Testing Documentation

Project Name: Passport Application System

Student Name: Stephen McDonagh

Supervisor: Alistair Sutherland

Module Code: CA400

Student Number: 14518233

Submission Date: 17-05-2019

Contents

0 Description

This document will outline some details of the testing covered as part of my CA400 final year project. This document will cover Unit Testing, Integration Testing, Angular UI Testing, Facial Detection System Testing, Stress Testing, Accessibility Testing & System Testing

1 Unit Testing

Unit Testing (component testing) involves testing each of the individual parts of the application. This takes place before integration testing.

I am using maven as my build tool which allows me to run the command 'mvn clean test' which will run all my unit & integration tests in a suite.

I was able to run all my tests in a pipeline, I have two jobs on my pipeline which is 'Build stage' & 'Test stage'. My .yml file can be found at the root of my directory.

I was able to implement HTML reporting with when my tests run. This will give me the coverage of my tests. A screen shot can be seen below. As you can see I have 56% coverage of my application.

Application

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.passport.Application.Services		3%		0%	3	4	12	13	1	2	0	1
com.passport.Application.controllers		60%		50%	3	13	12	33	2	11	0	3
com.passport.Application		33%		n/a	1	2	3	4	1	2	0	1
com.passport.Application.models		100%		n/a	0	26	0	44	0	26	0	2
Total	143 of 332	56%	6 of 8	25%	7	45	27	94	4	41	0	7

An example unit test can be seen below

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest(classes = Application.class)
@WebAppConfiguration
public class RestControllerUnitTest extends Abstract{

    @Autowired
    FacialDetectionController facialDetectionController;

    @Override
    @Before
    public void setUp() { super.setUp(); }

    //Unit Test to check we get correct response for all applications.
    @Test
    public void getValidResponse() throws Exception {
        String uri = "/api/v1/applications";
        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)
                .accept(MediaType.APPLICATION_JSON_VALUE)).andReturn();
        int status = mvcResult.getResponse().getStatus();
        assertEquals( expected: 200, status);
    }
```

2 Integration Testing

Integration testing involves combining components and then testing them together. For me this would involve testing my Springboot backend with my SQLite database.

An example can be seen below. In this test I am checking the REST API GET request, I am getting the first element in my database and using a JSONAssert to check I am getting the correct response.

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = Application.class, webEnvironment = SpringBootTest.WebEnvironment
public class IntegrationTests {

    @LocalServerPort
    private int port;

    TestRestTemplate restTemplate = new TestRestTemplate();

    HttpHeaders headers = new HttpHeaders();

    //Testing Get request for springboot with my Sqlite DB
    @Test
    public void testRetrieveApplication() throws JSONException {
        HttpEntity<String> entity = new HttpEntity<~>( body: null, headers);

        ResponseEntity<String> response = restTemplate.exchange(
                createURLWithPort( uri: "api/v1/applications/1"),
                HttpMethod.GET, entity, String.class);

        String expected = "{\"id\":1,\"name\":\"Stephen Mcdonagh\",\"dob\":\"01-20-2110\",\

        JSONAssert.assertEquals(expected, response.getBody(), strict: true);
    }

    private String createURLWithPort(String uri) {
        return "http://localhost:" + port + uri;
    }
}
```

3 Angular UI Testing (e2e Tests)

As I had gotten some experience using Selenium Web driver during my internship, I felt this would be a good opportunity to use the experience I had gained to my advantage. For Angular, I can use 'Protractor' for my User Interface (UI) tests. This mocks how the user will interact with the UI and checks that the system works as expected.

All these tests can be found under src\ApplicationSystem-Ui\e2e\src

3.1 Tests

These tests require 2 files:

- app.e2e-spec: This is where the tests are written.
- app.po: This is where the web elements I'm interacting with are defined.

