

# **FIT3164 - Data Science Software Project**

## **Software Test / QA Report**

**Workshop: Thursday, 2pm - 4pm**

**Group: FIT3163\_CL\_04**

**Members: Elaine Liong (ID: 29942357),**

**Jack Ooi (ID: 29037077),**

**Vionnie Tan (ID: 30092809)**

## **1.2. Table of Contents**

<b>Front Matter</b>	<b>1</b>
<b>Cover Sheet</b>	<b>1</b>
<b>Table of Contents</b>	<b>2</b>
<b>Introduction</b>	<b>3</b>
<b>Test Approach</b>	<b>3</b>
<b>Black Box Testing</b>	<b>4</b>
<b>Test 1</b>	<b>4</b>
<b>Test 2</b>	<b>6</b>
<b>Test 3</b>	<b>8</b>
<b>Test 4</b>	<b>10</b>
<b>Test 5</b>	<b>13</b>
<b>White Box Testing</b>	<b>15</b>
<b>Test 1</b>	<b>15</b>
<b>Test 2</b>	<b>16</b>
<b>Test 3</b>	<b>18</b>
<b>Test 4</b>	<b>22</b>
<b>Test 5</b>	<b>25</b>
<b>Security</b>	<b>27</b>
<b>Scalability</b>	<b>27</b>
<b>Usability Test</b>	<b>28</b>
<b>Software Limitations</b>	<b>29</b>
<b>Recommendation for Improvement</b>	<b>29</b>
<b>Limitations of Testing Process</b>	<b>30</b>
<b>Conclusion</b>	<b>31</b>

## **Introduction**

Extensive software testing is needed to ensure that deliverables are up to the required standard and quality. Our project's main objective is to build a predictive model to detect risks of gastrointestinal cancer through applications of transfer learning on Convolutional Neural Networks. The final product is distributed as a website, where our end-users are able to upload their tumour image and predict their risks of contracting gastrointestinal cancer. The penultimate stage before handing out our software to related health institutions is to substantially test our software, particularly in terms of Robustness, Security, Accessibility, Scalability, Documentation and Maintainability.

The purpose of this report is to show that the software that our team has delivered satisfies the software requirements as initially proposed. In this report, we will first discuss the test approach that we will be executing. For each of the tests, our team will provide descriptions on what we are testing, how we are testing including a snippet of code being executed, and the expected outcomes. In cases where expected outcomes aren't met, alternative methods such as Integration Testing can be done or be left as a software bug. Also included in this report is a Usability Test handed to respondents, limitations of our software, recommendations for future testing approaches and limitations towards the testing procedures.

## **Test Approach**

The test approach that our team would implement on our software is mainly through manual testing. As our developed software is still in its infancy, our team has not developed functionalities that would benefit immensely from automated testing on platforms such as PyTest. Primary Testing can be divided into 2 parts - Black Box Testing and White Box Testing. Black Box testing is done through accessing our deployed website at <https://cancerprediction-4.herokuapp.com/> whereas White Box testing is done through creation of a testing python script on our Website directory. White box testing primarily focuses on assertions and internal methodologies of our website component. In addition to this, our team would also be utilizing usability testing, where volunteers would be given our website for them to essentially "play-around".

# Black Box Testing

## Test 1

### What is being tested

Test website sign up functionality

### How it is being tested

Sign up into the website with an email address, first name and password

### Inputs to the part of code being tested



Email Address

First Name

Password

Password (Confirm)

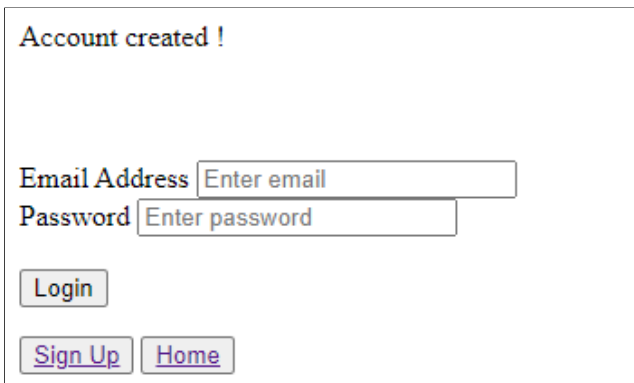
### Expected Outputs

The user should be able to sign up and show a confirmation message that says the account created.

### Actual Outputs Observed

Users signed up successfully and a confirmation message was shown.

### Test Result Screen Shot



Account created !

Email Address

Password

### Integration Testing

This integration testing is testing the integration between login and signup. Users should be able to log in with the signed up users' credentials.

Logged in successfully!

Email Address

Password

The users are able to log in successfully. Thus, this integration testing is successful.

