CPEN321 Milestone 9

PART 1

1. A high-level description of the progress since the previous milestone(M8)
    - Front-end:
        - Added partial implementation of complex logic which allows data sent from post page to main page to display results retrieved from back-end
    - Back-end:
        - Fixed bug for complex logic and write several test cases for the complex logic, Non-functional requirement test

2. The plans until the next milestone:

- Front-end:
    - Complete complex logic implementation
- Back-end:
    - Complete complex logic implementation

3. Major decisions and changes in the scope of the project (from the requirements and design document submitted when the groups were formed).

None

4. The contributions of individual team members to the work done so far.

Maxon Zhao:

- Implemented UI interaction, multi-user chat system, push notification functionality, and simple testing on UI components.
- communication between frontend and backend, backend components for multi-user chat system as well as fixing issues identified by Codacy.

Sheng Wang:

- Connect the front-end and back-end in the user case: “search”, send the request to the back-end, and get the search result. Send the “search” result from map activity to mainactivity.
- Fixed the left issues identified by Codacy.

Robin Lai:

- Non-functional requirement test
- Implemented and debugged the complex logic for the back-end part

Peter Lai:

- Implemented and debugged the complex logic for the back-end part
- Write an explanation of the complex logic for the back-end part
- Non-functional requirement test

PART 2
Back-end testing:

- A table summarizing the number of back-end tests per the tested modules and the number of integration tests, as specified in M7

Test List	**Number**
Modules 1 (test on Search functionality)	11
Modules 2 (test on Complex logic)	4
Integration Test	9

Modules 1(test on Search functionality): In total 11 test cases

Modules 1 Test	**Expected Behaviour**
Test: API Connect GET	Test if API connection is ok, the status code is expected to be 200
Test: API Connect POST	Test if API connection is ok, the status code is expected to be 200
Test: Search Good POST	Send a search request including price, location and types, the status code is expected to be 200
Test: Search Good GET	Check if it can get the correct search result by checking its price, location, types, phone, email and descript. If the result includes this information, we expect it to be true and the status code is expected to be 200
Test: Good Search Conditions	Entering a complete and correct search condition to see if we get the correct search result
Test: Empty Search conditions POST	the status code is expected to be 200
Test: Empty Search conditions GET	the status code is expected to be 401
Test: Empty Search conditions	Entering an empty search condition to see if we get the corresponding search result
Test: Empty Search result POST	the status code is expected to be 200
Test: Empty Search result GET	the status code is expected to be 402
Test: Empty Search result	The search result is empty

Modules 2(test on Complex logic): In total 4 test cases

Modules 2 Test	**Expected Behaviour**
Test: API GET	Test if API connection is ok, the status code is expected to be 200
Test: API POST	Test if API connection is ok, the status code is expected to be 200
Test: Good Recommendation	Pre-Enter the search history, and then do the recommendation logic. Then check if the recommendation result's price is the same as it predefined
Test: No history	No search history should return nothing, so expected nothing in the recommendation sting result

Integration tests: In total 9 test cases

Integration Test	**Expected Behaviour**
Test: GET	The status code is expected to be 200
Test: POST	The response-text is except to be "message: 3000 location: PV" as we post data to front end
Test: Search POST	The status code is expected to be 200
Test: Search GET	The excepted result is the data send in the POST process, and should include "price":3000,"location":"PV","types":"1","phone":"1","email":"1","descript":"1"
Test: Empty Search POST	The status code is expected to be 200
Test: Empty Search GET	The status code is expected to be 401
Test: Empty Result POST	The status code is expected to be 200
Test: Empty Result GET	The status code is expected to be 402
Test: Search Complex Logic GET	Test if complex logic work properly, and if it returns the correct result The status code is expected to be 200