This is an example of how I'm getting the Date of birth Input field Web element

```
getDateOfBirthInputField(){
  return element(by.id('Dob'));
}
```

This is an example Test, where I'm checking if the correct error message is displayed to the user when a Date of birth in the incorrect format is inputted.

```
it('Should Display DOB Error Message',() =>{
  page.navigateTo();
  page.getDateOfBirthInputField().sendKeys('03/10/1996')
  expect(page.getDobIncorrectFormat()).toEqual('Please enter a valid date of birth (DD-MM-YYYY)');
});
```

Using the command 'ng e2e' I can run all my tests which is displayed as follows

```
Passport Application System UI Tests
  √ Should Display Application Heading
  √ Should Display All Navigation Bar Buttons
  √ Should Display Submit Button
  √ Should Display Submit Button
  √ Should Route To About Page
  √ Should Route To Chatbot Page
  √ Should Check Footer LOCATION Heading
  √ Should Check Footer MY DETAILS Heading
  √ Should Check Footer ABOUT PASSPORT Heading
  √ Should Check LinkedIn & Github Images Are Present
  √ Should Check If Images Are Displayed
  √ Should Display DOB Error Message
  √ Should Display Email Error Message
  √ Should Display Pps Error Message
  √ Should Display Phonee Error Message

Executed 15 of 15 specs SUCCESS in 19 secs.
[18:01:24] I/launcher - 0 instance(s) of WebDriver still running
[18:01:24] I/launcher - chrome #01 passed
```

4 Facial Detection System Testing

4.1 Data

For testing my facial detection system, I needed a dataset of opensource images of faces. I was able to get this data from the Georgia Tech Face Database(Here). This dataset contains images of 50 different people. Some of the factors that changed in these images included:

- Gender
- Hair Length
- Nationality
- Lighting
- Facial Expression
- Placement of face
- Wearing glasses

4.2 Testing

I am running my tests within Postman as this allows for me to quickly send a post request to my application which in turn displays the result to me.

I have tested 50/50 of the different people within this dataset.

For each person there is 15 images. I have chosen 3 randomly from each person to include in my final dataset.

4.3Results

As of now 141/150 images were correctly detected.

In 141 of the POST requests I was correctly given the answer of 1 face detected.

In 9 of the POST requests I was incorrectly given the answer of 2 faces detected (when only 1 face was present)

Below are screenshots of the Springboot log files showing my results

```
CORRECT IMAGE TYPE
Successfully Uploaded: P2-02.jpg
Detected 2 faces
```

```
CORRECT IMAGE TYPE
Successfully Uploaded: P7-01.jpg
Detected 2 faces
```

```
CORRECT IMAGE TYPE
Successfully Uploaded: P18-03.jpg
Detected 2 faces
```

```
CORRECT IMAGE TYPE
Successfully Uploaded: P28-02.jpg
Detected 2 faces
```

```
CORRECT IMAGE TYPE
Successfully Uploaded: P34-02.jpg
Detected 2 faces
```

```
CORRECT IMAGE TYPE
Successfully Uploaded: p36-03.jpg
Detected 2 faces
```

```
CORRECT IMAGE TYPE
Successfully Uploaded: P42-03.jpg
Detected 2 faces
```

```
CORRECT IMAGE TYPE
Successfully Uploaded: P44-03.jpg
Detected 2 faces
```

```
CORRECT IMAGE TYPE
Successfully Uploaded: P48-01.jpg
Detected 2 faces
```

4.4 Conclusion

Based on the testing I have completed, I have seen that the LBP Classifier system is very sensitive. I am saying this because in 9 images inputted I was given the output that 2 faces were detected, even though only 1 was present.

141/150 = 94%

5 Stress Testing

Stress testing involves overloading your application to the point where performance goes down or the application stops working. I was curious to see how the performance of my application would be given a high number of users.

5.1 Stress Testing Tools Used

As I am using Springboot which is a java framework, I will be using JMeter to help me run my stress tests. J meter allows you to change the number of threads(users) as well as showing you reports of how your application handled the stress.