## Test 2

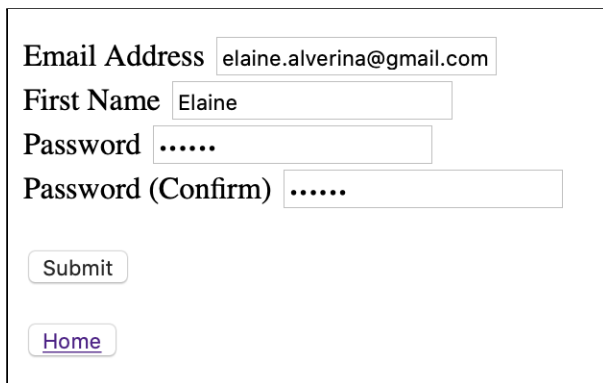
### What is being tested

Test website sign up with password less than 8 characters

### How it is being tested

Sign up on the website with password length 6, other sections of the form being properly completed.

### Inputs to the part of code being tested



A screenshot of a web form for user registration. The form contains four input fields: 'Email Address' with the value 'elaine.alverina@gmail.com', 'First Name' with the value 'Elaine', 'Password' with six dots, and 'Password (Confirm)' with six dots. Below the fields are two buttons: 'Submit' and 'Home'.

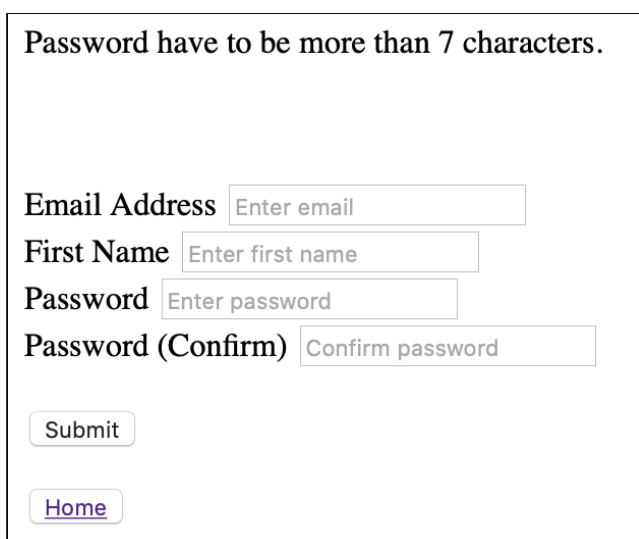
### Expected Outputs

The user will not be able to sign up and the website should show a text that tells the user what needs to be filled to be able to proceed.

### Actual Outputs Observed

A message saying that the password has to be more than 7 characters (at least 8) is shown, the user is not signed up to the website.

### Test Result Screen Shot



A screenshot of the same web form as above, but with an error message at the top: 'Password have to be more than 7 characters.' The input fields now contain placeholder text: 'Email Address' has 'Enter email', 'First Name' has 'Enter first name', 'Password' has 'Enter password', and 'Password (Confirm)' has 'Confirm password'. The 'Submit' and 'Home' buttons are still present at the bottom.

## Integration Testing

This integration testing is testing the integration between login and signup. As the users sign up with violated rules, the users' credentials are not saved in the database. Therefore login of users is not possible.

Email does not exist.

Email Address

Password

## Test 3

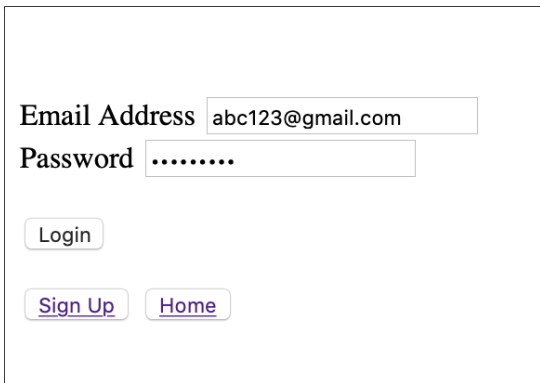
### What is being tested

Test website login functionality

### How it is being tested

Log on to the website and fill the email with a registered email address that has been used to sign up on the website.

