Test Execution Document

For

ONLINE CRIME INVESTIGATION SYTEMS PROJECT

Version 1.1 approved

Prepared by: SHIVAM SHANDIL

SUJOY ROY

RAJAT RAI

RACHIT TRIVEDI

VAIBHAV KAUSHIK

Organization: NIIT University

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The **Test Execution Document** made for the project that is ONLINE CRIME INVESTIGATION SYSTEM. The following document clearly displays the test outputs that has been performed on the product. The developer team is from NIIT University and are currently the students of 3rd year. The application made does not support hierarchy in case of development. This document consists of the test cases in different forms. Testing has been done mainly in two different ways.

· Unit Testing

· Integration Testing

1.2 Testing Process

Testing which are performed are mainly divided into two parts which:-
· Unit Testing

· Integration Testing

In case of Unit Testing:-
Application has been divided into small units and most importantly all the small units are tested separately. Each of the test cases are clustered into white and black box test cases, they are type of deriving test cases for the unit. All the test are then again briefly described with the reasons as well.

In case of Integration Testing:-
Application have been divided into integrated parts and all the parts are addition of units only. Therefore before starting the integration testing the unit testing should be done. Each test case of integration testing is mainly clustered into black box and white box test cases. Black box and white box is a type of deriving test cases and all the tests are again briefly described with reasons as well.

1. Unit Testing

1.1 Introduction

Complete code of the product is divided in to small units which can be tested under blocks. All the test cases can be derived/ divided from black box and white box testing methods. Each unit will be having two section named black box and white box testing. Each section will be having number of test cases and will be described appropriately.

1.2 Unit 1 → Login_click

Section 1 – Black Box Test Cases

Test Case	Function	Type
Test case 1	Login_click	Valid username
Test case 2	Login_click	Invalid username
Test case 3	Login_click	Valid password
Test case 4	Login_click	Invalid password

<u>Test Case 1</u>

Description: Login function directs the user to the home page of the website , once he fills the required correct credentials that match up to the ones stored in the database that was inputted while formation of the profile.

Test Approach: Testing would be done on various popular web browsers (examples: chrome, Firefox) using various input combinations as specified above.

The function has been used for admin, staff, and user

Test Inputs: suppose the username and password is admin, admin respectively.

The case that we have, according to that we obtain

1) Valid – admin

Similarly we tried various combinations of passwords and usernames, using numerical, alphabets, alphanumerical, special characters, and various combinations of strings. Permutation and combinations of various input methodologies have also been used to check the authenticity of the system.

Test Results:

Result: PASSED

Only when the combination are correct.

No BUGS, no faulty output has been received.

<u>Test Case 2</u>

Description: Login function directs the user to the home page of the website , once he fills the required correct credentials that match up to the ones stored in the database that was inputted while formation of the profile.

Test Approach: Testing would be done on various popular web browsers (examples: chrome, Firefox) using various input combinations as specified above.

The function has been used for admin, staff, and user

Test Inputs: suppose the username and password is admin, admin respectively.

The case that we have, according to that we obtain

1) invalid – @dmin

Similarly we tried various combinations of passwords and usernames, using numerical, alphabets, alphanumerical, special characters, and various combinations of strings. Permutation and combinations of various input methodologies have also been used to check the authenticity of the system.

Test Results:

Result: PASSED

Only when the combination are incorrect.

No BUGS, no faulty output has been received.

<u>Test Case 3</u>

Description: Login function directs the user to the home page of the website , once he fills the required correct credentials that match up to the ones stored in the database that was inputted while formation of the profile.

Test Approach: Testing would be done on various popular web browsers (examples: chrome, Firefox) using various input combinations as specified above.

The function has been used for admin, staff, and user

Test Inputs: suppose the username and password is admin, admin respectively.

The case that we have, according to that we obtain

1) valid - admin

Similarly we tried various combinations of passwords and usernames, using numerical, alphabets, alphanumerical, special characters, and various combinations of strings. Permutation and combinations of various input methodologies have also been used to check the authenticity of the system.

Test Results:

Result: PASSED

Only when the combination are correct.

No BUGS, no faulty output has been received.

<u>Test Case 4</u>

Description: Login function directs the user to the home page of the website , once he fills the required correct credentials that match up to the ones stored in the database that was inputted while formation of the profile.

Test Approach: Testing would be done on various popular web browsers (examples: chrome, Firefox) using various input combinations as specified above.

The function has been used for admin, staff, and user

Test Inputs: suppose the username and password is admin, admin respectively.

The case that we have, according to that we obtain

1) Invalid - @dmin

Similarly we tried various combinations of passwords and usernames, using numerical, alphabets, alphanumerical, special characters, and various combinations of strings. Permutation and combinations of various input methodologies have also been used to check the authenticity of the system.

Test Results:

Result: PASSED

Only when the combination are incorrect.

No BUGS, no faulty output has been received.

Section 2 – White Box Test Cases

Note:

White Box test cases included (for a given unit of code) path coverage, branch coverage, statement coverage that must have been achieved by the derived set of test cases. The above test cases which were covered (in the section Black Box Test Cases) . There is no need of extra test cases to complete or achieve the above coverage's that is path, statement and branch. So .we can conclude that the white box test cases have been executed as above.

