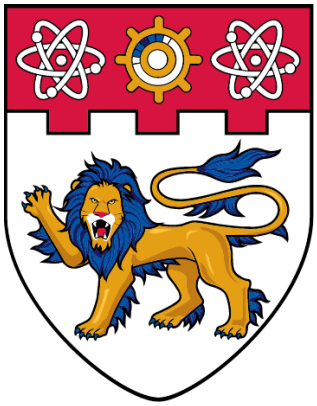

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Test Plan for “Go Where GaiGai” Web Application

Version: 1.0
Last Modified: 25/10/2022

Revision History

Version	Implemented By	Revision Date	Approved By	Approval Date	Description of Changes
1.0	RuiMin, Wee Kiat	25/10/22	Eugene Lim	11/3/22	Initial Test Plan

Table of Contents

1 Test Plan Identifier

The test plan for Go Where GaiGai contains the scope, approach and schedule of the testing activities to be conducted for Go Where GaiGai throughout its development lifecycle as well as during the maintenance phase. This is the initial version of the document. The test plan is currently written at the second level of the master plan checklist in software development. All fundamental testing criteria and test cases have been created and documented in the *Test Cases* documentation. The software's basic functionality has also been implemented and tested in the software application. This test plan will transition into the master plan further in the Software Development Life Cycle (SDLC) as the testing becomes more robust and complete.

The plan contains information about the resources and procedures required to test Go Where GaiGai adequately, and how the testing process is controlled. This documentation also serves as a supplement to the *Change Management Plan* to describe how the project is managed throughout Go Where GaiGai's SDLC.

2 Introduction

The test plan aims to provide a comprehensive overview of the scope, approach, resources and schedule of all testing processes and activities to be performed for Go Where GaiGai. The plan identifies and investigates the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan.

Four levels of testing will be carried out. Unit Testing, Integration Testing, System Testing from the V-model of Test Levels. User Acceptance Tests will also be done to gain measurable feedback from end-users of the software. Specifically, the testing done in this project will be carried with the black-box approach.

1. **Unit Testing**: Unit testing will be performed to test the individual units or components of Go Where GaiGai in isolation. Individual units will be scrutinized to test and determine if they work as intended before their integration into the main system. If done correctly, unit testing aims to detect early flaws in code which may be more difficult to find in later testing stages. Unit testing will be performed during the development phase as the Development Team writes the components of Go Where GaiGai.

2. **Integration Testing**: Integration tests will put together the different components of Go Where GaiGai and logically test them as a group. The tests will evaluate the compliance of the sub-systems within the software with specific functional requirements. Defects in the interaction between the components will be exposed in this phase.

3. **System Testing**: System testing aims to validate the complete and fully integrated software product. Go Where Gai Gai will be tested with external peripherals in order to check how components interact with one another in end-to-end testing scenarios. Thorough testing of possible inputs in the application will be checked to verify for the desired outputs. System tests will be conducted once all components required for the application to meet functional and non-functional requirements have been implemented.

4. **User Acceptance Testing**: User acceptance tests will be the final phase of testing performed before Go Where GaiGai goes live. Compared to system testing, the software will be tested in real-world situations, validating changes made and assessing adherence to the stakeholder's business requirements.

3 Test Items (Functions)

The following functional test items are identified for various testing techniques:

3.1 Unit Testing

3.1.1 **Import existing planner**: The planner page allows the user to import and export a schedule into a text file. The user should be able to import a planner into Go Where GaiGai, if he/she has a valid text file for import.

3.1.2 **Export planner**: The planner page allows the user to import and export a schedule into a text file. The user should be able to click on the "Export" button and begin a download of the text file.

3.1.3 **Remove existing plans**: The planner also allows the user to clear the page of any plans. The user should be able to remove at least one existing record on the planner page by clicking on the "bin" button on the page.

3.1.4 **Search location from View Map**: The View Map page allows the user to interact with a map, and perform searches on dining/leisure locations. The user should be able to enter the page and view the map. They should also be able to type in a category of choice, address and/or postal code. The map will then search for and display the specified location.

3.1.5 **Add to planner function from View Map**: The user can add a location from the View Map page to the Planner page. The user should be able to click on a location after a search has been performed, and add that location to the Planner page by clicking on a "add to planner" button. The location is displayed on the Planner page in a schedule.

3.1.6 **Add to planner from View Locations**: The user can add a location from the View Locations page to the Planner page. The user should be able to click on a location card on the View Locations page. He/she should then be able to click on an "Add to planner" button to move it into a plan on the Planner page.

