|  |  |  |
| --- | --- | --- |
| Name: | Student No | Reg No |
| NAKANWAGI VANESA | 2200706530 | 22/U/6530 |
| MPAIRWE LAUBEN | 2200721345 | 22/U/21345 |
| LUTALO ALLAN | 2200703330 | 22/U/3330/PS |
| RWEMERA DAVID | 2200705278 | 22/X/5278/PS |
| OKWEL EDGAR MARK | 2200706807 | 22/U/6807 |

**Test Plans & Test Cases**

**Test Plan Document for Booking a Meeting.**

**Part 1: What is being tested (units, subsystems, features)**

We will test the following units, subsystems, and features in the Planning System:

1. Booking a Meeting

* Unit: **Meeting.java**, **Calendar.java**, **PlannerInterface.java**
* Subsystem: Booking system
* Features:
* Creating a meeting (**scheduleMeeting()** in **PlannerInterface.java**)
* Modifying a meeting (if applicable)
* Cancelling a meeting (if applicable)
* Checking for time conflicts (**TimeConflictException.java**)
* Adding and Removing Attendees
* Assigning a Room to a Meeting
* Printing Meeting Details

**Part 2: How it will be tested (Test scenarios)**

The following scenarios will be executed to validate the Meeting class functionality:

* **Scenario 1:** Meeting Class Initialization

**How it will be tested**: Create a Meeting object with valid month, day, start and end time. Use getter methods to verify values are correctly stored.

* **Scenario 2:** Meeting Class with Attendees

**How it will be tested**: Initialize a Meeting with a predefined List of Person objects. Verify that all attendees are added by checking the size of the list and presence of specific names.

* **Scenario 3:** Meeting Class with Room

**How it will be tested**: Create a Meeting with a Room object and assert that the room ID assigned to the meeting matches the expected value.

* **Scenario 4:** Meeting Time Modification

**How it will be tested**: Use the setStartTime() and setEndTime() methods on a meeting object and then verify the updates using getter methods.

* **Scenario 5:** Meeting Room Modification

**How it will be tested**: Assign a new room to an existing meeting using setRoom(), and verify the update using the meeting’s getRoom() method.

* **Scenario 6:** Attendee Management in a Meeting

**How it will be tested**: Add a new attendee using addAttendee() and remove an existing one using removeAttendee(). Validate the list before and after using contains() and size assertions.

* **Scenario 7:** Validation of Day Input in Calendar

**How it will be tested**: Call Calendar.checkTimes() with an invalid day (e.g., 35). Use try-catch to assert a TimeConflictException is thrown and verify the exception message.

* **Scenario 8:** Validation of Start Time in Calendar

**How it will be tested**: Use Calendar.checkTimes() with a negative start time. Catch the expected TimeConflictException and verify that it matches the correct error condition.

* **Scenario 9:** Validation of Time Order in Meeting Scheduling

**How it will be tested**: Try scheduling a meeting where endTime < startTime. Use try-catch to ensure a TimeConflictException is raised with the appropriate message.

* **Scenario 10:** Room Booking Conflict Detection

**How it will be tested**: Schedule a valid meeting in a room. Attempt to add a new meeting in the same room that overlaps in time. Expect a TimeConflictException.

* **Scenario 11:** Input Validation – Month

**How it will be tested**: Call Calendar.checkTimes() with an invalid month (e.g., 13). Verify that a TimeConflictException is thrown with the expected error message.

* **Scenario 12:** Input Validation – Hour Range

**How it will be tested**: Call Calendar.checkTimes() with an invalid hour (e.g., 24). Confirm that a TimeConflictException is raised indicating illegal hour input.

The following scenarios will be executed to validate the Room class functionality:

1.**Room Initialization and ID Validation**

* Verify default constructor initializes ID as an empty string.
* Verify parameterized constructor correctly sets the ID (e.g., "LLT6A").

