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

List of Figures

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 $\neq$ 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

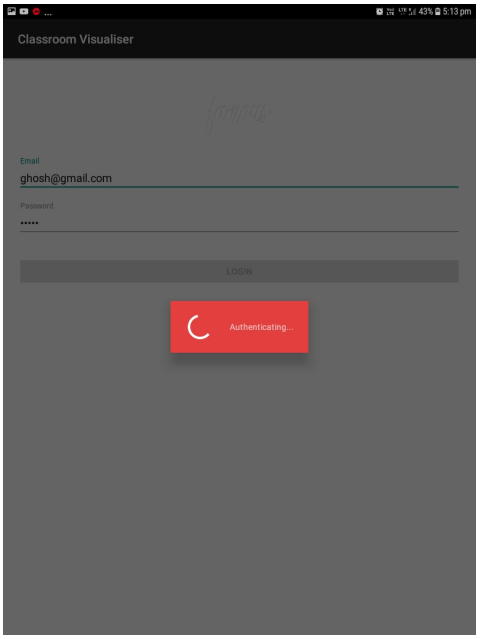
4. **Input**: More than one known students(1 to 5) are standing
   Expected Output: Students should be correctly identified.

- **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 marked with unknown.

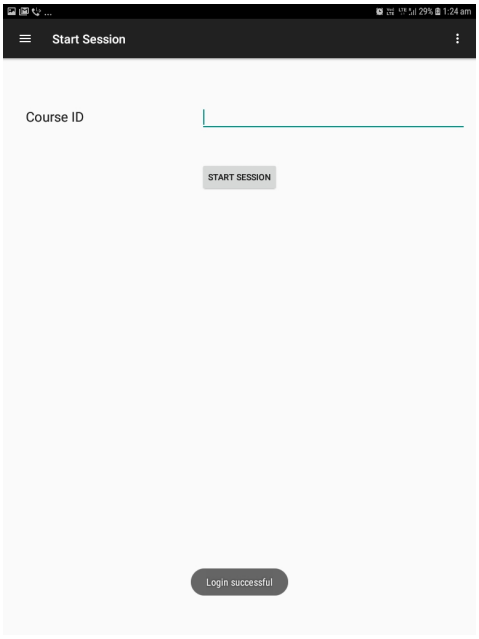
4 Blackbox Testing

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.

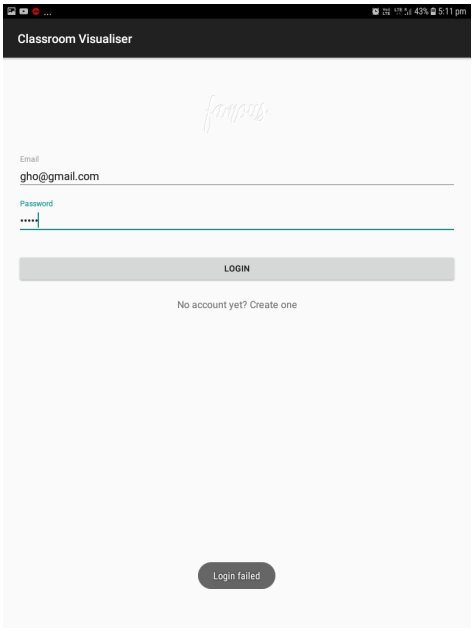


Figure 4.2: Input with wrong email id

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.

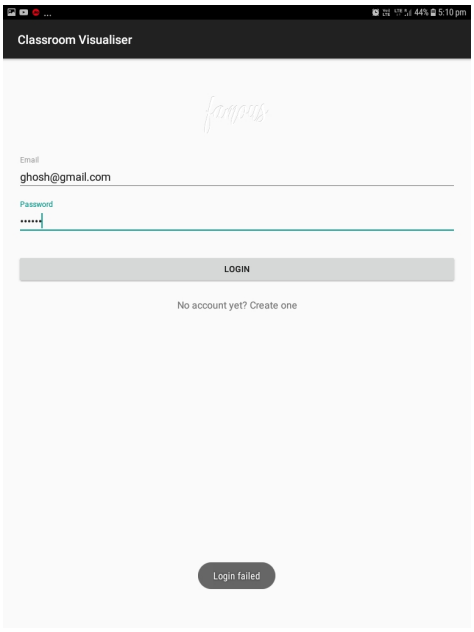


Figure 4.3: Input with wrong password

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.

