs.

2. Mobile Responsiveness

   - Objective: Ensure the application is usable on mobile devices.
   - Steps:
     1. Access application on various mobile devices and screen sizes.
     2. Test all major features on mobile browsers.
   - Expected Result: UI adapts appropriately to screen size with all functionality accessible.
   - Pass/Fail Criteria: Pass if mobile experience is consistent with desktop.

1.2.2.3 3. Reliability Testing

1. Data Persistence

- Objective: Verify that user data is saved correctly and persistently.
- Steps:
  1. Create study plans and timetables, then log out and back in.
  2. Check that all user-created data is preserved.
- Expected Result: All user data is persistent across sessions.
- Pass/Fail Criteria: Pass if no data loss occurs; fail otherwise.

2. Error Recovery

- Objective: Test application's ability to handle and recover from errors.
- Steps:
  1. Simulate network disruptions during operations.
  2. Intentionally submit invalid data to test error handling.
- Expected Result: Application provides meaningful error messages and recovers gracefully.
- Pass/Fail Criteria: Pass if app maintains usability after errors.

1.2.2.4 4. Security Testing

1. Authentication Security

- Objective: Verify secure login and session management.
- Steps:
  1. Attempt various authentication bypass techniques.
  2. Test password strength requirements and account lockout functionality.
- Expected Result: Unauthorized access attempts are prevented.
- Pass/Fail Criteria: Pass if authentication barriers cannot be circumvented.

2. Data Protection

- Objective: Ensure user data is protected.
- Steps:
  1. Inspect network traffic for sensitive data transmission.
  2. Test that API endpoints properly validate authentication.
- Expected Result: Sensitive data is encrypted and access-controlled.
- Pass/Fail Criteria: Pass if no unauthorized data access is possible.

1.2.2.5 5. Compatibility Testing

1. Browser Compatibility

- Objective: Ensure functionality across major browsers.
- Steps:

    1. Test application on Chrome, Firefox, Safari, and Edge.
    2. Verify feature parity and visual consistency.
  - Expected Result: Application works identically across all supported browsers.
  - Pass/Fail Criteria: Pass if no browser-specific issues exist.

2. Device Compatibility

  - Objective: Verify application works on different devices and screen sizes.
  - Steps:
    1. Test on desktop, tablet, and mobile devices.
    2. Check for layout or functionality issues.
  - Expected Result: Application is fully functional across all device types.
  - Pass/Fail Criteria: Pass if the experience is consistent across devices.

1.3 3. Resource Allocation

This section outlines the personnel, tools, environments and other resources required to execute the test plan effectively.

1.3.1 1. Team Roles and Responsibilities

1.3.1.1 Key Team Members: (TODO: Fix this part cuz we just share the work...)

1. Test Lead:

  - Oversees the entire testing process.
  - Creates test plans.
  - Reports progress to stakeholders.
  - Assigns tasks to team members and monitors progress.

2. Manual Testers:

  - Execute functional and non-functional test cases.
  - Document results.
  - Identity and report bugs.

3. Frontend Developer:

  - Assists with UI/UX testing.
  - Validate the user interface and user experience.
  - Resolves front-end related issues.

4. Backend Developer:

  - Assists with API testing.

- Resolves server-side issues.

5. Database Administrator:
    - Manages database configurations.

1.3.1.2 Resource Allocation Table: (TODO: Fix this)

Role	Name	Responsibilities	Contribution (%)
Test Lead	Cheung Wai Lok	Oversee testing, coordination, progress tracking	12.5%
Manual Testers	All members	Performance testing, log bugs	12.5%
Frontend Developer	Lam Leung Yiu	Validate UI/UX and accessibility	12.5%
Backend Developer	Cheung Ching Yu	Validate features and resolve identified issues	12.5%
Database Administrator	Liu Yun Hei	Database Design and Maintenance	12.5%

1.3.2 2. Tools and Software

1.3.2.1 Testing Tools:

- Browser Developer Tools: Chrome DevTools for frontend testing.
- MongoDB Compass: For database inspection and verification.
- Manual Testing: Since no framework is used for webapp building.
- Git: For version controlling test scripts/datasets/files and documents.