3.2 Integration Testing

3.2.1 **Session Data Sync**: Similar to unit test 3.1.5, the user should be able to add a location between the two pages. This test case ensures the View Map component and Planner components work together appropriately. The location should be transferred with the correct details.

3.2.2 **Session Data Sync**: Similar to unit test 3.1.6, the user should be able to add a location between the two pages. This test case ensures the View Locations component and Planner components work together appropriately. The location should be transferred with the correct details.

3.2.3 **Imported Data Sync**: The Planner page should be able to store the plan a user has imported in the current session. The user should be able to import a valid plan text file, navigate between the other two pages (View Map, View Locations) and return to the Planner page with the plan still loaded.

3.2.4 **Import/Export Data Sync**: The Planner page should not only be able to create, import and export a plan. It should also be able to make changes to a plan by removing, adding and updating existing records in a plan previously imported by the user without error.

3.3 System Testing

3.3.1 **Smoke Test**: Smoke testing is a software test to determine if a system is stable or not. The user should be able to access the website and access its features provided their machine satisfies the system requirements as specified in the *System Requirements Specifications* document. This includes an internet connection.

3.3.2 **Stress Test**: Stress testing is a software test to determine if a system is stable or not. Multiple users should be able to open a session on Go Where GaiGai, maintain the session and be able to access features of the website. The stress test will be conducted with 15,000 users.

3.3.3 **Scalability Test**: The scalability test ensures the back-end database used is able to function smoothly after being deployed for extended periods of time in a real-world scenario. The back-end API will be tested with 15,000 users. The database is expected to still be able to perform queries initiated by user interaction with the website's search algorithm.

3.3.4 **Security Test**: The system should be protected from basic malicious activity. The website should not import a malicious plan text file from a user.

4 Software Risk Issues

Some of the potential software risks which could be faced are:

1. Potential changes to third party tools, libraries and/or APIs might impact certain functions of Go Where GaiGai critical for the operation of the website.
2. Ability for existing and incoming members of the Development Team to understand new packages and tools.
3. Maintenance of certain complex functions.
4. Modification to components with a past history of failure or lack of updates.
5. Poorly documented high-level design architecture, software modules or change requests.

Unit testing will help to identify potential areas within the software that are risky. If the unit tests discover a large number of defects or a tendency toward defects in a particular area of the software, this will indicate areas where future problems might potentially occur. The team would then have to plan an approach to resolve such issues by defining the risks during weekly meetings during the development of Go Where GaiGai.

5 Features To Be Tested

The components of the system have been identified, and each component will be given a risk rating, which is one of three levels: High, Medium and Low. High level risks have the highest priority in testing and debugging. Medium risks features are to be tested only after high level defects, followed by Low level features. The components of Go Where GaiGai and their respective risk ratings are as shown below:

Item	Risk Rating
Homepage (Landing Page)	High
View Map	High
View Locations	Medium
Planner	Medium
Back-end Database	High

Table 1: System Component Risk Rating

6 Features Not To Be Tested

All features implemented in the application will be tested. This is because all features in Go Where GaiGai are high in priority necessary for the application to be operationally ready.

7 Approach

A test approach specifies how testing will be performed, and the test strategy the team will adopt during the implementation of the project. A test approach has two techniques:

Reactive: Reactive approach manages the issues once they emerge or being encountered, without prior arrangements on how to, what to, when to, and whom to report. In this approach, design and coding are completed first before the tests are conducted.

Proactive: A proactive approach includes planning for the future, taking into consideration the potential problems that on occurrence may disrupt the orders of processes in the system. Future threats are recognized and prevented with requisite actions and planning. In this approach, tests are designed as early as possible with an aim to find and fix any defects before the build is fully created.

7.1 Testing Approaches

Apart from reactive and proactive techniques, a test approach based on different context and perspective may be categorized into the following types:

- **Methodological Approach**
  This approach consists of all the methods which have been pre-determined and pre-defined, to carry out testing. The methods covered under this approach are used to test a software product from each different perspective and requirements, ranging from static analysis of the programming code to dynamic testing of the application.
- **Analytical Approach**
  Analytical approaches involve, selecting and defining the approaches based on the analysis of some factors or conditions associated with the software product, which may produce some significant changes to the testing environment, such as on requirement basis, risk based defect severity basis, defect priority basis, etc. For example, when defining and preparing the test approach based on risk, it may include an approach or method to consider the areas of higher risks, for the execution under the testing phase, whereas that of lower risk may be taken up at a later stage.