1.3 Unit → user sign up

Section 1 – Black Box Test Cases

Test Case	Function	Type
Test Case 1	User sign up	- Valid Name
Test Case 2	User sign up	- Invalid Name
Test Case 3	User sign up	- Valid Age
Test Case 4	User sign up	- Invalid Age
Test Case 5	User sign up	- Valid Address
Test Case 6	User sign up	- Invalid Address
Test Case 7	User sign up	- Valid gender
Test Case 8	User sign up	- Invalid gender
Test Case 9	User sign up	- Valid Password

Test Case 10	User sign up	- Invalid Password

Test Case 1

Description: this function type checks the value inputted by the user in the form of alphabetic entry only

Test Approach: testing will be done by inputting valid entries of alphabets only that will be accepted. User may input various alphabets in any form of permutation and combination that he/she may like in order to create a desired user name.

Test Inputs: If the name contains only alphabets then it's considered a valid entry

Test Results:

Result is successful, as the entry contains only alphabets.

Test Case 2

Description: this function type checks the value inputted by the user in the form of alphabetic entry.

Test Approach: testing will be done by inputting valid entries of alphabets only that will be accepted. User may input various alphabets in any form of permutation and combination that he/she may like in order to create a desired user name. If user inputs any other form of input for example a numerical or a special character, the entry becomes invalid or is considered invalid

Test Inputs: If the name contains numerical, alpha numerical characters, special characters and etc., then it's considered invalid input.

Test Results:

Result is successful, as the entry was detected as invalid.

Test Case 3

Description: this function type checks the value inputted by the user in the form of numerical entry only

Test Approach: testing will be done by inputting valid entries of numerical values only that will be accepted. User may input various numericals that showcase their valid ages. Since age is in numeric hence he/she can only input values in numeric and not any other form of character.

Test Inputs: If the age contains only numeric then it's considered a valid entry

Test Results:

Result is successful, as the entry contains only numeric.

Test Case 4

Description: this function type checks the value inputted by the user in the form of numerical entry only.

Test Approach: testing will be done by inputting invalid entries of various characters apart from numerical values only that will be accepted. Since age is in numeric hence he/she can only input values in numeric and not any other form of character , if any other character is inputted then it is considered invalid.

Test Inputs: If the age contains no numeric then it's considered a valid entry

Test Results:

Result is successful, as the entry contains no numeric and is classified as invalid.

Test Case 5

Description: this function type checks the value inputted by the user for the address field.

Test Approach: testing will be done by inputting valid entries which are genuine map locations else the input location will be truncated

Test Inputs: If the address is a valid map location then it will be considered as valid for the system to store in its database. For example , if we put NIIT UNIVERSITY , NEEMRANA , ALWAR , RAJASTHAN , this is a valid address. If we put NIIT UNIVERSITY, NEEMRANA, MIAMI, MARS. This location does not exist. Hence its considered invalid.

Test Results:

Result is successful, as the entry contains input which has valid map location.

Test Case 6

Description: this function type checks the value inputted by the user for the address field.

Test Approach: testing will be done by inputting invalid entries which aren't genuine map locations else the input location will be valid.

Test Inputs: If the address is a valid map location then it will be considered as valid for the system to store in its database. For example, if we put NIIT UNIVERSITY, NEEMRANA, ALWAR, RAJASTHAN, this is a valid address. If we put NIIT UNIVERSITY, NEEMRANA, MIAMI, MARS. This location does not exist. Hence it's considered invalid.

Test Results:

Result is successful, as the entry contains input which has invalid map location.

Test Case 7

Description: this function type checks the value inputted by the user for the gender field.

Test Approach: testing will be done by inputting valid entries which are single character M/F only which will be accepted by the system. No numeric, alpha numeric and string of alphabets are allowed

Test Inputs: If the input from the user is a single character m or f , it will be accepted by the system else any other string of characters is considered invalid. For example:

M –valid

F –valid

Male –invalid

Female –invalid

m@l* -invalid

Test Results:

Result is successful, as the entry contains input which has single character.

Test Case 8

Description: this function type checks the value inputted by the user for the gender field.

Test Approach: testing will be done by inputting invalid entries which are numeric, alpha numeric and string of alphabets that are invalid to the system.

Test Inputs: If the input from the user is a single character m or f , it will be accepted by the system else any other string of characters is considered invalid. For example:

M –valid

F –valid

Male –invalid

Female –invalid

m@l* -invalid

Test Results:

Result is successful, as the entry contains input which has invalid input.

<u>Test Case 9</u>

Description: this function type checks the value inputted by the user in the form of string of characters. The user must not only enter the password in the form of numerics. It should stay a combination of all characters.

Test Approach: testing will be done by inputting valid entries as specified above .

Test Inputs: If the password contains all required characters it is a valid entry and will be accepted for example: Niit@123 this is valid, invalid will be 123123.

Test Results:

Result is successful, as the entry contains specified character set.

Test Case 10

Description: this function type checks the value inputted by the user in the form of string of characters. The user must not only enter the password in the form of numerics. It should stay a combination of all characters if only one set of characters are used it’s invalid.

Test Approach: testing will be done by inputting invalid entries as specified above.

Test Inputs: If the password contains all required characters it is a valid entry and will be accepted for example: Niit@123 this is valid, invalid will be 123123.

Test Results:

Result is successful, as the entry contains invalid input in the form of single characters like in the above example.

Section 2 – White Box Test Cases

Note:

White Box test cases included (for a given unit of code) path coverage, branch coverage, statement coverage that must have been achieved by the derived set of test cases. The above test cases which were covered (in the section Black Box Test Cases) . There is no need of extra test cases to complete or achieve the above coverage’s that is path, statement and branch. So .we can conclude that the white box test cases have been executed as above.