2.**Meeting Management**

* Add a meeting successfully and confirm room availability reflects the booking.
* Attempt to add overlapping meetings and ensure a TimeConflictException is thrown with a message containing the room ID.
* Remove a meeting and verify the room becomes available for the removed timeslot.
* Ensure removing one meeting does not affect other scheduled meetings on the same day.

3. **Agenda Printing**

* Print agenda for a month with meetings and validate output includes meeting descriptions and room IDs.
* Print agenda for a specific day with no meetings and confirm output shows an empty agenda.

4. **Edge Cases and Error Handling**

* Check availability for invalid dates (e.g., February 30) and ensure a TimeConflictException is thrown.
* Verify non-existent days (e.g., April 31) are marked busy as per calendar initialization.
* Validate reversed time ranges (start > end) throw TimeConflictException.

5. **Integration with Calendar Class**

* Confirm isBusy() accurately reflects meetings added via addMeeting().
* Test retrieval of meetings by index after addition/removal..

**Part 4: When it will be tested (Required stage of Completion)**

**1.Unit testing Stage**

* As soon as individual units (Person, Calendar, and PlannerInterface) are implemented, unit tests will be written and executed.

**2. Integration Testing Stage**

* After all modules (Person, Calendar, etc.) are integrated, we will run tests to ensure the modules interact correctly.

**3. System Testing Stage**

* Once the entire scheduling system is complete, end-to-end tests will be conducted.

**4. Regression Testing**

* Before any major release or update, previously passed test cases will be re-run to ensure no existing features are broken

**Part 5: Where It Will Be Tested**

* **Environment**: Local development machine.
* **Tools**:
  + **IDE**: VS Code, IntelliJ, or Eclipse.
  + **Testing Framework**: JUnit 4.
  + **Build Tools**: Maven/Gradle (if applicable).
* **Execution**:
  + Tests are run manually or via IDE/test runner scripts.
  + Automated execution through JUnit to ensure repeatability.

**Part 6: Why We Are Testing It**

1. **Ensure Accuracy**:
   1. To guarantee that scheduling logic is working as expected.
2. **Detect Bugs Earlier**:
   1. Unit and integration tests help catch errors before deployment.
3. **Maintain User Trust**:
   1. Prevent scheduling conflicts that could affect productivity or cause frustration.
4. **Support Future Enhancements**:
   1. Regression tests ensure updates or new features don’t break existing functionality.

**Approval**  
*Prepared by*: LUTALO ALLAN  
*Reviewed by*: Group

**Test Plan Document for Vacation Time Scheduling (scheduleVacation() Method)**

**Part 1: How It Will Be Tested (Test Scenarios)**

The following scenarios will be executed to validate the Room class functionality:

**1.Valid Inputs**

* **1.1.1** Schedule vacation time with valid start and end dates (e.g., June 1 to June 5).

**Expected:** Vacation is added successfully, and the person’s agenda reflects the booked dates.

* **1.1.2** Schedule vacation time for a single day (e.g., July 10 to July 10).

**Expected:** Vacation is added for the specified day only.

**2. Invalid Inputs**

* 2.2.1 Enter an invalid start/end month (e.g., 0 or 13).

Expected: System rejects input with an error message (e.g., "Invalid month").

* 2.2.2 Enter an invalid start/end day (e.g., April 31 or February 30).

Expected: System rejects input with an error message (e.g., "Invalid day").

* 2.2.3 Enter a person name that does not exist.

Expected: System rejects input with an error message (e.g., "Person not found").

**3. Conflicts Handling**

* **3.3.1** Schedule vacation time that overlaps with an existing meeting.

**Expected:** System detects conflict, displays details of the conflicting meeting, and rejects the vacation request.

* **3.3.2** Schedule vacation time that spans multiple months (e.g., December 25 to January 5).

**Expected:** Vacation is added for all valid days across months, and conflicts (if any) are flagged.

**4. Edge Cases**

* **4.4.1** Schedule vacation time that starts and ends on the same day.

**Expected:** Treated as a single-day vacation and added successfully.

* **4.4.2** Attempt to schedule vacation time for the entire year (e.g., January 1 to December 31).