- A screenshot with success/fail condition of each test specified in M7

Modules 1 (test on Search functionality)

```
test("test search good POST", done => {
  mocked_Search_GET_good((s_price, s_location, s_types)=> {
    request(app)
    .post('/search')
    .send({
      price: s_price,
      location: s_location,
      types: s_types,
      })
    .then(response => {
      expect(response.statusCode).toBe(200);
      done();
    });
  });
});
test("test search good GET", done => {
    request(app)
        .get('/search')
        .then(response => {
            expect(response.text.includes("\"price\":3000,\"location\":\"PV\",\"types\":\"1\",\"phone\":\"1\",\"email\":\"1\",\"descript\":\"1\"")).toBe(true);
            expect(response.statusCode).toBe(200);
            done();
        });
});
test("good search conditions", done => {
  mocked_Search_GET_good((s_price, s_location, s_types)=> {
    mysearch(s_price, s_location, s_types,
      (err, data) => {
          expect(data[0].price).toBe(s_price);
          expect(data[0].location).toBe(s_location);
          expect(data[0].types).toBe(s_types);
          done();
      },
      ()=>{
        var bo = (s_price == "" || s_location == "" || s_types == "");
        expect(bo).toBe(true);done();
      }
      );
    });
});
test("empty search conditions POST", done => {
  mocked_Search_GET_bad1((s_price, s_location, s_types)=> {
    request(app)
    .post('/search')
    .send({
      price: s_price,
      location: s_location,
      types: s_types,
      })
    .then(response => {
      expect(response.statusCode).toBe(200);
      done();
    });
  });
});
```

```
test("empty search conditions GET", done => {
    request(app)
        .get('/search')
        .then(response => {
            expect(response.statusCode).toBe(401);
            done();
        });
});
```

```
test("empty search conditions", done => {
  mocked_Search_GET_bad1((s_price, s_location, s_types)=> {
  mysearch(s_price, s_location, s_types,
    (err, data) => {
        expect(data[0].price).toBe(s_price);
        expect(data[0].location).toBe(s_location);
        expect(data[0].types).toBe(s_types);
        done();
      },
    ()=>{
      var bo = (s_price == "" || s_location == "" || s_types == "");
      expect(bo).toBe(true);done();
    }
    );
  });
});

test("empty search result POST", done => {
  mocked_Search_GET_bad2((s_price, s_location, s_types)=> {
    request(app)
    .post('/search')
    .send({
      price: s_price,
      location: s_location,
      types: s_types,
      })
    .then(response => {
      expect(response.statusCode).toBe(200);
      done();
    });
  });

});
test("empty search result GET", done => {
  request(app)
      .get('/search')
      .then(response => {
          expect(response.statusCode).toBe(402);
          done();
      });
});
test("empty search result", done => {
  mocked_Search_GET_bad2((s_price, s_location, s_types)=> {
  mysearch(s_price, s_location, s_types,
    (err, data) => {
        expect(data.length).toBe(0);
        done();
    },
    ()=>{
      var bo = (s_price == "" || s_location == "" || s_types == "");
      expect(bo).toBe(true);done();
    }
    );
  });
});
```

Modules 2 (test on Complex logic)

```
test("test api connect GET", done => {
  request(app)
    .get('/')
    .then(response => {
      expect(response.text).toBe("data:");
      expect(response.statusCode).toBe(200);
      done();
    });
});
test("test api connect POST", done => {
  request(app)
    .post('/')
    .send(postdata)
    .then(response => {
      expect(response.statusCode).toBe(200);
      done();
    });
});
test("good recommendation", done => {
  mocked_recommendation_good((arr)=>{
    var re = [];
    var ids = arr;
    mylogic(re,ids,1,()=>{
      expect(re[0].price).toBe(3000);
      expect(re[0].location).toBe("PV");
      expect(re[0].types).toBe("1");
      expect(re[0].phone).toBe("1");
      expect(re[0].email).toBe("1");
      expect(re[0].descript).toBe("1");
      done();
    }, 1);
  });
});
test("no history", done => {
  mocked_recommendation_empty((arr)=>{
    var re = [];
    var ids = arr;
    mylogic(re,ids,1,()=>{
      expect(re).toStrictEqual([]);
      done();
    }, 0);
  });
});
```

