1. Project Title and Authors

a. Your assigned team number (from 1 to 76)

b. Optionally a cool name you select for your team

c. A list of all team members (names, emails, and USC ID numbers)

2. Preface

a. Describe this document in a nut-shell, the intended readers, etc.

The USCRecApp is an Android application designed for USC students to make reservations on USC recreational centers online. This document provides implementation documentations for our detailed implementation for the USCRecApp. This document is intended for both programmers and clints for reference purposes about the implementation.

3. Instruction

a. Instruction on how to execute your test cases

The USCRecApp allows USC students to make reservations before accessing recreational centers under USC's guidance. Students should be able to view each USC recreational center on the map. Students should be able to make reservations for an available time slot or join a waiting list if the time slot is full. Students should be able to track their reservation information and cancel an upcoming reservation. The USCRecApp would be more user-friendly for mobile users than the current web reservation system.

This document serves as the implementation document for the implementation of this app with two primary purposes. First, it records the changes we made during implementation (on architectural design, detailed class design, or requirements). Second, it provides a rationale behind each of these changes.

4. Black-box Testing

b. Each test case information:

**SDM Test1: BtestAlreadyLoginTest() in AlreadyLoginTest**

i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/AlreadyLoginTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the AlreadyLoginTest.java and select “run.” First, the program will log in using a test account then quite the program when it checks it is currently on the Map page (this is the setting up in AtestAlreadyLoginSetup() to get ready for the real test in BtestAlreadyLoginTest()). Then the program is opened again, but this time no login is required as expected as we directly have access to the Map page (because this device already logged in once and did not log out, so we don’t need to log in again). Then the profile button is clicked to check if the information displayed on the Profile page matched the content in the test account. Finally, the logout button on the Profile page is clicked to log out the test user. The whole process should contain no errors.
iii. Rationale behind the test case:
This test case aims to make sure a user that used a device to log in before (and the user did not log out) would not need to log in again if opening the application again on the same device. This test aims to test the “open app again does not require re-login” functionality in depth.
iv. Document the details of the bugs if there are any:
No bug

**SDM Test2: testLoginSuccessAndLogout() in LoginActivityTest**
i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/LoginActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the testLoginSuccessAndLogout() function in LoginActivityTest and select “run.”
First, the program will login using a test account (entering username & password, then clicking the login button) and check if it is currently on the Map page as expected. Then the profile button is clicked to check if the information displayed on the Profile page matched the test account. Finally, the logout button on the Profile page is clicked to log out the test user, and the program checks if the user is redirected to the Login page (indicating log out successfully). The whole process should contain no errors.
iii. Rationale behind the test case:
This test case aims to make sure when a user enters the correct username & password combination, they can log in successfully to the correct account corresponding to the username

entered. This test case also checks if the user can log out successfully. This test aims to test the “login & logout” functionality in depth.
iv. Document the details of the bugs if there are any:
No bug

**SDM Test3: testLoginFail() in LoginActivityTest**
i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/LoginActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the testLoginFail() function in LoginActivityTest and select “run.”
First, the program will login using a wrong account (entering a wrong combination of username & password, then clicking the login button) and check if it is currently still on the Login page as expected, and check if the error message is shown. The whole process should contain no errors.
iii. Rationale behind the test case:
This test case aims to make sure when a user enters the wrong username & password combination, it cannot log in, and an error message is prompted. This test aims to test the “login GUI” functionality in depth.
iv. Document the details of the bugs if there are any:
No bug

**SDM Test4: mapToLyon() in MapActivityTest**
i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/MapActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the mapToLyon() function in MapActivityTest and select “run.”
First, the program will login using a test account as usual and is directed to the Map page. Then it clicks the button at the Lyon Recreation Center located on the map and checks if the user is directed to the Booking page of Lyon Recreation Center (by checking if several pieces of information only visible on the Booking page such as the Lyon title is visible in the current page). Then it clicks the back button on the Booking page and checks if the user is redirected to the

Map page. Finally, it clicks the profile button and then the logout button to log out of the test account. The whole process should contain no errors.
iii. Rationale behind the test case:
This test case aims to make sure when a user clicks the button at the Lyon Recreation Center located on the map, it directs the user to the Booking page of Lyon Recreation Center and can go back to the Map page clicking the back button. This test aims to test the “map page GUI” functionality in depth.
iv. Document the details of the bugs if there are any:
No bug

**SDM Test5: mapToVillage() in MapActivityTest**

i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/MapActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the mapToVillage() function in MapActivityTest and select “run.”
First, the program will login using a test account as usual and is directed to the Map page. Then it clicks the button at the Village Recreation Center located on the map and checks if the user is directed to the Booking page of the Village Recreation Center (by checking if several pieces of information only visible on the Booking page such as the Village title is visible in the current page). Then it clicks the back button on the Booking page and checks if the user is redirected to the Map page. Finally, it clicks the profile button, then the logout button to log out of the test account. The whole process should contain no errors.
iii. Rationale behind the test case:
This test case aims to make sure when a user clicks the button at the Village Recreation Center location on the map, it directs the user to the Booking page of Village Recreation Center and can go back to the Map page clicking the back button. This test aims to test the “map page GUI” functionality in depth.
iv. Document the details of the bugs if there are any:
No bug