**Expected:** System processes the request but flags any conflicts with existing meetings.

**Part 2: When it will be tested (Required stage of Completion)**

**1.Unit testing Stage**

* As soon as individual units (Person, Calendar, and PlannerInterface) are implemented, unit tests will be written and executed.

**2. Integration Testing Stage**

* After all modules (Person, Calendar, etc.) are integrated, we will run tests to ensure the modules interact correctly.

**3. System Testing Stage**

* Once the entire scheduling system is complete, end-to-end tests will be conducted.

**4. Regression Testing**

* Before any major release or update, previously passed test cases will be re-run to ensure no existing features are broken

**Part 3: Where It Will Be Tested**

* **Environment**: Local development machine.
* **Tools**:
  + **IDE**: VS Code, IntelliJ, or Eclipse.
  + **Testing Framework**: JUnit 4.
  + **Build Tools**: Maven/Gradle (if applicable).
* **Execution**:
  + Tests are run manually or via IDE/test runner scripts.
  + Automated execution through JUnit to ensure repeatability.

**Part 4: Why We Are Testing It**

Functional Validation:

* Ensure vacations are scheduled correctly and reflected in agendas.
* Verify conflict detection works as expected (e.g., overlaps with meetings).

Input Robustness:

* Prevent invalid inputs (e.g., nonexistent dates) from corrupting data.

Edge Case Coverage:

* Handle edge cases like single-day or year-long vacations gracefully.

Integration Assurance:

* Confirm seamless interaction with Person and Meeting classes.

User Feedback:

* Validate error messages are clear and actionable (e.g., "Conflict detected on June 3").

**Approval**  
*Prepared by*: NAKANWAGI VANESA

Reviewed by: Group

Test Plan Document for Room Class Availability Checks

### **Part 1: How It Will Be Tested (Test Scenarios)**

The following scenarios will be executed to validate the Room class functionality:

#### **1. Room Initialization and ID Validation**

* 1.1.1 Verify default constructor initializes ID as an empty string.
* 1.1.2 Verify parameterized constructor correctly sets the ID (e.g., "LLT6A").

#### **2. Meeting Management**

* 2.2.1 Add a meeting successfully and confirm room availability reflects the booking.
* 2.2.2 Attempt to add overlapping meetings and ensure a TimeConflictException is thrown with a message containing the room ID.
* 2.2.3 Remove a meeting and verify the room becomes available for the removed timeslot.
* 2.2.4 Ensure removing one meeting does not affect other scheduled meetings on the same day.

#### **3. Agenda Printing**

* 3.3.1 Print agenda for a month with meetings and validate output includes meeting descriptions and room IDs.
* 3.3.2 Print agenda for a specific day with no meetings and confirm output shows an empty agenda.

#### **4. Edge Cases and Error Handling**

* 4.4.1 Check availability for invalid dates (e.g., February 30) and ensure a TimeConflictException is thrown.
* 4.4.2 Verify non-existent days (e.g., April 31) are marked busy as per calendar initialization.
* 4.4.3 Validate reversed time ranges (start > end) throw TimeConflictException.

#### **5. Integration with Calendar Class**

* 5.5.1 Confirm isBusy() accurately reflects meetings added via addMeeting().
* 5.5.2 Test retrieval of meetings by index after addition/removal.

### **Part 2: Where It Will Be Tested**

* Environment: Local development machine.
* Tools:
  + IDE: VS Code, IntelliJ, or Eclipse.
  + Testing Framework: JUnit 4.
  + Build Tools: Maven/Gradle (if applicable).
* Execution:
  + Tests are run manually or via IDE/test runner scripts.
  + Automated execution through JUnit to ensure repeatability.