5.2 Stress Testing Results

View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	17:23:39.485	Users 1-1	HTTP Request	16		2905	137	16	3

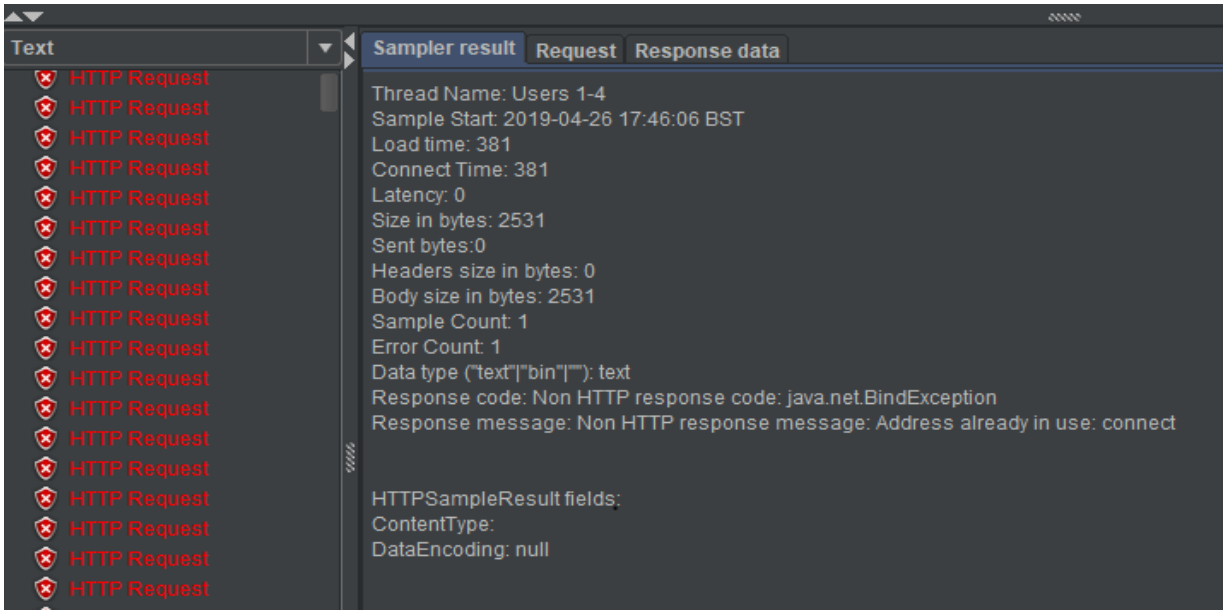
The Above image shows the results of sending a HTTP GET Request to my Springboot application.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	17:23:39.485	Users 1-1	HTTP Request	16		2905	137	16	3
2	17:28:24.746	Users 1-1	HTTP Request	21		2905	137	20	2
3	17:28:24.776	Users 1-1	HTTP Request	15		2905	137	14	1
4	17:28:24.792	Users 1-1	HTTP Request	13		2905	137	13	1
5	17:28:24.806	Users 1-1	HTTP Request	13		2905	137	12	1
6	17:28:24.819	Users 1-1	HTTP Request	17		2905	137	15	2
7	17:28:24.836	Users 1-1	HTTP Request	15		2905	137	15	2
8	17:28:24.847	Users 1-2	HTTP Request	18		2905	137	17	1
9	17:28:24.852	Users 1-1	HTTP Request	16		2905	137	14	1
10	17:28:24.865	Users 1-2	HTTP Request	17		2905	137	15	1
11	17:28:24.868	Users 1-1	HTTP Request	20		2905	137	18	1
12	17:28:24.883	Users 1-2	HTTP Request	15		2905	137	14	1
13	17:28:24.889	Users 1-1	HTTP Request	9		2905	137	8	0
14	17:28:24.898	Users 1-2	HTTP Request	12		2905	137	12	1
15	17:28:24.898	Users 1-1	HTTP Request	16		2905	137	16	1
16	17:28:24.915	Users 1-1	HTTP Request	12		2905	137	10	0
17	17:28:24.911	Users 1-2	HTTP Request	17		2905	137	16	0
18	17:28:24.928	Users 1-1	HTTP Request	11		2905	137	10	0
19	17:28:24.928	Users 1-2	HTTP Request	15		2905	137	14	1
20	17:28:24.940	Users 1-1	HTTP Request	13		2905	137	12	0
21	17:28:24.944	Users 1-3	HTTP Request	10		2905	137	8	1
22	17:28:24.943	Users 1-2	HTTP Request	22		2905	137	21	1
23	17:28:24.965	Users 1-2	HTTP Request	14		2905	137	13	1
24	17:28:24.955	Users 1-3	HTTP Request	38		2905	137	37	0
25	17:28:24.994	Users 1-3	HTTP Request	8		2905	137	7	0
26	17:28:24.954	Users 1-1	HTTP Request	63		2905	137	61	0
27	17:28:25.002	Users 1-3	HTTP Request	17		2905	137	9	1
28	17:28:24.979	Users 1-2	HTTP Request	45		2905	137	44	1
29	17:28:25.017	Users 1-1	HTTP Request	16		2905	137	15	1
30	17:28:25.024	Users 1-2	HTTP Request	18		2905	137	16	1
31	17:28:25.033	Users 1-1	HTTP Request	17		2905	137	17	1
32	17:28:25.044	Users 1-4	HTTP Request	15		2905	137	14	1
33	17:28:25.042	Users 1-2	HTTP Request	23		2905	137	22	1