1.4 Unit → btnregister_Click

Section 1 – Black Box Test Cases

Test Case	Function	Type
Test case 1	btnregister_Click	Valid citizenid
Test case 2	btnregister_Click	Invalid citizenid
Test case 3	btnregister_Click	Valid location
Test case 4	btnregister_Click	Invalid location
Test case 5	btnregister_Click	Valid Description

Test Case 1

Description: Citizen id function is the unique id generated for the user logging into the system. It doesn’t exist in the database. It is valid if it contains an auto incremented integer value

Test Approach: Testing will be done by checking the database for a citizen id and by using a valid citizen id a new entry is created in the database then the function would be assumed to work correctly.

Test Inputs: This field was supplied with valid citizen id's that already existed in the database eg 7,9 1 etc.

Test Results:

Result is successful, as the entry contains invalid input in the form valid integers like in the above example.

<u>Test Case 2</u>

Description: citizen id function is the unique id generated for the user logging into the system. It doesn't exist in the database. It is valid if it contains an auto incremented integer value

Test Approach: Testing will be done by checking the database for a citizen id and by using a valid citizen id a new entry is created in the database then the function would be assumed to work correctly.

Test Inputs: This field was supplied with invalid citizen id's that already existed in the database eg 777,z,123hy, 1a etc.

Test Results:

Result is successful, as the entry contains invalid input in the form of characters like in the above example.

Test Case 3

Description: this function type checks the value inputted by the user for the location field.

Test Approach: testing will be done by inputting valid entries which are genuine map locations else the input location will be truncated

Test Inputs: If the location is a valid map location then it will be considered as valid for the system to store in its database. For example, if we put NIIT UNIVERSITY, NEEMRANA, ALWAR, RAJASTHAN, this is a valid address. If we put NIIT UNIVERSITY, NEEMRANA, MIAMI, MARS. This location does not exist. Hence it's considered invalid.

Test Results:

Result is successful, as the entry contains input which has valid map location.

Test Case 4

Description: this function type checks the value inputted by the user for the location field.

Test Approach: testing will be done by inputting invalid entries which aren't genuine map locations else the input location will be valid.

Test Inputs: If the address is a valid map location then it will be considered as valid for the system to store in its database. For example, if we put NIIT UNIVERSITY, NEEMRANA, ALWAR, RAJASTHAN, this is a valid address. If we put NIIT UNIVERSITY, NEEMRANA, MIAMI, MARS. This location does not exist. Hence it's considered invalid.

Test Results:

Result is successful, as the entry contains input which has invalid map location.

Test Case 5

Description: this function type checks that the fir/ crime description is contained under the specified word limit.

Test Approach: word limit for the description being 4000 words max will be checked by having various descriptions under 4000 characters, and will be considered valid if it's less than 4000 and if more than 4000 invalid entry will be marked.

Test Inputs: if it exceeds 4000 characters it will not be accepted

Test Results:

Result is successful, as the entry contains characters less than specified max amount.

Section 2 – White Box Test Cases

Note:

White Box test cases included (for a given unit of code) path coverage, branch coverage, statement coverage that must have been achieved by the derived set of test cases. The above test cases which were covered (in the section Black Box Test Cases) . There is no need of extra test cases to complete or achieve the above coverage's that is path, statement and branch. So .we can conclude that the white box test cases have been executed as above.

1.5 Unit → btnUpdate_Click

Test Case	**Function**	**Type**
Test case 1	btnUpdate_Click	Existing FIR ID
Test case 2	btnUpdate_Click	NON- Existing FIR ID

Test Case 1

Description: update FIR function allows the user to view and make changes to the registered existing FIR.

Test Approach:when the update button is clicked the registered FIR with a particular register ID is updated from the existing, remaining registered FIR list in the system

Test Inputs: Inputs are divided into equivalence blocks. This blocks will be dealing with all the inputs of valid fir IDs

Test Results: Registered FIR updated

<u>Test Case 2</u>

Description: update FIR function allows the user to view and make changes to the registered existing FIR.

Test Approach:when the update button is clicked the registered FIR with a particular register ID is updated from the existing remaining registered FIR list in the system

Test Inputs: Inputs are divided into equivalence blocks. This blocks will be dealing with all the inputs of invalid fir IDs

Test Results: Registered FIR not found

Section 2 – White Box Test Cases

Note:

White Box test cases included (for a given unit of code) path coverage, branch coverage, statement coverage that must have been achieved by the derived set of test cases. The above test cases which were covered (in the section Black Box Test Cases) . There is no need of extra test cases to complete or achieve the above coverage's that is path, statement and branch. So .we can conclude that the white box test cases have been executed as above.

2.6 Unit →btnViewSelected_Click

Test Case	**Function**	**Type**
Test case 1	btnViewSelected_Click	Existing FIR ID
Test case 2	btnViewSelected_Click	NON- Existing FIR ID

<u>Test Case 1</u>

Description: view FIR function allows the user to view the registered existing FIR.

Test Approach:when the view button is clicked the registered FIR with a particular register ID is viewed in the system

Test Inputs: Inputs are divided into equivalence blocks. This blocks will be dealing with all the inputs of valid fir IDs

Test Results: Registered FIR can be viewed

Test Case 2

Description: view FIR function allows the user to view the registered existing FIR.

Test Approach:when the view button is clicked the registered FIR with a particular register ID is viewed in the system

Test Inputs: Inputs are divided into equivalence blocks. This blocks will be dealing with all the inputs of invalid fir IDs

Test Results: Registered FIR not found

Section 2 – White Box Test Cases

Note:

White Box test cases included (for a given unit of code) path coverage, branch coverage, statement coverage that must have been achieved by the derived set of test cases. The above test cases which were covered (in the section Black Box Test Cases) . There is no need of extra test cases to complete or achieve the above coverage's that is path, statement and branch. So .we can conclude that the white box test cases have been executed as above.