### **Part 3: Why We Are Testing It**

1. Functional Validation:
   * Ensure core features (booking, availability checks, agenda printing) work as specified.
   * Confirm the room’s calendar accurately reflects added/removed meetings.
2. Error Prevention:
   * Catch invalid inputs (e.g., non-existent dates, reversed time ranges) early to prevent runtime failures.
   * Validate exception messages provide actionable feedback (e.g., room-specific conflict details).
3. Edge Case Handling:
   * Ensure the system gracefully handles edge cases like invalid dates and all-day meetings.
4. Integration Assurance:
   * Verify seamless interaction between Room and Calendar classes (e.g., agenda formatting, conflict detection).
5. Maintainability:
   * Automated tests act as regression safeguards for future code changes.

Approval  
Prepared by: mpairwe lauben  
Reviewed by: Group

**Test Plan Document for Checking Availability for a person**

**Part 1: How It Will Be Tested (Test Scenarios)**

The following scenarios will be executed to validate the Room class functionality:

**1.Checking if a person is available (checkEmployeeAvailability() )**

* **1.1.1** Person has no meetings scheduled — should return available.

**2. Handling scheduling conflicts**

* **2.2.1** Add a meeting that overlaps with an existing one — should trigger a conflict error..

**3. Viewing and Managing Personal Schedules**

* **3.3.1** View full day schedule — should return all meetings/events for the day.

**4. Checking Availability for Recurring Meetings**

* **4.4.1** Weekly recurring meeting every Monday 9–10 AM — check availability for future Mondays should return unavailable during that time.

**5. Checking Availability for Multiple Meetings**

* **5.5.1** Suggest optimal available time slot for a batch of meetings — system should suggest based on least conflicts.

**6. Checking Availability for Multiple Meetings**

* **6.6.1** Person marks full-day vacation from March 10–12 — all availability checks during this period should return unavailable.

**Part 2: When it will be tested (Required stage of Completion)**

**1.Unit testing Stage**

* As soon as individual units (Person, Calendar, and PlannerInterface) are implemented, unit tests will be written and executed.

**2. Integration Testing Stage**

* After all modules (Person, Calendar, etc.) are integrated, we will run tests to ensure the modules interact correctly.

**3. System Testing Stage**

* Once the entire scheduling system is complete, end-to-end tests will be conducted.

**4. Regression Testing**

* Before any major release or update, previously passed test cases will be re-run to ensure no existing features are broken

**Part 3: Where It Will Be Tested**

* **Environment**: Local development machine.
* **Tools**:
  + **IDE**: VS Code, IntelliJ, or Eclipse.
  + **Testing Framework**: JUnit 4.
  + **Build Tools**: Maven/Gradle (if applicable).
* **Execution**:
  + Tests are run manually or via IDE/test runner scripts.
  + Automated execution through JUnit to ensure repeatability.

**Part 4: Why We Are Testing It**

1. **Ensure Accuracy**:
   * To guarantee that scheduling logic is working as expected.
2. **Detect Bugs Earlier**:
   * Unit and integration tests help catch errors before deployment.
3. **Maintain User Trust**:
   * Prevent scheduling conflicts that could affect productivity or cause frustration.
4. **Support Future Enhancements**:
   * Regression tests ensure updates or new features don’t break existing functionality.