**SDM Test6: mapToProfile() in MapActivityTest**

i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/MapActivityTest.java

ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the mapToProfile() function in MapActivityTest and select “run.”
First, the program will login using a test account as usual and is directed to the Map page. Then it clicks the profile icon at the top-left corner of the Map page and checks if the user is directed to the Profile page (by checking if several pieces of information only visible on the Profile page such as the name, username, email are visible in the current page). Then it clicks the back button on the Profile page and checks if the user is redirected to the Map page. Then again, it clicks the view profile button at the top-left corner of the Map page and checks if the user is directed to the Profile page since both the icon and the button should lead the user to the Profile page. Finally, it clicks the logout button to log out of the test account. The whole process should contain no errors.
iii. Rationale behind the test case:
This test case aims to make sure when a user clicks the profile icon OR views the profile button at the top-left corner of the Map page, it directs the user to the Profile page and can go back to the Map page by clicking the back button. This test aims to test the “map page GUI” functionality in depth.
iv. Document the details of the bugs if there are any:
No bug

**SDM Test7: mapToSummary() in MapActivityTest**
i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/MapActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the mapToSummary() function in MapActivityTest and select “run.”
First, the program will login using a test account as usual and is directed to the Map page. Then it clicks the small window (which usually shows some summary information) at the bottom of the Map page and checks if the user is directed to the Summary page (by checking if several pieces of information only visible on the Summary page are visible on the current page). Then it clicks the back button on the Summary page and checks if the user is redirected to the Map page. Finally, it clicks the profile button and then the logout button to log out of the test account. The whole process should contain no errors.
iii. Rationale behind the test case:

This test case aims to make sure when a user clicks the small window at the bottom of the Map page, it directs the user to the Summary page which includes all booking information, and can go back to the Map page by clicking the back button. This test aims to test the “map page GUI” functionality in depth.
iv. Document the details of the bugs if there are any:
No bug

**SDM Test8: profileContentTest() in ProfileActivityTest**
i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/ProfileActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the ProfileContentTest() function in ProfileActivityTest and select “run.”
First, the program will login using a test account as usual and is directed to the Map page. Then it clicks the profile icon and checks if the user is directed to the Profile page. On the Profile page, all information displayed is checked in detail to see if they match with the information about this particular user stored in the database. Finally, it clicks the logout button to log out of the test account. The whole process should contain no errors.
iii. Rationale behind the test case:
This test case aims to make sure when a user enters the Profile page, the correct and complete information about the user is shown on the page, the user can then go back to the Map page clicking the back button, or log out by clicking the logout button. This test aims to test the “view profile & profile GUI” functionality in depth.
iv. Document the details of the bugs if there are any:
No bug

**SDM Test9: NotificationTest (tested manually)**
i. Location of the test case:
Since this test require multiple devices, we test it manually as approved by Yannan Li
ii. Description of what it does and how to execute it:
Here are the steps to test manually. First, we need two devices, for simplicity we call them D1 and D2, also for simplicity we focus on a single timeslot T (which is set to have only 2 available

spots for testing). We need three accounts A1, A2, and A3. First, let A1 login to D1, make a reservation on T, then log out of D1. Second, let A2 login to D1, make a reservation on T, and stay in the app. Third, let A3 login to D2, now the timeslot T is full (since it already contains A1, A2), so A3 clicks the join waitlist button, then quit the app and waits. Now, let A2 cancel the reservation on T using D1 and check if A3 successfully receives a notification on D2. We tested this result multiple times on our own, it works all the time. We also showed this in our demo video as supplement material for project 2.3 for reference.
iii. Rationale behind the test case:
This test case aims to make sure a user on the waitlist can receive a notification when the timeslot corresponds to the waitlist has a new spot available as a prompt message on the phone. This test aims to test the “waitlist & notification” functionality in depth.
iv. Document the details of the bugs if there are any:
No bug

**XTC Test 1: onClickRefresh()---User Story 4**
i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/BookingActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the onClickRefresh() to run it, the program will automatically log in using a test account and go to one recreational center to view the timeslots (by clicking the buttons on the map). It will try to click the refresh button on the timeslots page to see if refresh works. After finishing, the program goes back to the main page and then goes to the profile page and clicks the logout button to log out. The whole process should contain no errors.
iii. Rationale behind the test case: Just to make sure the refresh button works.
iv. Document the details of the bugs if there are any: no bug

