Testing Document

SWEN90007

Musical Event System
Team: SDA 1

In charge of:

Name	Student id	Email	Unimelb username	Github username
Chengyi Huang	1173994	chengyih@student.unimelb.edu.a	chengyih	Kind0fblue
Ziqiang Li	1173898	ziqiangl1@student.unimelb.edu.au	ziqiangl1	johnnylild
Yuxin Liu	1408760	yuxliu20@student.unimelb.edu.au	YUXLIU20	YuxinLiu-unimelb
Hanming Mao	1257522	hanmingm@student.unimelb.edu.au	HanmingM	KkkellyM

SCHOOL OF
COMPUTING &
INFORMATION
SYSTEMS

Revision History

Date	Version	Description	Author

17/10/2023	01.00	Add half of the cases	Chengyi Huang
17/10/2023	02.00	Finish draft	Chengyi Huang

- **Introduction**

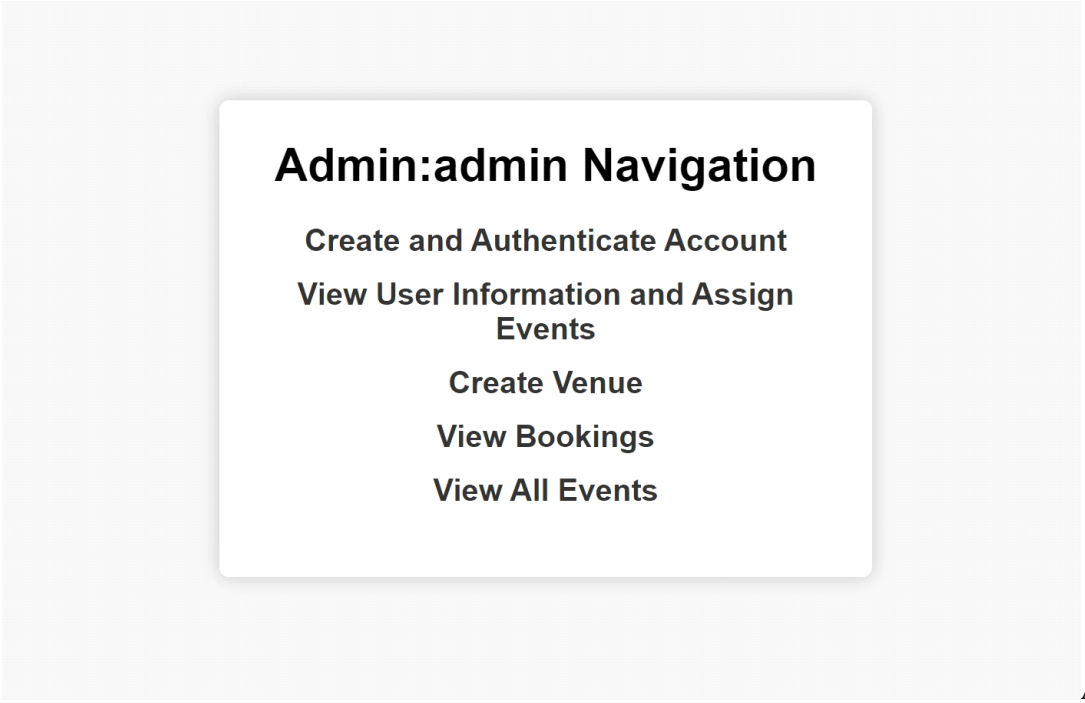
- **Proposal**

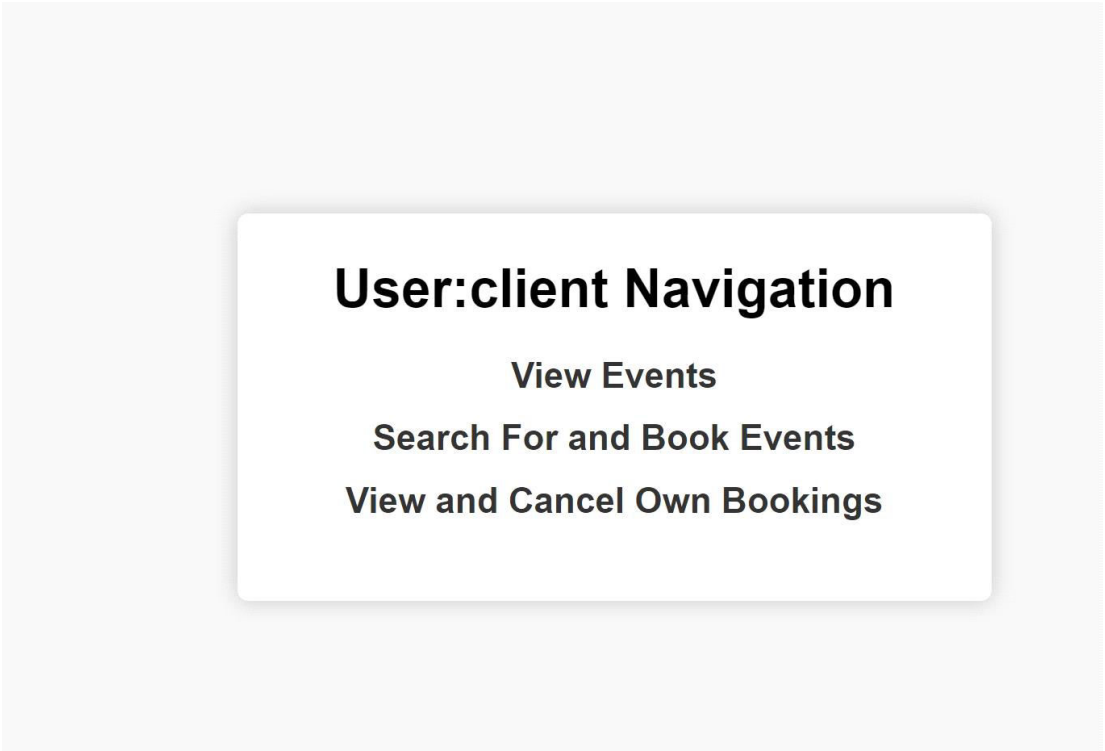
The purpose of this document is to define and present the test cases for **project Music Events System of team SDA 1**, covering the test cases for the system use cases.

- **Target Users**

This document in mainly designed for those responsible for executing the test cases in this project **Music Events System of team SDA 1.**