**Approval**  
*Prepared by*: RWEMERA David  
*Reviewed by*: Group

**Test Cases**

|  |  |
| --- | --- |
| **Software Under Test** | **The Planning System** |
| **Testing Report Type** | Unit Tests |
| **Automated Test Suite** | |
| **Environment** | Windows |
|  |  |
| **Scenario 1** |  |
| **Tested Feature** | Meeting Class Initialization |
| **Description** | To verify that the Meeting class correctly initializes a meeting object with the provided values for month, day, start time, and end time. |
| **Expected Result** | The Meeting object should correctly store the values provided (month = 4, day = 10, startTime = 14, endTime = 16), and the getter methods should return these values correctly. |
| **Actual Result** | All assertions were successful. The Meeting object was initialized correctly, and the getter methods returned the expected values. |
|  |  |
| **Scenario 2** |  |
| **Tested Feature** | Meeting Class with Attendees |
| **Description** | To verify that the Meeting class correctly initializes a meeting object with a list of attendees, ensuring that the attendees are properly added to the meeting. |
| **Expected Result** | The Meeting object should be initialized with the correct attendees, and the number of attendees should match the expected count (2). The attendees list should contain both "Alice" and "Bob". |
| **Actual Result** | All assertions were successful. The Meeting object was initialized correctly, with the expected number of attendees and the correct attendees ("Alice" and "Bob") included in the list. |
|  |  |
| **Scenario 3** |  |
| **Tested Feature** | Meeting Class with Room |
| **Description** | To verify that the Meeting class correctly initializes a meeting object with a room, ensuring the room is properly assigned to the meeting. |
| **Expected Result** | The Meeting object should be initialized with the correct room, and the room ID should match the provided room ID ("Room101"). |
| **Actual Result** | The assertion was successful. The Meeting object was initialized correctly, and the room ID is correctly set to "Room101". |
|  |  |
| **Scenario 4** |  |
| **Tested Feature** | Meeting Time Modification |
| **Description** | To verify that the Meeting class correctly updates the start and end times of a meeting when the setter methods setStartTime() and setEndTime() are used. |
| **Expected Result** | The start time should be updated to 15, and the end time should be updated to 17. |
| **Actual Result** | The assertions were successful. The start time was updated to 15, and the end time was updated to 17. |
|  |  |
| **Scenario 5** |  |
| **Tested Feature** | Meeting Room Modification |
| **Description** | To verify that the Meeting class correctly updates the meeting room when the setRoom() method is used. |
| **Expected Result** | The meeting room should be updated from "Room101" to "Room102". |
| **Actual Result** | The assertion was successful. The meeting room was correctly updated to "Room102". |
|  |  |
| **Scenario 6** |  |
| **Tested Feature** | Attendee Management in a Meeting |
| **Description** | To verify that attendees can be added to and removed from a meeting using the addAttendee() and removeAttendee() methods. |
| **Expected Result** | After adding, the meeting should include Bob; after removing, the meeting should no longer include Alice. |
| **Actual Result** | The assertions were successful. Bob was added and Alice was removed correctly. |
|  |  |
| **Scenario 7** |  |
| **Tested Feature** | Validation of Day Input in Calendar |
| **Description** | To verify that the system throws a TimeConflictException when an invalid day (e.g., 35th) is used during meeting time checking. |
| **Expected Result** | An exception should be thrown with a message indicating the day does not exist. |
| **Actual Result** | The exception was correctly thrown and the message confirmed the invalid day input. |
|  |  |
| **Scenario 8** |  |
| **Tested Feature** | Validation of Start Time in Calendar |
| **Description** | To verify that a TimeConflictException is thrown when an invalid (negative) start time is provided while checking meeting times. |
| **Expected Result** | The system should throw a TimeConflictException due to the negative start time. |
| **Actual Result** | The exception was correctly thrown, indicating invalid time input was handled properly. |
|  |  |
| **Scenario 9** |  |
| **Tested Feature** | Validation of Time Order in Meeting Scheduling |
| **Description** | To ensure the system throws a TimeConflictException when a meeting's end time is earlier than its start time. |
| **Expected Result** | A TimeConflictException is thrown with a message indicating the meeting ends before it starts. |
| **Actual Result** | Exception was successfully thrown with the correct message, confirming invalid time sequence was detected. |
|  |  |
| **Scenario 10** |  |
| **Tested Feature** | Room Booking Conflict Detection |
| **Description** | Validates that the system prevents overlapping meetings from being scheduled in the same room. |
| **Expected Result** | A TimeConflictException is thrown when attempting to add a conflicting meeting to an already booked room. |
| **Actual Result** | Exception was successfully thrown, indicating the conflict was correctly detected. |
|  |  |
| **Scenario 11** |  |
| **Tested Feature** | Input Validation – Month |
| **Description** | Ensures that the system throws an exception when an invalid month value (e.g., 13) is passed to the meeting creation logic. |
| **Expected Result** | A TimeConflictException is thrown with a message indicating "Month does not exist." |
| **Actual Result** | Exception was successfully thrown with the correct message. |
|  |  |
| **Scenario 12** |  |
| **Tested Feature** | Input Validation – Hour Range |
| **Description** | Validates that the system correctly identifies and rejects an invalid hour (e.g., 24) when checking meeting times. |
| **Expected Result** | A TimeConflictException is thrown with a message indicating "Illegal hour." |
| **Actual Result** | Exception was successfully thrown with the correct message. |
|  |  |