**XTC Test 2: makeReservationTest() —User story 5**
i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/BookingActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the makeReservationTest() to run it, the program will automatically log in using a test account and go to one recreational center to view the timeslots (by clicking the buttons on the map). It will try to make a reservation

in an available timeslot to see if making reservation works. After finishing, the program should give a successful message box to tell the user. Then it will try to make another reservation on the same day, this should fail because a user should only be allowed to make one reservation each day. A failure message box will be shown. Then it will cancel the reservation the user has made to ensure that the database is not changed, then it will log out the user after all the operations are done. The whole process should contain no errors.
iii. Rationale behind the test case:
To make sure making a reservation works properly when the user can/cannot make a reservation
iv. Document the details of the bugs if there are any: no bug

**XTC Test 3: joinWaitlistTest() — User story 6**
i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/BookingActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the joinWaitlistTest() to run it, the program will automatically log in using a test account and go to one recreational center to view the timeslots. It will try to join a waitlist in one timeslot that is full(we already made one timeslot each day be full to test the functionality) After finishing, the program should give a successful message box to tell the user. Then it will try to join the waitlist in the same timeslot on the same day, this will fail because the user is already on the waitlist. A failure message box will be shown. Then it will log out the user after all the two tries are done. The whole process should contain no errors.
iii. Rationale behind the test case:
To make sure joining a waitlist works properly when the user can/cannot join the waitlist
iv. Document the details of the bugs if there are any: no bug

**XTC Test 4: joinAfterMakeTest() — User story 6**
i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/BookingActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the joinAfterMakeTest() directly to run it, the program will automatically log in using a test account and go to one recreational center to view the timeslots. It will try to make a reservation in a timeslot that has one spot

left(already set by us on each day) After finishing, the program should give a successful message box to tell the user. Then the timeslot's button will be [join waitlist] instead of]book], it will try to join the waitlist in the same timeslot where it made the reservation in, this will fail because the user has made the reservation in this timeslot. A failure message box will be shown. Then it will cancel the reservation and then log out the user after all the two tries are done. The whole process should contain no errors.
iii. Rationale behind the test case:
To make sure the page refreshes and the button changes after the user take up the last spot in a timeslot. And then make sure the user cannot join the waitlist in the timeslot it already has a reservation in.
iv. Document the details of the bugs if there are any: no bug

**JY Test 1: DisplayReservation() - user story view booking information**

i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/BookingInformationActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the DisplayReservation() in BookingInformationActivityTest directly to run it, the program will automatically log in using a test account and go to the map page. It checks the contents shown in the future booking list to make sure it is correct. Then it goes to the booking information summary page. This test user has 2 future reservations and 1 history reservation already created in the database. The future reservation is with reservation id 129 and 130. The history reservation is with reservation id 128. The test checks the contents shown on the summary page. It checks whether the future booking information container contains exactly two elements, with one as reservation 129 and the other one as reservation 130. It also checks whether the history booking information container has exactly one element, with reservation 128. Then it will return to the map page by clicking the back button and go to the profile page by clicking the profile button. Finally, it will log out by clicking the logout button. The whole process should contain no errors.
iii. Rationale behind the test case:
To make sure the booking information page can correctly display all the future bookings and history bookings in the correct container with the correct information.
iv. Document the details of the bugs if there are any: no bug

**JY Test 2: CancelReservation() - user story cancel upcoming reservations**

i. Location of the test case:
app/java/com.example.uscrecapp_team28(androidTest)/BookingInformationActivityTest.java
ii. Description of what it does and how to execute it:
Just make sure your emulator has been set up and right-click on the CancelReservation() in BookingInformationActivityTest directly to run it, the program will automatically log in using a test account and go to the booking information summary page. This test user has 2 future reservations and 1 history reservation already created in the database. The future reservation is with reservation id 129 and 130. The history reservation is with reservation id 128. The test checks the contents shown on the summary page. It checks whether the future booking information container contains exactly two elements, with one as reservation 129 and the other one as reservation 130. It also checks whether the history booking information container has exactly one element, with reservation 128. Then it will return to the map page by clicking the back button. The next step is to go to the reservation page of a recreational center. I will try to make a reservation in an available time slot for tomorrow. After finishing, the program should give a successful message box to tell the user. Then it goes back to the map page and then the booking information page. It will cancel the newly created reservation. The test will check after the cancellation if the summary page displays the same information as before. Finally, it returns to the map page and goes to the profile page to log out. The whole process should contain no errors.
iii. Rationale behind the test case:
To make sure the booking information page can correctly display all the future bookings and history bookings in the correct container with the correct information. After it successfully makes the reservation, and cancels the reservation, make sure the updated booking summary page displays the correct booking information.
iv. Document the details of the bugs if there are any: no bug

5. White-box Testing

We have over 100 white-box test cases. Most of these test cases are testing something very simple to make sure we do not have easy-fix bugs in the code. Here, we ignore most of

these simple test cases but only focus on some most important ones that serves directly (or as core) to the functionalities of our application. // DS

b. Each test case information:

**Test1: testInit_info() in AgentTest = DS**

i. Location of the test case in your folder

testInit_info() in AgentTest

ii. Description of what it does and how to execute it

Right-click on AgentTest and select “run” to run this class of test cases, testInit_info() are one of them. If just want to run this specific test, you need to open AgentTest, navigate to testInit_info(), then right-click on the method, and select “run” to run this specific test case.

iii. Result of the test case

In this test case, we test if the Agent can successfully init all the information from device_id by calling init_info(). There are three sub-test cases included. The first case is to instantiate a new agent without assigning a divice_id and without calling init_info(), in this case, the agent’s corresponding fields should all be empty (aka null). The second case is to instantiate a new agent, assigning a divice_id, but not calling init_info(), in this case, the agent’s corresponding fields should still all be empty (aka null), except the device_id, because the assignment only happens when we call init_info(). The third case is instantiating a new agent, assigning a valid divice_id, and finally calling init_info(). In this case, the agent’s fields should be set up correctly corresponding to the information from the database. In all cases, the test results are the same as what we expected since the test passed.

iv. Document the details of the bugs if there are any

There is no bug found here.

**Test2: testUser_login() in AgentTest = DS**

i. Location of the test case in your folder

testUser_login() in AgentTest

ii. Description of what it does and how to execute it

Right-click on AgentTest, and select “run” to run this class of test cases, testUser_login() is one of them. If just want to run this specific test, you need to open AgentTest, navigate to testUser_login(), then right-click on the method, and select “run” to run this specific test case.

iii. Result of the test case

In this test case, we test if the Agent can successfully check if the username and password (in this case, it should be provided by the user and stored to the agent, but this is done in some Activities and can only be tested in the black-box testing) are correctly matched to a user in the database. There are two sub-test cases included. The first case is to instantiate a new agent and provide it with a correct username & password combination, then called user_login() which should return true. The second case also instantiates a new agent but provides it with an incorrect username & password combination, then called user_login() which should return false. In all cases, the test results are the same as what we expected since the test passed.
iv. Document the details of the bugs if there are any
There is no bug found here.

**Test3: testCheck_loggedin() in AgentTest = DS**
i. Location of the test case in your folder
testCheck_loggedin() in AgentTest
ii. Description of what it does and how to execute it
Right-click on AgentTest, and select “run” to run this class of test cases, testCheck_loggedin() is one of them. If just want to run this specific test, you need to open AgentTest, navigate to testCheck_loggedin(), then right-click on the method, and select “run” to run this specific test case.
iii. Result of the test case
In this test case, we test if the Agent can successfully check if the user using this device is already logged in as stored in the database (which should happen as long as this device logged in before and did not log out). There are three sub-test cases included. The first case is to instantiate a new agent without assigning a device_id then call testCheck_loggedin() which should return false because we cannot get the device information. The second case is to instantiate a new agent, assigning a device_id that does not exist in the database, then call testCheck_loggedin() which should return false because the device information is not stored in the database, which means this device is not logged in. The third case is to instantiate a new agent, assigning a device_id that exists in the database, then call testCheck_loggedin() which should return true because the device information is stored in the database, which means that the device is already logged in with a corresponding user assigned to it. In all cases, the test results are the same as what we expected since the test passed.
iv. Document the details of the bugs if there are any
There is no bug found here.

**Test4: testView_profile() in AgentTest = DS**

i. Location of the test case in your folder

testView_profile() in AgentTest

ii. Description of what it does and how to execute it

Right-click on AgentTest, and select “run” to run this class of test cases, testView_profile() is one of them. If just want to run this specific test, you need to open AgentTest, navigate to testView_profile(), then right-click on the method, and select “run” to run this specific test case.

iii. Result of the test case

In this test case, we test if a view_profile() correctly returns all information stored in the profile corresponding to a device_id (the information should be obtained from the database using this valid device_id). There are three sub-test cases included. The first case is to instantiate a new agent without assigning a device_id then call testView_profile() which should return an empty result because we cannot get the device information. The second case is to instantiate a new agent, assigning a device_id that does not exist in the database, then call testView_profile() which should return an empty result because the device information is not stored in the database, we cannot obtain any user information correspond to this device. The third case is to instantiate a new agent, assigning a device_id that exists in the database, then call testView_profile() which should return the corresponding user profile from the database. In all cases, the test results are the same as what we expected since the test passed.

iv. Document the details of the bugs if there are any

There is no bug found here.