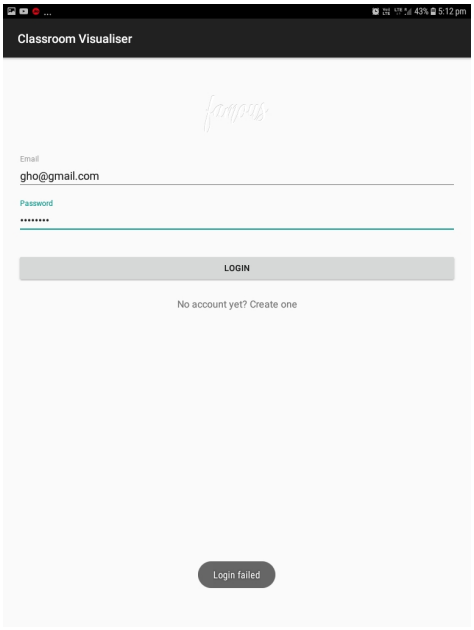
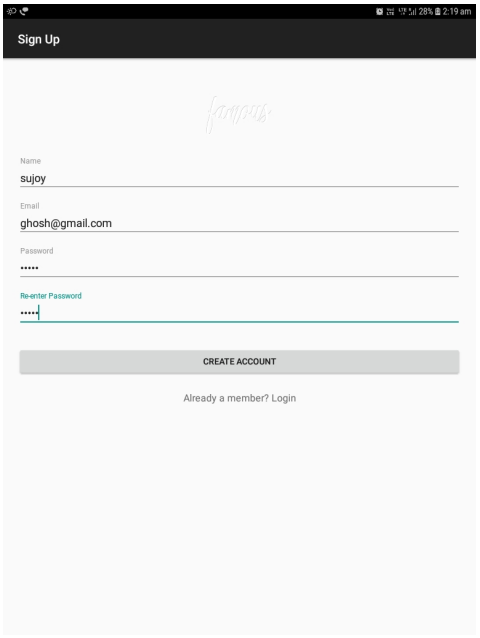


Figure 4.4: Input with wrong email id as well as wrong password

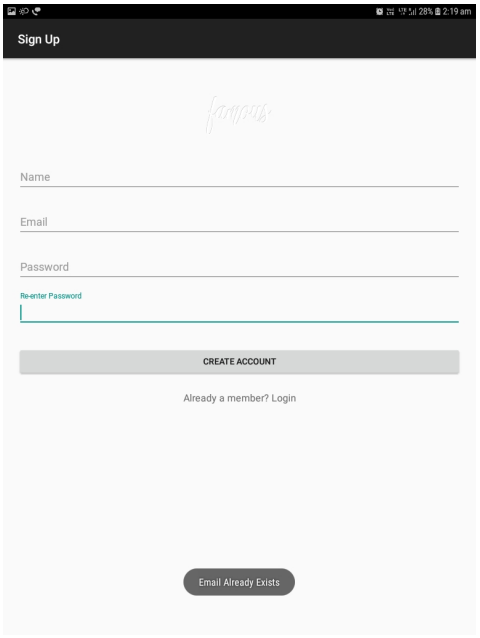
Output: 'Login Failed' is displayed. Hence we get our desired output.

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".