After I tested using 10 users and adding one in every second. As you can see from the samples below, the time doesn't change too much for the first few iterations.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
13030	17:28:47.149	Users 1-5	HTTP Request	35		2905	137	35	1
13031	17:28:47.182	Users 1-8	HTTP Request	4		2905	137	4	1
13032	17:28:47.183	Users 1-10	HTTP Request	3		2905	137	3	1
13033	17:28:47.184	Users 1-1	HTTP Request	4		2905	137	4	1
13034	17:28:47.184	Users 1-5	HTTP Request	4		2905	137	4	1
13035	17:28:47.169	Users 1-9	HTTP Request	21		2905	137	21	1
13036	17:28:47.186	Users 1-8	HTTP Request	4		2905	137	4	0
13037	17:28:47.186	Users 1-10	HTTP Request	5		2905	137	5	0
13038	17:28:47.188	Users 1-5	HTTP Request	3		2905	137	3	0
13039	17:28:47.164	Users 1-6	HTTP Request	28		2905	137	28	1
13040	17:28:47.191	Users 1-10	HTTP Request	3		2905	137	3	1
13041	17:28:47.192	Users 1-5	HTTP Request	2		2905	137	2	0
13042	17:28:47.192	Users 1-6	HTTP Request	4		2905	137	4	0
13043	17:28:47.194	Users 1-5	HTTP Request	3		2905	137	3	1
13044	17:28:47.194	Users 1-10	HTTP Request	3		2905	137	3	1
13045	17:28:47.163	Users 1-3	HTTP Request	34		2905	137	34	1
13046	17:28:47.197	Users 1-5	HTTP Request	2		2905	137	1	0
13047	17:28:47.197	Users 1-10	HTTP Request	2		2905	137	2	0
13048	17:28:47.162	Users 1-2	HTTP Request	38		2905	137	38	1
13049	17:28:47.197	Users 1-3	HTTP Request	3		2905	137	3	1
13050	17:28:47.199	Users 1-5	HTTP Request	2		2905	137	2	0
13051	17:28:47.200	Users 1-2	HTTP Request	1		2905	137	1	0
13052	17:28:47.196	Users 1-6	HTTP Request	6		2905	137	6	0
13053	17:28:47.200	Users 1-3	HTTP Request	8		2593	0	0	1
13054	17:28:47.202	Users 1-6	HTTP Request	13		2593	0	0	0
13055	17:28:47.201	Users 1-2	HTTP Request	25		2593	0	0	1
13056	17:28:47.190	Users 1-8	HTTP Request	34		2593	0	0	0
13057	17:28:47.201	Users 1-5	HTTP Request	22		2593	0	0	0

As you can see Above, the status changed after approximately 13000 samples. This then gave the following error in the JMeter GUI:



6 Accessibility Testing

Accessibility give developers an idea if their application is useable by people with disabilities such as hearing, colour blindness, old age etc. Taking this into consideration, I thought it would be a good idea to conduct some accessibility testing on my Web application. This would give me an idea of what changes need to be made on my frontend if work was to continue with this project after the deadline.

