

---

**CS-223**  
**CODE TESTING**  
**DOCUMENT**

for  
**Project 7**  
**Classroom Visualisation App-1**

Prepared by:  
Group-12  
Mitansh Jain - 160101042  
Sujoy Ghosh - 160101073  
Akul Agrawal - 160101085

**April 22, 2018**

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>Team profile for black box testing</b>	<b>7</b>
<b>3</b>	<b>Equivalence Class Partitioning</b>	<b>8</b>
3.1	Module Name: Login Module . . . . .	8
3.2	Module Name: Sign Up . . . . .	8
3.3	Module Name: Add Student . . . . .	8
3.4	Module Name: Add Images . . . . .	9
3.5	Module Name: Edit Student . . . . .	9
3.6	Module Name: View Student List . . . . .	10
3.7	Module Name: Delete Student Record . . . . .	10
3.8	Module name: Camera Session and Face Detection . . . . .	10
<b>4</b>	<b>Blackbox Testing</b>	<b>12</b>
4.1	Module Name: Login Module . . . . .	12
4.2	Module Name: Sign Up . . . . .	14
4.3	Module Name: Add Student . . . . .	17
4.4	Module Name: Add Images . . . . .	21
4.5	Module Name: Edit Student . . . . .	23
4.6	Module Name: View Student List . . . . .	27
4.7	Module Name: Delete Student Record . . . . .	29
4.8	Module name: Camera Session and Face Recognition . . . . .	31
<b>5</b>	<b>White Box Testing</b>	<b>40</b>
5.1	Module: Sign Up . . . . .	40
5.1.1	Funtion: signup() . . . . .	40
5.1.2	Funtion: CheckEmailAlreadyExists() . . . . .	42
5.1.3	Funtion: CheckFinalResult() . . . . .	43
5.1.4	Funtion: validate() . . . . .	45
5.2	Module: AddStudentRecord . . . . .	48
5.2.1	Funtion: AddData() . . . . .	48
5.3	Module: AddImages . . . . .	51
5.3.1	Funtions: onResume(), addimage(), btnTakePhotoClicker.OnClickListener() . . . . .	51
5.4	Module: DeleteStudent . . . . .	53
5.4.1	Funtion: deleteStudent() . . . . .	53
5.5	Module: EditStudent . . . . .	55
5.5.1	Funtion: UpdateData() . . . . .	55

5.5.2	Funtion: UpdateImages()	57
5.6	Module: Login	60
5.6.1	Funtion: login()	60
5.6.2	Funtion: validate()	62
5.7	Module: Camera session	64
5.7.1	Funtion: faceDetect()	64
5.7.2	Funtion: getboxColor()	67

<b>6 Conclusion</b>	<b>69</b>
---------------------	-----------

# List of Figures

4.2	Input with wrong email id . . . . .	13
4.3	Input with wrong password . . . . .	13
4.4	Input with wrong email id as well as wrong password . . . . .	14
4.8	Input with Password $\neq$ Confirm Password . . . . .	16
4.9	Successful signup . . . . .	17
4.12	Input with empty details . . . . .	20
4.13	We get desired output . . . . .	21
4.14	Number of photos left = 0 . . . . .	23
4.18	Updated List . . . . .	27
4.19	Input with incorrect course id . . . . .	28
4.20	Input with corrected course id . . . . .	29
4.22	List of students in class . . . . .	35
4.23	Students identitfied correctly . . . . .	35
5.1	Code for signup() function . . . . .	40
5.2	CFG for signup() function . . . . .	41
5.3	Code for CheckEmailAlreadyExists() function . . . . .	42
5.4	CFG for CheckEmailAlreadyExists() function . . . . .	42
5.5	Code for CheckFinalResult() function . . . . .	43
5.6	CFG for checkFinalResult() function . . . . .	44
5.7	Code for validate() function . . . . .	45
5.8	CFG for validate() function . . . . .	46
5.9	Code for addData() function . . . . .	48
5.10	CFG for addData() function . . . . .	49
5.11	Code for deleteStudent() function . . . . .	53
5.12	CFG for deleteStudent() function . . . . .	54
5.13	Code for UpdateData() function . . . . .	55
5.14	CFG for UpdateData() function . . . . .	56
5.15	Code for UpdateImages() function . . . . .	57
5.16	CFG for UpdateImages() function . . . . .	58
5.17	Code for login() function . . . . .	60
5.18	CFG for login() function . . . . .	61
5.19	Code for Validate() function . . . . .	62
5.20	CFG for Validate() function . . . . .	63
5.21	Code for faceDetect() function . . . . .	64
5.22	CFG for faceDetect() function . . . . .	65
5.23	Code for getboxColor() function . . . . .	67



# **1 Introduction**

This document contains the complete unit testing of our app Classroom Visualisation. Unit testing is undertaken after a module has been coded and successfully reviewed. The code testing team in isolation tests different units and modules of the system. Different members of the code testing team have submitted their reports. In short, this document is meant to equip the reader with the bugs and shortcomings in the working of the app.

## **2 Team profile for black box testing**

The Black box testing team comprises of the following members, all of whom are Undergraduates currently pursuing Bachelor of Technology at Indian Institute of Technology Guwahati, India in the Department of Computer Science and Engineering. All of the members are currently in the sophomore year.

1. Harshit Sharma
2. Harshit Gupta
3. Harshit Agrawal

# 3 Equivalence Class Partitioning

## 3.1 Module Name: Login Module

- Equivalence Classes:
  1. **Input:** Valid Email Id, Valid password  
**Expected Output:** Directs to start session activity. Displays "Login successful".
  2. **Input:** Email Id, Valid password  
**Expected Output:** Login Failed message displayed.
  3. **Input:** Valid Email Id, Invalid password  
**Expected Output:** Login Failed message displayed.
  4. **Input:** Invalid Email Id, Invalid password  
**Expected Output:** Login Failed message displayed.
- Boundary Cases: Equivalence classes here are discrete. There are no boundary cases.

## 3.2 Module Name: Sign Up

- Equivalence Classes:
  1. **Input:** Duplicate email Id is entered  
**Expected Output:** Displays "Email ID already exists."
  2. **Input:** If any one field is left empty  
**Expected Output:** Account is not created. Marks the fields left empty.
  3. **Input:** Password ≠ Confirm Password  
**Expected Output:** Displays "Passwords do not match".
  4. **Input:** Valid email ID is entered, Name is entered, passwords entered match  
**Expected Output:** Account created successfully.
- Boundary Cases: Equivalence classes here are discrete. There are no boundary cases.

## 3.3 Module Name: Add Student

- Equivalence Classes:

1. **Input:** Roll No. not integer  
**Expected Output:** Displays "Roll Number is not an integer".
  2. **Input:** Roll No. already exists in entered course ID  
**Expected Output:** Displays "Student already exists".
  3. **Input:** Roll No. already exists in other course ID  
**Expected Output:** Adds student successfully to database.
  4. **Input:** If any one field is left empty  
**Expected Output:** Shows error on empty fields.
  5. **Input:** If all fields are filled correctly and input does not belong to any of the above class.  
**Expected Output:** Displays "Student Record created successfully".
- **Boundary Cases:** Equivalence classes here are discrete. There are no boundary cases.

### 3.4 Module Name: Add Images

- **Equivalence Classes:**
  1. **Input:** No. of added images < 15  
**Expected Output:** Option to return to main menu is not shown.
  2. **Input:** No. of added images > 15  
**Expected Output:** Clicking more than 15 images should not be allowed.
- **Boundary Cases:**
  1. **Input:** No. of added images = 15  
**Expected Output:** Gives option to return to main screen.

### 3.5 Module Name: Edit Student

- **Equivalence Classes:**
  1. **Input:** Roll No. not integer  
**Expected Output:** Displays "Roll Number is not an integer".
  2. **Input:** Roll No. already exists but in other course ID  
**Expected Output:** Displays "Student does not exists".
  3. **Input:** Roll No. doesn't exist  
**Expected Output:** Displays "Student does not exists".
  4. **Input:** If any one field is left empty  
**Expected Output:** Shows error on empty fields.

- 5. **Input:** If all fields are filled correctly and input does not belong to any of the above class.  
**Expected Output:** Displays "Data successfully updated".
- **Boundary Cases:** Equivalence classes here are discrete. There are no boundary cases.

### 3.6 Module Name: View Student List

- **Equivalence Classes:**
  1. **Input:** Course ID field is left empty  
**Expected Output:** Shows error on empty field.
  2. **Input:** Invalid Course ID  
**Expected Output:** Displays "no such course Id exists".
  3. **Input:** Valid Course ID  
**Expected Output:** Displays list of all student that were added to data base.
- **Boundary Cases:** Equivalence classes here are discrete. There are no boundary cases.

### 3.7 Module Name: Delete Student Record

- **Equivalence Classes:**
  1. **Input:** Roll No. not an integer  
**Expected Output:** Displays "Roll Number not an integer".
  2. **Input:** Roll No. exists.  
**Expected Output:** Displays "Roll Number deleted". Roll Number should not be visible in table of students in from view student list module.
- **Boundary Cases:** Equivalence classes here are discrete. There are no boundary cases.

### 3.8 Module name: Camera Session and Face Detection

- **Equivalence Classes:**
  1. **Input:** State of student < 5.  
**Expected Output:** Bounding box color: red
  2. **Input:** 5 < State of student < 8  
**Expected Output:** Bounding box color: blue
  3. **Input:** 8 < State of student  
**Expected Output:** Bounding box color: green

4. **Input:** More than one known students(1 to 5) are standing  
**Expected Output:** Students should be correctly identified.

- **Boundary Cases:**

1. **Input:** State of student = 5.  
**Expected Output:** Bounding box color: blue
2. **Input:** State of student = 8  
**Expected Output:** Bounding box color: green
3. **Input:** Number of known students = 1  
**Expected Output:** Student should be correctly identified.
4. **Input:** Number of known students = 0  
**Expected Output:** Nothing should be shown.
5. **Input:** Number of unknown students = 1  
**Expected Output:** Student should be marked with unknown.

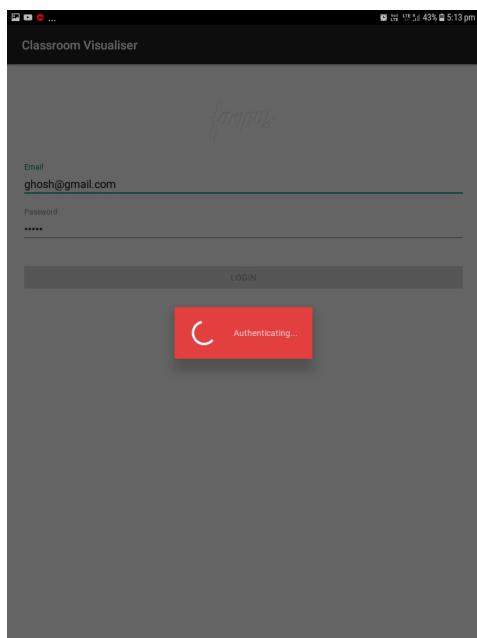
# 4 Blackbox Testing

## 4.1 Module Name: Login Module

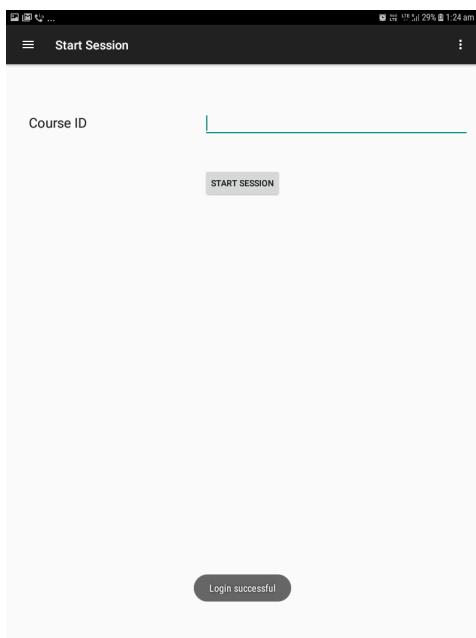
- Equivalence Classes:

1. **Input:** Valid Email : ghosh@gmail.com, Valid password : 12345

**Expected Output:** Directs to start session activity. Displays "Login successful".



(a) Input Credentials



(b) Output

**Output:** Directs to start session activity. Displays "Login successful". Hence we get our desired output.

2. **Input:** Invalid email : gho@gmail.com, Valid password : 12345

**Expected Output:** Login Failed message displayed.

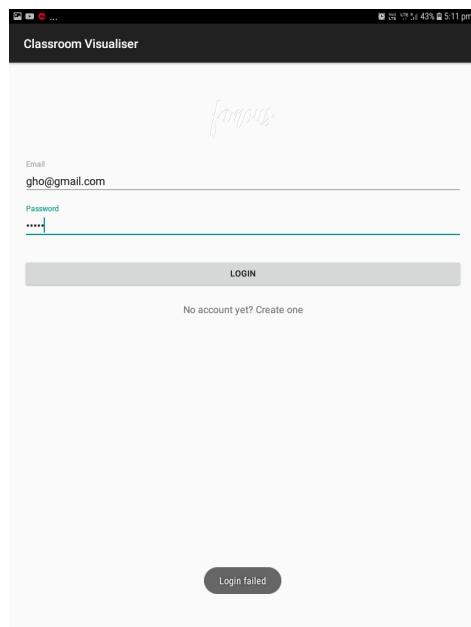


Figure 4.2: Input with wrong email id

**Output:** 'Login Failed' is displayed. Hence we get our desired output.

3. **Input:** Valid Email Id : ghosh@gmail.com, Invalid password : 123456  
**Expected Output:** Login Failed message displayed.