1.3.3 3. Testing Environments

1.3.3.1 Environment Types:

- Virtual Machine (Docker): Both developing, testing and production environments are containerized by Docker to avoid endpoint discrepency.
- Developing environment: Consists of test files and source codes, used by developers for developing, testing and debugging.
- Production Environment: Used for final validation during User Acceptance Testing. Mimics the live environment to ensure the system is ready for release.

1.3.3.2 Environment Setup:

- Hardware Requirements:
  - Devices/Browsers/DevTool mimicing to test behaviour of the web-application in desktop and laptop environments.
- Software Requirements:
  - Docker (Compose) Containers: For consistent testing environments.
  - Node.js: The backend of the web-application.
  - MongoDB Database: Stores user data and comments.

1.3.3.3 Environment Allocation Table: (TODO: Fix this)

Environment	Purpose	Configuration	Responsibility
Local	Individual feature testing	Local installations on each member's computer	Individual Team Members
Shared Test	Coordinated testing	Docker compose setup accessible to all members	Group Leader + Team Member with Docker knowledge
Production-like	Final validation	Deployment to free tier hosting (if available)	Group Leader

1.3.4 4. Time Allocation

1.3.4.1 Effort Estimation: (TODO: Fix this later)

- Effort Estimation:
  1. Test Planning: 3-4 days teseting
  2. Test Case Development: 1 week
  3. Functional Testing: 1-2 weeks
  4. Non-functional Testing: 1 week
  5. Bug Fixing and Verification: Ongoing throughout project

1.3.4.2 Time Allocation Table: (TODO: Fix this later)

Testing Phase	Duration	Team Members Involved	Deliverables
Test Planning	3-4 days	All team members, led by Group Leader	Test Plan Document
Test Case Development	1 week	All team members	Test Cases, Test Data
Authentication Testing	2-3 days	Team Member 1, supported by others	Test Results, Bug Reports
Course Management Testing	2-3 days	Team Member 2, supported by others	Test Results, Bug Reports
Study Plan & Timetable Testing	3-4 days	Team Member 3, supported by others	Test Results, Bug Reports
UI/UX Testing	2-3 days	Group Leader, supported by others	Test Results, Bug Reports
Performance & Security Testing	2-3 days	Shared responsibility	Performance Report
Compatibility Testing	1-2 days	Team Member 2, supported by others	Compatibility Report
Bug Fixing & Verification	Ongoing	All team members	Updated Application

1.3.5 5. Budget Allocation

1.3.5.1 Key Budget Considerations:

- Time Investment: Primary resource is team members' time.
- Free Tools: Utilizing free versions instead of premium versions of software/services.
- Personal Computing Resources: Using team members' own computers.
- Cloud Resources: Minimal use of free tier cloud services if needed (e.g. using `EtherealMail` + `NodeMailer` to mimic email sending).
- No Monetary Budget: Focus on free resources and time allocation.

1.4 4. Testing Approach

1.4.1 Types of Testing:

- Unit Testing: Validate individual components of the **AIO - Course Planner** (e.g., authentication module, course search functionality, comment system).
- Integration Testing: Test interactions between different modules (e.g., how adding courses affects timetable display, how study plan interacts with degree roadmap), from developers' point of view.

- System Testing: Verify the application as a whole from the user's perspective, including complete workflows like creating a study plan or generating a timetable.
- UI/UX Testing: Ensure the interface is intuitive, responsive, and functions correctly across all pages.
- Performance Testing: Evaluate load times, API response times, and behavior under concurrent user scenarios.
  - Page load time should be under 2 seconds.
  - API responses should take less than 500 milliseconds.
  - The app should support 100 concurrent users without crashing or slowing down significantly.
  - (TODO:[draft: Define performance targets (e.g., page load <2s, API response <500ms, handle 100 users). Document findings and bottlenecks for improvement.]
- Security Testing: Verify authentication mechanisms, user data protection, and access controls.
- Compatibility Testing: Confirm functionality across different browsers and devices.
- Regression Testing: Ensure that new features or bug fixes don't break existing functionality.