**Test5: testIf_already_login() in AuthenticationTest = DS**

i. Location of the test case in your folder

testIf_already_login() in AuthenticationTest

ii. Description of what it does and how to execute it

Right-click on AuthenticationTest, and select “run” to run this class of test cases, testIf_already_login() is one of them. If just want to run this specific test, you need to open AuthenticationTest, navigate to testIf_already_login(), then right-click on the method, and select “run” to run this specific test case.

iii. Result of the test case

In this test case, we test if the Authentication can successfully check if the device is logged in. This is called in the Agent’s check_loggedin(), but the Authentication serves as a base-level

class that interacts directly with the database, so we still need to test it. There are three sub-test cases included. The first case is to instantiate a new Authentication object with an incorrect device_id and assign it an unique_userid, then call if_already_login() which should return true because no matter what device_id is, as long as this class currently has a unique_userid, it is logged in (since this unique_userid corresponds to the user that is currently using this app). The second case is instantiating a new Authentication object with an incorrect device_id, then calling if_already_login() which should return false because we cannot get any useful information including the unique_userid from this incorrect device_id from the database. The second case is instantiating a new Authentication object with a correct device_id that exists in the database, then calling if_already_login() which should return true because we can successfully access the user information corresponding to this device_id as stored in the database and set the unique_userid correctly. In all cases, the test results are the same as what we expected since the test passed.

iv. Document the details of the bugs if there are any

There is no bug found here.

**Test6: testCheck_login() in LoginCheckerTest = DS**

i. Location of the test case in your folder

testCheck_login() in LoginCheckerTest

ii. Description of what it does and how to execute it

Right-click on LoginCheckerTest, and select “run” to run this class of test cases, testCheck_login() is one of them. If just want to run this specific test, you need to open LoginCheckerTest, navigate to testCheck_login(), then right-click the method, and select “run” to run this specific test case.

iii. Result of the test case

In this test case, we test if the LoginChecker can successfully check if the username and password provided by the user are correctly matched to a user in the database (although we also need a device_id in instantiating a LoginChecker class, that is not required for check_login() so we can ignore it here). This is called in the Agent’s user_login(), but the LoginChecker serves as a base-level class that interacts directly with the database, so we still need to test it. There are two sub-test cases included. The first case is to instantiate a new LoginChecker object with a correct username & password combination then called check_login() which should return true. The second case also instantiates a new LoginChecker object but with an incorrect username & password combination then calls check_login() which

should return false (as such combination does not match any row in the database). In all cases, the test results are the same as what we expected since the test passed.
iv. Document the details of the bugs if there are any
There is no bug found here.

**Test7: testDisplay_profile() in ProfileTest = DS**
i. Location of the test case in your folder
testDisplay_profile() in ProfileTest
ii. Description of what it does and how to execute it
Right-click on ProfileTest, and select “run” to run this class of test cases, testDisplay_profile() is one of them. If just want to run this specific test, you need to open ProfileTest, navigate to testDisplay_profile(), then right-click on the method, and select “run” to run this specific test case.
iii. Result of the test case
In this test case, we test if a display_profile() correctly returns all information stored in the profile corresponding to a device_id (the information should be obtained from the database using this valid device_id). This is called in the Agent’s view_profile(), but the Profile serves as a base-level class that interacts directly with the database, so we still need to test it. There are two sub-test cases included. The first case is to instantiate a new Profile object with a correct device_id that exists in our database then call display_profile() which should return the corresponding user profile from the database that matches with this device_id. The second case is to instantiate a new Profile object with an incorrect device_id that does not exist in our database then call display_profile() which should return an empty result because we cannot get the device information from the database. In all cases, the test results are the same as what we expected since the test passed.
iv. Document the details of the bugs if there are any
There is no bug found here.

**XTC Test 1: check_availability() in TimeSlotTest**
i. Location of the test case in your folder:
app/java/com.example.uscrecapp_team28(test)/TimeSlotTest.java
ii. Description of what it does and how to execute it
Right-click on TimeSlotTest, and select “run” to run this class of test cases, check_availability() is one of them. If just want to run this specific test, you need to open TimeSlotTest, navigate to

check_availability(), then right-click on the method, and select “run” to run this specific test case. It tests the functionality of the method which checks if a user is available on a specific date. When the user makes the reservation, if the user already has a reservation on that date, the user cannot make the reservation

iii. Result of the test case

There are some initialized records in our database. There are two sub-test cases here. I picked a user and two dates for these two cases. On one day, the user has a reservation, while on another the user does not have a reservation. The function returns true when the input is the date that the user has no reservation on and is available. The function returns false when the input is the date that the user has a reservation on and is not available. In all cases, the test results are the same as what we expected since the test passed.

iv. Document the details of the bugs if there are any

There is no bug found here.

**XTC Test 2: check_full() in TimeSlotTest**

i. Location of the test case in your folder:

app/java/com.example.uscrecapp_team28(test)/TimeSlotTest.java

ii. Description of what it does and how to execute it

Right-click on TimeSlotTest, and select “run” to run this class of test cases, check_full() is one of them. If just want to run this specific test, you need to open TimeSlotTest, navigate to check_full(), then right-click on the method, and select “run” to run this specific test case. It tests the functionality of the method which checks if a timeslot is available on a specific date. When the user views the timeslots, if the timeslot is full, it will give the user an option to join the waitlist. Otherwise, it allows the user to book.

iii. Result of the test case

There are some initialized records in our database. There are two sub-test cases here. I picked two timeslots, and I set the capacity of them to be 1 and 0. Both timeslots have no current reservations. When we check if the timeslot is full on the timeslot that has a capacity of 0, it is full of course and the function returns true. In another timeslot, it returns false since it has one spot left and is not full. In all cases, the test results are the same as what we expected since the test passed.

iv. Document the details of the bugs if there are any

There is no bug found here.