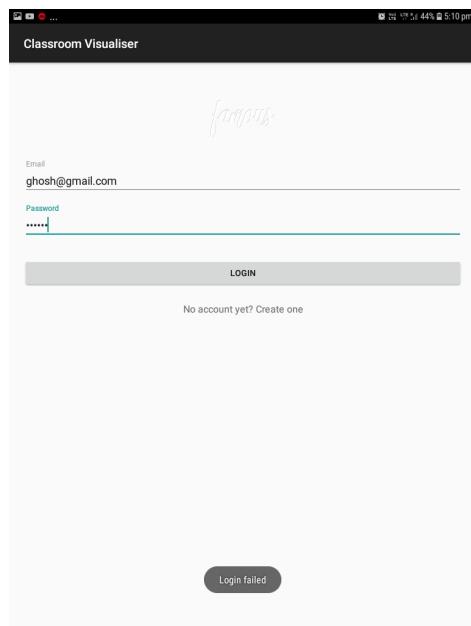


Figure 4.3: Input with wrong password

**Output:** 'Login Failed' is displayed. Hence we get our desired output.

4. **Input:** Invalid Email Id : gho@gmail.com, Invalid password : 12345678  
**Expected Output:** Login Failed message displayed.

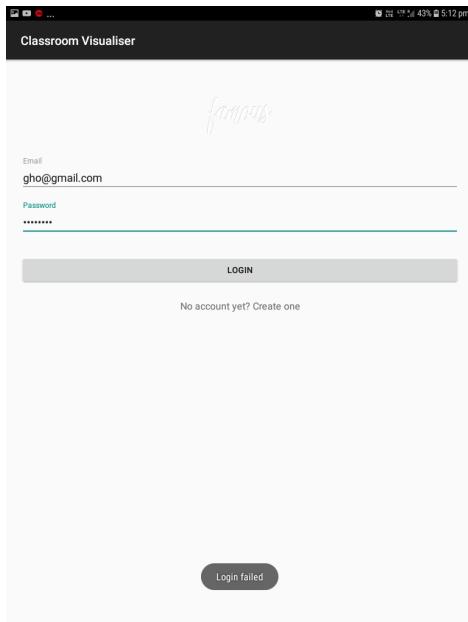


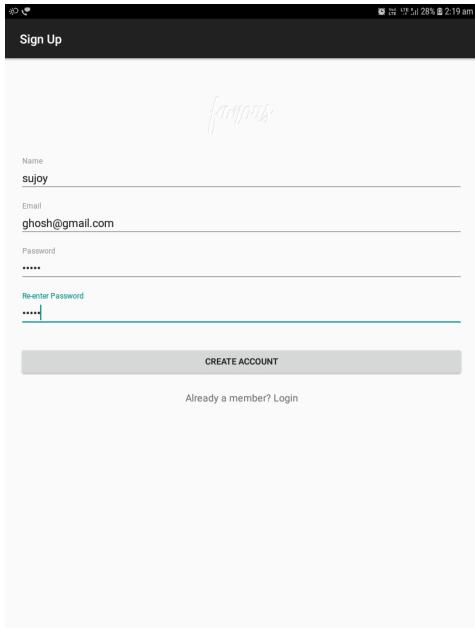
Figure 4.4: Input with wrong email id as well as wrong password

**Output:** 'Login Failed' is displayed. Hence we get our desired output.

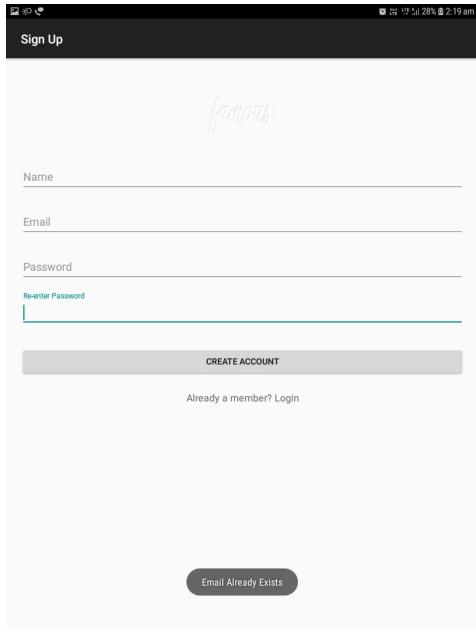
## 4.2 Module Name: Sign Up

- **Equivalence Classes:**

1. **Input:** Duplicate email Id is entered : ghosh@gmail.com  
**Expected Output:** Displays "Email ID already exists".



(a) Input Credentials

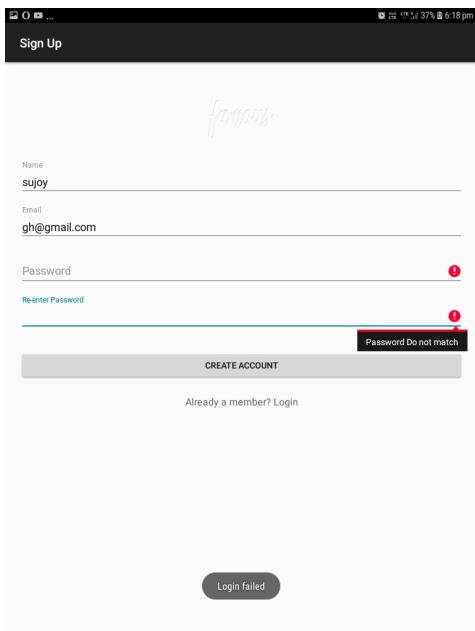


(b) Output : Email exists

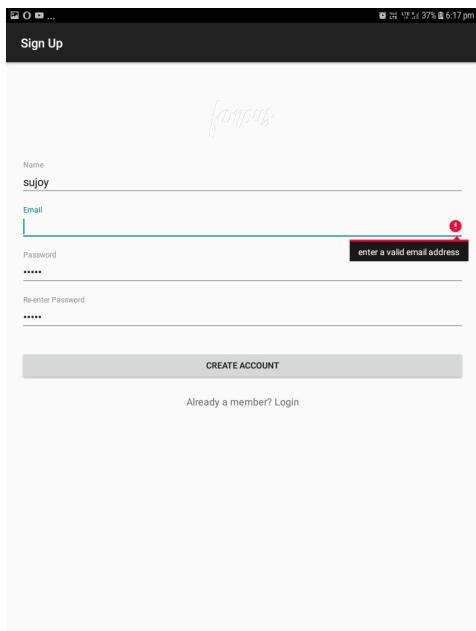
**Output:** Displays "Email already exists". Hence we get our expected output

## 2. Input: If any one field is left empty

**Expected Output:** Account is not created. Marks the fields left empty.



(a) Input with password empty



(b) Input with email empty

The screenshot shows a 'Sign Up' screen with a black header bar. Below it is a white form area with handwritten text 'famous' above it. The form fields are:

- Name:** (empty field)
- Email:** gh@gmail.com
- Password:** (empty field)
- Re-enter Password:** (empty field)

A red error message 'at least 3 characters' is displayed above the Name field. A red error message 'between 4 and 10 alphanumeric characters' is displayed above the Password field.

**CREATE ACCOUNT**

Already a member? Login

**Login failed**

(a) Input with name empty

The screenshot shows a 'Sign Up' screen with a black header bar. Below it is a white form area with handwritten text 'famous' above it. The form fields are:

- Name:** sujoy
- Email:** gh@gmail.com
- Password:** (empty field)
- Re-enter Password:** (empty field)

A red error message 'between 4 and 10 alphanumeric characters' is displayed above the Password field.

**CREATE ACCOUNT**

Already a member? Login

**Login failed**

(b) Input with password empty

**Output:** Account is not created. Hence we get our desired output.

### 3. Input: Password $\neq$ Confirm Password

**Expected Output:** Displays "Passwords do not match".

The screenshot shows a 'Sign Up' screen with a black header bar. Below it is a white form area with handwritten text 'famous' above it. The form fields are:

- Name:** sujoy
- Email:** gh@gmail.com
- Password:** (empty field)
- Re-enter Password:** \*\*\*\*\*

A red error message 'Password Do not match' is displayed above the Re-enter Password field.

**CREATE ACCOUNT**

Already a member? Login

**Login failed**

Figure 4.8: Input with Password  $\neq$  Confirm Password

**Output:** "Passwords do not match" is displayed. Hence we get our desired output.

4. **Input:** Valid email ID is entered, Name is entered, passwords entered match
- Expected Output:** Account created successfully.

(a) Input with correct credentials

(b) Output with successful login

Figure 4.9: Successful signup

**Output:** "Signup and login successful" is displayed. Hence we get our desired output

### 4.3 Module Name: Add Student

- **Equivalence Classes:**

1. **Input:** Roll No. not integer

**Expected Output:** Displays "Roll Number is not an integer".

The screenshot shows a 'Add Student' form with the following fields:

Name	sujoy
Surname	ghosh
Roll ID	1601010sd
Course	cs243

A button labeled 'ADD DATA' is at the bottom. A message box at the bottom center says 'Roll Number is not integer'.

**Output:** Displays "Roll number is not integer". We get our expected output.

2. **Input:** Roll No. already exists in entered course ID

**Expected Output:** Displays "Student already exists".

The screenshot shows a 'Add Student' form with the following fields:

Name	Gupta
Surname	Harshit
Roll ID	160101031
Course	cs243

A button labeled 'ADD DATA' is at the bottom. A message box at the bottom center says 'Roll Number already present in this course'.

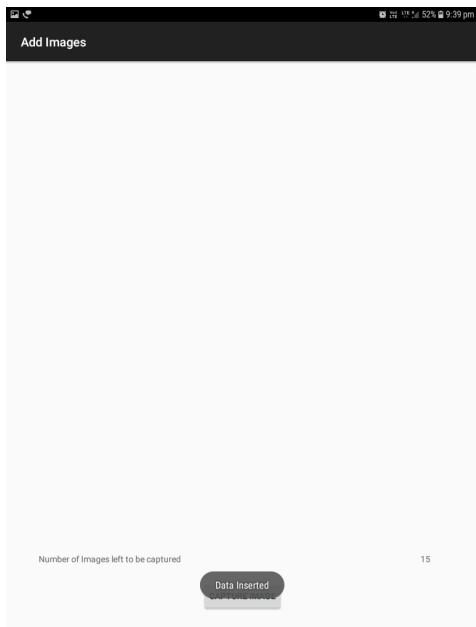
**Output:** Displays "Roll number already present in database". We get our desired output.

**3. Input:** Roll No. already exists in other course ID

**Expected Output:** Adds student successfully to database.

The screenshot shows a mobile application interface titled "Add Student". It contains four text input fields: "Name" (Sharma), "Surname" (Harshit), "Roll ID" (160101032), and "Course" (cs243). Below these fields is a button labeled "ADD DATA".

(a) Input with correct details



(b) Output

**Output:** Displays "Data inserted". Hence, we get our desired output.

**4. Input:** If any one field is left empty

**Expected Output:** Shows error on empty fields.

The figure consists of two side-by-side screenshots of a mobile application interface titled "Add Student".

**Screenshot 1 (Left):**

- Name: sujoy
- Surname: ghosh
- Roll ID: (empty)
- Course: cs243

**Screenshot 2 (Right):**

- Name: sujoy
- Surname: (empty)
- Roll ID: 160101067
- Course: cs243

In both screenshots, there is a small circular message at the bottom center stating "Field(s) is empty". Below the form is a grey button labeled "ADD DATA".

A single screenshot of a mobile application interface titled "Add Student".

- Name: (empty)
- Surname: (empty)
- Roll ID: 160101067
- Course: cs243

A small circular message at the bottom center states "Field(s) is empty". Below the form is a grey button labeled "ADD DATA".

Figure 4.12: Input with empty details

**Output:** Displays "Field(s) is empty" and doesn't insert it to database. Hence we get our desired output

5. **Input:** If all fields are filled correctly and input does not belong to any of the above class.