1.4.2 Methodologies:

- Manual Testing: Primary approach for functionality verification, user experience, and exploratory testing since no testing framework is being used for the web application.
- Browser Developer Tools: For frontend testing, performance analysis, and mobile responsiveness verification.
- Database Verification: Using MongoDB Compass to verify data persistence and integrity.
- Exploratory Testing: To discover edge cases and unexpected behaviors, particularly in complex workflows like timetable conflict detection. (TODO: ??) [draft: Document steps, findings, and edge cases discovered during each session.]
- Environment-Based Testing: Using Docker containers to ensure consistency across development, testing, and production environments.

1.5 5. Timeline and Schedule (TODO: This part NEEDS SEVERE FIX)

1.5.1 1. Agile Model

In our Agile model, testing is integrated throughout the development process with iterative cycles. Each sprint incorporates testing activities for newly developed features while maintaining regression testing for existing functionality.

1.5.1.1 Dependencies:

- Completion of feature development within each sprint.
- Availability of testing environments (Docker containers + Compose environment settings).
- Access to MongoDB for data verification.
- Timely resolution of reported bugs.

1.5.1.2 Key Phases and Timeline:

1. Sprint Planning and Preparation: Day 1 of each sprint

- Define acceptance criteria for course planner features (timetable, study plan, etc.)
- Create test cases for new functionality
- Set up test data and environments in Docker

2. Testing During the Sprint: Ongoing (Daily)

- Execute test cases for completed features
- Perform exploratory testing on UI/UX elements
- Log bugs in course search, timetable conflicts, study plan persistence
- Conduct regression testing for authentication and previously implemented features

3. Mid-Sprint Testing Milestones: Midpoint of each sprint

- Perform integration testing between modules (e.g., how course selection affects timetable)
- Update test cases based on interim feedback and UI/UX adjustments
- Verify data persistence in MongoDB

4. End-of-Sprint Testing Activities: Last 1–2 days of each sprint

- Conduct user acceptance testing for completed features
- Perform cross-browser compatibility testing
- Execute performance tests for API response times and page loads

5. Post-Sprint Activities: After each sprint

- Update the test suite with new test cases
- Document known issues and workarounds
- Plan testing activities for the next sprint
- [draft: update the progress of the last sprint]

1.5.1.3 Agile Model Timeline for a 2-Week Sprint:

Day	Activity
Day 1	Sprint planning, define acceptance criteria for new course planner features
Day 2	Prepare test data (course information, schedules), configure test environment
Day 3–4	Test user authentication, course search, and selection features
Day 5	Integration testing between course selection and timetable visualization
Day 6–7	Test study plan creation and persistence, verify email token functionality
Day 8	Test Google Calendar integration, verify data export functionality
Day 9	Perform UI/UX testing on multiple browsers and devices
Day 10	Sprint review/demo, verify all completed features work in the integrated environment
Day 11	Regression testing for core features (authentication, timetable, study plan)
Day 12	Document test results, update test cases, collaborate on next sprint planning

[darft:] | Day | Activity | |——|————————————————————————| | Day 1 | Set sprint goals and clarify acceptance criteria for new course planner features | | Day 2 | Prepare and validate test data, and set up the test environment | | Day 3 | Complete testing of user authentication functionality | | Day 4 | Finish testing course search and selection features | | Day 5 | Integrate and test course selection with timetable visualization | | Day 6 | Test study plan creation feature | | Day 7 | Ensure study plan data is saved and persists correctly | | Day 8 | Test email token generation and verification | | Day 9 | Complete testing of Google Calendar integration | | Day 10 | Verify data export functionality | | Day 11 | Conduct UI/UX testing across different browsers and devices | | Day 12 | Review and demo all completed features in the integrated environment | | Day 13 | Perform regression testing on core features and document results | | Day 14 | Gather feedback, hold sprint retrospective, and plan improvements for the next sprint |

[draft:] | Day | Activity | |——-|———————————————————————————-| | Day 1 | Set sprint goals and clarify acceptance criteria for new course planner features | | Day 2 | Prepare and validate test data, and set up the test environment | | Day 3 | Complete testing of user authentication functionality | | Day 4 | Finish testing course search and selection features | | Day 5 | Integrate and test course selection with timetable visualization | | Day 6 | Test study plan creation feature | | Day 7 | Ensure study plan data is saved and persists correctly | | Day 8 | Test email token generation and verification | | Day 9 | Complete testing of Google Calendar integration | | Day 10| Verify data export functionality | | Day 11| Conduct UI/UX testing across different browsers and devices | | Day 12| Review and demo all completed features in the integrated environment | | Day 13| Perform regression testing on core features and document results | | Day 14| Gather feedback, hold sprint retrospective, and plan improvements for the next sprint |