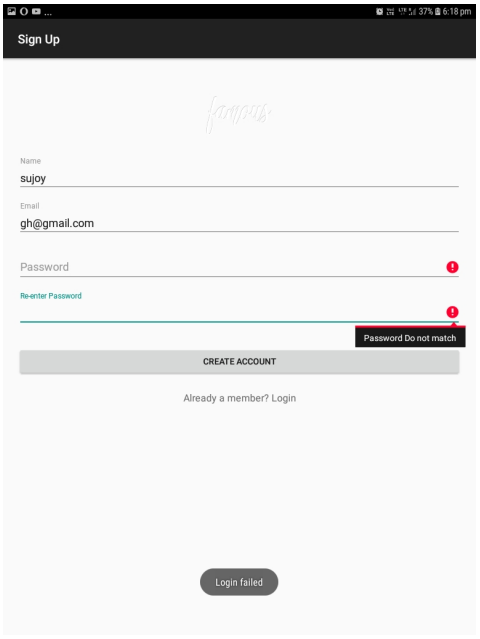
(a) Input Credentials



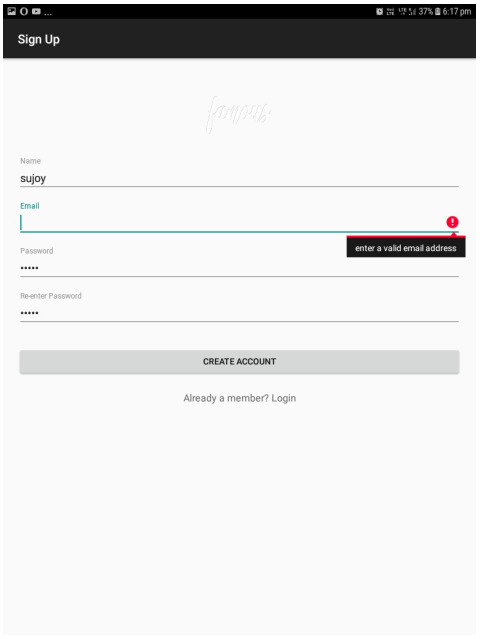
(b) Output : Email exists

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

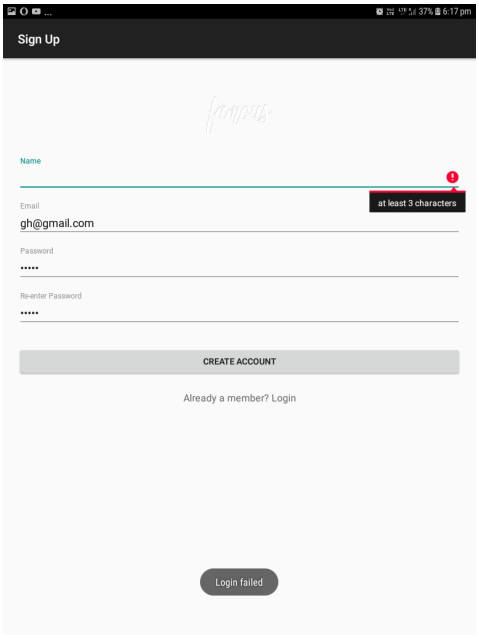
2. **Input**: If any one field is left empty
   Expected Output: Account is not created. Marks the fields left empty.



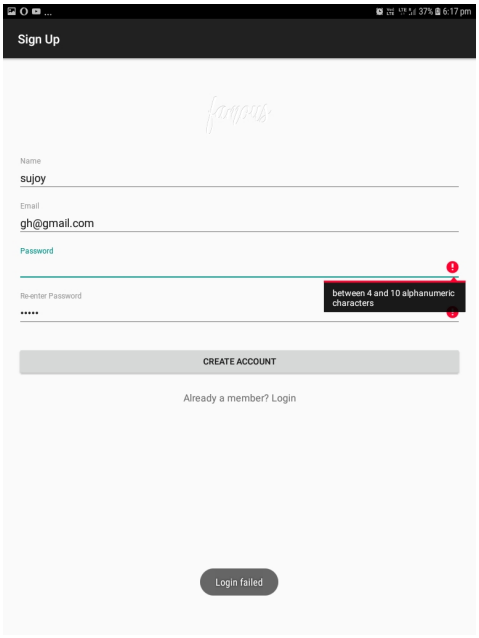
(a) Input with password empty



(b) Input with email empty



(a) Input with name empty



(b) Input with password empty

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

3. **Input**: Password ≠ Confirm Password
   Expected Output: Displays "Passwords do not match".