**Expected Output:** Displays "Student Record created successfully".

(a) Input with correct details

(b) Output

Figure 4.13: We get desired output

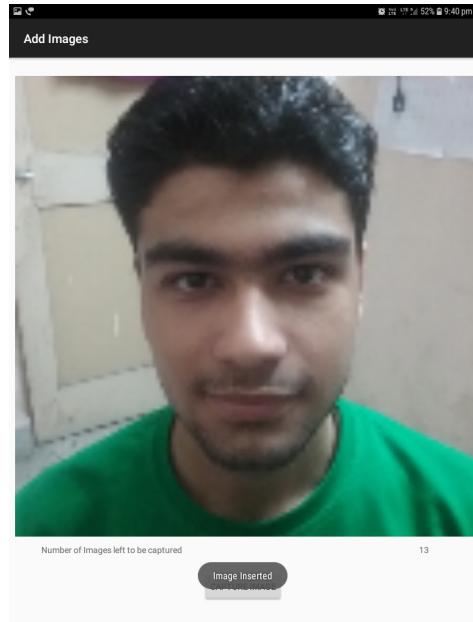
**Output:** It displays "Data inserted". Hence we get our desired output.

#### 4.4 Module Name: Add Images

- **Equivalence Classes:**

1. **Input:** No. of added images < 15

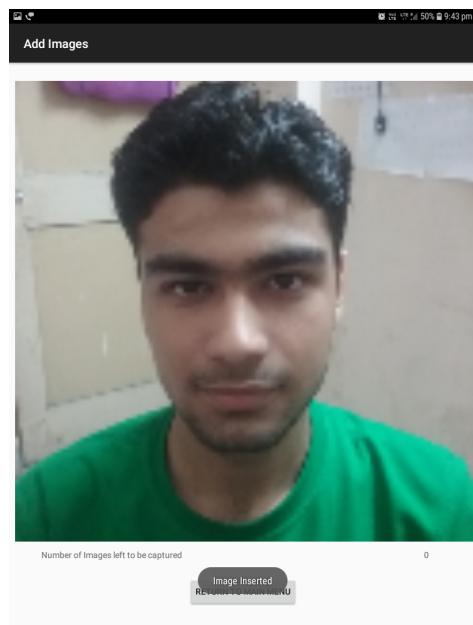
**Expected Output:** Option to return to main menu is not shown.



**Output:** Option to return to main menu is not shown. Hence we get our desired output.

2. **Input:** No. of added images > 15

**Expected Output:** Clicking more than 15 images should not be allowed.



**Output:** Return to main menu is shown when images=15. Hence we get our desired output.

- **Boundary Cases:**

1. **Input:** No. of added images = 15

**Expected Output:** Gives option to return to main screen.



Figure 4.14: Number of photos left = 0

**Output:** Return to main menu is shown when images=15. Hence we get our desired output.

## 4.5 Module Name: Edit Student

- **Equivalence Classes:**

1. **Input:** Roll No. not integer

**Expected Output:** Displays "Roll Number is not an integer".

The screenshot shows the 'Edit Student' form with the following fields and errors:

- Name: gupta
- Surname: harshit
- Roll ID: 16010103a (highlighted in red)
- Course: cs243

Below the form are two buttons: 'EDIT DATA' and 'EDIT IMAGES'. At the bottom center is a message: "Roll Number is not valid integer".

**Output:** Displays "Roll Number is not an integer". We get our desired output.

2. **Input:** Roll No. already exists but in other course ID

**Expected Output:** Displays "Student does not exists".

The screenshot shows the 'View Students' form with the following input:

Course ID: cs110

Below the input is a 'CLEAR' button. A table below shows student data:

Roll Number	Name	Course Code	Last Appear	Box Colour
160101073	Sujoy	cs110	13:13:52	4

(a) Input with correct details

The screenshot shows the 'Edit Student' form with the following fields:

- Name: Sujoy
- Surname: Ghosh
- Roll ID: 160101073
- Course: cs243

Below the form are two buttons: 'EDIT DATA' and 'EDIT IMAGES'. At the bottom center is a message: "No such data found".

(b) Output

**Output:** Displays "No such data found". We get our desired output.

**3. Input:** Roll No. doesn't exist

**Expected Output:** Displays "Student does not exists".

Name gupta

Surname harshit

Roll ID 160101033

Course cs243

EDIT DATA

EDIT IMAGES

No such data found

**Output:** Displays "No such data found". Hence, we get our desired output.

**4. Input:** If any one field is left empty

**Expected Output:** Shows error on empty fields.

Name gupta

Surname harshit

Roll ID

Course cs243

EDIT DATA

EDIT IMAGES

Field(s) is empty

(a) Input with empty fields

Name gupta

Surname

Roll ID 160101031

Course cs243

EDIT DATA

EDIT IMAGES

Field(s) is empty

(b) Output

Add Student

Name \_\_\_\_\_

Surname \_\_\_\_\_

Roll ID 160101067

Course cs243

**ADD DATA**

Field(s) is empty

**Output:** Displays "Field(s) is empty". We get our desired output.

5. **Input:** If all fields are filled correctly and it does not belong to any of the above class.

**Expected Output:** Displays "Data successfully updated".

View Students

Course ID cs243

**CLEAR**

Roll Number	Name	Course Code	Last Appear	Box Colour
160101031	Gupta	cs243	23:19:47	2
160101032	Sharma	cs243	23:11:16	9
160101042	Mitansh	cs243	23:14:11	4
160101030	Agrawal	cs243	23:14:11	5

(a) Student list

Edit Student

Name mitansh

Surname Jain

Roll ID 160101042

Course cs243

**EDIT DATA**

**EDIT IMAGES**

Data Updated

(b) Output

View Students					
Course ID					
<input type="text" value="cs243"/>					
<input type="button" value="CLEAR"/>					
Roll Number	Name	Course Code	Last Appear	Box Colour	
160101031	Gupta	cs243	23:19:47	2	
160101032	Sharma	cs243	23:11:16	9	
160101042	mitansh	cs243	13:12:18	4	
160101030	Agrawal	cs243	23:14:11	5	

Figure 4.18: Updated List

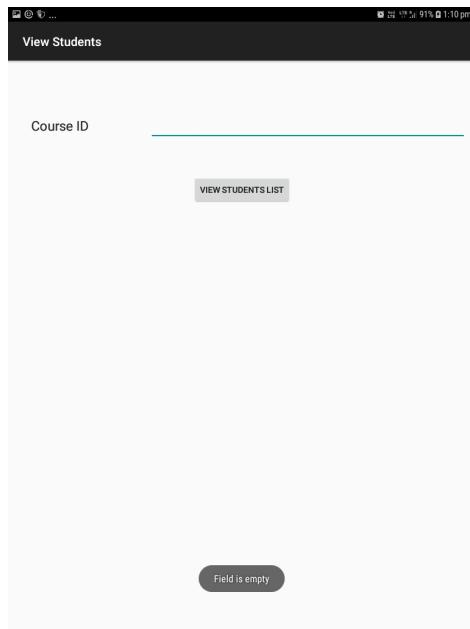
**Output:** Displays "Data Updated". Student list was also updated. We get our desired output.

## 4.6 Module Name: View Student List

- Equivalence Classes:

1. **Input:** Course ID field is left empty

**Expected Output:** Shows error on empty field.



**Output:** Displays "Field is empty". We get our desired output.

2. **Input:** Invalid Course ID

**Expected Output:** Displays "no such course Id exists".

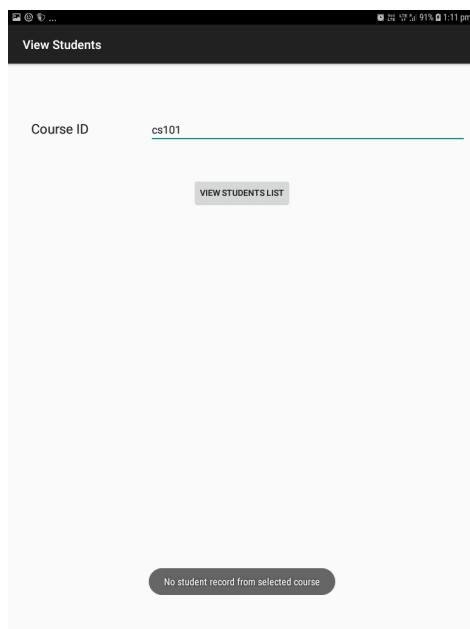


Figure 4.19: Input with incorrect course id

**Output:** Displays "No student record from selected course". We get our de-

sired output.

3. **Input:** Valid Course ID

**Expected Output:** Displays list of all student that were added to data base.

The screenshot shows a software window titled "View Students". At the top, there is a search bar labeled "Course ID" with the value "cs243". Below the search bar is a "CLEAR" button. A table below the search bar displays student records:

Roll Number	Name	Course Code	Last Appear	Box Colour
160101031	Gupta	cs243	23:19:47	2
160101032	Sharma	cs243	23:11:16	9
160101042	Mitansh	cs243	23:14:11	4
160101030	Agrawal	cs243	23:14:11	5

Figure 4.20: Input with corrected course id

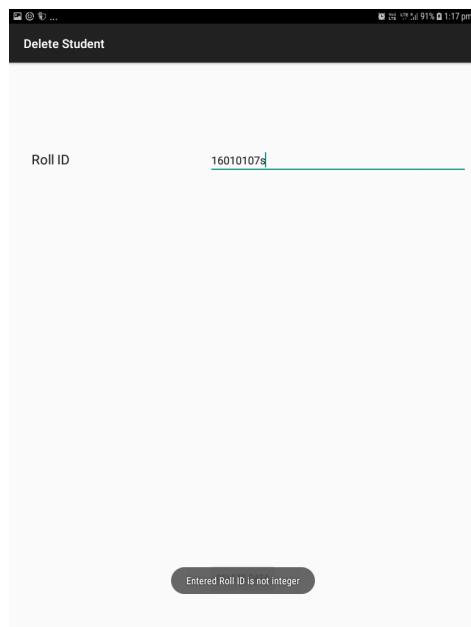
**Output:** Displays the list of students. We get our desired output.

## 4.7 Module Name: Delete Student Record

- **Equivalence Classes:**

1. **Input:** Roll No. not an integer

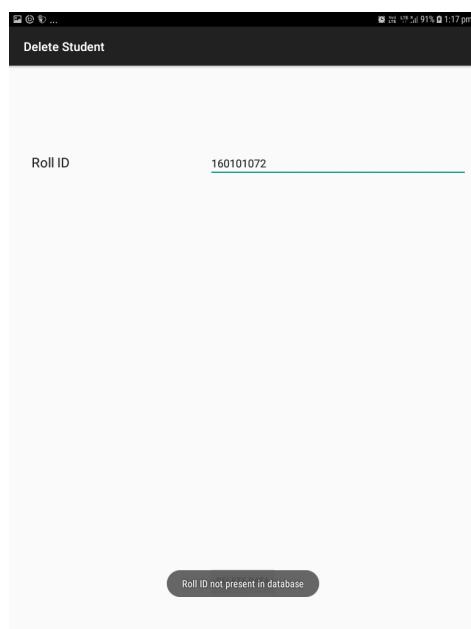
**Expected Output:** Displays "Roll Number not an integer".



**Output:** Displays "Roll Number is not an integer". We get our desired output.

2. **Input:** Roll No.doesn't exists.

**Expected Output:** Displays "Roll Number does not exist".



**Output:** Displays "Roll Number not in database". We get our desired output.

**3. Input:** Roll No. exists.

**Expected Output:** Displays "Data deleted". Roll Number should not be visible in table of students in from view student list module.

The screenshot shows a mobile application interface titled "View Students". At the top, there is a search bar with the placeholder "Course ID" and a value "cs110" entered. Below the search bar is a "CLEAR" button. A table below the search bar displays student information with columns: Roll Number, Name, Course Code, Last Appear, and Box Colour. One row in the table is highlighted, showing the details: Roll Number 160101073, Name Sujoy, Course Code cs110, Last Appear 13:13:52, and Box Colour 4.

(a) Student List

The screenshot shows a mobile application interface titled "Delete Student". At the top, there is a search bar with the placeholder "Roll ID" and a value "160101073" entered. Below the search bar is a "Data Deleted" button. The screen displays a confirmation message "Data Deleted" in a dark blue rounded rectangle at the bottom center.

(b) Output

**Output:** Displays "Data Deleted". We get our desired output.

## 4.8 Module name: Camera Session and Face Recognition

- **Equivalence Classes:**

1. **Input:** State of student < 5.