1.6 6. Risk Assessment and Mitigation

1.6.1 Potential Risks:

1. Data Loss in Study Plans or Timetables:
    - Risk: Users could lose carefully created course plans due to system errors, browser issues or network issues.
    - Mitigation: Implement auto-save functionality that makes use of browser's `localStorage` module or cookie sessions.
2. Course Data Inconsistency:
    - Risk: Course information (schedules, prerequisites, descriptions) may be outdated.
    - Mitigation: Implement auto data update functionality for backend/database system.
3. Browser Compatibility Issues:
    - Risk: For example, maybe drag-and-drop functionality for course planning might not work consistently across all browsers.
    - Mitigation: Refactor whole codebase using frontend framework in future.
4. Email Verification Challenges:
    - Risk: Email verification tokens might not be delivered or processed correctly.
    - Mitigation: Implement fallback verification options, and token resending functionality (This is implemented).
5. Performance Under Load:
    - Risk: System may slow down when handling many concurrent users or complex study plans.
    - Mitigation: Optimize database queries, implement pagination where appropriate, and use profiling tools to analyze functions.
6. Intrinsic Untestability of Course Conflict Detection:
    - Risk: Complex schedule conflict detection algorithms may be difficult to test comprehensively.
    - Mitigation: Create a suite of test cases covering various conflict scenarios, implement verbose logging in development mode, and conduct extensive manual testing of edge cases.
7. Google Calendar Integration Issues:
    - Risk: Exported timetable data might not format correctly for Google Calendar imports.
    - Mitigation: Rigorously test the CSV export format against Google Calendar import specifications with various course combinations.

1.7 7. Success Criteria

The AIO - Course Planner testing will be considered successful when the following criteria are met: 1. Functionality Objectives: - All critical and high-priority bugs are resolved. - All core features (timetable planning, study plan creation, course search, etc.) function as specified. - Course information is displayed accurately and completely. - Drag-and-drop course placement works reliably across supported browsers. - Email verification system functions correctly. - Google Calendar export produces valid, importable files.

2. Performance Objectives:
    - Page load times are within acceptable limits (under 2 seconds for main pages).
    - API responses return within 500ms on average.
    - Application remains responsive with multiple concurrent users.
    - Course search returns results quickly (within 1 second).
3. User Experience Objectives:
    - Positive feedback from usability testing participants.
    - New users can complete basic tasks without assistance.
    - UI adapts appropriately to different screen sizes.
    - Course data is visually clear and understandable.
4. Security Objectives:
    - User authentication works securely.
    - User data is properly protected (email and passwords etc.).
    - Email verification process is secure.
5. Reliability Objectives:
    - Study plans and timetables are saved and retrieved accurately.
    - No data loss occurs during normal usage.

1.8 8. Reporting Requirements

1.8.1 Documentation: (TODO: FIX THIS)

- Test Case Documentation: Detailed description of each test case in the GitHub repository, [draft: Notion workplace and discord], including steps to reproduce, expected results, and actual results.
- Bug Reports: Standardized bug reports with severity levels, steps to reproduce, screenshots or videos, and expected vs. actual behavior.
- Feature Test Coverage Reports: Documentation showing which features have been tested and their pass/fail status. [darft: Highlight untested or partially tested areas for transparency. Markdown the status such as Not started, On-hold, In progress and Done]

- Test Data: Sample course data, users, and study plans used for testing.
- Environment Configuration: Documentation of the Docker setup used for testing. [draft: Provide clear instructions for setting up and running the test environment on README.md.]
- Testing Checklists: For manual tests of core user flows like course planning and timetable creation.

1.8.2 Communication:

- Frequent Testing Updates/Meetings: Brief updates on testing progress, including the number of tests executed and bugs found.
- Issue Tracking: Using Notion dashboard to track and manage bugs, with clear assignments and status updates.
- User Acceptance Testing Feedback: Collection and reporting of feedback from UAT participants.
- Final Test Summary Report: Comprehensive report before release, detailing test coverage, outstanding issues, and recommendations.