2.7 Unit → btnDelete_Click()

Test Case	**Function**	**Type**
Test case 1	btnDelete_Click()	Existing FIR ID
Test case 2	btnDelete_Click()	NON- Existing FIR ID

Test Case 1

Description: Deletes the existing registered FIR by the user.

Test Approach:when the delete button is clicked the registered FIR with a particular register ID is deleted from the existing, remaining registered FIR list in the system.

Test Inputs: Inputs are divided into equivalence blocks. This blocks will be dealing with all the inputs of valid fir IDs

Test Results: Registered FIR deleted

Test Case 2

Description: Deletes the existing registered FIR by the user.

Test Approach:when the delete button is clicked the registered FIR with a particular register ID is deleted from the existing remaining registered FIR list in the system.

Test Inputs: Inputs are divided into equivalence blocks. This blocks will be dealing with all the inputs of invalid fir IDs

Test Results: Registered FIR not found

Section 2 – White Box Test Cases

Note:

White Box test cases included (for a given unit of code) path coverage, branch coverage, statement coverage that must have been achieved by the derived set of test cases. The above test cases which were covered (in the section Black Box Test Cases) . There is no need of extra test cases to complete or achieve the above coverage's that is path, statement and branch. So .we can conclude that the white box test cases have been executed as above.

2.8 Unit →btnView1_Click() (view FIR status)

Test Case	**Function**	**Type**
Test case 1	btnView1_Click()	Existing FIR ID
Test case 2	btnView1_Click()	NON- Existing FIR ID

Test Case 1

Description: View the registered FIR status by the user.

Test Approach:when the view button is clicked the status of the registered is seen.

Test Inputs: Inputs are divided into equivalence blocks. This blocks will be dealing with all the inputs of valid fir IDs

Test Results: Registered FIR status seen

Test Case 2

Description: View the registered FIR status by the user.

Test Approach when the view button is clicked the status of the registered is seen.

Test Inputs: Inputs are divided into equivalence blocks. This blocks will be dealing with all the inputs of invalid fir IDs

Test Results: Registered FIR status is not seen.

Section 2 – White Box Test Cases

Note:

White Box test cases included (for a given unit of code) path coverage, branch coverage, statement coverage that must have been achieved by the derived set of test cases. The above test cases which were covered (in the section Black Box Test Cases) . There is no need of extra test cases to complete or achieve the above coverage's that is path, statement and branch. So .we can conclude that the white box test cases have been executed as above.

2.9 Unit → viewCitizen/deleteCitizen

Section 1 – Black Box Test Cases

Test Case for Citizen Model

Description: viewCitizen()/deleteCitizen() function is triggered radio button for Citizen clicked and submit button is pressed

Test Approach: When Citizen Radio Button is clicked and submit button is pressed then Citizen1 Table is queried for all records present in it. After the query is successfully executed then all the values in that table is collected and displayed in a tabular form on the page itself as an output. Also, this information that is displayed ia modified on updating classes like registerFir, UpdateFir, DeleteFir and hence the changes should be updated on the current page too when queried again.

Test Inputs: Since there are particular radio buttons for each case, this eliminates the possibility of wrong input being transmitted. Therefore each radio button has been duly clicked and checked to display the correct data as present in the database.

Test Results:

Result: PASSED

Output: Function is returning appropriate result for the given arguments.

No BUGS, Faulty output has been received.

2.10 Unit → viewStaff/deleteStaff

Section 1 – Black Box Test Cases

Test Case for Staff Model

Description: viewStaff()/deleteStaff() function is triggered radio button for Staff is clicked and submit button is pressed

Test Approach: When Staff Radio Button is clicked and submit button is pressed then Staff1 Table is queried for all records present in it. After the query is successfully executed then all the values in that table is collected and displayed in a tabular form on the page itself as an output. Also, this information that is displayed is modified on updating classes like registerFir, UpdateFir, DeleteFir and hence the changes should be updated on the current page too when queried again.

Test Inputs: Since there are particular radio buttons for each case, this eliminates the possibility of wrong input being transmitted. Therefore each radio button has been duly clicked and checked to display the correct data as present in the database.

Test Results:

Result: PASSED

Output: Function is returning appropriate result for the given arguments.

No BUGS, Faulty output has been received.

2.11 Unit → viewCriminal/deleteCriminal

Section 1 – Black Box Test Cases

<u>Test Case for Criminal Model</u>

Description: viewCriminal()/deleteCriminal() function is triggered radio button for Criminal is clicked and submit button is pressed

Test Approach: When Criminal Radio Button is clicked and submit button is pressed then Criminal Table is queried for all records present in it. After the query is successfully executed then all the values in that table is collected and displayed in a tabular form on the page itself as an output. Also, this information that is displayed is modified on updating classes like registerFir, UpdateFir, DeleteFir and hence the changes should be updated on the current page too when queried again.

Test Inputs: Since there are particular radio buttons for each case, this eliminates the possibility of wrong input being transmitted. Therefore each radio button has been duly clicked and checked to display the correct data as present in the database.

Test Results:

Result: PASSED

Output: Function is returning appropriate result for the given arguments.

No BUGS, Faulty output has been received.