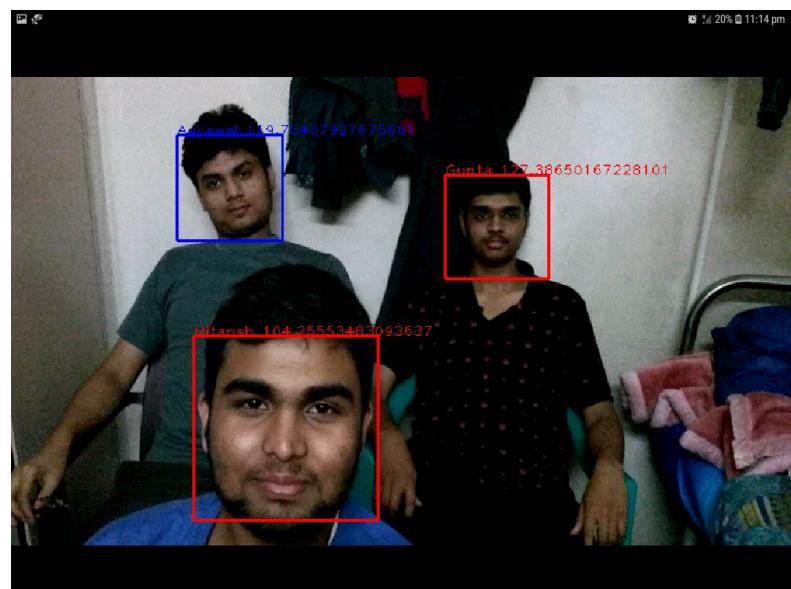
**Expected Output:** Bounding box color: red



Course ID

**CLEAR**

Roll Number	Name	Course Code	Last Appear	Box Colour
160101031	Gupta	cs243	23:13:49	1
160101032	Sharma	cs243	23:11:16	9
160101042	Mitansh	cs243	23:14:11	4
160101030	Agrawal	cs243	23:14:11	5



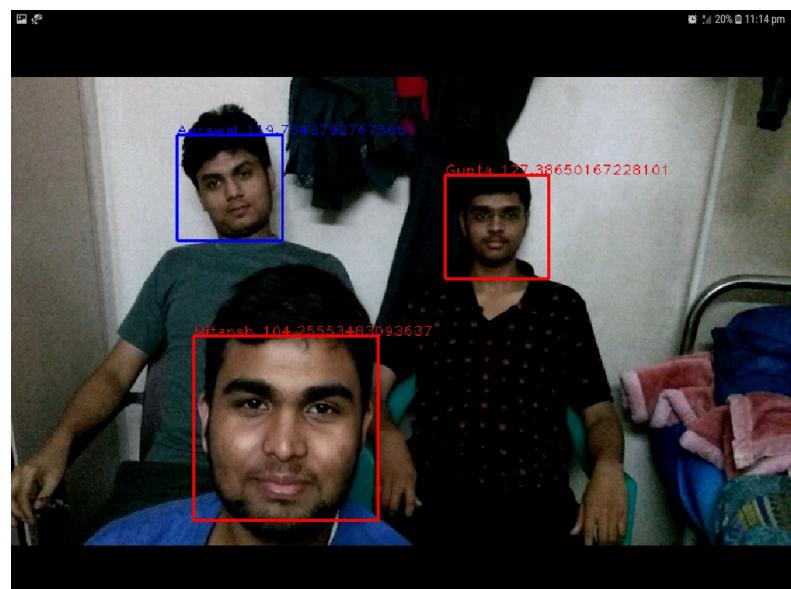
**Output:** Displays person recognised with corresponding bounding box. We get our desired output.

2. **Input:**  $5 < \text{State of student} < 8$

**Expected Output:** Bounding box color: blue



Roll Number	Name	Course Code	Last Appear	Box Colour
160101031	Gupta	cs243	23:13:49	1
160101032	Sharma	cs243	23:11:16	9
160101042	Mitansh	cs243	23:14:11	4
160101030	Agrawal	cs243	23:14:11	5



**Output:** Displays person recognised with corresponding bounding box. We get our desired output.

3. **Input:** State of student > 8

**Expected Output:** Bounding box color: green

The screenshot shows a mobile application interface. At the top, it says "View Students". Below that is a search bar with "Course ID" and "cs243" entered. There is a "CLEAR" button next to the search bar. A table below the search bar displays student information:

Roll Number	Name	Course Code	Last Appear	Box Colour
160101031	Gupta	cs243	23:11:26	4
160101032	Sharma	cs243	23:11:16	9
160101042	Mitansh	cs243	23:11:22	9
160101030	Agrawal	cs243	23:11:17	9



**Output:** Displays person recognised with corresponding bounding box. We get our desired output.

4. **Input:** More than one known students(1 to 5) are standing  
**Expected Output:** Students should be correctly identified.

View Students										
Course ID	cs243									
<input type="button" value="CLEAR"/>										
<hr/>										
Roll Number	Name	Course Code	Last Appear	Box Colour						
160101031	Gupta	cs243	23:13:49	1						
160101032	Sharma	cs243	23:11:16	9						
160101042	Mitansh	cs243	23:14:11	4						
160101030	Agrawal	cs243	23:14:11	5						

Figure 4.22: List of students in class

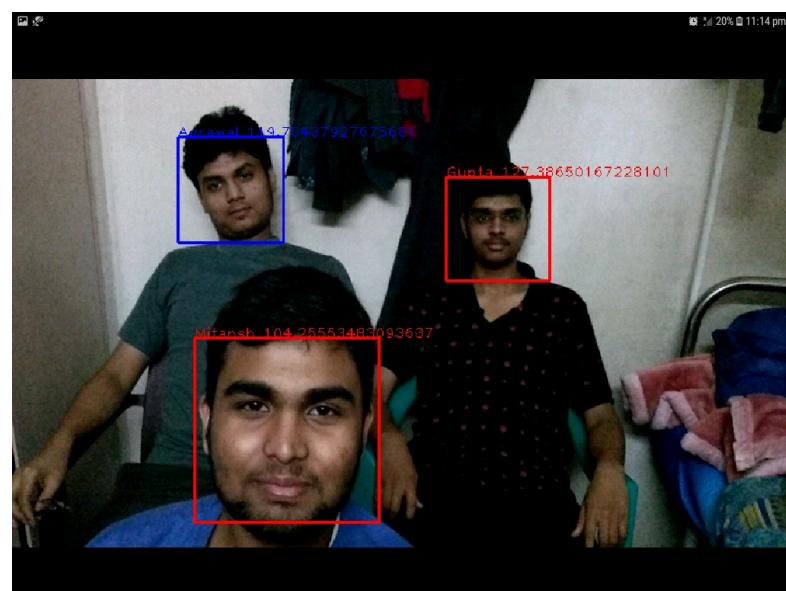


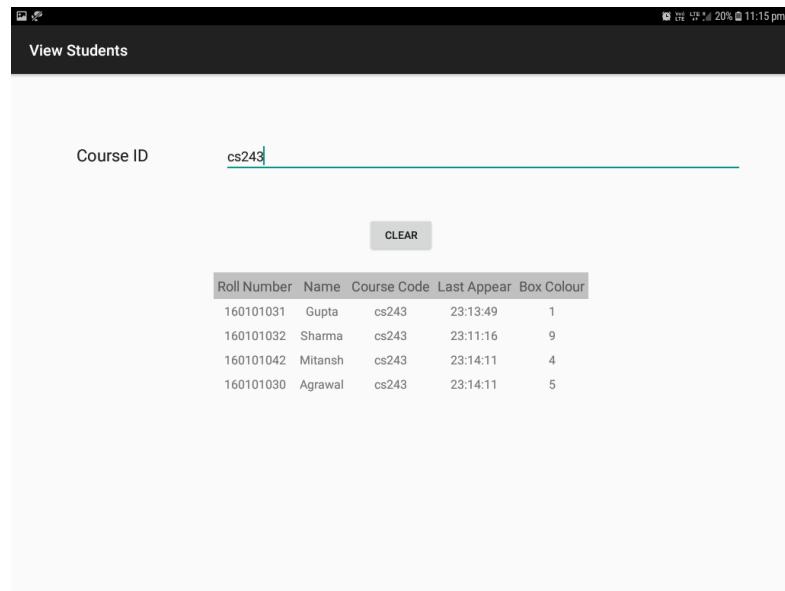
Figure 4.23: Students identified correctly

**Output:** Displays correctly recognised person with corresponding bounding box. We get our desired output.

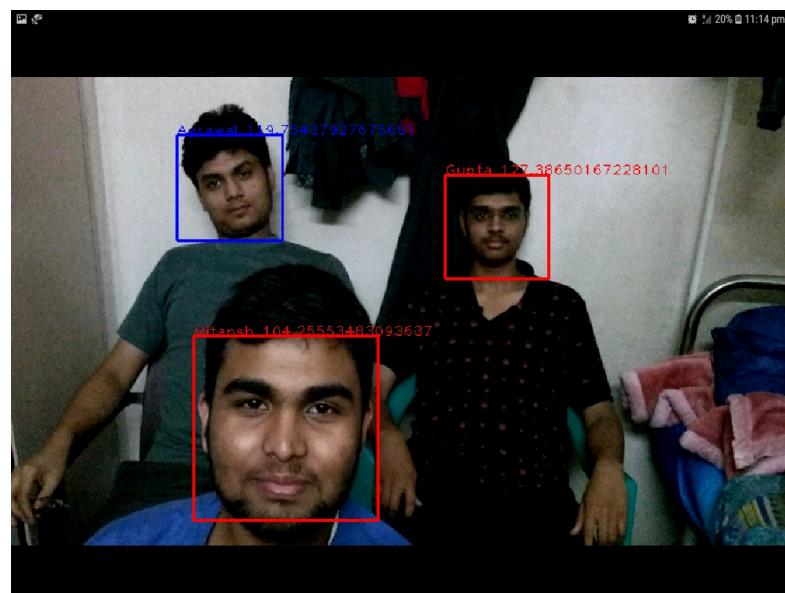
- **Boundary Cases:**

1. **Input:** State of student = 5.

**Expected Output:** Bounding box color: blue



Roll Number	Name	Course Code	Last Appear	Box Colour
160101031	Gupta	cs243	23:13:49	1
160101032	Sharma	cs243	23:11:16	9
160101042	Mitansh	cs243	23:14:11	4
160101030	Agrawal	cs243	23:14:11	5



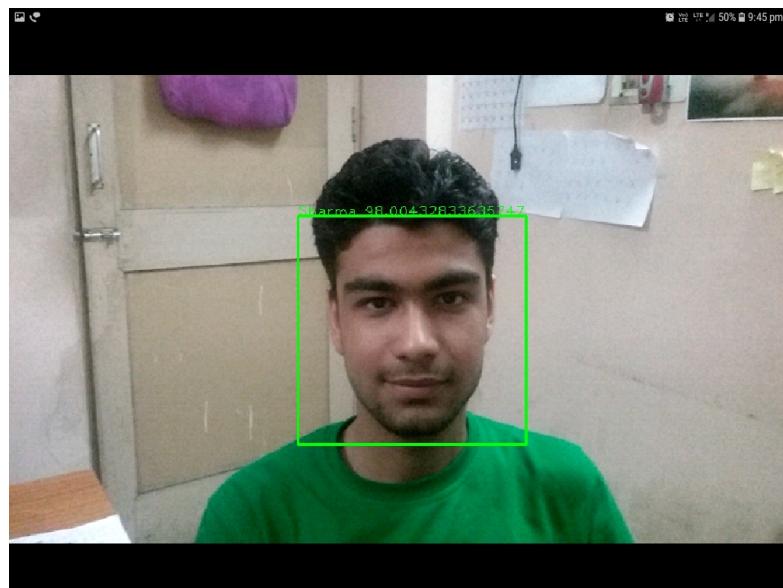
**Output:** Displays correct bounding box color for state=5. We get our desired output.

2. **Input:** State of student = 8

**Expected Output:** Bounding box color: green

The screenshot shows a software window titled "View Students". At the top, there is a search bar with the text "Course ID" and "cs243" entered. Below the search bar is a "CLEAR" button. A table displays student information:

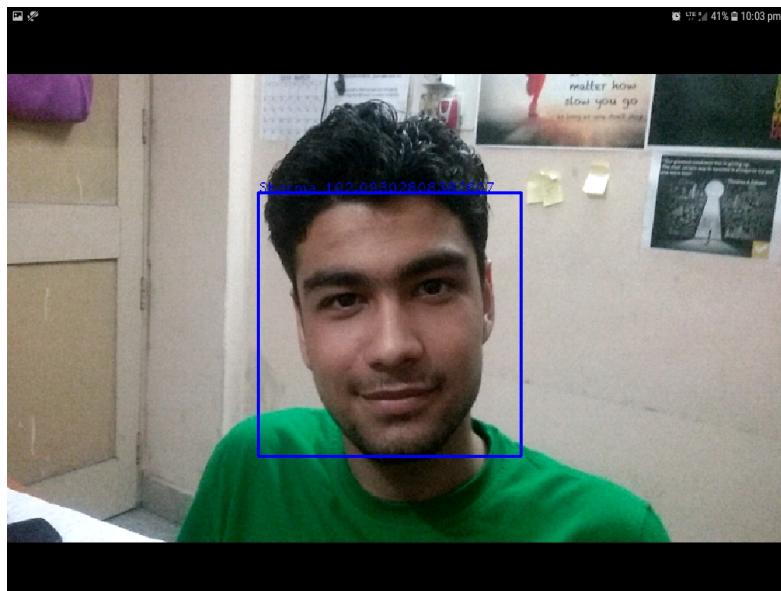
Roll Number	Name	Course Code	Last Appear	Box Colour
160101031	Gupta	cs243	21:56:51	9
160101030	Agrawal	cs243	21:56:45	8
160101032	Sharma	cs243	21:56:46	8



**Output:** Displays correct bounding box color for state=8. We get our desired output.

3. **Input:** Number of known students = 1

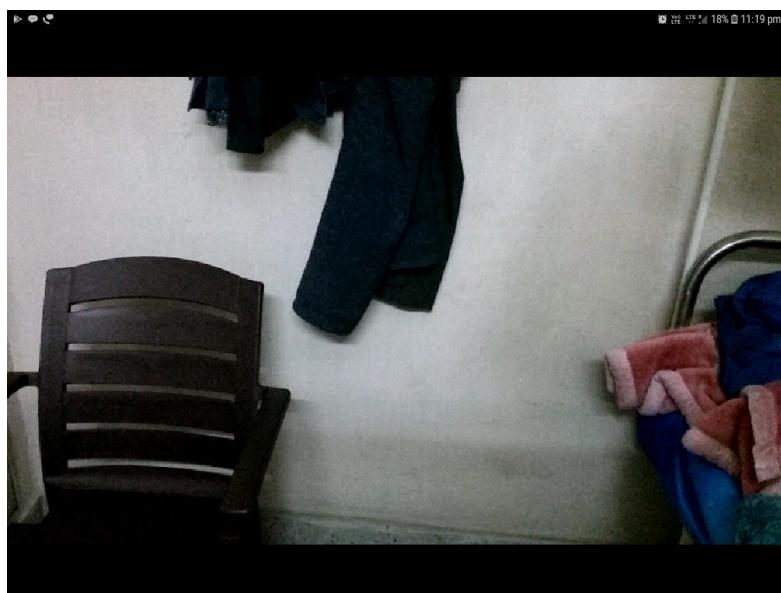
**Expected Output:** Student should be correctly identified.