Integration Test

```
test("test GET", done => {
  request(app)
    .get('/')
    .then(response => {
      expect(response.text).toBe("data:");
      expect(response.statusCode).toBe(200);
      done();
    });
});
test("test POST", done => {
  request(app)
    .post('/')
    .send(postdata)
    .then(response => {
      expect(response.any.toBe("{\"message\":\"price:3000  location:PV\"}");
      expect(response.statusCode).toBe(200);
      done();
    });

});


test("test search POST", done => {
  request(app)
    .post('/search')
    .send(data)
    .then(response => {
      expect(response.statusCode).toBe(200);
      done();
    });

});
test("test search GET", done => {
  request(app)
      .get('/search')
      .then(response => {
          expect(response.text.includes("\"price\":3000,\"location\":\"PV\",\"types\":\"1\",\"phone\":\"1\",\"email\":\"1\",\"descript\":\"1\"")).toBe(true);
          expect(response.statusCode).toBe(200);
          done();
      });
});

test("test empty search POST", done => {
  request(app)
    .post('/search')
    .send({price: "",
      location: "",
      types: ""})
    .then(response => {
      expect(response.statusCode).toBe(200);
      done();
    });

});
```

```
test("test empty search GET", done => {
  request(app)
      .get('/search')
      .then(response => {
          expect(response.statusCode).toBe(401);
          done();
      });
});
test("test empty result POST", done => {
  request(app)
    .post('/search')
    .send({price: "5000",
      location: "PV",
      types: "1"})
    .then(response => {
      expect(response.statusCode).toBe(200);
      done();
    });

});
test("test empty result GET", done => {
  request(app)
      .get('/search')
      .then(response => {
          expect(response.statusCode).toBe(402);
          done();
      });
});

test("test complex logic GET", done => {
  request(app)
      .get('/logic')
      .then(response => {
          expect(response.text.includes("\"price\":3000,\"location\":\"PV\",\"types\":\"1\",\"phone\":\"1\",\"email\":\"1\",\"descript\":\"1\"")).toBe(true);
          expect(response.statusCode).toBe(200);
          done();
      });
})
```

- A statement-and method-level coverage report for tests of each of the two modules under test.

Module 1 Test:

```
------------|---------|----------|---------|---------|-------------------
File        | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
------------|---------|----------|---------|---------|-------------------
All files   |   76.54 |     61.9 |   76.47 |   78.67 |
 app.js     |   91.67 |      100 |   83.33 |   91.11 | 91-94
 mylogic.js |   34.78 |        0 |   33.33 |      40 | 15-31
 mysearch.js|     100 |      100 |     100 |     100 |
------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       13 skipped, 11 passed, 24 total
Snapshots:   0 total
Time:        3.343 s
Ran all test suites with tests matching "Module1".
```

Module 2 Test:

```
------------|---------|----------|---------|---------|-------------------
File        | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
------------|---------|----------|---------|---------|-------------------
All files   |    64.2 |    47.62 |   47.06 |   65.33 |
 app.js     |   45.83 |       25 |   33.33 |   48.89 | 45-48,54-85,91-94
 mylogic.js |     100 |      100 |     100 |     100 |
 mysearch.js|      70 |        0 |      50 |      70 | 12-19
------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       20 skipped, 4 passed, 24 total
Snapshots:   0 total
Time:        3.102 s
Ran all test suites with tests matching "Module2".
```

- A statement-and method-level coverage report for each integration test separately and all integration tests together

```
------------|---------|----------|---------|---------|-------------------
File        | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
------------|---------|----------|---------|---------|-------------------
All files   |   95.06 |    80.95 |     100 |   97.33 |
 app.js     |     100 |      100 |     100 |     100 |
 mylogic.js |   82.61 |       50 |     100 |      90 | 29-31
 mysearch.js|     100 |      100 |     100 |     100 |
------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       15 skipped, 9 passed, 24 total
Snapshots:   0 total
Time:        3.261 s
Ran all test suites with tests matching "inttest".
```

Front-end testing:

A list of tests we created:

Use case	**Multi-user chat**
Logic Test Case	Expected Results
Click on Chat on the navigation bar	Check UI is updated to chat activity
Enter “Maxon” in the input text field and click on the enter room button	Check UI is updated to chatroom activity
	Check image button is displayed
Enter “hello” as input	Check that the send button is visible and image button is invisible
Click on send button	Check that the send button is invisible and image button is visible again
	Check hello is sent and visible on the recycler view
Press back	Check UI is updated to chat activity
	Check the username text is shown up below as: “username: Maxon”

How do we encoded success/failure criteria:

In this particular test on multi-user chat system, we define success as being able to navigate to the chat system page from main page, entering username as user specified to enter the chatroom, as well as being able to send custom messages with send button available, and in the end, being able to navigate back to the home page. In contrast, a failure would occur if any information mismatch, such as entered username does not match the username shown up in the chatroom, or the system crashed in the steps mentioned above.