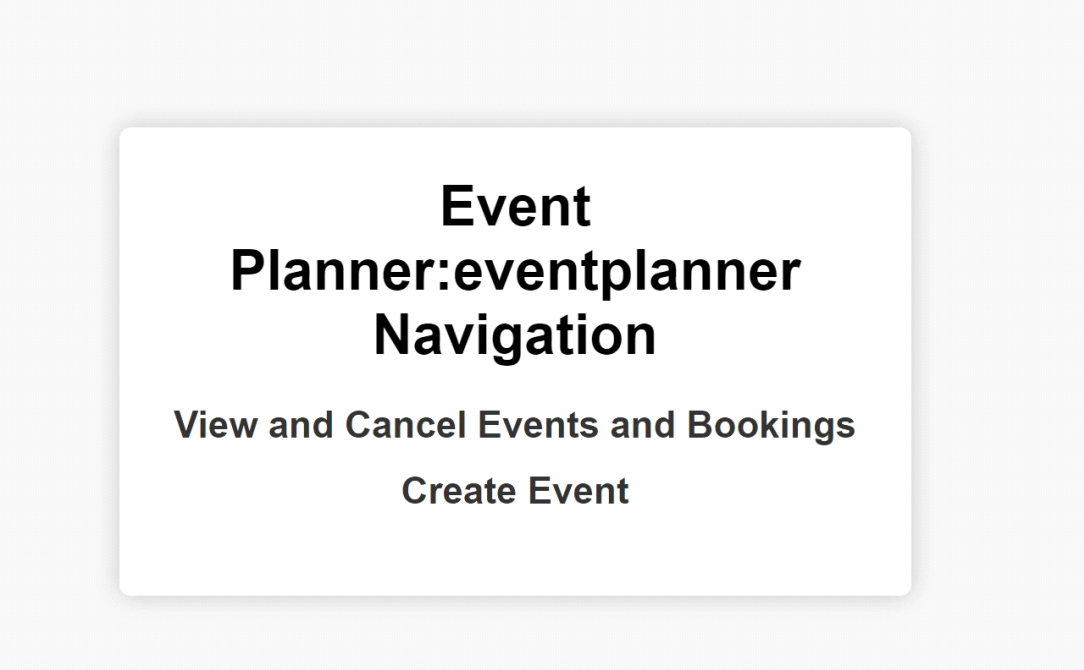
- **Interface for Users (After login)**

- 

Admin
- 

User

1.3.2 Event Planner



- **Covered Requirements**

This section lists the system requirements covered in the test cases.

- **Functional or Product Requirements**

Note: All tests should start from https://nine0007-1b.onrender.com/musicsystem1017/login

- **Functional Test Cases**

This section describes the test cases that cover the product requirements of the system.

- **UC001: EventPlanner adds an event to the system concurrently**
- **TC001: EventPlanner concurrently adds two distinct events without clashes**

Test Type:	**Execution Type:**
Functional	Manual

Objective:

Verify if the system can handle and process multiple concurrent event addition requests without any clashes.

Setup:
The system is running and connected to the database.
No existing events in the time slots being tested.

.

Pre-Conditions:
The system is operational and running.
EventPlanner has successfully logged into the system.

Notes:

1. The EventPlanner will need to initiate concurrent requests to add events.
2. Events being added should have distinct timeslots to avoid any clashes.
3. After both threads complete, the system should reflect both the added events.
4. EventPlanner can concurrently add distinct events successfully

Data sample:
First event:
venueId: 5
eventName: NewTestEvent01
startTime: 2023-10-20 03:00:00
endTime: 2023-10-20 04:00:00

Second event:
venueId: 5
eventName: NewTestEvent02
startTime: 2023-10-20 05:00:00
endTime: 2023-10-20 06:00:00

Time constraint:

Minimum: 5 min
Maximum: 25 min

- **UC002: EventPlanner concurrently adds and updates events to the system**
- **TC002: EventPlanner concurrently adds a new event and updates an existing event that might cause a clash**

Test Type:	**Execution Type:**
Functional	Manual

Objective:

Verify if the admin can assign events to the event planner

Setup:

The system is running and connected with the database.

There are some events already stored in the database.

The “updateAddEventTest” file is available for reference..

Pre-Conditions:

• The system is up and running.

• An event planner is logged into the system.

• There is an existing event in the database that matches the venueId and time frame of the new event being added.

Notes:

• Mock requests are set up to simulate the addition and update operations.

• Junit and Mockito frameworks are used to mock necessary objects and define behaviors.

• There's a latch that ensures concurrency for the two threads responsible for adding and updating.

• After the tasks are executed, exceptions, if any, are caught and logged.

Data sample:

For addition:

Username: “eventPlanner1”,

Event: NewTestEvent05

venueId: 23

startTime: 2023-10-20 03:00:00

endTime: 2023-10-20 04:00:00

For update:

Username: “eventPlanner2”,

eventId: 96

venueId: 23

Event: OldEvent

startTime: 2023-10-20 03:00:00

endTime: 2023-10-20 04:00:00

Note: Other attributes of the sample data do not have any effect on the test results thus are not explicitly listed here.

Time constraint:
Minimum: 10 min Maximum: 25 min

- **UC003: Multiple clients attempt to book limited tickets concurrently**
- **TC003: Multiple clients concurrently book the last few available tickets for a specific event section**

Test Type:	**Execution Type:**
Functional	Manual

Objective:

Verify if the system can prevent concurrent bookings that exceed the available ticket capacity for a particular event section.

Setup:

- The system is running and connected to the database.
- There are some events with limited ticket capacities already stored in the database.
- The ACID structure has been implemented in the system to handle the booking process.

Pre-Conditions:

- The system is up and running.
- Multiple clients are logged into the system.
- There are limited tickets available for a particular section of an event.

Notes:

• The entire ticket booking process is treated as a long transaction.

• The system sets the connection to not auto-commit and ensures TRANSACTION_REPEATABLE_READ.

• OrderIDs are primary keys, and any conflicts will trigger SQLExceptions, leading to rollbacks.