2.12 Unit → viewVictim/deleteVictim

Section 1 – Black Box Test Cases

<u>Test Case for Victim Model</u>

Description: viewVictim()/deleteVictim() function is triggered radio button for Victim is clicked and submit button is pressed

Test Approach: When Victim Radio Button is clicked and submit button is pressed then Victim Table is queried for all records present in it. After the query is successfully executed then all the values in that table is collected and displayed in a tabular form on the page itself as an output. Also, this information that is displayed is modified on updating classes like registerFir, UpdateFir, DeleteFir and hence the changes should be updated on the current page too when queried again.

Test Inputs: Since there are particular radio buttons for each case, this eliminates the possibility of wrong input being transmitted. Therefore each radio button has been duly clicked and checked to display the correct data as present in the database.

Test Results:

Result: PASSED

Output: Function is returning appropriate result for the given arguments.

No BUGS, Faulty output has been received.

2.13 Unit → viewWitness/deleteWitness

Section 1 – Black Box Test Cases

<u>**Test Case for Criminal Model**</u>

Description: viewWitness ()/deleteWitness() function is triggered radio button for Witness is clicked and submit button is pressed

Test Approach: When Witness Radio Button is clicked and submit button is pressed then Witness Table is queried for all records present in it. After the query is successfully executed then all the values in that table is collected and displayed in a tabular form on the page itself as an output. Also, this information that is displayed is modified on updating classes like registerFir, UpdateFir, DeleteFir and hence the changes should be updated on the current page too when queried again.

Test Inputs: Since there are particular radio buttons for each case, this eliminates the possibility of wrong input being transmitted. Therefore each radio button has been duly clicked and checked to display the correct data as present in the database.

Test Results:

Result: PASSED

Output: Function is returning appropriate result for the given arguments.

No BUGS, Faulty output has been received.

2.14 Unit → SignUpBtn_Click [valid for all]

Section 1 – Black Box Test Cases

Test Case	Function	Type
Test Case 1	SignUpBtn_Click()	- Valid Name
Test Case 2	SignUpBtn_Click()	- Invalid Name

Test Case 3	SignUpBtn_Click()	- Valid Age
Test Case 4	SignUpBtn_Click()	- Invalid Age
Test Case 5	SignUpBtn_Click()	- Valid Address
Test Case 6	SignUpBtn_Click()	- Invalid Address
Test Case 7	SignUpBtn_Click()	- Valid gender
Test Case 8	SignUpBtn_Click()	- Invalid gender
Test Case 9	SignUpBtn_Click()	- Valid Password
Test Case 10	SignUpBtn_Click()	- Invalid Password
Test Case 11	SignUpBtn_Click()	- Valid username
Test Case 12	SignUpBtn_Click()	- Invalid username

Test Case 1

Description: this function type records the value inputted by the user in the form of alphabetic entry only

Test Approach: testing will be done by inputting valid entries of alphabets only that will be accepted. User may input various alphabets in any form of permutation and combination that he/she may like in order to create a desired user name.

Test Inputs: If the name contains only alphabets then it's considered a valid entry

Test Results:

Result is successful, as the entry contains only alphabets.

<u>Test Case 2</u>

Description: this function type records the value inputted by the user in the form of alphabetic entry.

Test Approach: testing will be done by inputting valid entries of alphabets only that will be accepted. User may input various alphabets in any form of permutation and combination that he/she may like in order to create a desired user name. If user inputs any other form of input for example a numerical or a special character, the entry becomes invalid or is considered invalid

Test Inputs: If the name contains numerical, alpha numerical characters, special characters and etc., then it's considered invalid input.

Test Results:

Result is successful, as the entry was detected as invalid.

<u>Test Case 3</u>

Description: this function type records the value inputted by the user in the form of numerical entry only

Test Approach: testing will be done by inputting valid entries of numerical values only that will be accepted. User may input various numericals that showcase their valid ages. Since age is in numeric hence he/she can only input values in numeric and not any other form of character.

Test Inputs: If the age contains only numeric then it's considered a valid entry

Test Results:

Result is successful, as the entry contains only numeric.

<u>Test Case 4</u>

Description: this function type records the value inputted by the user in the form of numerical entry only.

Test Approach: testing will be done by inputting invalid entries of various characters apart from numerical values only that will be accepted. Since age is in numeric hence he/she can only input values in numeric and not any other form of character , if any other character is inputted then it is considered invalid.

Test Inputs: If the age contains no numeric then it’s considered a valid entry

Test Results:

Result is successful, as the entry contains no numeric and is classified as invalid.

Test Case 5

Description: this function type records the value inputted by the user for the address field.

Test Approach: testing will be done by inputting valid entries which are genuine map locations else the input location will be truncated

Test Inputs: If the address is a valid map location then it will be considered as valid for the system to store in its database. For example , if we put NIIT UNIVERSITY , NEEMRANA , ALWAR , RAJASTHAN , this is a valid address. If we put NIIT UNIVERSITY, NEEMRANA, MIAMI, MARS. This location does not exist. Hence its considered invalid.

Test Results:

Result is successful, as the entry contains input which has valid map location.

Test Case 6

Description: this function type records the value inputted by the user for the address field.

Test Approach: testing will be done by inputting invalid entries which aren't genuine map locations else the input location will be valid.

Test Inputs: If the address is a valid map location then it will be considered as valid for the system to store in its database. For example, if we put NIIT UNIVERSITY, NEEMRANA, ALWAR, RAJASTHAN, this is a valid address. If we put NIIT UNIVERSITY, NEEMRANA, MIAMI, MARS. This location does not exist. Hence it's considered invalid.

Test Results:

Result is successful, as the entry contains input which has invalid map location.

Test Case 7

Description: this function type records the value inputted by the user for the gender field.