**XTC Test 3: display_all_timeslot_info() in TimeSlotTest**

i. Location of the test case in your folder:

app/java/com.example.uscrecapp_team28(test)/TimeSlotTest.java

ii. Description of what it does and how to execute it

Right-click on TimeSlotTest, select “run” to run this class of test cases, display_all_timeslot_info() is one of them. If just want to run this specific test, you need to open TimeSlotTest, navigate to display_all_timeslot_info(), then right-click on the method, and select “run” to run this specific test case. It tests the functionality of the method which gets the information of all the timeslots in one center on one specific date. Then we need to display this information on our page.

iii. Result of the test case

There are two possible conditions (so two sub-test cases). One is that either the date or the center or maybe both are invalid, then we cannot get any information on the timeslots. What this function returns is a list of TimeSlotItem(a class that we defined to store the needed information of timeslots). If the date or the center is invalid, the returned array will have no element in it, and the size is 0. So we check if the size is 0 and it turns out to be true. Then we choose a date and center that are both valid. A center should have 6 timeslots each day, so we check if the list has 6 TimeSlotItems storing the information of the 6 timeslots. It turns out to be true. We also check some of the information of the items in the list to check if the information is derived correctly. The result is that all the information we want for a timeslot is derived correctly. In all cases, the test results are the same as what we expected since the test passed.

iv. Document the details of the bugs if there are any

There is no bug found here.

**XTC Test 4: add_user() in TimeSlotTest**

i. Location of the test case in your folder:

app/java/com.example.uscrecapp_team28(test)/TimeSlotTest.java

ii. Description of what it does and how to execute it

Right-click on TimeSlotTest, and select “run” to run this class of test cases, add_user() is one of them. If just want to run this specific test, you need to open TimeSlotTest, navigate to add_user() then right-click on the method, and select “run” to run this specific test case. It tests the functionality of the method which lets the user make a reservation in a timeslot. According to the three possible results of the trial, our app will give different messages to the user and do different operations in the database.

iii. Result of the test case

A user's reservation try can be successful or failed. If it's failed, it's probably because the timeslot just becomes full or because the user already has a reservation today. If it's successful, the timeslot must not be full and the user should be available. This method will return 0, 1, and 2 for the three different conditions. So I initialize three non-existent timeslots in the database (to test each of the three sub-test cases). One is full, another two are available. Then I try to call the method on the full timeslot, but the user cannot make the reservation and returns the corresponding integer for this condition. Then I call the method on an available timeslot, and it returns the right integer too. At last, I call the method on the last timeslot(on the same date as the second) and it fails to add the user. The method returns the integer corresponding to this condition too. The result is correct. In all cases, the test results are the same as what we expected since the test passed.

iv. Document the details of the bugs if there are any

There is no bug found here.

**XTC Test 5: join_waitlist() in TimeSlotTest**

i. Location of the test case in your folder:

app/java/com.example.uscrecapp_team28(test)/TimeSlotTest.java

ii. Description of what it does and how to execute it

Right-click on TimeSlotTest, and select "run" to run this class of test cases, join_waitlist() is one of them. If just want to run this specific test, you need to open TimeSlotTest, navigate to join_waitlist() then right-click on the method, and select "run" to run this specific test case. It tests the functionality of the method which lets the user join a waitlist in a timeslot. According to the 3 possible results, our app will give different messages to the user and do different operations in the database.

iii. Result of the test case

A user's join to a waitlist try can be successful or failed. If it's failed, it's probably because the user is already on the waitlist or because the user already made a reservation in the timeslot. If it's successful, the user should not have a reservation or be on the waitlist of the timeslot. This method will return 0, 1, and 2 for the three different conditions. So I initialize three non-existent timeslots in the database (to test each of the three sub-test cases). The user has one reservation in one of the timeslots, and the user already joined the waitlist of another timeslot. And the method returns the correct integers associated with the two failure conditions. And for the third timeslot, the user neither has a reservation nor joins the waitlist in the timeslot, and the

result is a success. In all cases, the test results are the same as what we expected since the test passed.

iv. Document the details of the bugs if there are any

There is no bug found here.