### Inputs to the part of code being tested



Email Address

Password

### Expected Outputs

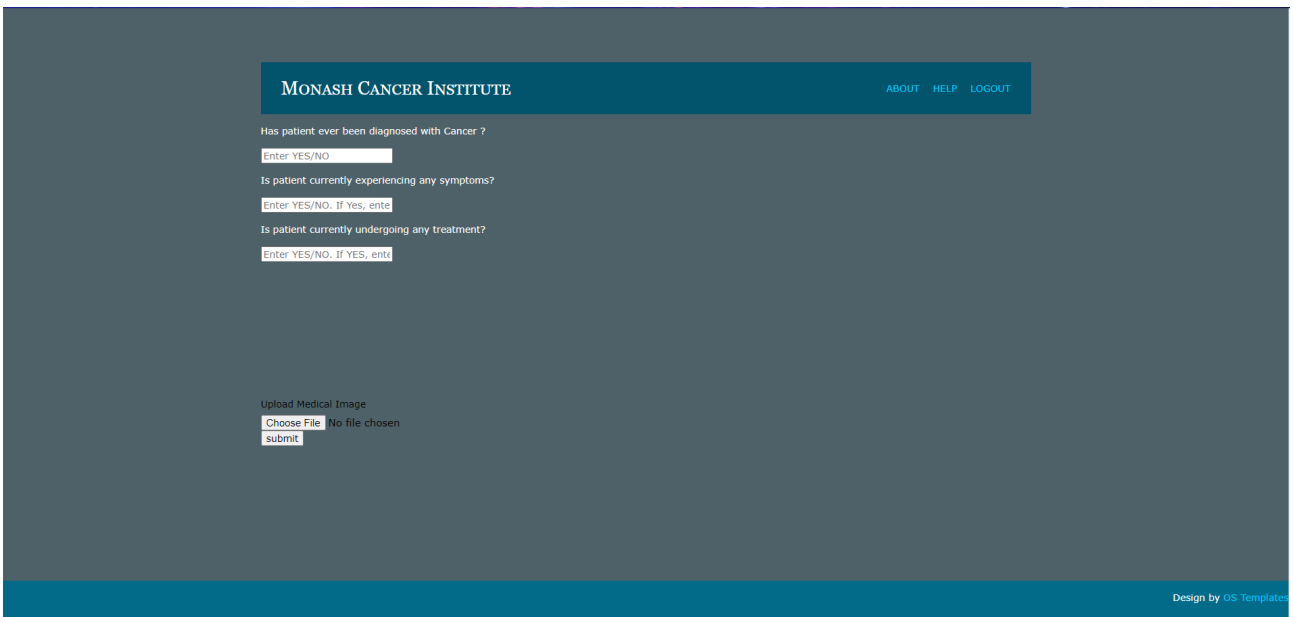
The user should be able log into the website and redirect the users to its homepage

### Actual Outputs Observed

Users are logged in and redirected to the homepage of the users.

### Test Result Screen Shot

(Logout button shown on top right means logged in successful)



MONASH CANCER INSTITUTE [ABOUT](#) [HELP](#) [LOGOUT](#)

Has patient ever been diagnosed with Cancer ?

Is patient currently experiencing any symptoms?

Is patient currently undergoing any treatment?

Upload Medical Image

Design by OS Templates



## Integration Testing

This integration testing is testing the integration between login and signup. If the users try to login with a not registered email address. It should not be able to log in as the users' details are not saved in the database.



The screenshot shows a web form with the following elements:

- An error message at the top: "Email does not exist."
- An "Email Address" label followed by a text input field containing "elaine.alverina@gmail.com".
- A "Password" label followed by a password input field containing seven dots ".....".
- A "Login" button below the password field.
- Two links, "Sign Up" and "Home", at the bottom of the form.

The users are not able to log in. Thus, this integration testing is successful.

## Test 4

### What is being tested

Test signup function with an incomplete form

### How it is being tested

Try signing up on the website:

1. Without providing email address
2. Without providing first name
3. Without providing password

### Inputs to the part of code being tested

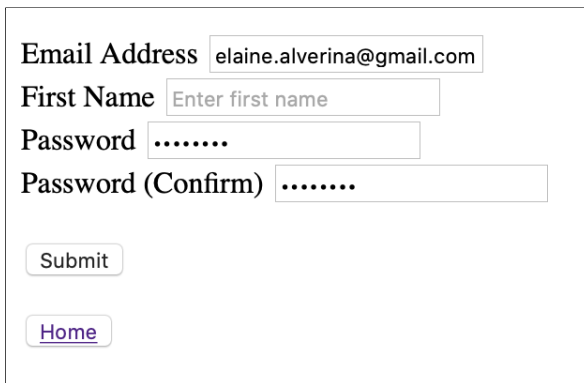
1. Without providing email address



A screenshot of a web form for user registration. The form contains four input fields: 'Email Address' (empty), 'First Name' (containing 'Abc'), 'Password' (containing seven dots), and 'Password (Confirm)' (containing seven dots). Below the fields are two buttons: 'Submit' and 'Home'.

Email Address	Enter email
First Name	Abc
Password	.....
Password (Confirm)	.....
<input type="button" value="Submit"/>	
<a href="#">Home</a>	

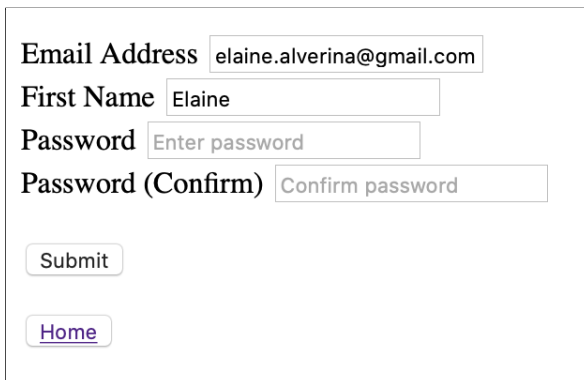
2. Without providing first name



A screenshot of the same web form. The 'Email Address' field now contains 'elaine.alverina@gmail.com', and the 'First Name' field is empty with the placeholder text 'Enter first name'. The other fields and buttons remain the same.