6.1 Accessibility Tools used

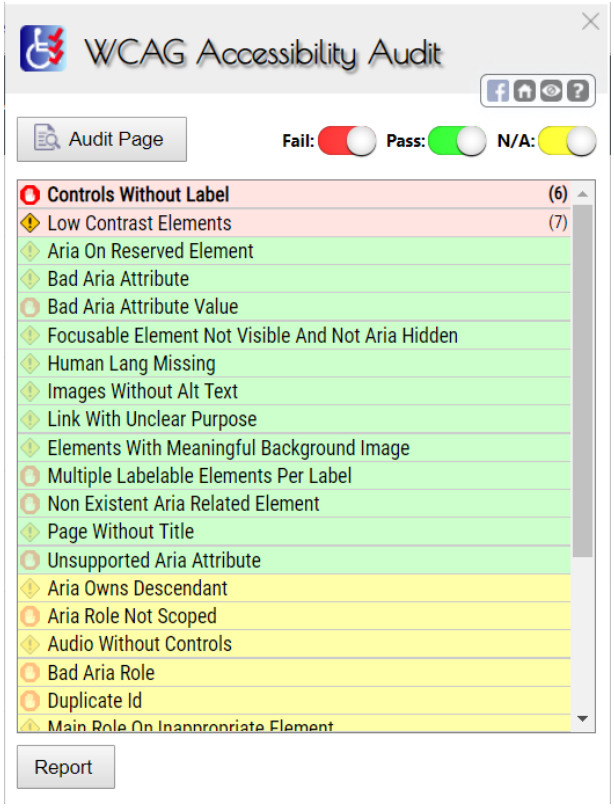
After some research I have found two Google Chrome Plugins available that can help with Accessibility testing. These two were called 'Axe' & 'WCAG Accessibility Audit Developer UI'. These tools help developers see changes that could be made to their application to improve its accessibility.

6.2 Accessibility Testing Results

Below are the results I have gathered from my Accessibility testing. They have been broken up by the two different tools used.

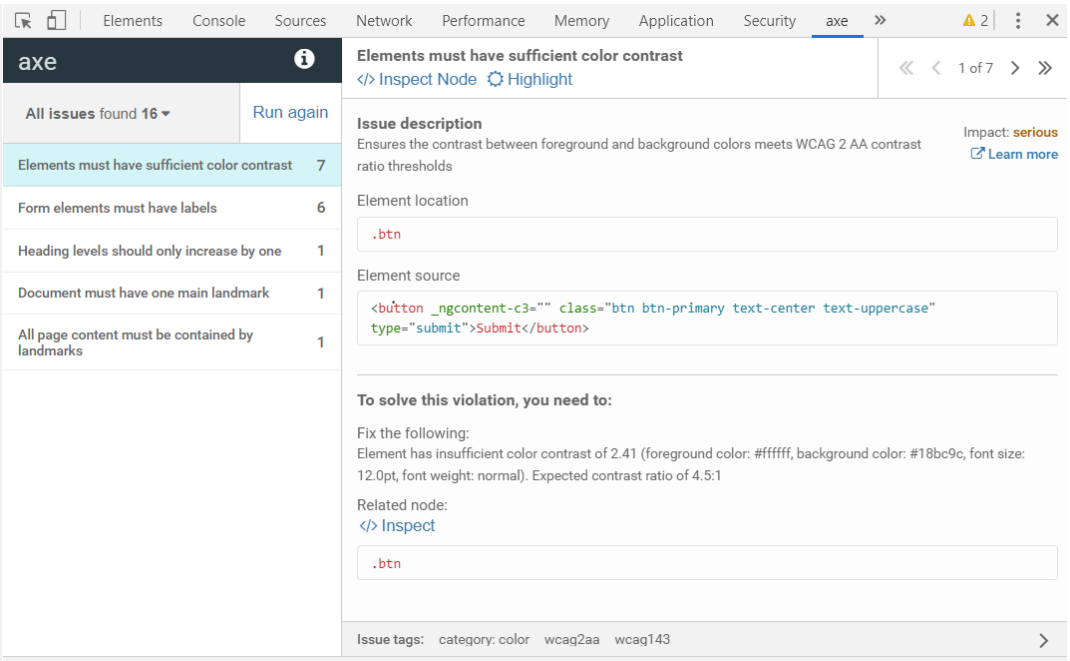
6.2.1 WCAG Accessibility Audit Developer UI Results

Using the above tool, I was able to generate this report of what I passed & failed.