|  |  |
| --- | --- |
| **Software Under Test** | **The Planning System** |
| **Testing Report Type** | Unit Testing |
| **Automated Test Suite** | |
| **Environment** | Windows |
|  |  |
| **Scenario 1** |  |
| **Tested Feature** | Checking if a person is available |
| **Description** | Check availability when the person has no meetings scheduled |
| **Expected Result** | Display: True |
| **Actual Result** | Displayed: True |
|  |  |
| **Scenario 2** |  |
| **Tested Feature** | Handling scheduling Conflicts |
| **Description** | Attempt to schedule two overlapping meetings |
| **Expected Result** | First meeting should be scheduled, second one should fail |
| **Actual Result** | First meeting scheduled: true  Second overlapping meeting scheduled: false |
|  |  |
| **Scenario 3** |  |
| **Tested Feature** | Viewing and Managing Personal Schedules |
| **Description** | View all meetings scheduled on a specific day |
| **Expected Result** | Only one meeting should appear |
| **Actual Result** | David’s schedule showed 1 meeting — Actual: 1 |
|  |  |
| **Scenario 4** |  |
| **Tested Feature** | Checking Availability for Recurring Meetings |
| **Description** | Check availability during a recurring weekly meeting time |
| **Expected Result** | Person should be unavailable during recurring meeting |
| **Actual Result** | David was not available — “Is David available on the following Monday 9–10 AM? False” |
|  |  |
| **Scenario 5** |  |
| **Tested Feature** | Checking Availability for Multiple Meetings |
| **Description** | Find optimal time slots for a meeting (60 minutes) within a day |
| **Expected Result** | Return multiple time slots, excluding 10–11 AM |
| **Actual Result** | Several optimal time slots were returned, excluding 10–11 AM — “Actual: X slots” |
|  |  |
| **Scenario 6** |  |
| **Tested Feature** | Checking Availability for Multiple Meetings |
| **Description** | Check availability during scheduled vacation period |
| **Expected Result** | Person should be unavailable |
| **Actual Result** | David was not available — “Is David available during vacation? False” |
|  |  |