- **Regression Averse Approach**
  A regression averse strategy may involve automating functional tests prior to the release of the function, in which case it requires early testing. However, the tests are almost entirely focused on testing functions that have already been released, which is in some sense a form of post release test involvement.

- **Model-based Approach**
  Model-based test strategies often involve the creation of some formal or informal mathematical or statistical model of critical system behaviors, usually during the early stages of the project such as requirement specification or design stages. Hence, it may generally be seen as a preventive approach, which should begin as soon as the models used in the SDLC are selected.

- **Dynamic or Heuristic Approach**
  Dynamic or heuristic based approaches involve testing types such as exploratory testing, where tests are designed, created and executed as per the current scenario and are not pre-planned like in other approaches. It is a reactive test approach which carries out the simultaneous execution and evaluation of respective test cases.

- **Standard Compliant Approach**
  Standard compliant approaches make use of existing IEEE standards to fill in methodological gaps. Alternatively, agile methodologies such as extreme programming may be adopted. The testing activities covered under this approach like designing and creation of test cases follows and adheres to the given specified standards and may have an early or late point of involvement for testing.

- **Consultative or Directed Approach**
  This approach is based on the recommendations and suggestions given by a group of non-testers to guide or perform the testing efforts. They might involve experts or other professionals from the business or the technical domain outside the boundary of the organization, which may include end users. Consultative approaches emphasize on the later stages of testing due to the lack of value in early testing with this approach.

7.2 Steps in Testing

A test plan approach can be summarized with six steps as illustrated in Figure 1 below.

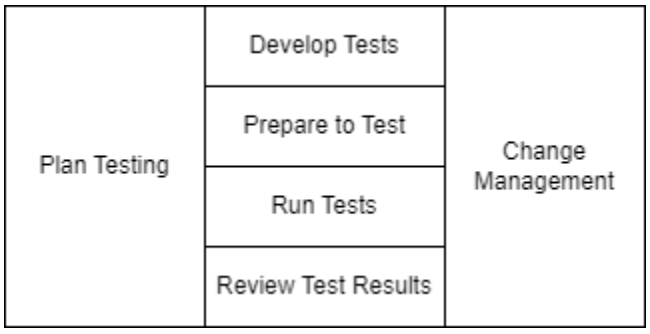

Figure 1: Steps of the Test Plan

The activities performed during each step is summarized as such:

1. **Plan Testing**
   This is where the timescale, scope, quality, risks and resources are decided in advance, and are kept up to date throughout the UAT, including entry criteria and exit criteria.

2. **Develop Tests**
   This involves numerous activities shown in Figure 2 in order to develop the formal tests required to run any form of testing:

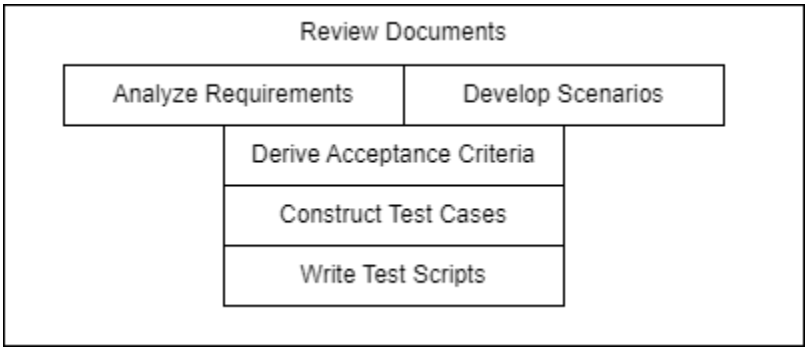

Figure 2: Activities in the Develop Tests step

- **Review Documents**: This is a key quality process of checking all documentation produced during the development life cycle of Go Where GaiGai.

- **Develop Scenarios**: This involves the preparation of scenarios that occur when users interact with the system. This uses techniques such as use cases models and descriptions.

- **Analyze Requirements**: This involves understanding the formal tests and looking for what is inconsistent and missing from what is actually required.

- **Derive Acceptance Criteria**: Once the previous two activities are underway or completed then the set of questions to be asked about the system to see if it matches the capability needed are prepared.

- **Construct Test Cases**: Test cases are the set of specific inputs and expected results which enable one or more acceptance criteria to be proved.