Test Approach: testing will be done by inputting valid entries which are single character M/F only which will be accepted by the system. No numeric, alpha numeric and string of alphabets are allowed

Test Inputs: If the input from the user is a single character m or f , it will be accepted by the system else any other string of characters is considered invalid. For example:

M –valid

F –valid

Male –invalid

Female –invalid

m@l* -invalid

Test Results:

Result is successful, as the entry contains input which has single character.

Test Case 8

Description: this function type records the value inputted by the user for the gender field.

Test Approach: testing will be done by inputting invalid entries which are numeric, alpha numeric and string of alphabets that are invalid to the system.

Test Inputs: If the input from the user is a single character m or f , it will be accepted by the system else any other string of characters is considered invalid. For example:

M –valid

F –valid

Male –invalid

Female –invalid

m@l* -invalid

Test Results:

Result is successful, as the entry contains input which has invalid input.

Test Case 9

Description: this function type records the value inputted by the user in the form of string of characters. The user must not only enter the password in the form of numerics. It should stay a combination of all characters.

Test Approach: testing will be done by inputting valid entries as specified above .

Test Inputs: If the password contains all required characters it is a valid entry and will be accepted for example: Niit@123 this is valid, invalid will be 123123.

Test Results:

Result is successful, as the entry contains specified character set.

Test Case 10

Description: this function type records the value inputted by the user in the form of string of characters. The user must not only enter the password in the form of numerics. It should stay a combination of all characters if only one set of characters are used it’s invalid.

Test Approach: testing will be done by inputting invalid entries as specified above.

Test Inputs: If the password contains all required characters it is a valid entry and will be accepted for example: Niit@123 this is valid, invalid will be 123123.

Test Results:

Result is successful, as the entry contains invalid input in the form of single characters like in the above example.

Test Case 11

Description: this function type records the value inputted by the user in the form of string of characters that will be displayed at the user profiles home page. This is a uniquely generated name and every name is different in order to differentiate b/w the people with a greater sense of ease.

Test Approach: testing will be done by inputting valid entries as specified above. If its unique the username will be generated

Test Inputs: the username inputted by the user should be unique using various characters of different types. For example : if niit123 exists , then the user will not be able to access this name instead he would need to make a unique one like niit678. This will be accepted by the system and the username will be generated

Test Results:

Result is successful, as the entry contains specified character set and the username has been generated for the user.

Test Case 12

Description: this function type records the value inputted by the user in the form of string of characters that will be displayed at the user profiles home page. This is a uniquely generated name and every name is different in order to differentiate b/w the people with a greater sense of ease.

Test Approach: testing will be done by inputting valid entries as specified above. If its unique the username will be generated. Else the system would require the user to try a new username that is not existing in the system.

Test Inputs: the username inputted by the user should be unique using various characters of different types. For example: if niit123 exists, then the user will not be able to access this name instead he would need to make a unique one like niit678. But if still the user types niit123 he would be informed on the system in the form of a pop up that the username has been taken and cannot exist, therefore he would have to try again.

Test Results:

Result is successful, as the entry was a preexisting username and wasn't generated, the user was informed so as to generate a new username.

Section 2 – White Box Test Cases

Note:

White Box test cases included (for a given unit of code) path coverage, branch coverage, statement coverage that must have been achieved by the derived set of test cases. The above test cases which were covered (in the section Black Box Test Cases) . There is no need of extra test cases to complete or achieve the above coverage's that is path, statement and branch. So .we can conclude that the white box test cases have been executed as above.

2.15 Unit--RESET( )

Description: this function would reset all the fields to their default values and its applicable for all sorts of users and on all fields.

Test Approach: the user is to click on the reset button to wipe out all his saved fields to the default empty fields so that he can input values and descriptions at ease , as its not expected out of the user to delete each and every field one by one causing him/her inconvenience if they commit any mistake or error earlier.

Test Inputs: button is to be clicked causing the fields to get empty.

Test Results:

Result is successful, as the reset button emptied all the fields .

2.16 Unit → updateCitizen/updateStaff

Section 1 – Black Box Test Cases

Test Case	Function	Type
Test Case 1	updateCitizen()/updateStaff()	- Valid Name
Test Case 2	updateCitizen()/updateStaff()	- Invalid Name
Test Case 3	updateCitizen()/updateStaff()	- Valid Age
Test Case 4	updateCitizen()/updateStaff()	- Invalid Age
Test Case 5	updateCitizen()/updateStaff()	- Valid Address
Test Case 6	updateCitizen()/updateStaff()	- Invalid Address
Test Case 7	updateCitizen()/updateStaff()	- Valid gender
Test Case 8	updateCitizen()/updateStaff()	- Invalid gender

Test Case 1

Description: this function type update the value inputted by the user in the form of alphabetic entry only

Test Approach: testing will be done by inputting valid entries of alphabets only that will be accepted. User may input various alphabets in any form of permutation and combination that he/she may like in order to create a desired user name.

Test Inputs: If the name contains only alphabets then it's considered a valid entry

Test Results:

Result is successful, as the entry contains only alphabets.

<u>**Test Case 2**</u>

Description: this function type update the value inputted by the user in the form of alphabetic entry.

Test Approach: testing will be done by inputting valid entries of alphabets only that will be accepted. User may input various alphabets in any form of permutation and combination that he/she may like in order to create a desired user name. If user inputs any other form of input for example a numerical or a special character, the entry becomes invalid or is considered invalid

Test Inputs: If the name contains numerical, alpha numerical characters, special characters and etc., then it's considered invalid input.

Test Results:

Result is successful, as the entry was detected as invalid.