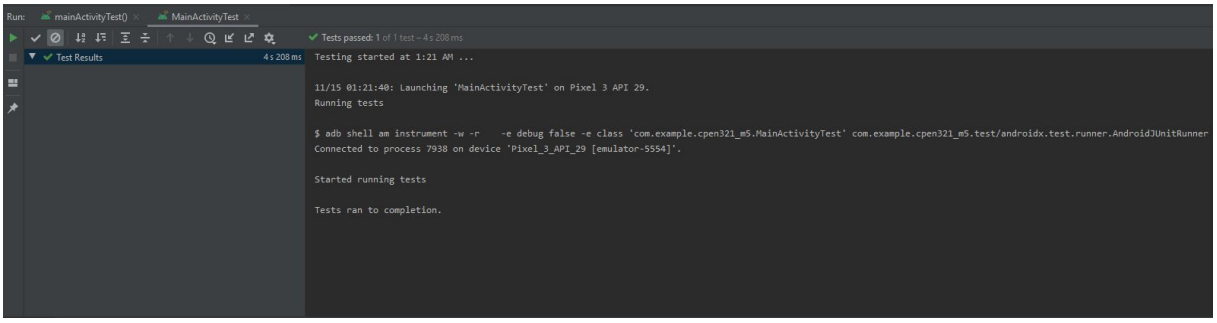
3 Examples of how we define/check success/failure status:

```
@Test
public void mainActivityTest() {
    ViewInteraction bottomNavigationItemView = onView(
            allOf(withId(R.id.nav_chat), withContentDescription( text: "Chat"),
                    childAtPosition(
                            childAtPosition(
                                    withId(R.id.bottomNavigationView),
                                     position: 0),
                             position: 2),
                    isDisplayed()));
    bottomNavigationItemView.perform(click());

    ViewInteraction appCompatEditText = onView(
            allOf(withId(R.id.editText),
                    childAtPosition(
                            childAtPosition(
                                    withId(android.R.id.content),
                                     position: 0),
                             position: 0),
                    isDisplayed()));
    appCompatEditText.perform(replaceText( stringToBeSet: "Maxon"), closeSoftKeyboard());
```

```
ViewInteraction appCompatButton = onView(
        allOf(withId(R.id.enterBtn), withText("Enter room"),
                childAtPosition(
                        childAtPosition(
                                withId(android.R.id.content),
                                 position: 0),
                         position: 1),
                isDisplayed()));
appCompatButton.perform(click());

ViewInteraction imageView = onView(
        allOf(withId(R.id.pickImgBtn),
                isDisplayed()));
imageView.check(matches(isDisplayed()));

ViewInteraction editText = onView(
        allOf(withId(R.id.messageEdit), withText("Message..."),
                isDisplayed()));

ViewInteraction appCompatEditText2 = onView(
        allOf(withId(R.id.messageEdit),
                isDisplayed()));
appCompatEditText2.perform(replaceText( stringToBeSet: "hello"), closeSoftKeyboard());

ViewInteraction textView = onView(
        allOf(withId(R.id.sendBtn), withText("Send"),
                isDisplayed()));
textView.check(matches(isDisplayed()));
```

```
    appCompatTextView.perform(click());

    ViewInteraction textView2 = onView(
            allOf(withId(R.id.sentTxt), withText("hello"),
                    isDisplayed()));
    textView2.check(matches(withText("hello")));

    ViewInteraction appCompatImageView = onView(
            allOf(withId(R.id.pickImgBtn),
                    isDisplayed()));

    pressBack();

    ViewInteraction textView3 = onView(
            allOf(withId(R.id.username), withText("username: Maxon"),
                    isDisplayed()));
    textView3.check(matches(withText("username: Maxon")));

    ViewInteraction bottomNavigationItemView2 = onView(
            allOf(withId(R.id.nav_home), withContentDescription( text: "Home"),
                    childAtPosition(
                            childAtPosition(
                                    withId(R.id.bottomNavigationView),
                                     position: 0),
                             position: 0),
                    isDisplayed()));
    bottomNavigationItemView2.perform(click());
}
```