6.2.2Axe Results

Once Again Colour contrast is another factor within my web application.



6.3Conclusion

Based on these results, one common error I am getting is about the colour contrast. Going forward with this project, I will need to consider changing the colour scheme and choose higher contrasting colours. Something I was able to change the 'alt' tags for HTML images. Previously these were not set, which would cause problems for people using screen readers.

7 System Testing

The final type of testing for my application is System Testing. This involves testing the fully integrated application works as it is expected. I will need to test every input to check for the desired output. The results of my system tests are below

7.1 Application Section

TASK	EXPECTED RESULT	PASS / FAIL
Open the application	User should see Application section loaded	PASS
Check header & footer	Both should be visible	PASS
User enters name	No error message displayed	PASS
User does not enter a name	Error message displayed	PASS
User enter DOB in correct format	No error message should be displayed	PASS
User enters the DOB '12/12/2000'	Error message displayed	PASS
User enters correct email address	No error message should be displayed	PASS
User Enters incorrect email	Error message displayed	PASS
User uploads picture containing multiple faces	Error message displayed	PASS
User uploads image containing no faces	Error message displayed	PASS
User uploads image containing one face	User told image has been accepted	PASS
User enters PPS number with 4 character	Error Message displayed	PASS
User enters PPS number with 8 characters	No error message displayed	PASS
User enters 'hello' as input for Phone number	Error message displayed	PASS
User enters all integers for phone number	No error message displayed	PASS
User Presses submit button	Application is finished, Form is reset	PASS
After user presses submit	Email confirmation is sent	PASS

7.2 Administrator Section

TASK	EXPECTED RESULT	PASS / FAIL
Admin Opens the admin section	Admin section should load	PASS
Admin checks previous applications	Admin should be able to see completed passport Applications	PASS

7.3Chatbot Section

TASK	EXPECTED RESULT	PASS / FAIL
User Opens chatbot section	Chatbot section should load	PASS
User enters 'hello'	Chatbot responds appropriately	PASS
User enters 'Where can I find my PPS number	Chatbot responds appropriately	PASS