Email Address	elaine.alverina@gmail.com
First Name	Enter first name
Password	.....
Password (Confirm)	.....
<input type="button" value="Submit"/>	
<a href="#">Home</a>	

3. Without providing password



A screenshot of the same web form. The 'Email Address' field contains 'elaine.alverina@gmail.com' and the 'First Name' field contains 'Elaine'. Both the 'Password' and 'Password (Confirm)' fields are empty with their respective placeholder texts 'Enter password' and 'Confirm password'. The buttons remain the same.

Email Address	elaine.alverina@gmail.com
First Name	Elaine
Password	Enter password
Password (Confirm)	Confirm password
<input type="button" value="Submit"/>	
<a href="#">Home</a>	

## Expected Outputs

The user should not be able to sign up without a complete form. The website should show a text that tells the user what needs to be filled to be able to proceed.

## Actual Outputs Observed

When an email is not provided, the website shows “Email must be greater than 3 characters.” When the first name is not provided, the website shows “First Name must be greater than 1 character.” Lastly, when the password is not provided, the website shows “Password have to be more than 7 characters.”

## Test Result Screen Shot

When the email is not provided by the user:

Email must be greater than 3 characters.

Email Address

First Name

Password

Password (Confirm)

[Home](#)

When the name is not provided by the user:

First Name must be greater than 1 character.

Email Address

First Name

Password

Password (Confirm)

[Home](#)

When the password is not provided by the user:

Password have to be more than 7 characters.

Email Address

First Name

Password

Password (Confirm)

[Home](#)

### Integration Testing

This integration testing is testing the integration between login and signup. If the users try to login with a not registered email address. It should not be able to log in as the users' details are not saved in the database.

Email does not exist.

Email Address

Password

[Sign Up](#) [Home](#)

The users are not able to log in. Thus, this integration testing is successful.

## Test 5

### What is being tested

Test predict feature with a standard medical image.

### How it is being tested

This test case is carried out in the local environment. Upload a standard medical image to the website and predict.

### Part of code being tested

```
try:
    #run the predictive model with the submitted image
    img_bytes = file.read()
    prediction_name, percentage = predict(img_bytes)

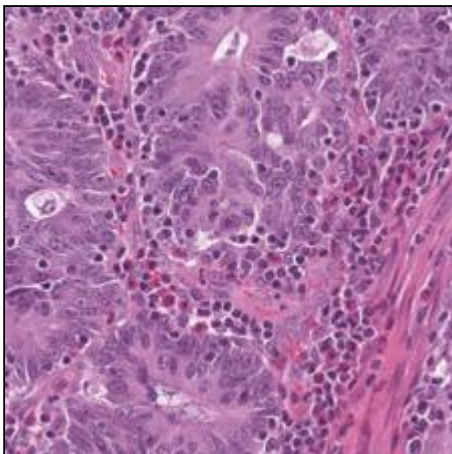
except:
    return render_template("error_file.html")

#add users responses to the database
if current_user.is_authenticated:
    update_user = User.query.filter_by(email=
current_user.email).first()
    update_user.result = percentage
    db.session.commit()

    return render_template("result.html", name= prediction_name,
prediction = percentage)
    return render_template("index.html", user = current_user)
```

### Inputs to the part of code being tested

A standard medical image



### Expected Outputs

A prediction result should produce correctly and show properly after submission

## Actual Outputs Observed

A prediction result has been produced and show properly

## Test Result Screen Shot

This is your prediction result.

**Prediction: MSIMUT 83.82528686523438%**

[Predict Again](#)

## Integration Testing

This integration testing is testing the integration between user authentication system and result prediction. After the users are logged in, the result is saved in the database which can be seen in the screenshot below.

```
Name: Jack, Email: wennchowchu@gmail.com, Password: sha256$pzLEYmDs79K4Jne7$5ee1164b7d61c6f0ec56998e2b01bf7f92b1afbe442fd38cabe8a646ba1f6028, Result: 83.82528686523438}
Name: hello, Email: testing@gmail.com, Password: sha256$Ly8Wynlqn4ZLUQh3$D567f83d123d108f8d8cf478a5436e810b56a6c4e7d72483e3a3af18c3aab46, Result: 90.1951675415039}
Name: 123, Email: jackkang1999@gmail.com, Password: sha256$drUmDTV0TX1VM6uOSd863e6b0d77bbc8fd0def3e9d05901f6e50231342a7f581bfe29cae1c038df7f, Result: 91.75254821777344}
Name: Elaine, Email: elaine.alverina@gmail.com, Password: sha256$XRTToEgqGkxWs5i3v$D5c6c57177f254422c901f3098c4613a7bb198536faad9554a2a2bc093d891a9, Result: 99.53870391845703}
```

The integration testing shows the integration between user authentication system and result prediction is successful as the results are saved in the database.

# White Box Testing

## Test 1

### What is being tested

Test the database is created properly and all the fields are defined correctly. This test case is carried out in the local environment.

### How it is being tested

By using assert in a test function located at test\_model.py. This test case is carried out in the local environment.