A log of the automated execution of these tests on the device, including their pass/fail status:



In the "search" main user case, the user can choose anyone within the given address selection range. Among the room types, the user can also choose whatever he wants, and can make changes before submitting it. At the same time, in the price column, The user can enter an amount greater than zero and less than 10,000, and the seek bar can correspond to the amount, click the submit button, this activity should be finished, and return to the main interface. If the UI cannot be displayed, the address and room type cannot be selected and changed, the amount of price cannot be entered, clicking the submit button cannot return to the main page, and the map cannot be displayed normally, it will be regarded as a failure.

```
ViewInteraction textView = onView(
        allOf(withId(android.R.id.text1), withText("Totem Park"),
                childAtPosition(
                        allOf(withId(R.id.search_loc_spi),
                                childAtPosition(
                                        withId(R.id.linearLayout),
                                        position: 1)),
                                position: 0),
                isDisplayed()));
textView.check(matches(withText("Totem Park")));
```

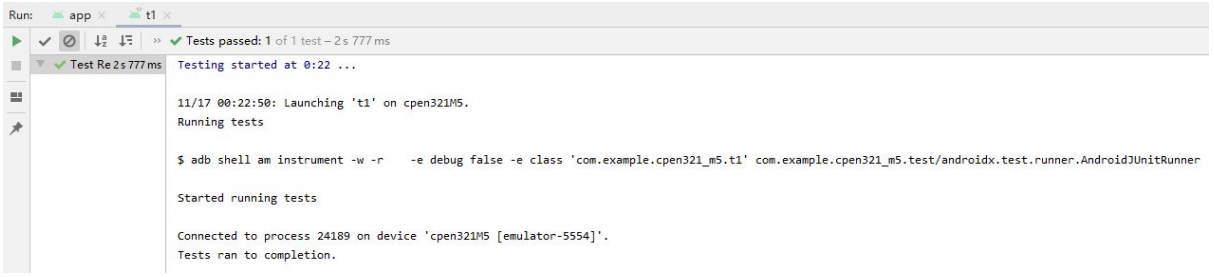
This part of the test checks whether the address can be selected.

```
ViewInteraction editText2 = onView(
        allOf(withId(R.id.search_price_edi), withText("1234"),
                childAtPosition(
                        allOf(withId(R.id.linearLayout),
                                childAtPosition(
                                        withId(R.id.wholepage),
                                        position: 0)),
                                position: 5),
                isDisplayed()));
editText2.perform(pressImeActionButton());

ViewInteraction editText3 = onView(
        allOf(withId(R.id.search_price_edi), withText("1234"),
                childAtPosition(
                        allOf(withId(R.id.linearLayout),
                                childAtPosition(
                                        withId(R.id.wholepage),
                                        position: 0)),
                                position: 5),
                isDisplayed()));
editText3.check(matches(withText("1234")));
```

This part of the test checks whether the price can be entered.

```
ViewInteraction view = onView(
        allOf(withContentDescription( text: "Google Map"),
                childAtPosition(
                        childAtPosition(
                                withId(R.id.map),
                                 position: 0),
                        position: 0),
                isDisplayed()));
view.check(matches(isDisplayed()));
```

This part of the test checks whether the google map can be displayed normally.



This picture shows that all the tests are passed, this main user case success.