<u>Test Case 3</u>

Description: this function type update the value inputted by the user in the form of numerical entry only

Test Approach: testing will be done by inputting valid entries of numerical values only that will be accepted. User may input various numericals that showcase their valid ages. Since age is in numeric hence he/she can only input values in numeric and not any other form of character.

Test Inputs: If the age contains only numeric then it's considered a valid entry

Test Results:

Result is successful, as the entry contains only numeric.

<u>Test Case 4</u>

Description: this function type update the value inputted by the user in the form of numerical entry only.

Test Approach: testing will be done by inputting invalid entries of various characters apart from numerical values only that will be accepted. Since age is in numeric hence he/she can only input values in numeric and not any other form of character , if any other character is inputted then it is considered invalid.

Test Inputs: If the age contains no numeric then it's considered a valid entry

Test Results:

Result is successful, as the entry contains no numeric and is classified as invalid.

<u>Test Case 5</u>

Description: this function type update the value inputted by the user for the address field.

Test Approach: testing will be done by inputting valid entries which are genuine map locations else the input location will be truncated

Test Inputs: If the address is a valid map location then it will be considered as valid for the system to store in its database. For example , if we put NIIT UNIVERSITY , NEEMRANA , ALWAR , RAJASTHAN , this is a valid address. If we put NIIT UNIVERSITY, NEEMRANA, MIAMI, MARS. This location does not exist. Hence its considered invalid.

Test Results:

Result is successful, as the entry contains input which has valid map location.

<u>Test Case 6</u>

Description: this function type update the value inputted by the user for the address field.

Test Approach: testing will be done by inputting invalid entries which aren’t genuine map locations else the input location will be valid.

Test Inputs: If the address is a valid map location then it will be considered as valid for the system to store in its database. For example, if we put NIIT UNIVERSITY, NEEMRANA, ALWAR, RAJASTHAN, this is a valid address. If we put NIIT UNIVERSITY, NEEMRANA, MIAMI, MARS. This location does not exist. Hence it's considered invalid.

Test Results:

Result is successful, as the entry contains input which has invalid map location.

Test Case 7

Description: this function type update the value inputted by the user for the gender field.

Test Approach: testing will be done by inputting valid entries which are single character M/F only which will be accepted by the system. No numeric, alpha numeric and string of alphabets are allowed

Test Inputs: If the input from the user is a single character m or f , it will be accepted by the system else any other string of characters is considered invalid. For example:

M –valid

F –valid

Male –invalid

Female –invalid

m@l* -invalid

Test Results:

Result is successful, as the entry contains input which has single character.

Test Case 8

Description: this function type update the value inputted by the user for the gender field.

Test Approach: testing will be done by inputting invalid entries which are numeric, alpha numeric and string of alphabets that are invalid to the system.

Test Inputs: If the input from the user is a single character m or f , it will be accepted by the system else any other string of characters is considered invalid. For example:

M –valid

F –valid

Male –invalid

Female –invalid

m@l* -invalid

Test Results:

Result is successful, as the entry contains input which has invalid input.

Section 2 – White Box Test Cases

Note:

White Box test cases included (for a given unit of code) path coverage, branch coverage, statement coverage that must have been achieved by the derived set of test cases. The above test cases which were covered (in the section Black Box Test Cases) . There is no need of extra test cases to complete or achieve the above coverage's that is path, statement and branch. So .we can conclude that the white box test cases have been executed as above.

2.15 LOGOUT()

Description: this function would let the user end its active session and return to the page asking for login information

Test Approach: The user is to click on the logout button to wipe out all his active sessions. The user then should be prompted for is credentials again if he would like to use the system again

Test Inputs: button is to be clicked causing the user to get redirected to the home page prompting for his credentials again and all functionalities from the user is withdrawn.

Test Results:

Result is successful, as the logout button ended the session of the user.

INTEGRATING TESTING

Nextly, We have Integrating Testing which

Unit1 →`ViewUserDetailsbyStaff/ViewCitizen()/ViewDetails`

Test Case for : `viewCitizen();`

Description:viewCitizen()function is triggered when View User Details is clicked.

Test Approach: When Citizen Button is clicked thenCitizen Details are retrieved directly from the database. Further thereafter the query is successfully executed then all the values in that table is collected and displayed in a tabular form on the page itself as an output. Also, this information that is displayed is modified on Viewingfunction like ViewFIRbyStaff() and hence the changes should be displayed on the current page too when queried again.

Test Inputs: Since there are particular radio buttons for each case, this eliminates the possibility of wrong input being transmitted. Therefore each radio button has been duly clicked and checked to display the correct data as present in the database.

Test Results:

Result: PASSED

Output: Function is returning appropriate result for the given arguments.

No BUGS, Faulty output has been received.

Test Case	Function	Type
Test case 1	`Button1_Click`	Valid citizenid
Test case 2	`Button1_Click`	Invalid citizenid
Test case 3	`Button1_Click`	Valid location
Test case 4	`Button1_Click`	Invalid location
Test case 5	`Button1_Click`	Valid Description

Unit2→`ViewUserDetailsbyStaff/ViewUserDetailsbyStaff/ViewCitizen()/ViewDetails`

Test Case for : `viewStaff();`

Description:viewStaff()function is triggered when View User Details is clicked.

Test Approach: When Staff Button is clicked thenCitizen Details are retrieved directly from the database. Further thereafter the query is successfully executed then all the values in that table is collected and displayed in a tabular form on the page itself as an output. Also, this information that is displayed is modified on Viewingfunction like ViewFIRbyStaff() and hence the changes should be displayed on the current page too when queried again.