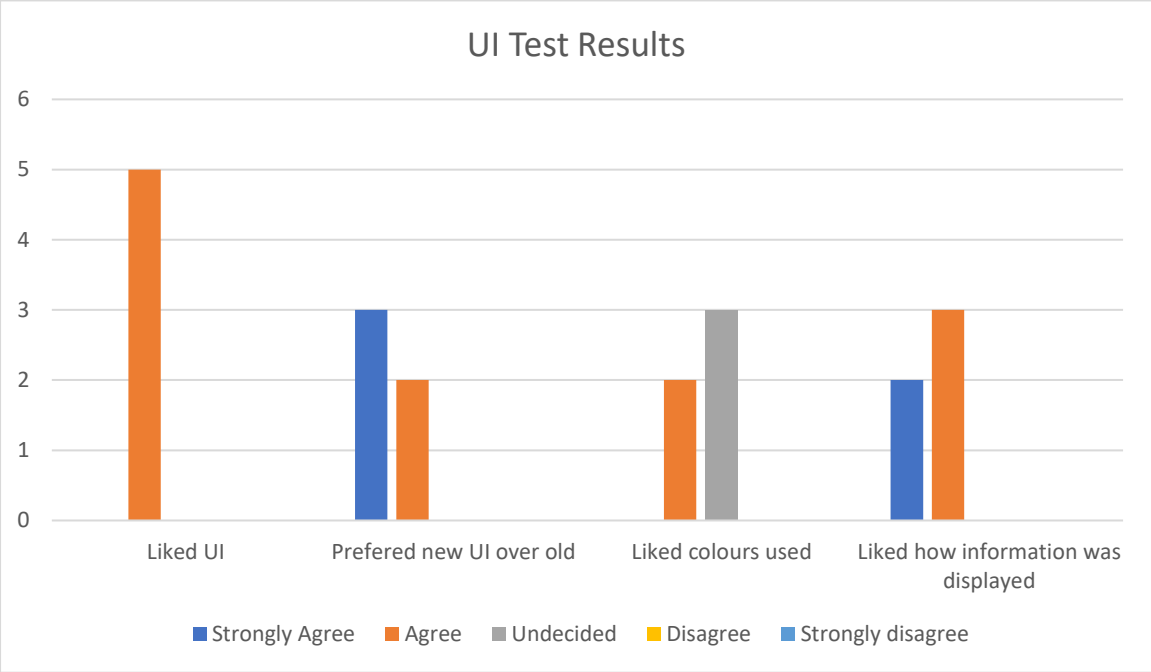
7.4About Section

TASK	EXPECTED RESULT	PASS / FAIL
User Opens about section	About section should load	PASS
User clicks link	Redirect appropriately	PASS

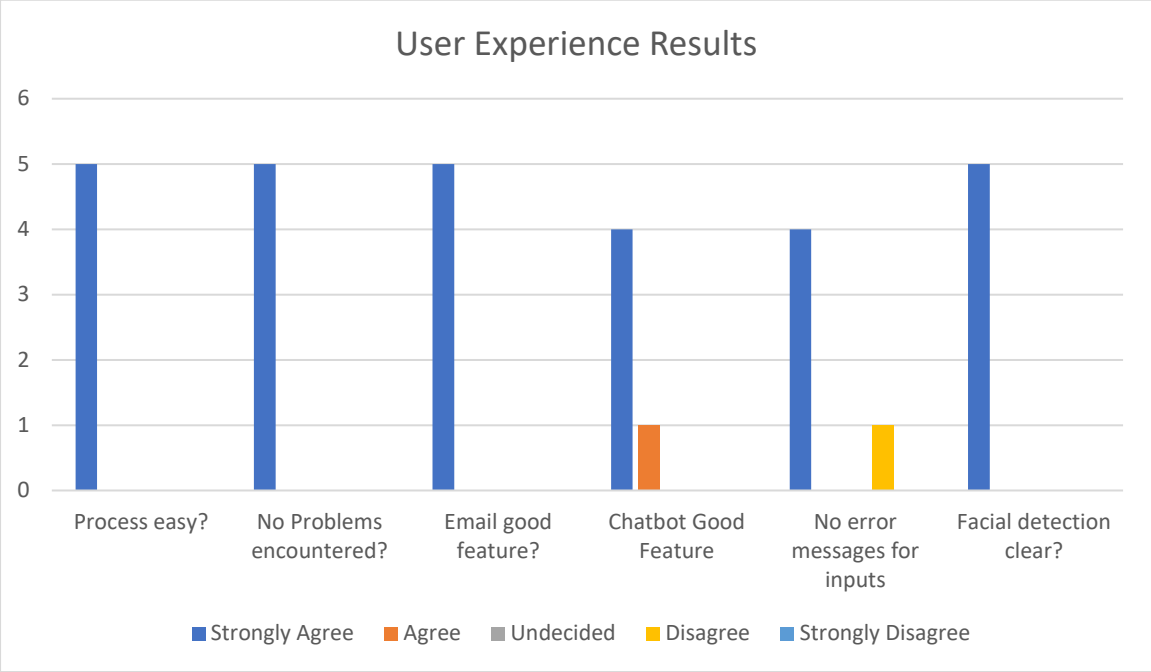
8 User Testing Results

Below Is the results of my User Tests.

8.1 UI Tests



8.2 UX Tests


User Experience Results
6
5
4
3
2
1
0
Process easy?
No Problems encountered?
Email good feature?
Chatbot Good Feature
No error messages for inputs
Facial detection clear?
Strongly Agree
Agree
Undecided
Disagree
Strongly Disagree