In the main use case of “post”, similar to user case“search”, the user can choose anyone within the given address selection range. Among the room types, the user can also choose whatever he wants, and can make changes before submitting. In the price column, the user can enter an amount greater than zero and less than 10,000, but the post will have three more parts, and the user can enter a valid mobile phone number less than 10 digits. In the mailbox column, the user can enter his own mailbox number, besides, The user can write a description of fewer than 450 words in the description column, click the submit button, this interface should be launched and return to the main interface. If the UI cannot be displayed, the address and room type cannot be selected, the amount cannot be entered, the submit button cannot return to the main interface, the user can not enter the phone number, email, description. It will be regarded as a failure.

```
ViewInteraction textInputEditText4 = onView(
        allOf(withId(R.id.editphonetie), withText("2368670720")));
textInputEditText4.perform(scrollTo(), replaceText( stringToBeSet: "23686707201"));

ViewInteraction textInputEditText5 = onView(
        allOf(withId(R.id.editphonetie), withText("23686707201")));
textInputEditText5.perform(closeSoftKeyboard());

ViewInteraction editText5 = onView(
        allOf(withId(R.id.editphonetie), withText("23686707201")));
editText5.check(matches(withText("23686707201")));

ViewInteraction textInputEditText6 = onView(
        allOf(withId(R.id.editphonetie), withText("23686707201")));
textInputEditText6.perform(scrollTo(), replaceText( stringToBeSet: "2368670720"));
```

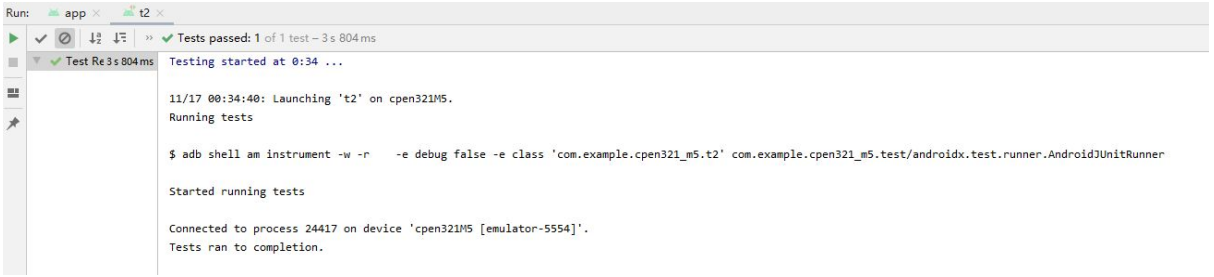
This part of the test checks the user can not enter more than 10 numbers in the phone number editor.

```
ViewInteraction textInputEditText2 = onView(
        allOf(withId(R.id.editemailtie)));
textInputEditText2.perform(scrollTo(), replaceText( stringToBeSet: "wangshengwalter@gmail.com"), closeSoftKeyboard());

ViewInteraction editText3 = onView(
        allOf(withText("wangshengwalter@gmail.com")));
editText3.check(matches(withText("wangshengwalter@gmail.com")));
```

This part of the test checks users can enter email.

```
ViewInteraction textInputEditText3 = onView(
        withId(R.id.editdescriptie));
textInputEditText3.perform(scrollTo(), replaceText( stringToBeSet: "test pass..."), closeSoftKeyboard());

ViewInteraction editText4 = onView(
        allOf(withText("test pass...")));
editText4.check(matches(withText("test pass...")));
```

This part of the test checks the user can enter a description.