Test Inputs: Since there are particular radio buttons for each case, this eliminates the possibility of wrong input being transmitted. Therefore each radio button has been duly clicked and checked to display the correct data as present in the database.

Test Results:

Result: PASSED

Output: Function is returning appropriate result for the given arguments.

No BUGS, Faulty output has been received.

Test Case	Function	Type
Test case 1	`Button1_Click`	Valid citizenid
Test case 2	`Button1_Click`	Invalid citizenid

Test case 3	`Button1_Click`	Valid location
Test case 4	`Button1_Click`	Invalid location
Test case 5	`Button1_Click`	Valid Description

Unit3→`ViewUserDetailsbyStaff/ViewUserDetailsbyStaff/ViewCitizen()/ViewDetails`

Test Case for : `viewCriminal();`

Description: viewCitizen()/deleteCitizen() function is triggered radio button for Citizen clicked and submit button ispressed.

Test Approach: When Citizen Radio Button is clicked and submit button is pressed then Citizen1 Table is queried for all records present in it. After the query is successfully executed then all the values in that table is collected and displayed in a tabular form on the page itself as an output. Also, this information that is displayed is modified on updating classes like registerFir, UpdateFir, DeleteFir section of three segments of Admin, Citizen, Staff and hence the changes should be updated on the current page too when queried again.

Test Inputs: Since there are particular radio buttons for each case, this eliminates the possibility of wrong input being transmitted. Therefore each radio button has been duly clicked and checked to display the correct data as present in the database.

Test Results:

Result: PASSED

Output: Function is returning appropriate result for the given arguments.

No BUGS, Faulty output has been received.

Test Case	Function	Type
Test case 1	`Button1_Click`	Valid citizenid
Test case 2	`Button1_Click`	Invalid citizenid
Test case 3	`Button1_Click`	Valid location
Test case 4	`Button1_Click`	Invalid location
Test case 5	`Button1_Click`	Valid Description

Unit4 →`ViewUserDetailsbyStaff/ViewUserDetailsbyStaff/ViewCitizen()/ViewDetails`

Test Case for : `viewVictim();`

Description: viewCitizen()/deleteCitizen() function is triggered radio button for Citizen clicked and submit button ispressed.

Test Approach: When Citizen Radio Button is clicked and submit button is pressed then Citizen1 Table is queried for all records present in it. After the query is successfully executed then all the values in that table is collected and displayed in a tabular form on the page itself as an output. Also, this information that is displayed is modified on updating classes like registerFir, UpdateFir, DeleteFir and hence the changes should be updated on the current page too when queried again.

Test Inputs: Since there are particular radio buttons for each case, this eliminates the possibility of wrong input being transmitted. Therefore each radio button has been duly clicked and checked to display the correct data as present in the database.

Test Results:

Result: PASSED

Output: Function is returning appropriate result for the given arguments.

No BUGS, Faulty output has been received.

Test Case	Function	Type
Test case 1	`Button1_Click`	Valid citizenid
Test case 2	`Button1_Click`	Invalid citizenid
Test case 3	`Button1_Click`	Valid location
Test case 4	`Button1_Click`	Invalid location
Test case 5	`Button1_Click`	Valid Description

Unit5 →`ViewUserDetailsbyStaff/ViewUserDetailsbyStaff/ViewCitizen()/ViewDetails`

<u>Test Case for :</u> `viewWitness();`

Description: viewWitness()/deleteWitness() function is triggered radio button for Citizen clicked and submit button ispressed.

Test Approach: When Citizen Radio Button is clicked and submit button is pressed then Citizen1 Table is queried for all records present in it. After the query is successfully executed then all the values in that table is collected and displayed in a tabular form on the page itself as an output. Also, this information that is displayed is modified on updating classes like registerFir, UpdateFir, DeleteFir and hence the changes should be updated on the current page too when queried again.

Test Inputs: Since there are particular radio buttons for each case, this eliminates the possibility of wrong input being transmitted. Therefore each radio button has been duly clicked and checked to display the correct data as present in the database.

Test Results:

Result: PASSED

Output: Function is returning appropriate result for the given arguments.

No BUGS, Faulty output has been received.

Test Case	Function	Type
Test case 1	`Button1_Click`	Valid citizenid
Test case 2	`Button1_Click`	Invalid citizenid
Test case 3	`Button1_Click`	Valid location
Test case 4	`Button1_Click`	Invalid location
Test case 5	`Button1_Click`	Valid Description

Description: viewCitizen()/deleteCitizen() function is triggered radio button for Citizen clicked and submit button ispressed.

Test Approach: When Citizen Radio Button is clicked and submit button is pressed then Citizen1 Table is queried for all records present in it. After the query is successfully executed then all the values in that table is collected and displayed in a tabular form on the page itself as an output. Also, this information that is displayed is modified on updating classes like registerFir, UpdateFir, DeleteFir and hence the changes should be updated on the current page too when queried again.

Test Inputs: Since there are particular radio buttons for each case, this eliminates the possibility of wrong input being transmitted. Therefore each radio button has been duly clicked and checked to display the correct data as present in the database.

Test Results:

Result: PASSED

Output: Function is returning appropriate result for the given arguments.

No BUGS, Faulty output has been received.

Test Case	Function	Type
Test case 1	`Button1_Click`	Valid citizenid
Test case 2	`Button1_Click`	Invalid citizenid
Test case 3	`Button1_Click`	Valid location
Test case 4	`Button1_Click`	Invalid location
Test case 5	`Button1_Click`	Valid Description