- **Write Test Scripts**: Test scripts are the operational instructions for running test cases. It includes what has to be done to the system, what has to be measured and how to perform the measurement accurately.

3. **Prepare to Test**
   These are the activities other than developing the tests that are needed for successful execution of the testing process:

   Preparing test data:
   Building the data files that are required to run the test cases.

Preparing the environment to run the tests:
Making sure that the people, processes, tools etc. are all in place to enable the testing to take place.

Creating three key documents:
3.1 Test Procedure - The instructions about how to run the tests on the test system including test scripts.
3.2 Entry Criteria - What must be done before testing starts.
3.3 Test Item Transmittal Report - The instructions from the developers about the system version released for testing.

4. **Run Tests**
   Run tests comprises of executing the tests and documenting the results:
   - Test Scripts that comply with inputs and expected outputs from each of the Test Cases must be designed and used to run the tests.
   - Documenting the results entail recording the order in which the tests were performed, and the outcome into the Test Log. If the outputs of the tests do not match the expected output from the Test Cases, the event must be recorded in an incidence report. The probability and impact of the error will also be decided and documented.

5. **Review Test Results**
   Once all the tests have been performed and all defects have been identified, the system will be assessed on its acceptability. The Quality Assurance team will decide if the system is suitable to be deployed based on the impact and probability of defects found. As there will always be defects in the system, defects will then be ordered based on how critical they are to the system. Defects with a higher impact to the system will be prioritized for fixes. The team will decide which defects must be fixed before Go Where GaiGai goes live, and which defects can be fixed after its release. Therefore, the test results need to be checked and traced to see what effect they have on:

   a. Business or System impact,
   b. Requirements and
   c. Scenarios

   This analysis ensures that a suitable decision can be made as to whether the system passes the tests, and provides effective recommendations to resolve the defects found. The results of this activity are then documented in a *Test Summary Report*.

6. **Change Management**
   New changes can be made at any time during the SDLC of Go Where GaiGai. Hence, it is very important to accommodate these changes during the testing process to ensure that the testing is effective in identifying the defects.

7.3 Approach for Go Where GaiGai

Team ONE will adopt a systematic approach when testing the application. The QA Manager and QA Engineer will design the test cases to be atomic, verifiable and traceable and track them in the *Test Cases* document. The QA Engineer will then conduct the tests from the test summary report, and mark them as Pass/Fail. If the tester finds that the actual output of the tests do not match the expected output as documented in the *Test Cases*, the test is considered to have failed. If so, the member of the Dev Team responsible for the tested component will be notified of the results so that he/she may rectify the error in the code. The test summary reports will be made available to the developers on the Kanban Board hosted on Github Project Board.

The QA Manager and QA Engineer will review the test summary report once the test cases for each phase of development have been completed. Once the test summary report is completed, the QA Manager will send the report to the Project Manager for approval. The Project Manager will then check the test summary report, and mark testing to be completed once it is approved.

These are the factors the QA Team will consider when selecting the test approach:

- The nature and domain of the product.
- Risks of product or risk of failure of the environment and the team.
- Regulatory and legal aspects, such as external and internal regulations of the development process.
- Experience and expertise of team members in the tools and techniques proposed for this project.

8 Item Pass/Fail Criteria

This section of the report seeks to define how a component of the overarching system is deemed to have passed or failed a test during the testing phase. Aside from checking the output of the test cases, the team will use two factors to assess the compliance of a module. They are as follows:

a. Severity - The impact that the defect will have on the application.
    i. Critical
    ii. Major
    iii. Moderate
    iv. Minor
b. Probability - The probability of the defect occurring.
    i. High
    ii. Medium
    iii. Low

Each test case will contain entry requirements, steps to conduct the test, and the expected correct output of the component. If the actual output matches the expected output the tester will mark the test case with a pass without any issues. If the actual output does not match the expected output, the tester will assess the severity and probability of the component failing during operation. The QA Manager will design a rubric to be used to prevent cases of ambiguity when assessing a failed test case. The rubric below describes how a Pass/Fail will be given based on the two factors:

	Critical	**Major**	**Moderate**	**Minor**
High	Fail	Fail	Fail	Fail
Medium	Fail	Fail	Fail	Pass
Low	Fail	Fail	Pass	Pass

Table 2: Severity and Probability chart

The completion criteria for this plan are:

- 100% of unit testing cases are complete.
- 80% of integration testing cases are complete and less than 10% of the test cases have minor defects.
- A minimum of 250 users have given their feedback in User Acceptance Testing
- System testing is done with at least 10 external testers (not part of Team ONE).