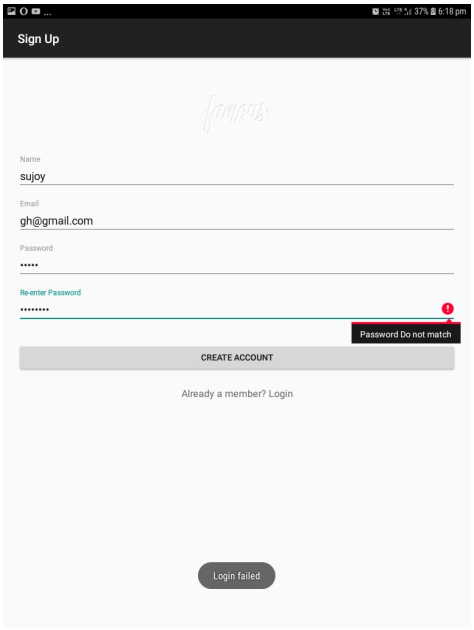
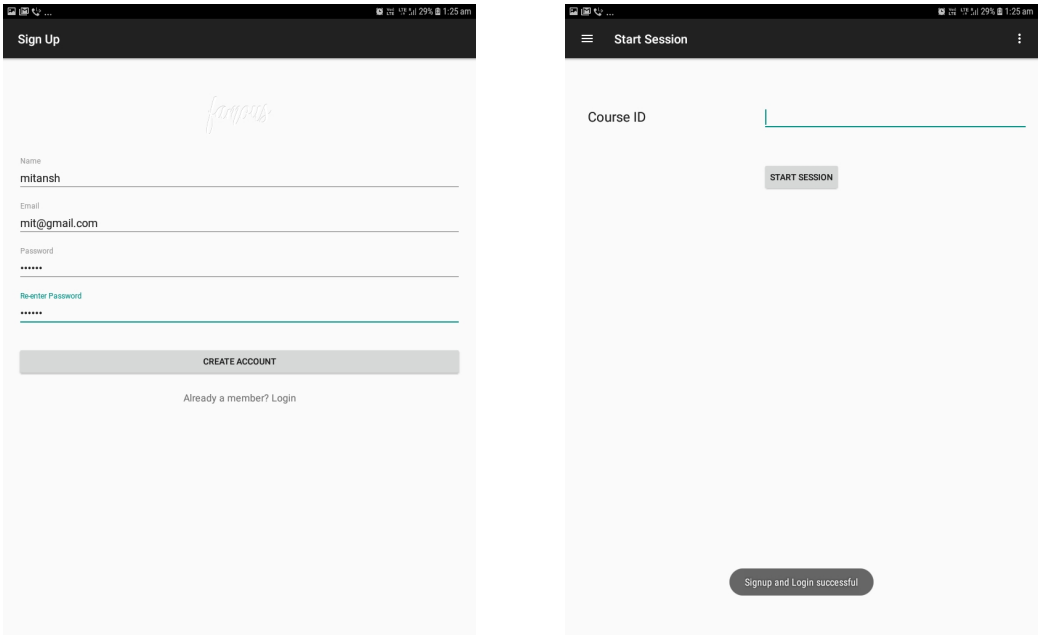


Figure 4.8: Input with Password ≠ Confirm Password

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.



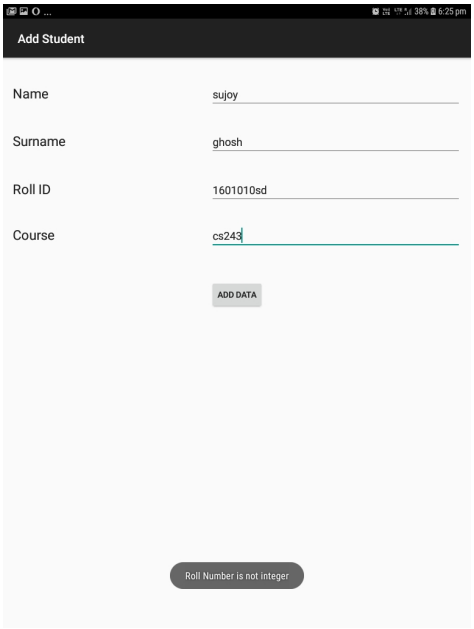
(a) Input with correct credentials (b) Output with successful login