**JY Test 1: testCancel_reservation() in AgentTest - scenario 6 cancel upcoming reservations**

i. Location of the test case in your folder:

app/java/com.example.uscrecapp_team28(test)/AgentTest.java

ii. Description of what it does and how to execute it

Right-click on AgentTest, and select “run” to run this class of test cases, testCancel_reservation() is one of them. If just want to run this specific test, you need to open AgentTest, navigate to testCancel_reservation() then right-click on the method, and select “run” to run this specific test case. It makes a reservation and then cancels the added reservation. It tests the correctness of the database after we cancel the reservation. The canceled reservation should not appear on the reservation and availability tables.

iii. Result of the test case

The test case passes every time. We use the user with id 6 to make the reservation of time slot 104 and then cancel this reservation. The database will not contain any record for time slot 104. Do make sure the internet connection is stable so that the result is observable.

iv. Document the details of the bugs if there are any

There is no bug found here.

**JY Test 2: testView_all_reservations() in AgentTest - scenario 5 view booking information**

i. Location of the test case in your folder:

app/java/com.example.uscrecapp_team28(test)/AgentTest.java

ii. Description of what it does and how to execute it

Right-click on AgentTest, select “run” to run this class of test cases, testView_all_reservation() is one of them. If just want to run this specific test, you need to open AgentTest, navigate to testView_all_reservation() then right-click on the method, and select “run” to run this specific test case. It tests the correctness to view a user’s all reservations in the past and the future. The database is populated with specific reservations for the test user. The data acquired should be identical to the data in the database.

iii. Result of the test case

The test case passes every time. We display user “6”’s future preservation and history reservation. All of that information is correct. Do make sure the internet connection is stable so that the result is observable.
iv. Document the details of the bugs if there are any
There is no bug found here.

**JY Test 3: display_all_reservation_info()in ReservationTest - scenario 5 view booking information**

i. Location of the test case in your folder:
app/java/com.example.uscrecapp_team28(test)/ReservationTest.java
ii. Description of what it does and how to execute it
Right-click on ReservationTest, and select “run” to run this class of test cases, display_all_reservation_info() is one of them. If just want to run this specific test, you need to open ReservationTest, navigate to display_all_reservation_info() then right-click on the method, and select “run” to run this specific test case. It tests the correctness to view a valid user’s all reservations in the past and the future. The database is populated with specific reservations for the test user. The data acquired should be identical to the data in the database.
iii. Result of the test case
The test case passes every time. We display user “6”’s future reservation and history reservation. All of that information is correct. Do make sure the internet connection is stable so that the result is observable.
iv. Document the details of the bugs if there are any
There is no bug found here.

**JY Test 4: display_all_reservation_info_for_nonexist_user() in ReservationTest - scenario 5 view booking information**

i. Location of the test case in your folder:
app/java/com.example.uscrecapp_team28(test)/ReservationTest.java
ii. Description of what it does and how to execute it
Right-click on ReservationTest, select “run” to run this class of test cases, display_all_reservation_info_for_nonexist_user() is one of them. If just want to run this specific test, you need to open ReservationTest, navigate to display_all_reservation_info_for_nonexist_user() then right-click on the method, and select “run” to run this specific test case. It tests the extreme case to view a nonexistent user’s reservation

information. The program should not throw any error but retrieve a history and future booking item list with size 0.

iii. Result of the test case

The test case passes every time. We display user “-1”’s future reservation and history reservation, which is not valid. All of that information is correct. Do make sure the internet connection is stable so that the result is observable.

iv. Document the details of the bugs if there are any

There is no bug found here.

**JY Test 5: cancel_reservation_reservation_table() in ReservationTest - scenario 6 cancel upcoming reservations**

i. Location of the test case in your folder:

app/java/com.example.uscrecapp_team28(test)/ReservationTest.java

ii. Description of what it does and how to execute it

Right-click on ReservationTest, select “run” to run this class of test cases, cancel_reservation_reservation_table() is one of them. If just want to run this specific test, you need to open ReservationTest, navigate to cancel_reservation_reservation_table() then right-click on the method, and select “run” to run this specific test case. It makes a reservation and then cancels the added reservation. It tests the correctness of the database after we cancel the reservation. The canceled reservation should not appear on the reservation tables.

iii. Result of the test case

The test case passes every time. We use the user with id 6 to make the reservation of time slot 104 and then cancel this reservation. The reservation table will not contain any record for time slot 104. Do make sure the internet connection is stable so that the result is observable.

iv. Document the details of the bugs if there are any

There is no bug found here.

**JY Test 6: cancel_reservation_availability_table() in ReservationTest - scenario 6 cancel upcoming reservations**

i. Location of the test case in your folder:

app/java/com.example.uscrecapp_team28(test)/ReservationTest.java

ii. Description of what it does and how to execute it

Right-click on ReservationTest, select “run” to run this class of test cases, cancel_reservation_availability_table() is one of them. If just want to run this specific test, you

need to open ReservationTest, navigate to cancel_reservation_availability_table() then right-click on the method, and select “run” to run this specific test case. It makes a reservation and then cancels the added reservation. It tests the correctness of the database after we cancel the reservation. The canceled reservation should not appear in the availability tables.

iii. Result of the test case

The test case passes every time. We use the user with id 6 to make the reservation of time slot 104 and then cancel this reservation. The availability table will not contain any record for time slot 104. Do make sure the internet connection is stable so that the result is observable.

iv. Document the details of the bugs if there are any

There is no bug found here.

