Schedule++: Second Iteration

Advanced Software Engineering

Team: Scoreoverflow

Cindy Le (xl2738), Henry Xing (hx2209), David Wan (dw2735), Zichuan Wang (zw2395)

GitHub Link: https://github.com/Zichuan-Wang/ScoreOverFlow

Use cases:

Name	Login
Actors	All users
Triggers	Start the program
Preconditions	None
Postconditions	The user is logged in with the correct, user-specific information
Main Course	1. The user enters username and password 2. The user clicks login button 3. The system will direct to the main page for the user
Alternate Courses	2A1: The user enters the wrong username and/or password 1. System will prompt that there is an error 2. It will repeat step 1 from the main course until the user has given a correct username and password combination 3. The use case continues

Name	Search Rooms
Actors	All users

Triggers	The user has clicked the button for reserving room
Preconditions	None
Postconditions	A list of reservable rooms is displayed in the middle of the program's GUI. If it is a high priority user, the list of overridable rooms are shown below the reservable rooms. If no room is found, a message will be displayed.
Main Course	1. User selects the desired date, if necessary (default date is today's date) 2. User selects the desired start time and end time, if necessary (default start time is current time and end time is 10 minutes after) 3. User types the desired capacity, if necessary (can leave blank) 4. User types the desired name of the room, if necessary (can leave blank) 5. User selects the desired facilities, if necessary (can select none) 6. For high priority user, user can check the box for showing reserved rooms to override 7. Click Search
Alternate Courses	None

Name	Reserve Rooms
Actors	All users
Triggers	The user has clicked the button for reserving room
Preconditions	User is in the reserve panel and has clicked search
Postconditions	Desired Rooms is reserved. Other rooms can still be reserved.
Main Course	1. User selects the reserve button for the desired room 2. Success message will be displayed 3. The room clicked will be unavailable to be clicked again
Alternate Courses	2A1. Failure message is displayed 1. Workflow ends

Name	Override Rooms
Actors	User (high priority)
Triggers	The user has clicked the button for reserving room and checked the button
Preconditions	User is in the reserve panel and has clicked search
Postconditions	Desired Rooms is overridden and reserved under the name of the high priority user. Other rooms can still be overridden.
Main Course	1. User selects the override button for the desired room 2. A message is displayed asking for user's acknowledgement of this action 3. Success message will be displayed 4. Email will be sent to the user who originally reserved 5. The room clicked will be unavailable to be clicked again
Alternate Courses	2A1. The user decides not to override 1. Workflow ends 3A1. Failure message is displayed 2. Workflow ends

Name	Cancel Reservations
Actors	All user
Triggers	The user has clicked the button for viewing reservations
Preconditions	User is in the view reservation panel
Postconditions	Desired rooms are cancelled
Main Course	1. A list of reservations is displayed in the middle of the program's GUI. 2. User select the desired reservation to cancel 3. Success message is prompted 4. List of reservation is refreshed
Alternate Courses	1A1. Empty list of reservation

	1. The middle Panel will be empty 2. Workflow ends 3A1. Failure message is displayed 3. Workflow ends 4. List of reservation is refreshed

Name	Batch reservation
Actors	User (program supervisor)
Triggers	The user has clicked the button for batch reserve
Preconditions	User is in the correct panel
Postconditions	List of rooms are reserved
Main Course	1. The user uploads the correctly supported file 2. The user clicks reserve button 3. The system parse the file and reserve as many rooms as possible 4. Message displays information about what rooms are reserved and not reserved
Alternate Courses	1A1. Not supported file 1. Workflow ends

Name	Add new classroom
Actors	Administrator
Triggers	The user has clicked the button for adding information
Preconditions	User is in the correct panel
Postconditions	Rooms are added
Main Course	1. Administrator selects room 2. Administrator enters the room name 3. Administrator enters the room capacity 4. Administrator selects the facilities

	5. Administrator selects add button 6. Returns to the initial stage (panel to select which to add)
Alternate Courses	None

Name	Add new facility
Actors	Administrator
Triggers	The user has clicked the button for adding information
Preconditions	User is in the correct panel
Postconditions	Facilities are added
Main Course	1. Administrator selects facility 2. Administrator enters the facility name 3. Administrator selects add button 4. Returns to the initial stage (panel to select which to add)
Alternate Courses	None

Name	Change classroom
Actors	Administrator
Triggers	The user has clicked the button for changing information
Preconditions	User is in the correct panel
Postconditions	Room information is changed
Main Course	1. Administrator selects change room 2. Administrator enters room name 3. Administrator enters the room capacity, if needed 4. Administrator selects the facilities, if needed 5. Administrator selects search button 6. Administrator selects the desired room 7. A box will be displayed with the information 8. Administrator changes the name, if needed

	9. Administrator changes the capacity, if needed 10. Administrator changes the list of facilities, if needed 11. Administrator confirms the change 12. Returns to the initial stage (panel to select what to change)
Alternate Courses	5A1. No rooms found 1. Error message prompts 2. Back to step 2 11A1. Administrator cancels the change 1. Changes are not applied 2. Workflow ends

Name	Change facility
Actors	Administrator
Triggers	The user has clicked the button for changing information
Preconditions	User is in the correct panel
Postconditions	Facilities are changed
Main Course	1. Administrator selects change facility 2. Administrator enters facility name 3. Administrator selects search button 4. Administrator selects the desired facility 5. A box will be displayed with the information 6. Administrator changes facility name 7. Administrator confirms the change 8. Returns to the initial stage (panel to select which to add)
Alternate Courses	3A1. No rooms found 3. Error message prompts 4. Back to step 2 6A1. Administrator cancels the change 3. Changes are not applied 4. Workflow ends

Test Plan:

The test suite of our project resides in src/test/ directory. We use JUnit with AssertJ support to achieve the unit testing. The major routines and business logic of our app is implemented in the src/main/server/action/ directory. Other major helper and service methods are implemented in src/main/email/ for email sending and src/main/security/ for user login and registration. We primarily provide test cases corresponding to equivalence classes and boundary conditions for these procedures.

