MEETING PLANNER TEST PLAN

TABLE OF CONTENTS

1. Introduction and Scope.

This test plan serves as a blueprint for evaluating the main functionalities of the calendar app. This plan outlines the testing strategies aimed at uncovering defects and ensuring accurate execution of features.

The calendar app is a critical component of organizational scheduling, facilitating the booking of meetings, management of vacation time, checking of room and employee availability, and printing of agendas. As such, this test plan is instrumental in validating the app's adherence to specifications and robustness in handling diverse inputs.

Through a combination of unit testing and exploratory testing, this plan aims to deliver a thorough assessment of the app's functionalities. It establishes clear guidelines for test execution and the utilization of appropriate tools to achieve testing objectives effectively.

2. Test Strategy.

- **Unit Testing**: Modules of the program shall be independently tested to ensure that they run as expected. In this case we shall be looking at the several methods of the classes in the program.

- **Exploratory Testing**: The app's user interface and functionality shall be manually explored to not only verify expected behavior but also identify faults. This includes comprises the use of various combinations of inputs and actions.

3. Test Scenarios.

a. Normal Execution

i. Book a meeting on a valid date and time. This will also be with the use of boundary inputs. ii. Book vacation time for an existing employee. This will also be with the use of boundary inputs.

iii. Check availability for an existing room and a person.

iv. Print agenda from the calendar for either a day or a month.

b. Illegal Inputs and Actions

i. Attempt to add a meeting for an invalid date (e.g. November 30th).

ii. Attempt to book conflicting meetings. iii. Check availability for a non-existent room.

iv. Check availability for a non-existent person.

4. **Unit Tests**

a. Normal Execution

i. Test Case 1: Book a Meeting

Input: Date, start and end time, room, person, meeting details.

Expected Output: Meeting is successfully booked and added to the calendar.

ii. Test Case 2: Book Vacation Time

Input: employee, start date, end date.

Expected Output**:** Vacation time is marked as busy for the employee during the specified period.

iii. Test Case 3: Check if room gets busy after scheduling meeting

Input**:** room, meeting for specified time frame

Expected Output**:** Room is marked as busy during that time frame.

iv. Test Case 3: Check if person gets busy after scheduling meeting

Input**:** person, meeting for specified time frame

Expected Output**:** Person is marked as busy during that time frame.

v. Test Case 5: Print Agenda from Calendar on a specified date

Input: date.

Expected Output: Agenda is printed showing meetings scheduled on the specified date.

vi. Test Case 5: Print Agenda from Calendar for a month

Input: month.

Expected Output: Agenda is printed showing meetings scheduled in that month. **vii.**

Test Case 6: Add attendees to meeting

Input: person

Output: Person is added to the list of attendees scheduled for a meeting **viii.**

Test Case 7: Remove attendees from meeting

Input: person

Output: Person is removed from the list of attendees scheduled for a meeting.

ix. Test Case 8: Test all class constructors

This is a complimentary test that used to tests whether the initialization of class objects works as expected. This applies for Calendar, Meeting, Person, Room and Organization.

b. Illegal inputs/Actions

x. Test Case 7: Add Meeting with Invalid Date

Input: Invalid date, valid start and end time, room, employee, meeting details.

Expected Output: TimeConflictException.

xi. Test Case 8: Book Conflicting Meetings

Input: Valid date, time, room, employee, meeting details for each of the conflicting meetings.

Expected Output: TimeConflictException. **xii. Test Case**

9: Print agenda with invalid time frame

Input: Invalid date

Expected Output: TimeConflictException

xiii. Add and remove nonexistent employees from meetings

Input: Nonexistent employees.

Output: Exception

5. Environment

Development Environment: Visual Studio Code for with the jUnit framework.

6. Test Execution

i. Design and write test cases based on the test plan.

ii. Execute unit tests using the jUnit framework with appropriate input data, and record results.

iii. Verify the actual output against the expected output for each test case. iv. Conduct exploratory testing sessions to discover issues not covered by unit tests.

v. Record any failures and faults encountered during testing.