### Part of code being tested

```
from main import User

def test_new_user():
    """
    GIVEN a User model
    WHEN a new User is created
    THEN check the email, password, firstname, Symptoms, Cancer, Treatment
    and results fields are defined correctly
    """
    user = User(email = 'jack123@gmail.com', password = '12345678ba',
first_name='Jack',vCancer='YES',vTreatment='YES',vSymptoms='YES',result='50%'
')
    assert user.email == 'jack123@gmail.com'
    assert user.password == '12345678ba'
    assert user.first_name == 'Jack'
    assert user.vSymptoms == 'YES'
    assert user.vCancer == 'YES'
    assert user.vTreatment == 'YES'
    assert user.result == '50%'
    print("Test passed")
```

### Expected Outputs

No errors and print test passed

### Actual Outputs Observed

No errors and Test passed is printed

### Test Result Screen Shot

```
2021.10.1336267007\pythonFiles\lib\python\debugpy\lau
Test passed
PS C:\University\FYP\Project\FIT3164\Website>
```

## Test 2

### What is being tested

Test that user's passwords are protected by a hash function

### How it is being tested

This test case is carried out in a local environment. To test this test case, we created a view.html file to be able to view the content of the database. The view.html is not deployed to prevent potential data breach.

### Inputs to the part of code being tested

```
{% block title %} View All {% endblock %}
{% block content %}
    {% for item in values %}
        <p>Name: {{item.first_name}}, Email: {{item.email}}, Password:
{{item.password}}</p>
    {% endfor %}
{% endblock %}

@app.route("/view/")
def view():
    return render_template("view.html", values = User.query.all())
```

### Expected Outputs

Shows that password is being protected by a hash function

### Actual Outputs Observed

Password is actually protected by SHA 256

### Test Result Screenshot

View All
Name: Jack, Email: jackkang1994@outlook.com, Password: sha256\$I9FwK4nCSPyznzbdr\$951713e3028668160e6ae5c5228e4b8d63cf6bac07c0aa6d6f94005364a6eb3d
Name: 123456, Email: jooi0007@student.monash.edu, Password: sha256\$GzF7125NOeJCoypY\$ab52c72661c61c14e6678bdecc23c7e9e30e655ebd2d2eb8f9c505fcac8ff8ca
Name: Jack, Email: wennchowchu@gmail.com, Password: sha256\$PzLEyMDs79K4Jne7\$5ee1164b7d61c6f0ec56998e2b01bf7f92b1afbe442fd38cabe8a646ba1f6028
Name: hello, Email: testing@gmail.com, Password: sha256\$Ly8Wynlqn4ZLUQh3\$D567f83d123d108f68d8cf478a5436e810b56a6c4e7d72483e3a3af18c3aab46
Name: 123, Email: jackkang1999@gmail.com, Password: sha256\$drUmDTV0TX1VM6uOSd863e6b0d77bbc8fd0def3e9d05901f6e50231342a7f581bfe29cae1c038df7f
Name: Elaine, Email: elaine.alverina@gmail.com, Password: sha256\$XRT0EgqGkxWs5i3v\$D5c6c57177f254422c901f3098c4613a7bb198536faad9554a2a2bc093d891a9
Name: Iak, Email: 123@gmail.com, Password: sha256\$10BvRYMaBaId2ckh\$e02c86d3de3f5b619602a17b519a1b09a884a9f086e66b0e5e180c4501e37742

### Integration Testing

After knowing that the password is protected by a secured hash function. This integration testing is testing whether users' is still able to log in despite their password not being saved in plain text anymore.

Check the functionality of this line:



```
if check_password_hash(user.password, password):
```

This integration testing is successful as the users are still able to log into their account after their password is encrypted.

Logged in successfully!

Email Address

Password

## Test 3

### What is being tested

This test case is to test the calling of different API endpoints that are functioning correctly.

### How it is being tested

This test case is carried out in the local environment. A test case is considered as successful if the http status code from the terminal is shown 200.