Test1:

i. Location of the test case in your folder

ii.  Description of what it does and how to execute it

iii. Result of the test case

iv. Document the details of the bugs if there are any

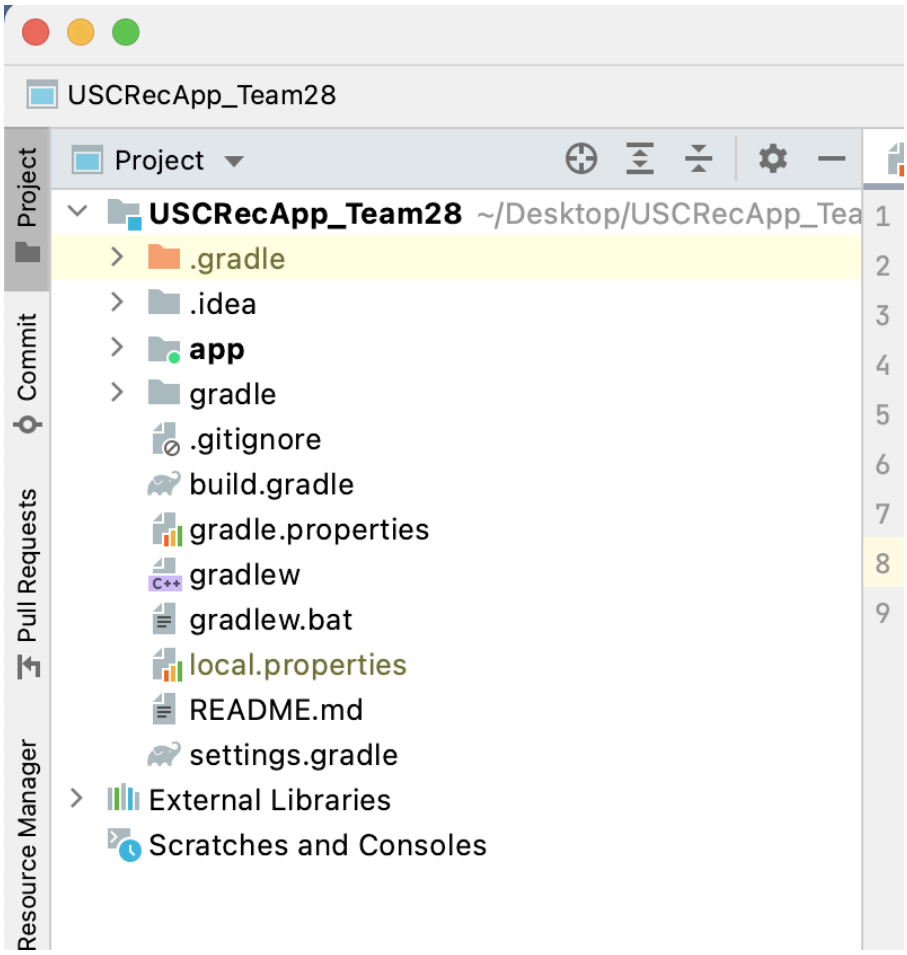
c. Explain the coverage criteria used in your white-box testing and what coverage level you are able to achieve in the end

For Activity Files and some Helper Adapter files, white-box testing cannot cover them well, and their behaviors will be tested by our black-box testing, so for our coverage criteria, we care about the classes where we implemented the core methods of our application（Agent.java, Authentication.java, BookingItem.java, LoginChecker.java, Profile.java,  Reservation.java, TimeSlot.java, and TimeslotItem.java, Logout.java)

You can see that we divide our java files into three categories, and our white-box tests focus on the files in the class folder. Overall, we have a 92% class coverage, 93% method coverage, and

86% line coverage. For most of the classes, the class and method coverage is 100% or very close to 100%, except for the Reservation class. The reason is that in reservation class, we connect to a firebase database to implement the functionality of notification, and this functionality is hard to test with white-box tests and black-box tests. We cover the tests of the notification in our demo for project 2.3. The missing percentage of Line coverage is comprehensible because some lines are used to catch exceptions and will not be executed. And the hardness to test notification functionality with white-box tests also contributes to the missing percentage of Line coverage, but it's a normal condition.

**If sdk is not found, click on the project and click the local.properties.**




Then change the sdk.dir to the directory where it stores the sdk.

local.properties

```
## This file must *NOT* be checked into Version Control Systems,
# as it contains information specific to your local configuration.
#
# Location of the SDK. This is only used by Gradle.
# For customization when using a Version Control System, please read the
# header note.
#Mon Apr 04 19:16:27 PDT 2022
sdk.dir=/Users/jiangyi/Library/Android/sdk

```