• Tickets are added to the database with a ticketFlag to ensure correct number additions.

• On rollback, users are redirected to the ticket booking page to view updated capacities.

• The transaction is finalized only when all conditions are met without errors.

Data sample:

User1:

Username: "test0918"

Password: "test0918"

User2:

Username: "client"

Password: "client"

User3:

Username: "client2"

Password: "client2"

Event Details:

eventName: test1

eventId: 73

vipcapacity: 1

moshcapacity: 2

standingcapacity: 3

seatedcapacity: 4

Concurrency Details:

24 threads, 1 loop each, booking 1 VIP ticket and 1 mosh ticket simultaneously.

- **UC004: Multiple event planners concurrently update the same event**
- **TC004: Two event planners attempt to modify the details of the same event simultaneously**

Test Type: Functional	**Execution Type:** Manual
Objective: To ensure that when multiple event planners concurrently modify the same event, the system prevents data inconsistencies and avoids lost updates.	
Setup: • The music event booking system is up and running. • The system has events already stored, with event planners assigned by administrators. • The Optimistic offline lock pattern has been integrated into the system.	
Pre-Conditions: • Two event planners are logged in and can access the same event. • The event has a version number in the database to track modifications.	
Notes: **• Optimistic offline lock allows concurrent access but locks the data only at commit time.** **• Version numbers are checked before an update is committed.** **• If a version mismatch is detected, indicating another event planner updated the event, the current transaction is rolled back.** **Data sample:** **Event planner 1:** **Username: "eventplanner"** **Password: "eventplanner"** **Event to modify: event20** **Changes:** **Event name: event_updated 20** **Artist name: artist_968** **Event planner 2:** **Username: "EventPlaner1"** **Password: "EventPlanner1"** **Event to modify: event20** **Changes:** **Artist name: artist_007**	

- **UC005: Event planners concurrently attempt to delete the same event**
- **TC005: Two event planners attempt to delete the same event at the same time**

Test Type:	**Execution Type:**
Functional	Manual

Objective:
To validate that when multiple event planners simultaneously try to delete the same event, the system correctly handles concurrency, allowing only one deletion and preventing data inconsistencies.

Verify if the administrator can creates venues

Setup:

• The music event booking system is active.

• Event planners have been assigned and granted access to events by administrators.

• The pessimistic offline lock pattern has been integrated into the system.

Pre-Conditions:

• Two event planners are logged in with access to the same event.

• A lock manager is set up and functioning correctly to manage lock requests.

Notes:

• Pessimistic offline lock blocks other users from accessing a resource once one user is operating on it.

• Lock manager uses a singleton pattern, ensuring one instance.

• The method findEvent in the event mapper class has been locked. This lock is engaged when an event planner accesses this method and released post deletion.

• The lock manager checks if an event is already locked when a deletion request is made. If so, a runtime exception is thrown, preventing subsequent deletions.

Data Sample:

Event Planner 1:

Username: “eventplanner”

Password: “eventplanner”

Event Planner 2:

Username: “EventPlanner1”

Password: “EventPlanner1”

Event to be Deleted:

Event ID: 28

Venue ID: 9

- **UC006: Conflict between client booking tickets and event planner deleting the event**
- **TC006: Client books tickets for an event concurrently with event planner attempting to delete the event**

Test Type:	**Execution Type:**
Functional	Manual

Objective:

To validate that when a client is in the process of booking tickets for an event, the system correctly handles concurrency, preventing the event planner from deleting the event to avoid data inconsistencies and disruption for the client.

Setup:

• The music event booking system is operational.

• Events are available for clients to book tickets.

• The pessimistic offline lock pattern has been integrated into the system.

Pre-Conditions:

• A client is logged in and ready to book tickets for an event.

• Event planners are logged in and have access to manage and delete events.

• The lock manager, operating on a singleton pattern, is active and able to manage lock

Notes:

• The lock manager, upon request, prevents event planners from accessing an event (for deletion) while a client is booking tickets for that event.

• The method findEvent in the event mapper class has been equipped with the pessimistic offline lock. This lock is engaged when a client attempts to book tickets for an event and released post booking.