[1] **Sending Email**

Sending email to the user whose room has been overridden is one of the major subroutine of the software. This process invokes the Java mail API and relies on the external library to perform the function. The email sending function takes three parameters: to (the address the email sends to), subject and the body.

The equivalence partition of the "to" argument is: 1) a valid address (Scheduleplusplus@gmail.com) 2) an invalid address (invalid). The boundary conditions are that the argument is an empty string or a Java null object. We test all four of the cases in the testAddress function in src/test/email/EmailSenderTest.java. We expect no exception in the first equivalence partition and SendFailedException in the invalid equivalence class as well as the two boundary conditions.

There are no equivalence partition for the subject argument. The boundary conditions are that the argument is an empty string or a Java null object. We test a nonempty subject and the boundary cases in the testSubject function in src/test/email/EmailSenderTest.java. We expect no exception in any of the conditions.

There are no equivalence partition for the body argument. The boundary conditions are that the argument is an empty string or a Java null object. We test a nonempty body and the boundary cases in the testBody function in src/test/email/EmailSenderTest.java. We expect no exception in any of the conditions.

[2] **Searching Rooms**

A very important subroutine of the software is to search for rooms that satisfy certain constraints including date and time, room name, and room capacity. We test the searchRooms function in the RoomAction class as follows: 1) given date and time; 2) given only date; 3) given room name; 4) given capacity; 5) given room name and date combined; and 6) given capacity and date

combined. For each item, we test both when a room satisfies the constraint and when the room does not satisfy. Blank fields, if not given, should not reduce the number of the search results.

The boundary conditions for the time constraint are when time slots are right next to another one that is already reserved. For example, if Room whose ID is 1 is reserved from 10am to 1pm on Nov. 21, 2018, then we test if a user can see Room 1 in the search result when she inputs 1pm-3pm on Nov. 21, 2018.

The boundary conditions for the capacity constraint are when the input capacity is exactly the same as the room's capacity recorded in database. We test and see if a user can see the Room 2 with a capacity of 50 when she inputs the constraint as to have capacity of at least 50.

[3] **Finding Users**

One of the software's subroutines is to query the database for the user with the given UNI through the findUserByUni function in the UserAction class. Instead of having equivalence partition for this test, we choose to have a normal case when there is one matching user and a boundary case when there is no user in the database with the given UNI. We expect the function to return null in the case that the user with the specified UNI does not exist. The test cases are written in the UserActionTest.java file under src/test/server/action/UserActionTest.java.

[4] **Finding Facilities**

One subroutine of the software is to query for all the available facilities in the database via the findAllFacilities function in the FacilityAction class. There are no equivalence partition for this function. However, a boundary condition arises when there are no facilities in the database in which case the function should return an empty list. We test for the normal case and the boundary case in the testFindAllFacilities function in the src/test/server/action/FacilityActionTest.java.

[5] **Creating Reservation**

One subroutine of the software is to create a reservation using the ReservationAction class. There are no equivalence partition for this function. However, a boundary condition arises when we try to reserve a null object. We test for the normal case and the boundary case in the testReserveRoom function in the src/test/server/action/ReservationActionTest.java.

[6] **Cancelling Reservation**

One subroutine of the software is to cancel a reservation using the ReservationAction class. The equivalence partitions of the function are 1) the reservation is in the database (valid) and 2) the reservation is not in the database (invalid). In addition, a boundary condition arises when we try to cancel a null object. We test for all the three different cases in the testCancelReservation function in the src/test/server/action/ReservationActionTest.java.

.

[7] **Searching Reservation**

One subroutine of the software is to search for all the reservation for a given user using the ReservationAction class. There are no obvious equivalence partitions for this function. However, a boundary condition arises when there are no reservations corresponding to the user. We test for the normal case when there are reservations and the boundary case. For the normal case, we test for the condition just above the boundary case, that is, when there is exactly one reservations for the user. We test for both cases in the testSearchReservations function in the src/test/server/action/ReservationActionTest.java.

[8] **Security**

One subroutine of the software is to ask the user to login. The equivalence partitions for this function is correct user name & password are provided, correct user name & incorrect password are provided, incorrect user name & some password are provided, no user name & some password are provided, no password & some user name are provided and no user name & no password are provided. We test those cases in src/test/security/SecurityServiceTest.java.

Branch Coverage

For coverage, we added JaCoCo to the post-commit CI process, and the reports are located in reports/jacoco. Please open the index.html file for the result. Overall, we achieved a 62% instructions coverage and a 26% branch coverage (at the time of writing). The branch coverage is low because we used an external database, and for most of the testing we create mock database. In this case, many of the real exceptions are not covered. Thus, most of the try-catch branches are not covered by the test case. There are also many if statements around real and mock databases, and for those statements we cannot real test them in both branches. Also, branches are sometimes very hard to test when they are in private methods. Some if statements are based on

database errors, which are also really hard to test. We will be adding more test cases for in interaction with the real database in order to achieve higher branch coverage with the test suite.