### Part of code being tested

```
@app.route("/about/")
def about():
    """
    Route of about page, display the about page to the user
    @return: render the about page HTML
    """
    return render_template("about.html")

@app.route("/help/")
def help():
    """
    Route of help page, display the help page to the user
    @return: render the help page HTML
    """
    return render_template("help.html")

@app.route("/signup/", methods = ['GET', 'POST'])
def signup():
    """
    Route of signup, display the signup page to the user and listen to GET
    and POST
    Added data submitted by users into database
    @return: render the signup HTML page
    """
    if request.method == 'POST':
        email = request.form.get('email')
        firstName = request.form.get('firstName')
        password1 = request.form.get('password1')
        password2 = request.form.get('password2')

        user = User.query.filter_by(email = email).first()
        if user:
            flash('Email already exists.', category= 'error')
        elif len(email) < 4:
            flash('Email must be greater than 3 characters.',
category='error')
        elif len(firstName) < 2:
```

```

        flash('First Name must be greater than 1 character.',
category='error')
        elif password1 != password2:
            flash('Password not matched', category='error')
        elif len(password1) < 7:
            flash('Password have to be more than 7 characters.',
category='error')
        else:
            new_user = User(email=email, first_name=firstName,
password=generate_password_hash(password1, method='sha256'), vCancer= "",
vSymptoms="",vTreatment="",result = "")
            db.session.add(new_user)
            db.session.commit()
            flash("Account created !", category='success')
            return redirect(url_for("login"))

    return render_template("signup.html", user = current_user

@app.route("/login/", methods = ['GET', 'POST'])
def login():
    """
    Route of login, display the login page to the user and listen to GET and
POST
    check user authentication when login
    @return: render the login HTML page
    """
    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get('password')

        user = User.query.filter_by(email = email).first()
        if user:
            if check_password_hash(user.password, password):
                flash('Logged in successfully!', category = 'success')
                login_user(user, remember = True)
                return redirect(url_for("home"))
            else:
                flash('Incorrect password.', category= 'error')
        else:
            flash('Email does not exist.', category = 'error')

    return render_template("login.html", user = current_user)

```

### Inputs to the part of code being tested

https://127.0.0.1:5000/  
https://127.0.0.1:5000/signup  
https://127.0.0.1:5000/about  
https://127.0.0.1:5000/help  
https://127.0.0.1:5000/login

### Expected Outputs

To get HTTP status code 200 from route signup, about, help, login and home.

### Actual Outputs Observed

All HTTP status code from route signup, about, help, login and home are shown 200. All test cases are successful.

### Test Result Screen Shot

```
127.0.0.1 - - [14/Oct/2021 13:49:22] "GET / HTTP/1.1" 200 -  
127.0.0.1 - - [14/Oct/2021 13:49:22] "GET /static/styles/index.css HTTP/1.1" 304 -  
127.0.0.1 - - [14/Oct/2021 13:49:22] "GET /static/styles/%7B%7B%20url_for('fontawesome-free',filename='css/all.css')%20%7D HTTP/1.1" 404 -  
127.0.0.1 - - [14/Oct/2021 13:49:22] "GET /static/styles/fontawesome.css HTTP/1.1" 200 -  
127.0.0.1 - - [14/Oct/2021 13:52:13] "GET /about HTTP/1.1" 308 -  
127.0.0.1 - - [14/Oct/2021 13:52:13] "GET /about/ HTTP/1.1" 200 -  
127.0.0.1 - - [14/Oct/2021 13:52:13] "GET /static/styles/about.css HTTP/1.1" 200 -  
127.0.0.1 - - [14/Oct/2021 13:52:27] "GET /help HTTP/1.1" 308 -  
127.0.0.1 - - [14/Oct/2021 13:52:27] "GET /help/ HTTP/1.1" 200 -  
127.0.0.1 - - [14/Oct/2021 13:52:27] "GET /static/styles/about.css HTTP/1.1" 304 -  
127.0.0.1 - - [14/Oct/2021 13:52:36] "GET /signup HTTP/1.1" 308 -  
127.0.0.1 - - [14/Oct/2021 13:52:36] "GET /signup/ HTTP/1.1" 200 -  
127.0.0.1 - - [14/Oct/2021 13:52:39] "GET /static/styles/framework.css HTTP/1.1" 304 -  
127.0.0.1 - - [14/Oct/2021 13:52:44] "GET /login HTTP/1.1" 308 -  
127.0.0.1 - - [14/Oct/2021 13:52:44] "GET /login/ HTTP/1.1" 200 -
```

### Integration Testing

After successful signup and login, the program is still running smoothly and shows HTTP status code 200.

Account created !

Email Address jackkang1994@outlook.com

Password .....

Login

[Sign Up](#)

[Home](#)

```
127.0.0.1 - - [14/Oct/2021 13:52:36] "GET /signup HTTP/1.1" 308 -  
127.0.0.1 - - [14/Oct/2021 13:52:36] "GET /signup/ HTTP/1.1" 200 -
```

## Test 4

### What is being tested

Test that user's responses are saved in the database

### How it is being tested

This test case is carried out in a local environment. To test this test case, we created a view.html file to be able to view the content of the database. The view.html is not deployed to prevent potential data breach.

Firstly, we login with an account and then fill up the users' responses. After that, the content of the database is checked to see whether the responses are saved or not.