**Output:** Displays correctly identified person with its bounding box. We get our desired output.

4. **Input:** Number of known students = 0

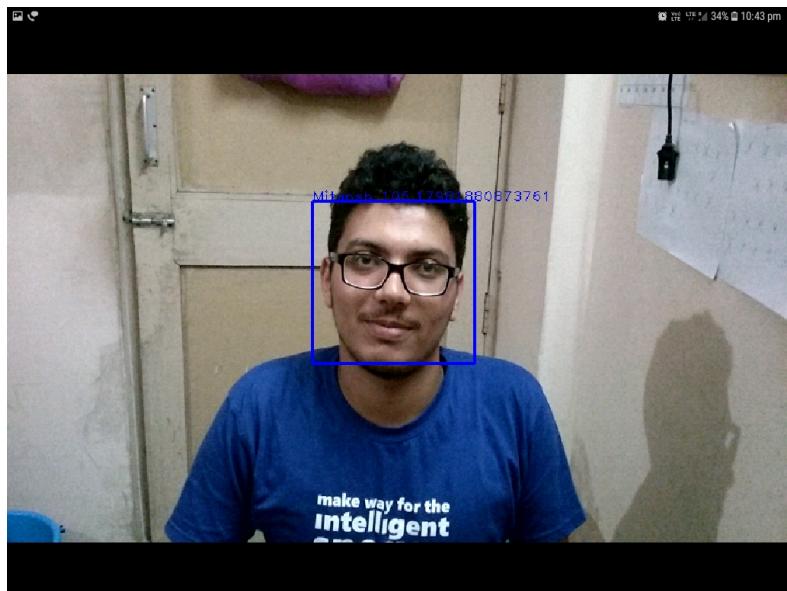
**Expected Output:** Nothing should be shown.



**Output:** Does not display any recognised person name. We get our desired output.

5. **Input:** Number of unknown students = 1

**Expected Output:** Student should be marked with unknown.



**Output:** Recognises the person to be "Mitansh" although he is an unknown student. It may be due to light conditions and face orientation as openCV is very sensitive to these aspects. If any unknown person comes, it compares with all persons in database and finds nearest possible match.

# 5 White Box Testing

## 5.1 Module: Sign Up

### 5.1.1 Function: signup()

```
1 public void signup() {
2     EditTextEmptyHolder = false;
3     if (!validate()) {
4         onSignupFailed();
5         return;
6     }
7     EditTextEmptyHolder = true;
8     _signupButton.setEnabled(false);
9     final ProgressDialog progressDialog = new ProgressDialog(SignupActivity.this,
10             R.style.AppTheme_Dark_Dialog);
11     progressDialog.setIndeterminate(true);
12     progressDialog.setMessage("Creating Account...");
13     progressDialog.show();
14     name = _nameText.getText().toString();
15     email = _emailText.getText().toString();
16     password = _passwordText.getText().toString();
17     reEnterPassword = _reEnterPasswordText.getText().toString();
18     SQLiteDataBaseBuild();
19     SQLiteTableBuild();
20     EmptyEditTextAfterDataInsert();
21     new android.os.Handler().postDelayed(
22         new Runnable() {
23             public void run() {
24                 CheckingEmailAlreadyExistsOrNot();
25             }
26         }, 3000);
27     });
28 }
29 }
```

Figure 5.1: Code for signup() function

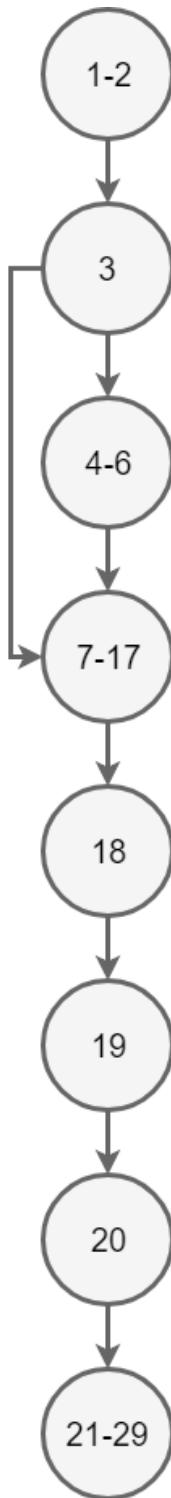


Figure 5.2: CFG for `signup()` function

### 5.1.1.1 Calculation of Linearly Independent Paths

Maximum number of Linearly independent paths = Number of Bounded Regions  
+ 1 = 2

### 5.1.1.2 Linearly Independent Paths

- (1-2)->3->(4-6)

**Testcase:** validate = False

**Expected Output:** Call onSignupFailed() and thus display "signup failed" message.

**Observed Output:** Displays "signup failed" message.

- (1-2)->3->(7-17)->(18-29)

**Testcase:** validate = True

**Expected Output:** Call database function to build database, table and insert data.

**Observed Output:** Data is inserted according to code.

### 5.1.2 Function: CheckEmailAlreadyExists()

```
1 public void CheckingEmailAlreadyExistsOrNot(){
2     boolean ifEmailExists = sqliteHelper.ifEmailExists(email);
3     if(ifEmailExists){
4         F_Result = "Email Found";
5     }
6     CheckFinalResult();
7 }
```

Figure 5.3: Code for CheckEmailAlreadyExists() function

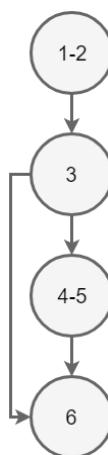


Figure 5.4: CFG for CheckEmailAlreadyExists() function

### 5.1.2.1 Calculation of Linearly Independent Paths

Maximum number of Linearly independent paths = Number of Bounded Regions  
+ 1 = 2

### 5.1.2.2 Linearly Independent Paths

- (1-2)->3->(4-6)

**Testcase:** ifEmailExists is False(i.e. if email does not exists in database)

**Expected Output:** F\_RESULT = "Not Found"

**Observed Output:** F\_RESULT = "Not Found"

- (1-2)->3->(7-17)->(18-29)

**Testcase:** ifEmailExists is True(i.e. if email exists in database)

**Expected Output:** F\_RESULT = "Email Found"

**Observed Output:** F\_RESULT = "Email Found"

### 5.1.3 Function: CheckFinalResult()

```
1 public void CheckFinalResult(){
2     if(F_Result.equalsIgnoreCase("Email Found")) {
3         Toast.makeText(SignupActivity.this,"Email Already Exists",Toast.LENGTH_LONG).show();
4         onSignupFailed();
5     } else {
6         boolean val = sqliteHelper.insertData(name, email, password);
7         if(val) {
8             onSignupSuccess();
9         } else {
10            onSignupFailed();
11        }
12    }
13    F_Result = "Not_Found" ;
14 }
```

Figure 5.5: Code for CheckFinalResult() function

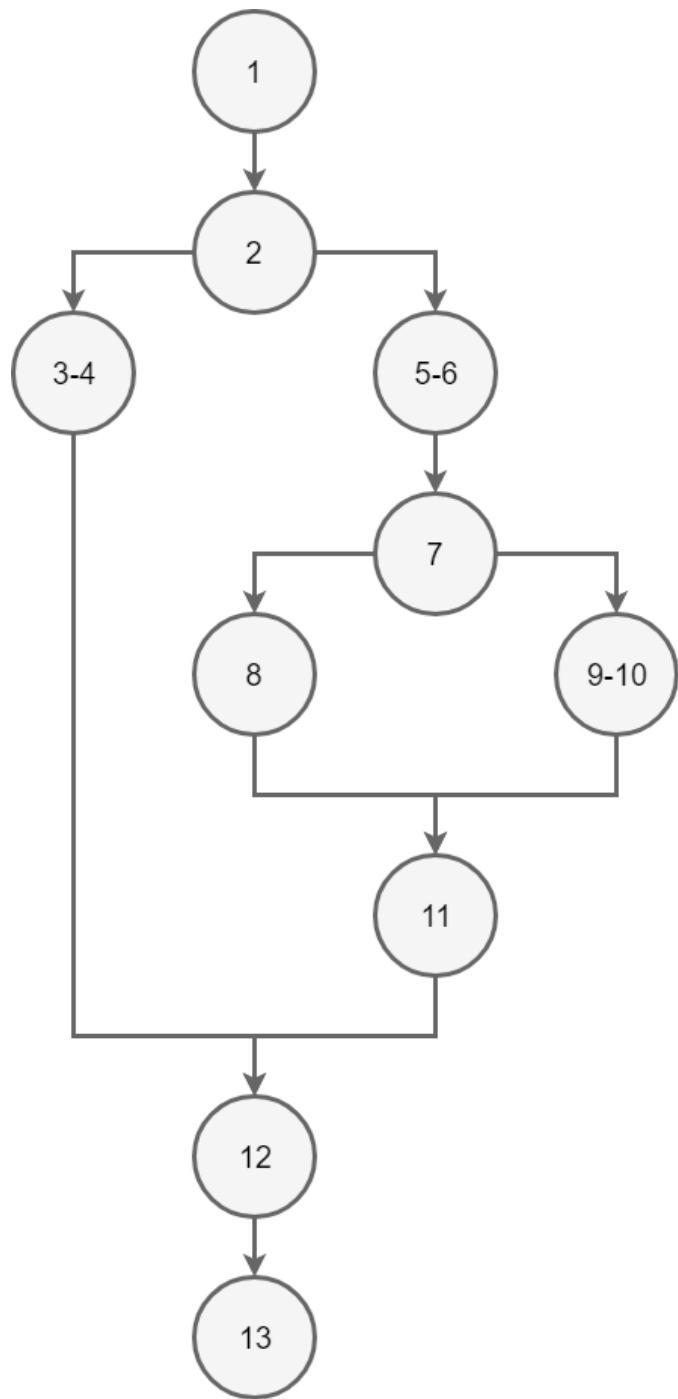


Figure 5.6: CFG for `checkFinalResult()` function

### 5.1.3.1 Calculation of Linearly Independent Paths

Maximum number of Linearly independent paths = Number of Bounded Regions  
+ 1 = 3

### 5.1.3.2 Linearly Independent Paths

- 1->2->(3-4)->12->13

**Testcase:** F\_RESULT = "Email Found"

**Expected Output:** Toast.text = "Email Already Exists"

**Observed Output:** F\_RESULT = "Email Already Exists"

- 1->2->(5-6)->7->8->11->12->13

**Testcase:** F\_RESULT = "Not Found" and val = False

**Expected Output:** Call onSignupFailed()

**Observed Output:** Calls onSignupFailed()

- 1->2->(5-6)->7->(9-10)->11->12->13

**Testcase:** F\_RESULT = "Not Found" and val = False

**Expected Output:** Call onSignupSuccess()

**Observed Output:** Call onSignupSuccess()

### 5.1.4 Function: validate()

```
1 public boolean validate() {
2     boolean valid = true;
3     String name = _nameText.getText().toString();
4     String email = _emailText.getText().toString();
5     String password = _passwordText.getText().toString();
6     String reEnterPassword = _reEnterPasswordText.getText().toString();
7     if (name.isEmpty() || name.length() < 3) {
8         _nameText.setError("at least 3 characters");
9         valid = false;
10    } else {
11        _nameText.setError(null);
12    }
13    if (email.isEmpty() || !android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
14        _emailText.setError("enter a valid email address");
15        valid = false;
16    } else {
17        _emailText.setError(null);
18    }
19    if (password.isEmpty() || password.length() < 4 || password.length() > 10) {
20        _passwordText.setError("between 4 and 10 alphanumeric characters");
21        valid = false;
22    } else {
23        _passwordText.setError(null);
24    }
25    if (reEnterPassword.isEmpty() || reEnterPassword.length() < 4 || reEnterPassword.length() > 10 || !(reEnterPassword.equals(password))) {
26        _reEnterPasswordText.setError("Passwords do not match");
27        valid = false;
28    } else {
29        _reEnterPasswordText.setError(null);
30    }
31    return valid;
32 }
```