Figure 4.9: Successful signup

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

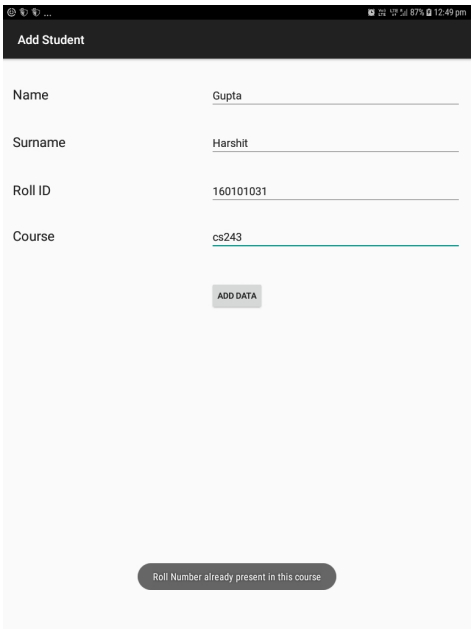
4.3 Module Name: Add Student

- **Equivalence Classes**:
  1. **Input**: Roll No. not integer
     Expected Output: Displays "Roll Number is not an integer".



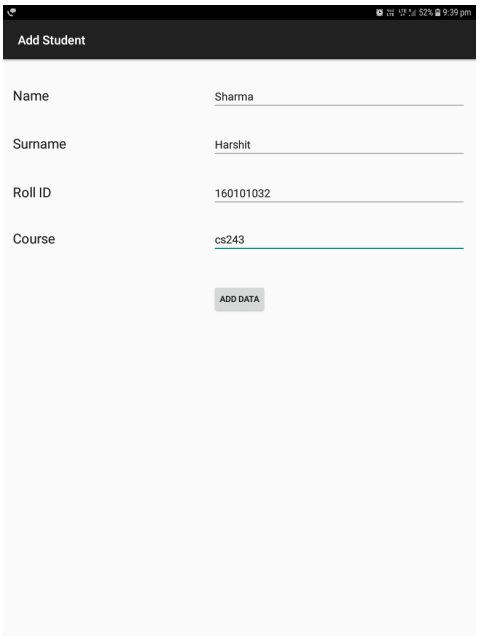
Output: Displays "Roll number is not integer". We get our expected output.

2. **Input**: Roll No. already exists in entered course ID
   Expected Output: Displays "Student already exists".



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.



(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.