### Part of code being tested

```
if request.method == 'POST':
    email = request.form.get('email')
    firstName = request.form.get('firstName')
    password1 = request.form.get('password1')
    password2 = request.form.get('password2')

    user = User.query.filter_by(email = email).first()
    if user:
        flash('Email already exists.', category= 'error')
    elif len(email) < 4:
        flash('Email must be greater than 3 characters.',
category='error')
    elif len(firstName) < 2:
        flash('First Name must be greater than 1 character.',
category='error')
    elif password1 != password2:
        flash('Password not matched', category='error')
    elif len(password1) < 7:
        flash('Password have to be more than 7 characters.',
category='error')
    else:
        new_user = User(email=email, first_name=firstName,
password=generate_password_hash(password1, method='sha256'), vCancer= "",
vSymptoms="",vTreatment="",result = "")
        db.session.add(new_user)
        db.session.commit()
        flash("Account created !", category='success')
        return redirect(url_for("login"))

return render_template("signup.html", user = current_user)
```

```
@app.route("/login/", methods = ['GET', 'POST'])
```

```

def login():
    """
    Route of login, display the login page to the user and listen to GET and
    POST
    check user authentication when login
    @return: render the login HTML page
    """
    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get('password')

        user = User.query.filter_by(email = email).first()
        if user:
            if check_password_hash(user.password, password):
                flash('Logged in successfully!', category = 'success')
                login_user(user, remember = True)
                return redirect(url_for("home"))
            else:
                flash('Incorrect password.', category= 'error')
        else:
            flash('Email does not exist.', category = 'error')

    return render_template("login.html", user = current_user)

{% block title %} View All {% endblock %}
{% block content %}
    {% for item in values %}
        <p>Name: {{item.first_name}}, Email: {{item.email}}, Password:
        {{item.password}}, Cancer: {{item.vCancer}}, Treatment:{{item.vTreatment}},
        Symptoms: {{item.vSymptoms}}</p>
    {% endfor %}
{% endblock %}

@app.route("/view/")
def view():
    return render_template("view.html", values = User.query.all())

```

### Expected Outputs

Shows that the users' responses are saved in the database by viewing the content of the database.

### Actual Outputs Observed

Users' responses are saved in the database

## Test Result Screenshot

Name: Testing, Email: 123456@outlook.com, Password: sha256\$HHqve70HpikWPIMU\$9eaf89f7313a3c5d22ad34adc0fe96707abd0c5241081f6869fc7d3e4c32eb60, Cancer: Testing, Treatment:Testing, Symptoms: Testing

## Integration Testing

After knowing the users' responses are saved in the database. This integration testing is testing whether other details of users' details such as first name and email will be saved.

After viewing the content of the database, the result is shown below and shows that all the users' details and responses are saved in the database.

Name: Testing, Email: 123456@outlook.com, Password: sha256\$HHqve70HpikWPIMU\$9eaf89f7313a3c5d22ad34adc0fe96707abd0c5241081f6869fc7d3e4c32eb60, Cancer: Testing, Treatment:Testing, Symptoms: Testing



## Test 5

### What is being tested

Test predict feature with a non-medical image

### How it is being tested

Upload a random non-medical image from Google to the website and predict its risk.

### Inputs to the part of code being tested



Upload Medical Image

blue-white-gru...in-820x532.jpg

### Expected Outputs

The software should show an error file and show an error message

### Actual Outputs Observed

The software shows an error file and shows an error message related to the content of the image files.

### Test Result Screen Shot

Please only submit a relevant medical SVS image file !

[Home](#)

### Integration Testing

This integration test is to test after users are logged in, the software will still produce the same error and error message.

Please only submit a relevant medical SVS image file !

[Home](#)

The software is still producing the same result. Thus, this integration testing is successful.

## **Security**

Scanned Medical Image is considered private and confidential and therefore in our system, none of the submitted medical images will be saved, only the prediction result, name and email will be saved inside the database. Besides, when users sign up for an account on our website, their password will be saved securely with the help of SHA 256 hash function. This can prevent any potential password data breach in our system.

## **Scalability**

A scalable web application is able to handle growth in users and load. Since we are using Heroku's free tier account, we have only 512 MB on Heroku's server and we have used around 315 MB of storage to deploy our website. When the storage usage exceeds 512 MB, we have no choice but to redeploy our website with a higher tier account on Heroku to continue our web application.

## **Usability Test**

The ease of access and use of our software is one of our priorities in the development of this project. In the development of the software, our team tries to ensure that we use consistent user interface layout throughout the software, buttons are grouped and well-positioned, colours have good contrast, error messages are meaningful, and no unnecessary items are added to the website.

To run the usability test effectively, we require third party participants, which may include the potential users of our software, to test our software. The participants will be given a complete set of user guides to help them operate the software.

From this test, we aim to be able to:

1. Learn whether participants are able to complete the task given on the user guide
2. Measure the participants' satisfaction with our software interface
3. Receive valuable feedback on how the usability can be improved
4. Identify modifications required to improve the usability

## **Result**

The usability test was implemented on three different respondents. Overall, our respondents were able to understand the functionality of our website, but had several complaints regarding the website. Almost all of the respondents found that the website's layout was not consistent with each page. For example, the landing page, about and help pages have dark background colour, however, the sign up, login, and predict result pages have bright background colour. Furthermore, respondents were also unclear with what MSS or MSIMUT meant, and had to reach out to us for clarification. Respondents felt greatly relieved that an end-user guide was given and the website ran smoothly during all three instances.