9 Suspension Criteria and Resumption Requirements

9.1 Suspension Criteria

The suspension criteria defines the criteria under which the software testing will be suspended. Suspensions will apply to parts of the testing process or a complete stop to all testing until a resumption is approved by the QA Manager. The suspension will occur should the Test-Stop Criteria find that certain major components of the system are not readily available for testing or when a serious defect is found in an implemented component.

Testing processes will mainly occur due to failures of test cases with a high severity score. This implies that a major component of the software is defective and/or unavailable for testing. Hence there will be a very high probability of the faulty module affecting the operation and testing of components that interact with it and components that are implemented after the faulty module.

In Go Where GaiGai, the QA Manager will suspend any of the testing streams should there be an appropriate reason to do so. The documentation which defines the acceptable level of the defect will be provided. There may be various scenarios to suspend the testing process. For example, network, hardware, or defects in the source code. Below are a few common criteria that will lead to the suspension in the software testing phase:

- Hardware or software not available.
- Testing resources are not available.
- A defect with a Major or Critical severity score is found (refer to Table 1 in the previous section).
- The testing process is limited due to defects in the build.
- Network connectivity issues are found.
- Schedule of the project cannot facilitate testing due to high priority deliverables.
- Major functionality does not work as specified in the *Software Requirement Specification* (SRS) document.

If more than 40% of the test cases are marked as Failed by the QA Engineer, testing will be suspended until the pass rate of all test cases reaches a threshold of less than 15%. The Lead Developer will be informed of the suspension, and the Dev Team will prioritize patching the defects or implementation of minor features.

9.2 Resumption

Resumption refers to the lifting of a suspension during the testing phase. It involves verification of the defect by which suspension was invoked during the testing process. As with suspensions, a resumption can only occur with the approval of the QA Manager. Below are the criteria the QA Manager will follow to allow for the resumption of testing:

- Hard or software are made available for testing.
- Testing resources are made available for testing.
- No further defect has been found in the resumption technique.
- Issue due to which a suspension occurs is resolved.
- Other major defects with a high severity score are patched by the Dev Team, and the QA Team verifies that the component is compliant via a retest.

10 Test Deliverables

Test deliverables are the artifact documents provided to the project stakeholders during the SDLC of Go Where GaiGai. Documents that serve as a guideline for how testing will be conducted will be submitted before the testing phase, will documents that summarize the test results and test coverage will be submitted once testing is fully completed. The deliverables included in this Test Plan are:

1. **Test Cases**
   Test cases are the set of descriptions containing steps to recreate a test scenario with entry conditions, test data, expected output results, exit conditions and actual results.

2. **Test Report**
   This document contains the test results and the summary of test execution activities.

3. **Test Plan Document**
   The test plan document is a document which contains the plans for all the testing activities to be done when delivering a quality product. The Test Plan references Go Where GaiGai's Project Plan, SRS, Use Case model and descriptions for all activities of the project.

4. **Revision Logs**
   The revision history of all documents. The revision history is to give a reader a summary of changes made during each revision.

5. **Defect Logs**
   All defects identified will be documented and logged in the appropriate documents as well as the solution implemented for the respective defect.

11 Test Tasks

The following activities are to be completed:

1. Preparation of the Test Plan.
2. Identification of the features to be tested.
3. Select the appropriate method to conduct tests.
4. Assignment of testers to each test case.
5. Conduct testing.
6. Fix bugs and errors that are discovered.
7. Creation of the defect logs.
8. Completion of the test report.

12 Environmental Needs

The following requirements have to be met for the testing of Go Where GaiGai:

- Windows device
- Git (used for source version control)
- Any appropriate Integrated Developer Environment (IDE). Visual Studio Code is recommended.

13 Staffing and Training Needs

Training will be required to set up the application on the tester's machines as well as to be proficient in conducting the tests accurately.

The Lead Developer will be responsible for providing training on how to set up the environment and software. The front-end developer will create training manuals on how to set up the libraries required to run Go Where GaiGai locally, while the back-end developer will do so for setting up the third party API. The QA Manager will be responsible for providing training on how to conduct the tests accurately, and documenting the actual output during the tests.