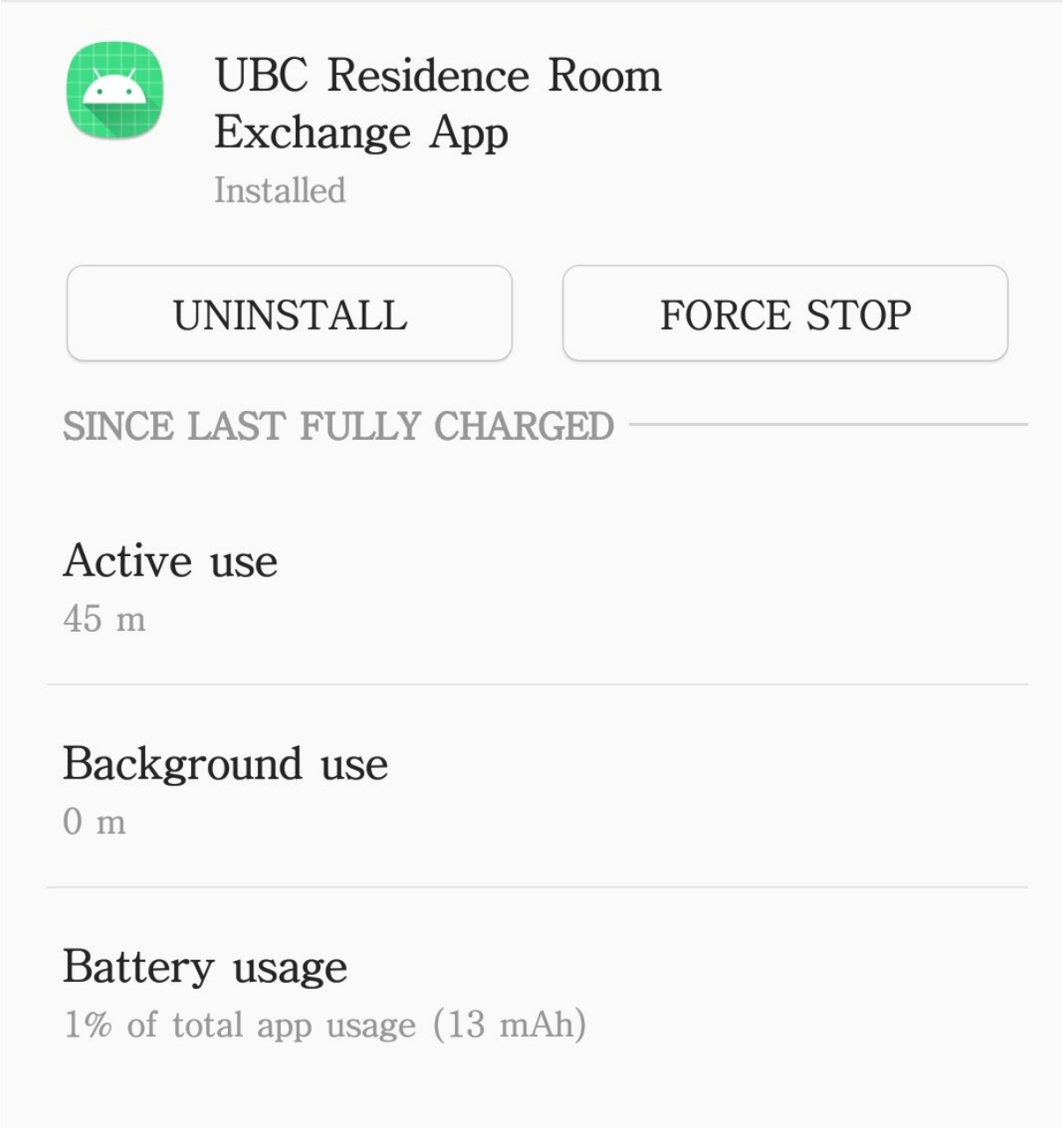


Tests ran to completion and success.

Non-functional requirements:

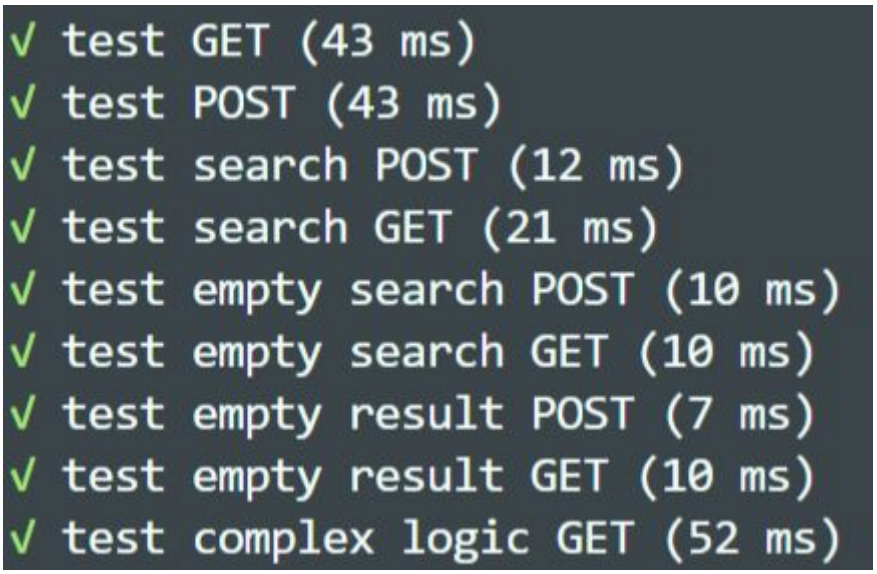
- For each non-functional requirements identified in M7, show the log with your verification results. Explain in 1-2 sentences how you verified each requirement.