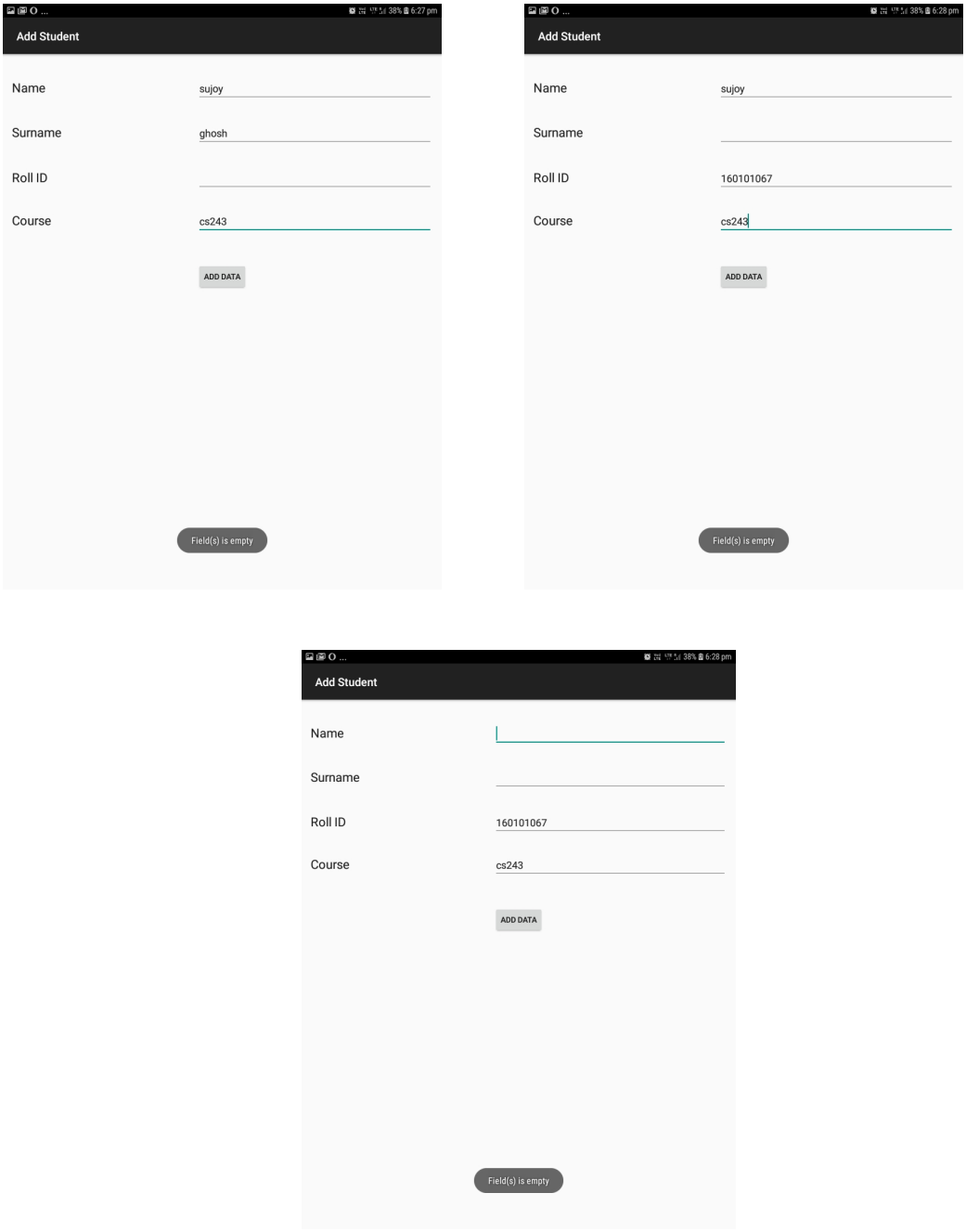
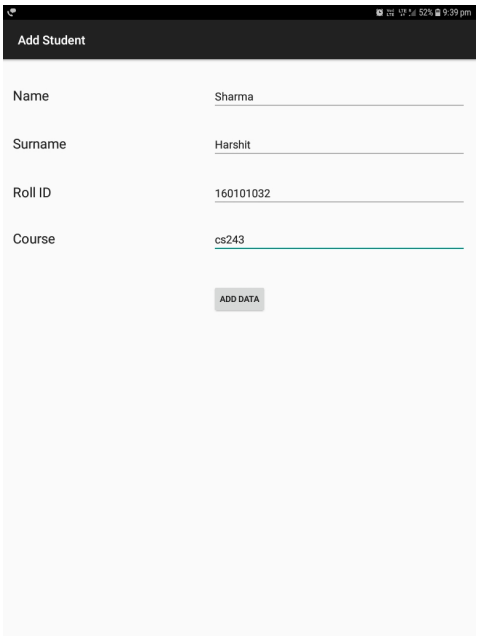