Figure 5.7: Code for validate() function

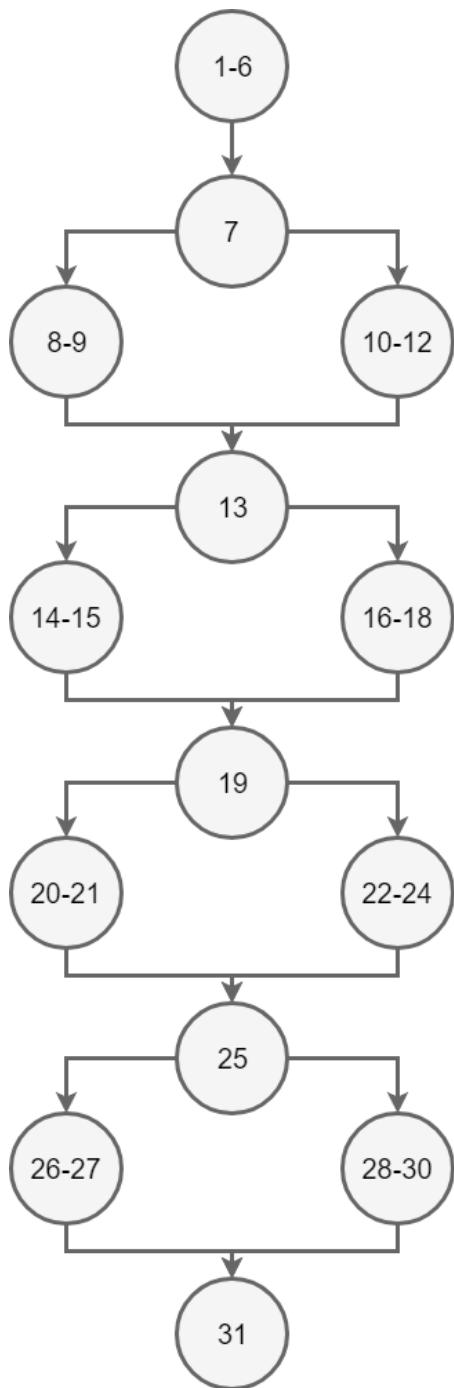


Figure 5.8: CFG for validate() function

#### **5.1.4.1 Calculation of Linearly Independent Paths**

**Maximum number of Linearly independent paths** = Number of Bounded Regions  
+ 1 = 5

#### **5.1.4.2 Linearly Independent Paths**

- (1-6)->7->(8-9)->13->(16-18)->19->(22-24)->25->(28-30)->31

**Testcase:** name = "", email = "a@gmail.com", password="abcde", reEnterPassword="abcde"

**Expected Output:** valid = false

**Observed Output:** valid = false

- (1-6)->7->(10-12)->13->(14-15)->19->(22-24)->25->(28-30)->31

**Testcase:** name = "abcde", email = "", password="abcde", reEnterPassword="abcde"

**Expected Output:** valid = false

**Observed Output:** valid = false

- (1-6)->7->(10-12)->13->(16-18)->19->(20-22)->25->(28-30)->31

**Testcase:** name = "abcde", email = "a@gmail.com", password="", reEnterPassword="abcde"

**Expected Output:** valid = false

**Observed Output:** valid = false

- (1-6)->7->(10-12)->13->(16-18)->19->(22-24)->25->(26-27)->31

**Testcase:** name = "abcde", email = "a@gmail.com", password="abcde", reEnterPassword="abcdefg"

**Expected Output:** valid = false

**Observed Output:** valid = false

- (1-6)->7->(10-12)->13->(16-18)->19->(22-24)->25->(28-30)->31

**Testcase:** name="abcde", email="a@gmail.com", password="abcde", reEnterPassword="abcde"

**Expected Output:** valid = true

**Observed Output:** valid = true

## 5.2 Module: AddStudentRecord

### 5.2.1 Function: AddData()

```
1 public void AddData() {
2     btnAddData.setOnClickListener(
3         new View.OnClickListener() {
4             @Override
5             public void onClick(View v) {
6                 boolean isInteger = false;
7                 String courseID = editCourse.getText().toString();
8                 String name = editName.getText().toString();
9                 String surname = editSurname.getText().toString();
10                String rollID = editTextId.getText().toString();
11                int roll_number=0;
12                try{
13                    roll_number = Integer.parseInt(editTextId.getText().toString());
14                    if (roll_number > 0) {
15                        isInteger = true;
16                    }
17                } catch ( NumberFormatException e ) {
18                    isInteger = false;
19                }
20                if (courseID.isEmpty() || surname.isEmpty() || name.isEmpty() || rollID.isEmpty()) {
21                    Toast.makeText(AddstudentActivity.this, "Field(s) is empty", Toast.LENGTH_LONG).show();
22                } else if(isInteger) {
23                    if(myDb.ifStudentExists(roll_number, courseID) == Boolean.FALSE) {
24                        boolean isInserted = myDb.insertdata( roll_number,
25                                editName.getText().toString(),
26                                editSurname.getText().toString(),
27                                editCourse.getText() );
28                        if(isInserted == true) {
29                            Toast.makeText(AddstudentActivity.this, "Data Inserted", Toast.LENGTH_LONG).show();
30                            Intent intent = new Intent(AddstudentActivity.this, AddImages.class);
31                            intent.putExtra("id", editTextId.getText().toString());
32                            startActivity(intent);
33                            finish();
34                        } else {
35                            Toast.makeText(AddstudentActivity.this,"Data not Inserted",Toast.LENGTH_LONG).show();
36                        }
37                    } else {
38                        Toast.makeText(AddstudentActivity.this, "Roll Number is not integer", Toast.LENGTH_LONG).show();
39                    }
40                } else {
41                    Toast.makeText(AddstudentActivity.this,"Roll Number is not integer",Toast.LENGTH_LONG).show();
42                }
43            }
44        });
45    });
46 }
47 }
48 }
```

Figure 5.9: Code for addData() function

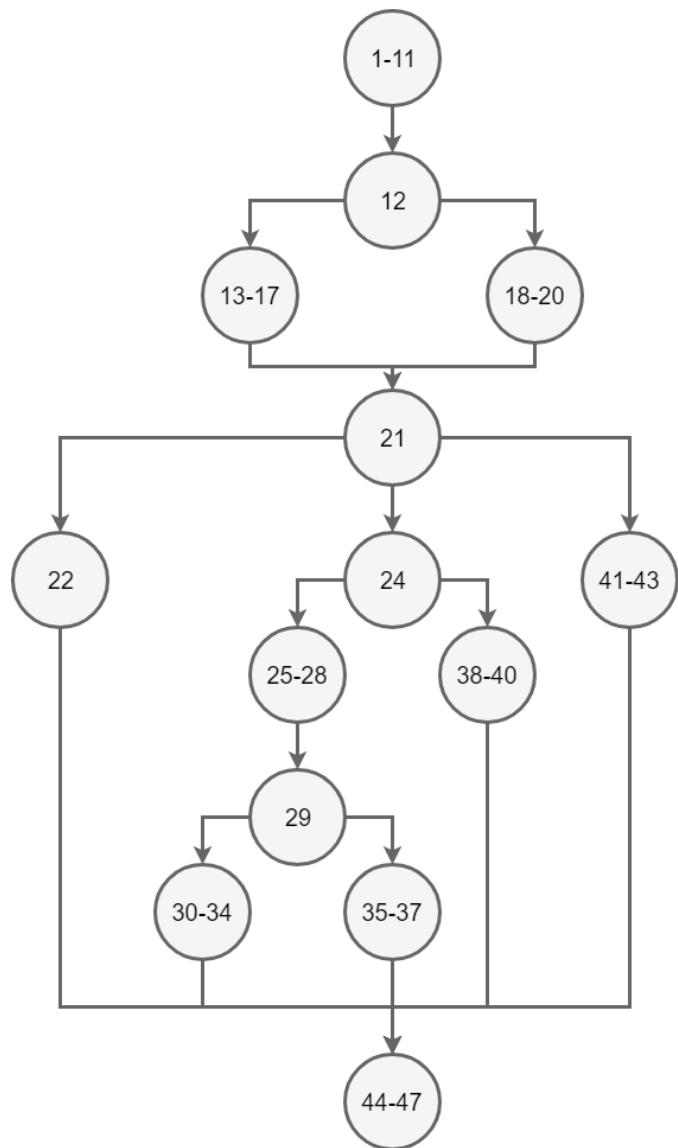


Figure 5.10: CFG for `addData()` function

### 5.2.1.1 Calculation of Linearly Independent Paths

Maximum number of Linearly independent paths = Number of Bounded Regions  
+ 1 = 6

### 5.2.1.2 Linearly Independent Paths

- (1-11)->12->(13-17)->21->22->(44-47)

**Testcase:** courseID=""", name="abc", surname="def", rollID=123

**Expected Output:** Toast.text = "Field is Empty"

**Observed Output:** Toast.text = "Field is Empty"

- (1-11)->12->(13-17)->21->24->(25-26)->29->(30-34)->(44-47)

**Testcase:** courseID="cs243", name="abc", surname="def", rollID=123,  
isInserted=true

**Expected Output:** Toast.text = "Data inserted"

**Observed Output:** Toast.text = "Data inserted"

- (1-11)->12->(13-17)->21->24->(25-26)->29->(35-37)->(44-47)

**Testcase:** courseID="cs243", name="abc", surname="def", rollID=123,  
isInserted=false

**Expected Output:** Toast.text = "Data not inserted"

**Observed Output:** Toast.text = "Data not inserted"

- (1-11)->12->(13-17)->21->24->(38-40)->(44-47)

**Testcase:** courseID="cs243", name="abc", surname="def", rollID=123,  
isInserted=false

**Expected Output:** Toast.text = "Data not inserted"

**Observed Output:** Toast.text = "Data not inserted"

- (1-11)->12->(18-20)->21->(41-43)->(44-47)

**Testcase:** courseID="cs243", name="abc", surname="def", rollID="123",  
isInserted=false

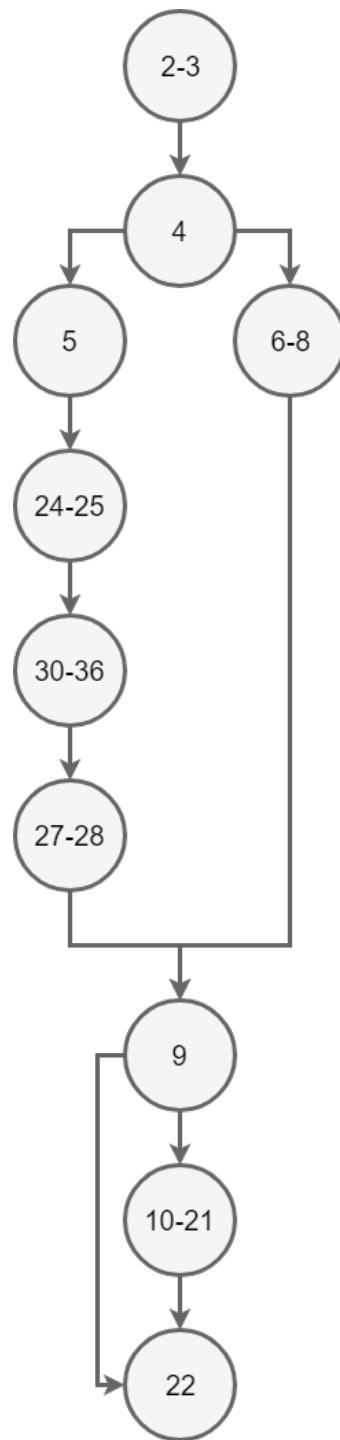
**Expected Output:** Toast.text = "Roll Number is not Integer"