Battery:



As you can see from the image, the App is actively used in 45 mins, and use only 13 mAh battery, which is only 1% of total app usage. We can calculate that: our app cost battery 0.289 mAh/min

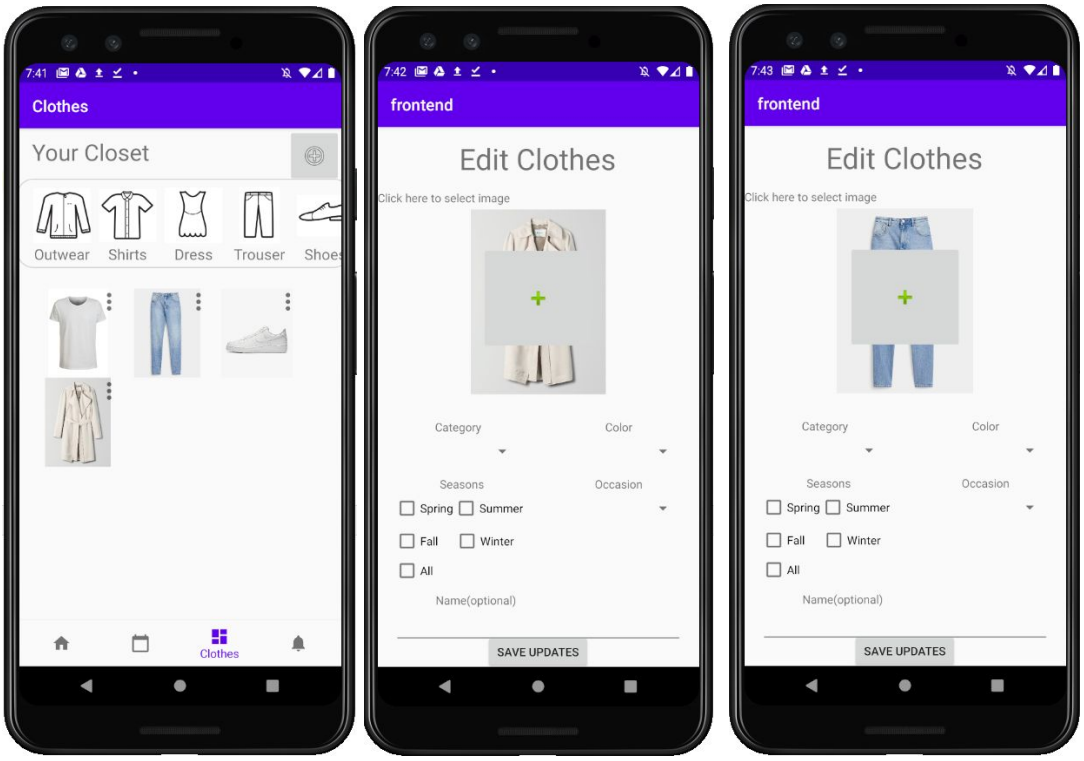
Request speed:



All of the requests are within 100ms, which is very fast responds

Manual customer acceptance testing:

- Report one major fault you found in your partner's team app:



(Displayed) (Expected)

When we have different clothes in the closet page and click "edit" on any piece of clothes, the clothes image displayed in the "edit clothes" page is always the last clothes in the closet.

For example, when we edit the jeans, the displayed image should be jeans, but now the image is the coat. The issue is only with the image display, but the edit functionality is still working.

- Report one major fault that your peer-team found in your app:

When submit search request to find a list of rooms that meets certain criterias such as listing all shared rooms whose price is under 3000 located in Place Vanier,
the results of queries, such as price, image, host contact information, are supposed to show up on the main page. However, when returning back to the main page, no results are available.