14 Responsibilities

Role	Responsibility
Project Manager	● Overseeing the different testing streams to ensure smooth delivery of the final product. ● Analyzing the severity score of each feature picked by the Lead Developer for testing. ● Designing the test schedule for the testing phases.
Lead Developer	● Selecting features to be tested. ● Providing training required for setting up the environment. ● Conducting white-box testing on the integration level.
Front-end Developer	● Conducting white-box testing on front-end components. ● Specifying entry/exit conditions and expected output for front-end components.
Back-end Developer	● Conducting white-box testing on front-end components. ● Specifying entry/exit conditions and expected output for back-end components.
QA Manager	● Producing the risk plan for the different phases of testing. ● Monitoring testing activities and ensuring resources required for testing are managed well. ● Providing training on testing procedures.
QA Engineer	● Conducting black-box testing. ● Reporting test logs and defect logs to the QA Manager. ● Generating the Test Cases document.
Release Manager	● Collecting details of test runs to ensure all components meet the completion criteria adopted.

Table 3: Roles and Responsibilities during testing

15 Schedule

Due to the short time period for the development of Go Where GaiGai, testing phases will begin once the major features are implemented in the system prototype, and will carry on in parallel with the development of the software. The details of which are given in the *Project Plan*. The table below summarizes the details:

Task	Estimated Time Taken (days)	Start Date
Produce Test Plan	5	17/10
Unit Test	14	17/10
Integration Test	14	17/10
Load Test	7	24/10
Produce Requirements Test Coverage Report	5	24/10

Table 4: Test Plan Schedule

Aside from the activities mentioned in Table 3, several activities will be planned for testing. They are as follows:

- A daily backlog will be maintained to keep track of items which are yet to be completed on the project team's Kanban board. The QA Manager will monitor the backlog and will use the Kanban board to assign members and/or testers to each testing task.
- Number of days assigned to each test case will depend on their severity score. Critical and Major impact items will receive five days, while Moderate and Minor impact items will receive four and three days respectively. This is to ensure testing of high priority components are done with completeness and accuracy in mind.
- For any defects that are discovered, the respective test cases will be pinned on the Kanban board, notifying the Dev Team of any suspension and the developer responsible for the component can analyze and fix the error.

16 Risks and Contingencies

The risks and their respective responses to be taken are specified in the table below:

Risk	Contingency
Testers are unavailable.	Members of the Dev Team will be assigned to assist the QA Team in testing. However, front-end developers will only be allowed to test back-end components and vice versa. This is to prevent feature blindness, where developers work on a component and are unable to think of unintended ways to interact with a feature.
Testers do not receive adequate training.	The QA Manager and Lead Developers are responsible for providing training on testing procedures and test environment set up respectively.
Delay in finishing the test items due to a delay in the implementation phases.	Project Manager will monitor the delays during implementation, and work with the QA Manager to edit the testing streams as appropriate. This is done to allow enough time for testing to be conducted accurately.
Lack of communication of failed tasks to the Dev Team.	The Kanban board hosted on Github Project Board, will notify team members when the QA Manager allocates tasks and reports back failed test cases. Testers will report failed test cases immediately following the end of every test phase.

Table 5: Risks and Contingencies during testing

17 Approvals

The table below describes the role responsible for each task in the testing phase, the criteria by which a task is to be considered as completed, and who to report to for approval of completion of said task:

Role	Approval Criteria	Approver
Front-end Developer	Unit testing of front-end components meet acceptance criteria.	Lead Developer
Back-end Developer	Unit testing of back-end components meet acceptance criteria.	Lead Developer

Release Manager	User Acceptance Testing meets acceptance criteria.	Lead Developer
Lead Developer	Integration Tests meet acceptance criteria. Application works as required based on functionality checks on the SRS document.	Project Manager
QA Engineer and External Testers	Black-box testing of all test cases meet acceptance criteria.	QA Manager
QA Manager	Test logs and defect logs are documented correctly.	Project Manager

Table 6: Approval Chart

18 Referenced Documents

Referenced Document	Link
Project Plan	https://github.com/GallonKnight/CZ3002/blob/main/Documentation/Project_Plan.pdf
System Requirements Specification (SRS)	https://github.com/GallonKnight/CZ3002/blob/main/Documentation/System_Requirements_Specification.pdf
Change Management Plan	https://github.com/GallonKnight/CZ3002/blob/main/Documentation/Change_Management_Plan.pdf
Test Cases / Test Coverage Report	https://github.com/GallonKnight/CZ3002/blob/main/Documentation/Test_Cases.pdf
IEEE 829 Test Plan Standard	https://ntulearn.ntu.edu.sg/bbcswebdav/pid-2965316-dt-content-rid-26443380_1/xid-26443380_1