**Observed Output:** Toast.text = "Roll Number is not Integer"

## 5.3 Module: AddImages

### 5.3.1 Functions: onResume(), addImage(), btnTakePhotoClicker.OnClickListener()

```
1 private static final int IMG_COUNT = 15;
2 public void onResume() {
3     super.onResume();
4     if (tempCounter != IMG_COUNT){
5         addImage();
6     } else {
7         button.setText("Return to Main Menu");
8     }
9     if (tempCounter == IMG_COUNT) {
10         button.setOnClickListener(
11             new View.OnClickListener() {
12                 @Override
13                 public void onClick(View v) {
14                     Intent nextPage = new Intent(AddImages.this,
15                         MainActivity.class);
16                     startActivity(nextPage);
17                     finish();
18                 }
19             });
20     }
21 }
22 |
23 public void addImage() {
24     button.setOnClickListener(
25         new AddImages.btnTakePhotoClicker()
26     );
27 }
28
29 class btnTakePhotoClicker implements Button.OnClickListener {
30     @Override
31     public void onClick(View view) {
32         Intent isPhotoAcceptable = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
33         startActivityForResult(isPhotoAcceptable, CAM_REQUEST);
34     }
35 }
```



### 5.3.1.1 Calculation of Linearly Independent Paths

Maximum number of Linearly independent paths = 3

### 5.3.1.2 Linearly Independent Paths

- (2-3)->4->(5)->(24-25)->(30-36)->(27-28)->9->22

**Testcase:** tempCounter = 5

**Expected Output:** Call addImage()

**Observed Output:** Calls addImage()

- (2-3)->4->(6-8)->9->(10-21)->22

**Testcase:** tempCounter = 15

**Expected Output:** Set button.text = "Return to Main Menu"

**Observed Output:** Set button.text = "Return to Main Menu"

## 5.4 Module: DeleteStudent

### 5.4.1 Function: deleteStudent()

```
1  public void DeleteData() {
2      btnDelete.setOnClickListener(
3          new View.OnClickListener() {
4              @Override
5              public void onClick(View v) {
6                  boolean isInteger = false;
7                  String rollID = editTextId.getText().toString();
8                  int roll_number=0;
9                  try
10                 {
11                     roll_number = Integer.parseInt(editTextId.getText().toString());
12                     if (roll_number > 0) {
13                         isInteger = true;
14                     }
15                 } catch ( NumberFormatException e ) {
16                     isInteger = false;
17                 }
18                 if (rollID.isEmpty()) {
19                     Toast.makeText(DeleteStudentActivity.this, "Field(s) is empty", Toast.LENGTH_LONG).show();
20                 } else if(isInteger) {
21                     Integer deletedRows = myDb.deleteData(roll_number);
22                     Integer deletedImg = imDb.deleteImg(roll_number);
23                     if (deletedRows > 0 && deletedImg > 0){
24                         Toast.makeText(DeleteStudentActivity.this, "Data Deleted", Toast.LENGTH_LONG).show();
25                     } else{
26                         Toast.makeText(DeleteStudentActivity.this, "Roll ID not present in database", Toast.LENGTH_LONG).show();
27                     }
28                 } else {
29                     Toast.makeText(DeleteStudentActivity.this,"Entered Roll ID is not integer",Toast.LENGTH_LONG).show();
30                 }
31             }
32         );
33     });
34 }
```

Figure 5.11: Code for deleteStudent() function

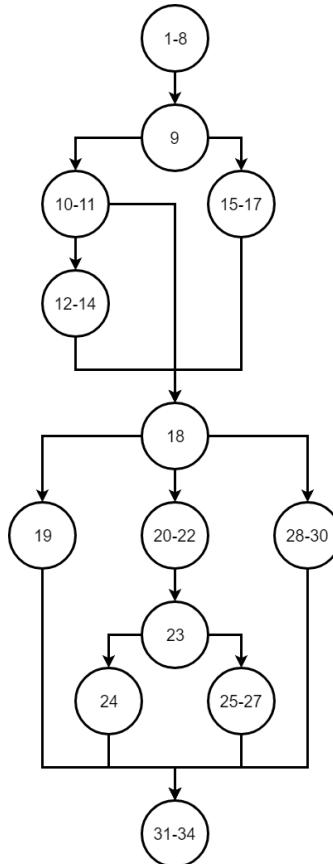


Figure 5.12: CFG for deleteStudent() function

#### 5.4.1.1 Calculation of Linearly Independent Paths

**Maximum number of Linearly independent paths** = Number of Bounded Regions + 1 = 6

#### 5.4.1.2 Linearly Independent Paths

- (1-8)->9->(10-11)->(12-14)->18->19->(31-34)

**Testcase:** roll\_number = ""

**Expected Output:** Toast.text = "Field(s) is Empty"

**Observed Output:** Toast.text = "Field(s) is Empty"

- (1-8)->9->(10-11)->(12-14)->18->(20-22)->23->24->(31-34)

**Testcase:** roll\_number = 160101042

**Expected Output:** Toast.text = "Data is Deleted"

**Observed Output:** Toast.text = "Data is Deleted"

- (1-8)->9->(10-11)->(12-14)->18->(20-22)->23->(25-27)->(31-34)

**Testcase:** roll\_number = 160101042

**Expected Output:** Toast.text = "Roll ID is not present in database"

**Observed Output:** Toast.text = "Roll ID is not present in database"

- (1-8)->9->(15-17)->18->(28-30)->(31-34)

**Testcase:** roll\_number = "abcde"

**Expected Output:** Toast.text = "Entered Roll ID not integer"

**Observed Output:** Toast.text = "Entered Roll ID not integer"

- (1-8)->9->(10-11)->18->(28-30)->(31-34)

**Testcase:** roll\_number = "abcde"

**Expected Output:** Toast.text = "Entered Roll ID not integer"

**Observed Output:** Toast.text = "Entered Roll ID not integer"

## 5.5 Module: EditStudent

### 5.5.1 Function: UpdateData()

```

1  public void UpdateData() {
2      btnviewupdate.setOnClickListener(
3          new View.OnClickListener() {
4              @Override
5              public void onClick(View v) {
6                  boolean isinteger = false;
7                  int roll_number=0;
8                  try {
9                      roll_number = Integer.parseInt(editTextid.getText().toString());
10                     isinteger = true;
11                 } catch ( NumberFormatException e ) {
12                     isinteger = false;
13                 }
14                 if(isInteger) {
15                     boolean isupdate = myDb.updateData(roll_number,
16                         editname.getText().toString(),
17                         editsurname.getText().toString(), editcourse.getText().toString());
18                     if (isupdate == true) {
19                         Toast.makeText(EditstudentActivity.this, "Data Updated", Toast.LENGTH_LONG).show();
20                     } else {
21                         Toast.makeText(EditstudentActivity.this, "Data not updated", Toast.LENGTH_LONG).show();
22                     }
23                 } else {
24                     Toast.makeText(EditstudentActivity.this,"Roll Number is not integer",Toast.LENGTH_LONG).show();
25                 }
26             }
27         );
28     });
29 }
30 }
```

Figure 5.13: Code for UpdateData() function

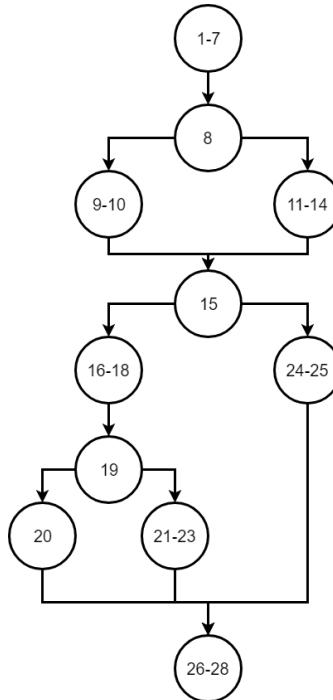


Figure 5.14: CFG for UpdateData() function

#### 5.5.1.1 Calculation of Linearly Independent Paths

**Maximum number of Linearly independent paths** = Number of Bounded Regions + 1 = 4

#### 5.5.1.2 Linearly Independent Paths

- (1-7)->8->(9-10)->(16-18)->19->20->(20-28)

**Testcase:** roll\_number=160101085

**Expected Output:** Toast.text = "Data Updated"

**Observed Output:** Toast.text = "Data Updated"

- (1-7)->8->(9-10)->(16-18)->19->(21-23)->(26-28)

**Testcase:** roll\_number=123

**Expected Output:** Toast.text = "Data not Updated"

**Observed Output:** Toast.text = "Data not Updated"

- (1-7)->8->(11-14)->15->(24-25)->(26-28)

**Testcase:** roll\_number="abcde"

**Expected Output:** Toast.text = "Rol Number is not integer"

**Observed Output:** Toast.text = "Rol Number is not integer"

### 5.5.2 Function: UpdateImages()

```
1 public void UpdateImage() {
2     button.setOnClickListener(
3         new View.OnClickListener() {
4             @Override
5             public void onClick(View v) {
6                 boolean isInteger = false;
7                 String courseID = editCourse.getText().toString();
8                 String name = editName.getText().toString();
9                 String surname = editSurname.getText().toString();
10                String rollID = editTextId.getText().toString();
11                int roll_number=0;
12                try {
13                    roll_number = Integer.parseInt(editTextId.getText().toString());
14                    isInteger = true;
15                } catch ( NumberFormatException e ) {
16                    isInteger = false;
17                }
18                if (courseID.isEmpty() || surname.isEmpty() || name.isEmpty() || rollID.isEmpty()) {
19                    Toast.makeText(EditStudentActivity.this, "Field(s) is empty", Toast.LENGTH_LONG).show();
20                } else if(isInteger) {
21                    if ( myDb.ifStudentExists(roll_number, courseID) ) {
22                        Integer deletedImg = imDb.deleteImg(roll_number);
23                        if (deletedImg > 0) {
24                            Toast.makeText(EditStudentActivity.this, "Images Deleted", Toast.LENGTH_LONG).show();
25                            Intent intent = new Intent(EditStudentActivity.this, AddImages.class);
26                            intent.putExtra("id", editTextId.getText().toString());
27                            startActivity(intent);
28                            finish();
29                        } else {
30                            Toast.makeText(EditStudentActivity.this, "Roll ID not present in database", Toast.LENGTH_LONG).show();
31                        }
32                    } else {
33                        Toast.makeText(EditStudentActivity.this, "Roll ID with given course not present", Toast.LENGTH_LONG).show();
34                    }
35                } else {
36                    Toast.makeText(EditStudentActivity.this, "Entered Roll ID is not integer",Toast.LENGTH_LONG).show();
37                }
38            }
39        });
40    });
41 }
42 }
43 );
44 );
45 };
```

Figure 5.15: Code for UpdateImages() function

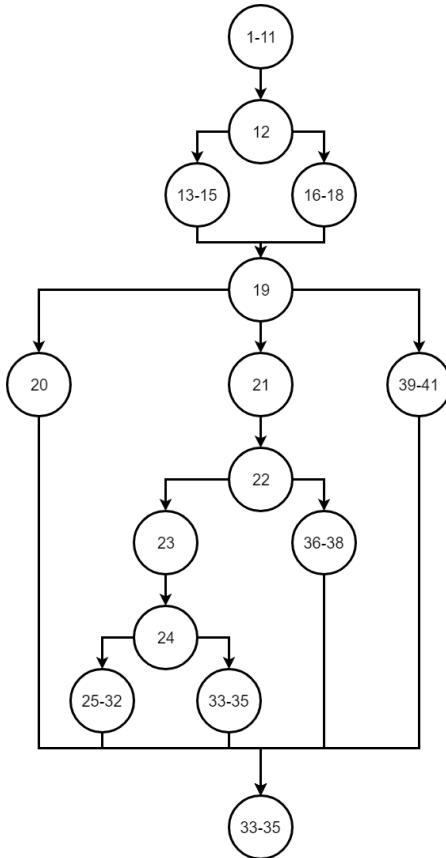


Figure 5.16: CFG for UpdateImages() function

#### 5.5.2.1 Calculation of Linearly Independent Paths

**Maximum number of Linearly independent paths** = Number of Bounded Regions + 1 = 6

#### 5.5.2.2 Linearly Independent Paths

- (1-11)->12->(13-15)->20->(42-45)

**Testcase:** roll\_number=""

**Expected Output:** Toast.text = "Field(s) is empty"

**Observed Output:** Toast.text = "Field(s) is empty"

- (1-11)->12->(13-15)->19->21->22->23->24->(25-32)->(42-45)

**Testcase:** roll\_number=160101032

**Expected Output:** Toast.text = "Images Deleted" and prompt to click new Images

**Observed Output:** Toast.text = "Images Deleted" and prompt to click new Images

- (1-11)->12->(13-15)->19->21->22->24->23->(33-35)->(42-45)

**Testcase:** roll\_number=160101026

**Expected Output:** Toast.text="Roll ID with given course not in database"

**Observed Output:** Toast.text="Roll ID with given course not in database"

- (1-11)->12->(13-15)->20->(36-38)->(42-45)

**Testcase:** roll\_number=160101031

**Expected Output:** Toast.text="Roll ID with given course not present"

**Observed Output:** Toast.text="Roll ID with given course not present"

- (1-11)->12->(13-15)->20->(36-38)->(42-45)