|  |  |
| --- | --- |
| **Software Under Test** | **The Planning System** |
| **Testing Report Type** | Unit Testing |
| **Automated Test Suite** | |
| **Environment** | Java 8+,  JUnit 4,  Custom Calendar System |
|  |
|  |  |
| **Scenario 1** |  |
| **Tested Feature** | Room Constructor with Default ID |
| **Description** | Verify that the default constructor initializes the room ID as an empty string. |
| **Expected Result** | Room ID should be empty (""). |
| **Actual Result** | Room ID initialized as empty string. |
|  |  |
| **Scenario 2** |  |
| **Tested Feature** | Room Constructor with Specific ID |
| **Description** | Verify that the constructor initializes the room with a specific ID. |
| **Expected Result** | Room ID should be "LLT6A". |
| **Actual Result** | Room ID initialized correctly as "LLT6A". |
|  |  |
| **Scenario 3** |  |
| **Tested Feature** | Adding a Meeting Successfully |
| **Description** | Add a valid meeting and verify if the room is marked busy. |
| **Expected Result**   |  | | --- | |  | | Meeting added, room marked as busy. |
| **Actual Result** | Meeting successfully added, room marked as busy. |
|  |  |
| **Scenario 4** |  |
| **Tested Feature** | Meeting Conflict Detection |
| **Description** | Add overlapping meetings and check for conflict. |
| **Expected Result** | Should throw TimeConflictException including the room ID. |
| **Actual Result** | Exception thrown with room ID "LLT6A" in the message. |
|  |  |
|  |  |
| **Scenario 5** |  |
| **Tested Feature** | Remove a Meeting |
| **Description** | Add and then remove a meeting. |
| **Expected Result** | Meeting should be removed, room no longer busy. |
| **Actual Result** | Meeting removed successfully, room no longer busy. |
|  |  |
| **Scenario 6** |  |
| **Tested Feature** | Remove One Meeting From Multiple |
| **Description** | Add two meetings, remove one, and ensure the other remains. |
| **Expected Result** | One meeting removed, other still present. |
| **Actual Result** | Remaining meeting unaffected, as expected. |
|  |  |
| **Scenario 7** |  |
| **Tested Feature** | Agenda for Empty Day |
| **Description** | Print agenda for a day with no meetings. |
| **Expected Result** | "Agenda for 3/10:\n"Printed as expected. |
| **Actual Result** | Printed as expected. |
|  |  |
| **Scenario 8** |  |
| **Tested Feature** | Agenda for a Month with MeetingPrint agenda for March when meetings exist. |
| **Description** | Print agenda for March when meetings exist. |
| **Expected Result** | Agenda should include meeting description and room ID. |
| **Actual Result** | Agenda correctly printed with expected content. |
|  |  |
| **Scenario 9** |  |
| **Tested Feature** | Agenda for a Specific Day with Meeting |
| **Description** | Print agenda for a day with meetings. |
| **Expected Result** | Agenda includes correct day, meeting details, and room ID. |
| **Actual Result** | Agenda correctly printed with expected content. |
|  |  |
|  |  |
| **Scenario 10** |  |
| **Tested Feature** | Get a Meeting by IndexRetrieve a meeting from the schedule by index. |
| **Description** | Retrieve a meeting from the schedule by index. |
| **Expected Result** | Meeting at index matches added meeting. |
| **Actual Result** | Meeting retrieved successfully. |
|  |  |
| **Scenario 11** |  |
| **Tested Feature** | Invalid Date Check (Feb 30) |
| **Description** | Check behavior when querying an invalid date. Throws TimeConflictException. |
| **Expected Result** | Throws TimeConflictException. |
| **Actual Result** | Exception thrown as expected. |
|  |  |
| **Scenario 12** |  |
| **Tested Feature** | Non-existent Date Check (April 31) |
| **Description** | Verify that room marks non-existent date (April 31) as busy. |
| **Expected Result.** | Room should be busy. |
| **Actual Result** | Room marked busy correctly. |

|  |  |
| --- | --- |
| **Software Under Test** | **The Planning System** |
| **Testing Report Type** | Unit TestingAutomated Test Suite |
| **Automated Test Suite** | |
| **Environment** | Java 8+,  JUnit 4, |
|  |
|  |  |
| **Scenario 1** |  |
| **Tested Feature** | Schedule a vacation with no conflict |
| **Description** | Add a vacation on 6/15 for the full day. |
| **Expected Result** | Vacation added successfully; agenda contains 'vacation'. |
| **Actual Result** | Vacation added successfully; agenda contains 'vacation'. |
|  |  |
| **Scenario 2** |  |
| **Tested Feature** | Schedule a vacation across two day |
| **Description** | Add vacation on 9/1 and 9/2 for full days. |
| **Expected Result** | Vacation appears on both days in agenda. |
| **Actual Result** | Vacation appears on both days in agenda. |
|  |  |
| **Scenario 3** |  |
| **Tested Feature** | Add vacation on a day with a meeting |
| **Description** | Add vacation overlapping with a scheduled meeting. |
| **Expected Result**   |  | | --- | |  | | Throws TimeConflictException. |
| **Actual Result** | TimeConflictException thrown. |
|  |  |
| **Scenario 4** |  |
| **Tested Feature** | Add meeting during a scheduled vacation |
| **Description** | Add a meeting during an existing vacation. |
| **Expected Result** | Throws TimeConflictException. |
| **Actual Result** | TimeConflictException thrown. |
|  |  |
|  |  |