Data Sample:

Client:

Username: “client”
Password: “client”
Event Planner 1:

Username: “eventplanner”
Password: “eventplanner”
Event Planner 2:

Username: “EventPlanner1”
Password: “EventPlanner1”
Ticket to Book:

Event ID: 26
Buy VIP capacity: 1
Buy Mosh capacity: 1
Buy Standing capacity: 0
Buy Seated capacity: 0
Total Amount: 7
Event to Delete:

ID: 26
Venue ID: 9

- **Entry Data**

This section describes the entry data that will be used by more than one test case, avoiding data replication. These data are referenced by the test cases.

• DATA...: EventPlanner concurrently adds distinct events to the system

Description:
This data set is used to test if two event planners can add and update distinct events at the same time.

Field Value
1. addEventTask - venueId 23
2. addEventTask - eventName "NewTestEvent05"

3. addEventTask - startTime "2023-10-20 03:00:00"
4. addEventTask - endTime "2023-10-20 04:00:00"
5. updateEventTask - eventId 96
6. updateEventTask - venueId 23
7. updateEventTask - eventName "OldEvent"
8. updateEventTask - startTime "2023-10-20 03:00:00"
9. updateEventTask - endTime "2023-10-20 04:00:00"

• DATA...: EventPlanner concurrently adds and updates events to the system

Description:
This data set is used to test if an event planner can add an event while another updates a different event.

Field Value
1. Username (Addition) "eventPlanner1"
2. Event (Addition) "NewTestEvent05"
3. venueId (Addition) 23
4. startTime (Addition) "2023-10-20 03:00:00"
5. endTime (Addition) "2023-10-20 04:00:00"
6. Username (Update) "eventPlanner2"
7. eventId (Update) 96
8. venueId (Update) 23
9. Event (Update) "OldEvent"
10. startTime (Update) "2023-10-20 03:00:00"
11. endTime (Update) "2023-10-20 04:00:00"

• DATA...: Multiple clients attempt to book limited tickets concurrently

Description:
This data set is used to test if multiple clients can book limited tickets at the same time.

Field Value
1. Username (User1) "test0918"
2. Password (User1) "test0918"
3. Username (User2) "client"
4. Password (User2) "client"
5. Username (User3) "client2"
6. Password (User3) "client2"
7. eventName "test1"
8. eventId 73

9. vipcapacity 1
10. moshcapacity 2
11. standingcapacity 3
12. seatedcapacity 4

• DATA...: Multiple event planners concurrently update the same event

Description:
This data set is used to test if multiple event planners can update the same event concurrently.

Field	Value
1. Username (EP1)	"eventplanner"
2. Password (EP1)	"eventplanner"
3. Event to modify (EP1)	"event20"
4. Event name (Change EP1)	"event_updated 20"
5. Artist name (Change EP1)	"artist_968"
6. Username (EP2)	"EventPlaner1"
7. Password (EP2)	"EventPlanner1"
8. Event to modify (EP2)	"event20"
9. Artist name (Change EP2)	"artist_007"

• DATA...: Event planners concurrently attempt to delete the same event

Description:
This data set is used to test if two event planners can delete the same event at the same time.

Field	Value
1. Username (EP1)	"eventplanner"
2. Password (EP1)	"eventplanner"
3. Username (EP2)	"EventPlanner1"
4. Password (EP2)	"EventPlanner1"
5. Event ID to delete	28
6. Venue ID	9

• DATA...: Conflict between client booking tickets and event planner deleting the event

Description:
This data set is used to test conflicts between a client booking tickets and an event planner

deleting the event.

Field	Value
1. Username (Client)	"client"
2. Password (Client)	"client"
3. Username (EP1)	"eventplanner"
4. Password (EP1)	"eventplanner"
5. Username (EP2)	"EventPlanner1"
6. Password (EP2)	"EventPlanner1"
7. Ticket - Event ID	26
8. Ticket - VIP capacity	1
9. Ticket - Mosh capacity	1
10. Ticket - Standing capacity	0
11. Ticket - Seated capacity	0
12. Ticket - Total Amount	7
13. Event to delete - ID	26
14. Event to delete - Venue ID	9