**Testcase:** roll\_number="abcde"

**Expected Output:** Toast.text="Enteres Roll ID is not integer"

**Observed Output:** Toast.text="Roll ID with given course not present"

## 5.6 Module: Login

### 5.6.1 Function: login()

```
1 public void login() {
2     if (!validate()) {
3         onLoginFailed();
4         return;
5     }
6     loginButton.setEnabled(false);
7     final ProgressDialog progressDialog = new ProgressDialog(LoginActivity1.this,
8             R.style.AppTheme_Dark_Dialog);
9     progressDialog.setIndeterminate(true);
10    progressDialog.setMessage("Authenticating...");
11    progressDialog.show();
12    String email = _emailText.getText().toString();
13    final String password = _passwordText.getText().toString();
14    Cursor result = sqliteHelper.getPassword(email);
15    if(result.getCount() == 0)
16    {
17        new android.os.Handler().postDelayed(
18            new Runnable() {
19                public void run() {
20                    onLoginFailed();
21                    progressDialog.dismiss();
22                }
23            }, 3000);
24        return;
25    }
26    result.moveToFirst();
27    TempPassword = result.getString(3);
28    new android.os.Handler().postDelayed(
29        new Runnable() {
30            public void run() {
31                if(TempPassword.equals(password)) {
32                    onLoginSuccess();
33                }
34                else {
35                    onLoginFailed();
36                }
37            }
38        }, 3000);
39    }
40 }
```

Figure 5.17: Code for login() function

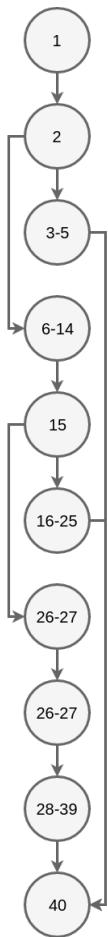


Figure 5.18: CFG for `login()` function

### 5.6.1.1 Calculation of Linearly Independent Paths

Maximum number of Linearly independent paths = Number of Bounded Regions  
+ 1 = 3

### 5.6.1.2 Linearly Independent Paths

- 1->2->(6-14)->15->(26-27)->(28-39)->40

**Testcase:** validate = false

**Expected Output:** Call onLoginFailed and set toast.text = "Login Failed"

**Observed Output:** Call onLoginFailed and set toast.text = "Login Failed"

- 1->12->(3-5)->40

**Testcase:** validate = true, result.getCount() = 0

**Expected Output:** Call onLoginFailed and set toast.text = "Login Failed"

**Observed Output:** Call onLoginFailed and set toast.text = "Login Failed"

- 1->2->(6-14)->15->(16-25)->40

**Testcase:** validate = true, result.getCount() = 0

**Expected Output:** Call onLoginSuccess and set toast.text = "Login Successful"

**Observed Output:** Call onLoginSuccess and set toast.text = "Login Successful"

### 5.6.2 Function: validate()

```
1 public boolean validate() {
2     boolean valid = true;
3     String email = _emailText.getText().toString();
4     String password = _passwordText.getText().toString();
5     if (email.isEmpty() || !android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
6         _emailText.setError("enter a valid email address");
7         valid = false;
8     } else {
9         _emailText.setError(null);
10    }
11    if (password.isEmpty() || password.length() < 4 || password.length() > 10) {
12        _passwordText.setError("between 4 and 10 alphanumeric characters");
13        valid = false;
14    } else {
15        _passwordText.setError(null);
16    }
17    return valid;
18 }
```

Figure 5.19: Code for Validate() function

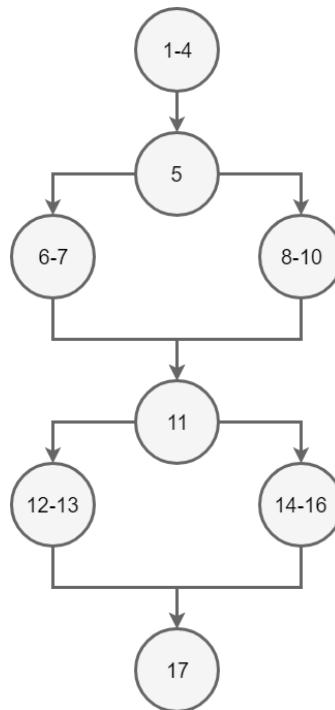


Figure 5.20: CFG for Validate() function

#### 5.6.2.1 Calculation of Linearly Independent Paths

**Maximum number of Linearly independent paths** = Number of Bounded Regions + 1 = 6

#### 5.6.2.2 Linearly Independent Paths

- (1-4)->5->(6-7)->11->(12-13)->17

**Testcase:** email="a@gmail.com", password=12345

**Expected Output:** valid=true

**Observed Output:** valid=true

- (1-4)->5->(8-10)->11->(12-13)->17

**Testcase:** email="", password=12345

**Expected Output:** \_emailText.error = "enter a valid email address"

**Observed Output:** \_emailText.error = "enter a valid email address"

- (1-4)->5->(6-7)->11->(14-16)->17

**Testcase:** email="a@gmail.com", password=""

**Expected Output:** passwordText.error = "between 4 and 10 character"

**Observed Output:** passwordText.error = "between 4 and 10 character"

## 5.7 Module: Camera session

### 5.7.1 Function: faceDetect()

```
1 public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
2     mRgba = inputFrame.rgba();
3     mGray = inputFrame.gray();
4     if(isTrainComplete){
5         training();
6     }
7     String globalTime = getTime();
8     MatOfRect faces = new MatOfRect();
9     mJavaDetector.detectMultiScale(mGray, faces, 1.1, 2, 
10             new Size(mAbsoluteFaceSize, mAbsoluteFaceSize), new Size());
11    int x=0,y=0;
12    Rect[] facesArray = faces.toArray();
13    for (int i = 0; i < facesArray.length; i++) {
14        x = facesArray[i].x;
15        y = facesArray[i].y;
16        Rect roi = new Rect(facesArray[i].tl(), facesArray[i].br());
17        Mat cropped = new Mat(mGray, roi);
18        int[] label = new int[1];
19        double[] conf = new double[1];
20        try{
21            faceRecognizer.predict(cropped, label, conf);
22        } catch (Exception e) {
23            Log.d(TAG, e.toString());
24        }
25        Random randomVal = new Random();
26        if(label[0] < 0 || conf[0] > 150.0) {
27            Imgproc.rectangle(mRgba, facesArray[i].tl(), facesArray[i].br(), new Scalar(255,255,255), 2);
28            Imgproc.putText(mRgba,"Unknown "+conf[0],new Point(x,y),Core.FONT_HERSHEY_PLAIN, 1.0, new Scalar(255,255,255));
29        } else {
30            int roll = personIndexList.get(label[0]);
31            String playerTime = lastModifiedTime.get(roll);
32            int boxColor = boundaryBoxColor.get(roll);
33            if(getTimeDiff(globalTime, playerTime) > 60000){
34                int newValue = randomVal.nextInt(9) + 1;
35                boxColor = newValue;
36                lastModifiedTime.put(roll, globalTime);
37                boundaryBoxColor.put(roll, newValue);
38            }
39            Imgproc.rectangle(mRgba, facesArray[i].tl(), facesArray[i].br(), getBoxColor(boxColor), 2);
40            Imgproc.putText(mRgba,personName.get(roll)+" "+conf[0],new Point(x,y),Core.FONT_HERSHEY_PLAIN, 1.0, getBoxColor(boxColor));
41        }
42    }
43    return mRgba;
44 }
```

Figure 5.21: Code for faceDetect() function

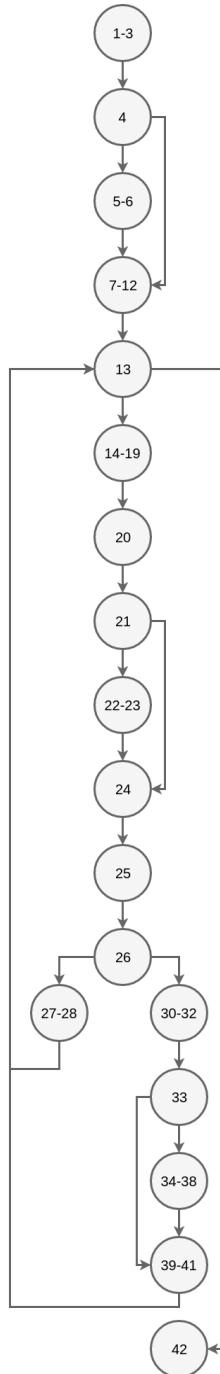


Figure 5.22: CFG for `faceDetect()` function

### 5.7.1.1 Calculation of Linearly Independent Paths

Maximum number of Linearly independent paths = Number of Bounded Regions + 1 = 6

### 5.7.1.2 Linearly Independent Paths

- (1-3)->4->(5-6)->(7-12)->(13)->(14-20)->21->24->25->26->(27-28)->13->42

**Testcase:** isTrainComplete=false, faceArray.length() < 0, conf[i] for some face is less than 150.

**Expected Output:** Assigns colour and return edited RGB frame

**Observed Output:** Assigned colour and return edited RGB frame

- (1-3)->4->(7-12)->(13)->(14-20)->21->24->25->26->(30-32)->(34-38)->(39-41)->13->42

**Testcase:** isTrainComplete=true, faceArray.length() > 0, conf[i] for some face is greater than 150 and has greater than 0 label, time difference in statement 33 > 60000

**Expected Output:** Assigns new random colour and return edited RGB frame

**Observed Output:** Assigns new random colour and return edited RGB frame

- (1-3)->4->(7-12)->(13)->(14-20)->21->24->25->26->(30-32)->(39-41)->13->42

**Testcase:** isTrainComplete=true, faceArray.length() > 0, conf[i] for some face is greater than 150 and has greater than 0 label, time difference in statement 33 < 60000

**Expected Output:** Assigns new random colour and return edited RGB frame

**Observed Output:** Assigns new random colour and return edited RGB frame

- (1-3)->4->(7-12)->(13)->(14-20)->21->(22-23)->24->25->26->(30-32)->(39-41)->13->42

**Testcase:** isTrainComplete=true, faceArray.length() > 0, conf[i] for some face is greater than 150 and has greater than 0 label, time difference in statement 33 < 60000, if app was not able to predict due to internal error(this is not an input but just covered for sake of path coverage, this will not occur if app is working properly)

**Expected Output:** Shows error in LOG, no sensible output

**Observed Output:** Can't be verified

### 5.7.2 Function: getboxColor()

```
1  private Scalar getBoxColor(int boxColor) {  
2      if(boxColor < 5){  
3          return new Scalar(255, 0, 0);  
4      } else if(boxColor < 8) {  
5          return new Scalar(0, 0, 255);  
6      } else {  
7          return new Scalar(0, 255, 0);  
8      }  
9  }
```

Figure 5.23: Code for getboxColor() function

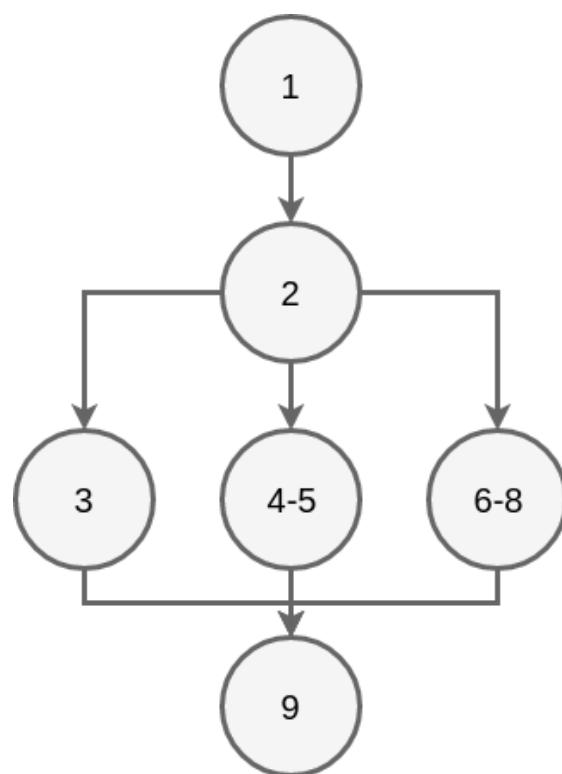


Figure 5.24: CFG for getboxColor() function

#### 5.7.2.1 Calculation of Linearly Independent Paths

Maximum number of Linearly independent paths = Number of Bounded Regions + 1 = 3

### 5.7.2.2 Linearly Independent Paths

- 1->2->3->9

**Testcase:** boxColor<5

**Expected Output:** color returned red

**Observed Output:** color returned red

- 1->2->(4-5)->9

**Testcase:** boxColor<8

**Expected Output:** color returned blue

**Observed Output:** color returned blue

- 1->2->(6-8)->9

**Testcase:** boxColor>=8

**Expected Output:** color returned green

**Observed Output:** color returned green

## **6 Conclusion**

The black-box testing and white-box testing were quite successful. In black-box testing it was found that app was able to identify people with a good success rate(80%). It was not able to label unknown people with label "unknown". Almost all the exception handling were covered. In white-box testing done by path coverage all the statement of almost all functions were tested. Expected output matched with observed one in all cases.