## **Evaluation**

From the test result mentioned above, we have received a number of useful feedback from the test participants. The test participants are all able to complete the tasks as shown in the user guide smoothly. However, they are not quite satisfied with the user interface since there is a mix of dark and light background colours in different pages of the website. Additionally, the users do not quite understand what the prediction results meant due to the lack of information provided in our website. To improve the usability of our software, in the future, we would like to modify the background colours of the software so that all pages are consistent. In addition, we would add more details on the result page to further explain what the result meant.

## **Software Limitations**

Based on the results stemmed from the variation of tests done above on our software, several limitations were exposed from our software. In particular, one limitation that has the potential to affect the integrity of our software is White Box Testing 5 - Test predict feature with a non-medical image. Although during integration testing, an additional exception handling is able to catch some of the invalid non-medical images, the application isn't foolproof, as there is a small chance that the software is not able to handle the error and still produce a result to classify the image as MSS/MSIMUT. This means that, there is a small probability that our users would get an invalid prediction result that is completely irrelevant to the uploaded image.

The security present in our website is also in its infancy, as our team has only implemented a single SHA 256 encryption algorithm on user's passwords, but other sensitive information such as their names, email addresses and uploaded images remain unencrypted. This means that the potential risk for a data breach by attackers is significantly increased.

Another limitation posed by our software is that our users aren't able to view their uploaded medical image, and the lack of overall information meant that our users were mostly confused by what our website is doing. For example, during usability testing, we had not told our users what MSS and MSIMUT meant, so users did not understand their prediction results.

## **Recommendation for Improvement**

With the limitations present in our website software, it leaves the door open for further improvements to be made on our software. In terms of technical functionalities of our website, our team would like to be able to differentiate medical and non-medical images, so as to prevent our users from having invalid prediction results. One solution to this is to develop an algorithm that would detect the similarity between the uploaded images and dataset images, and if it reaches a certain threshold, then the website would accept the uploaded images. In terms of security, it is also vital for us to securely handle user's private information. This means that all of their sensitive details should be encrypted with more than one layer of security to prevent the chances of data breach from attackers. The feedback that we received during usability testing can also be integrated to provide a better user experience for our end users. This involves changing the website typeface, layout and colour to be consistent with all of the pages. Our website currently lacks relevant information to be shown to users, it would be a good idea to add descriptions regarding a patient's predictive results, and solutions that can be offered to them.

## **Limitations of Testing Process**

Most of the testing methodologies - Black Box, White Box and Usability testing done on our deployed software were mostly simple and straightforward. As a result, our team did not utilize automated testing from platforms such as PyTest and regression testing. This is mainly due to the fact that our team has had no prior experience in testing a delivered software, and in addition, the functionality served by our service is still in its preliminary stages, where no substantial benefits would be obtained from doing automated testing. However, it remains to note that there are possible undetected bugs that our team has not manually tested in depth. For example, our team had not conducted White Box Testing relating to the developed machine learning model from PyTorch. Possible tests relating to the overall performance of the developed model such as exploratory testing of each fine-tuned layer in the convolutional neural network could be done. The following test was not accomplished as our team required further expertise on PyTorch's overall infrastructure which needed a lot of time and resources that our team was running low on. If the test had been done, a more detailed view of our network could be made such as whether each layer is being optimized which in turn could increase the overall performance of our model.

Another limitation posed by our testing process relates to the input testing on our website infrastructure. Since users are able to submit various content types of images inputs, including non-medical images, we are not able to predict which content types of image are going to be submitted by the users, so only a portion of content types will be tested. This means that not all errors will be caught and some content types will still produce a result. When the software receives a non-medical image as an input, the software is supposed to produce an error but due to limited testing not all incorrect content types errors will be caught, some of them might still produce a result as the software recognised the image as MSS/MSIMUT images. It is not possible to test our software with all types of images that the users are going to upload since there are an unlimited amount of images.

## **Conclusion**

In conclusion, this report has briefly discussed the software testing and quality assurance done on our deployed website that is used to predict a patient's risk of contracting gastrointestinal cancer. The testing approaches done to ensure every piece of code is working as its requirements is through manual testing that varies from Black Box, White Box and Usability Testing. Black Box testing was mostly done on the front-end of the website, whereas White Box was mostly done on the back-end of the website.

Throughout the testing process, our team came to recognize the flaws that had the potential to affect the integrity of our produced software. This meant that our team had the chance to update and make changes to the code that would help increase the robustness of our software. For example, prior to testing our team had not implemented any exception handling for mishandling of uploaded images. This was quickly rectified and integrated onto our website. Conducting usability testing also gave our team the chance to gain useful feedback from our end-users that can be implemented in future developments of our software.

An overall reflection on the limitations of our testing approaches meant that there were several aspects of our code that were not thoroughly tested, such as the developed predictive model. The lack of developed technical functionalities present on our website coupled with the lack of testing expertise from our team meant that these testings could not be implemented on our given software. All in all the performance of our testing approaches were successful assuming that the tested methodologies consisted only of the preliminary goals mentioned in our project proposal. However, on an industrial level, the performance of our testing approaches were at most average. This means that, in order to deliver a better product, substantial improvements to both the software and testing approaches have to be done.