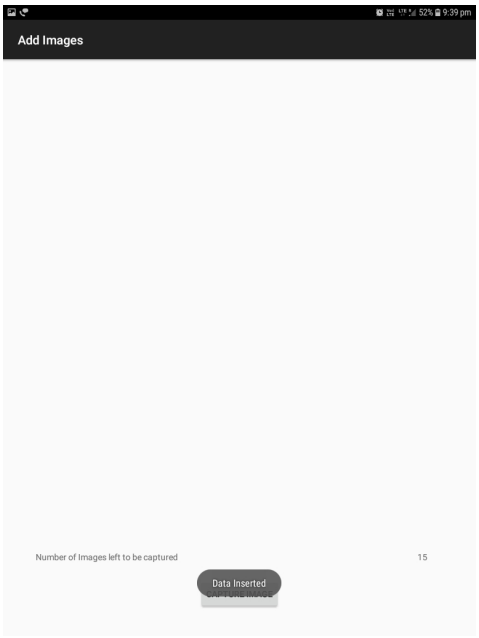
Figure 4.12: Input with empty details

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



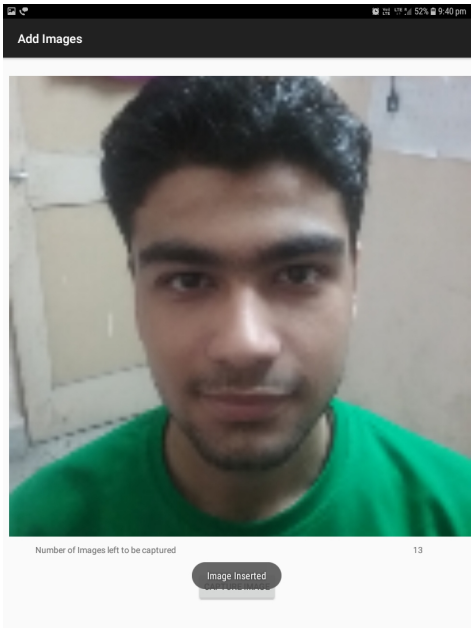
(b) Output

Figure 4.13: We get desired output

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

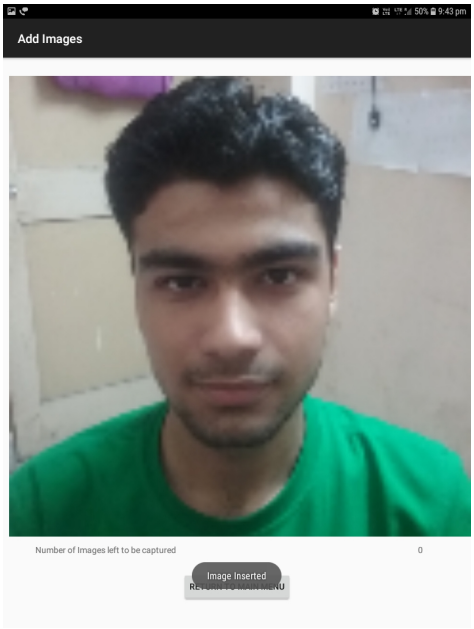
4.4 Module Name: Add Images

- **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.

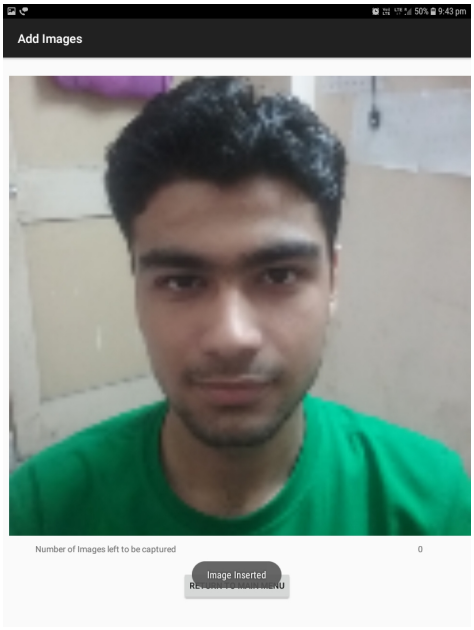
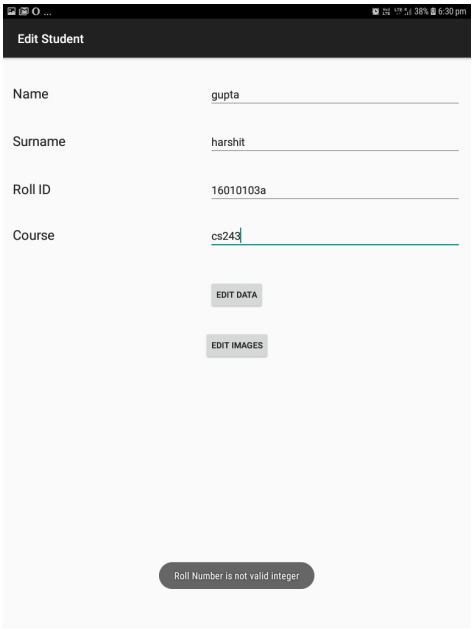


Figure 4.14: Number of photos left = 0

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

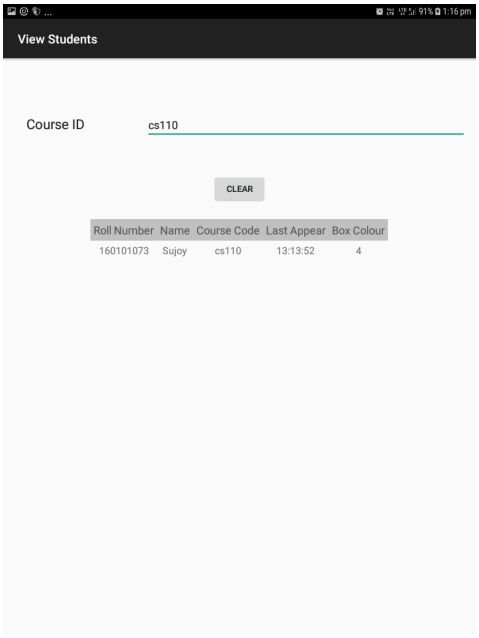
4.5 Module Name: Edit Student

- **Equivalence Classes**:
  1. **Input**: Roll No. not integer
     Expected Output: Displays "Roll Number is not an integer".

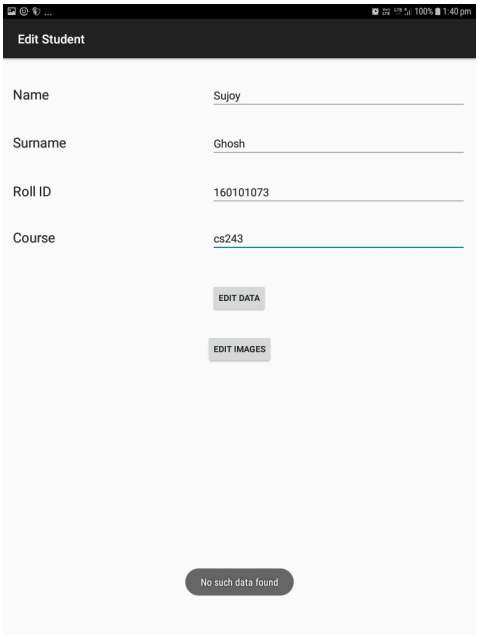


Output: Displays "Roll Number is not an integer".We get our desired output.

2. **Input**: Roll No. already exists but in other course ID
   Expected Output: Displays "Student does not exists".



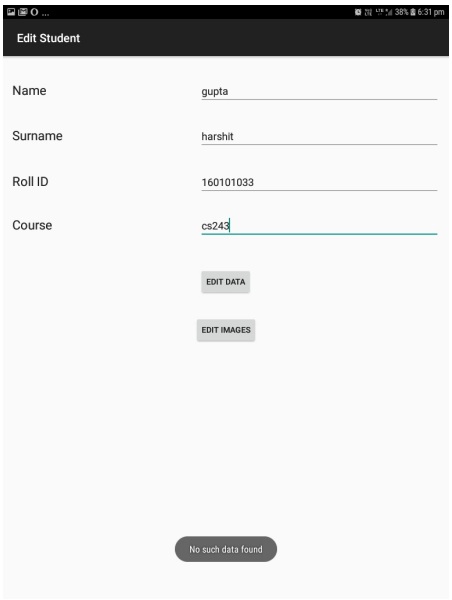
(a) Input with correct details



(b) Output

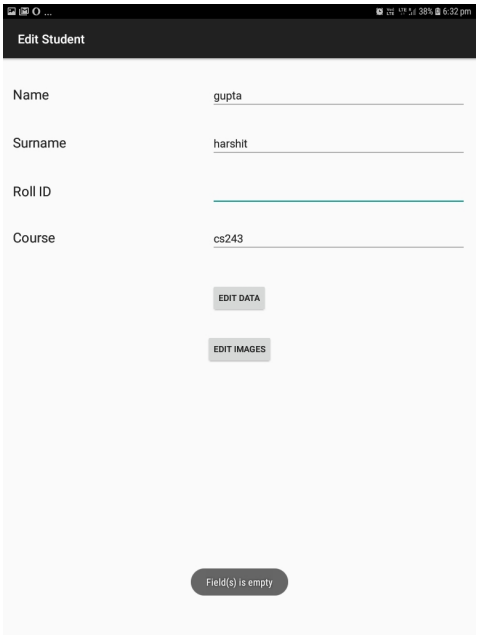
Output: Displays "No such data found".We get our desired output.

3. **Input**: Roll No. doesn't exist
   Expected Output: Displays "Student does not exists".

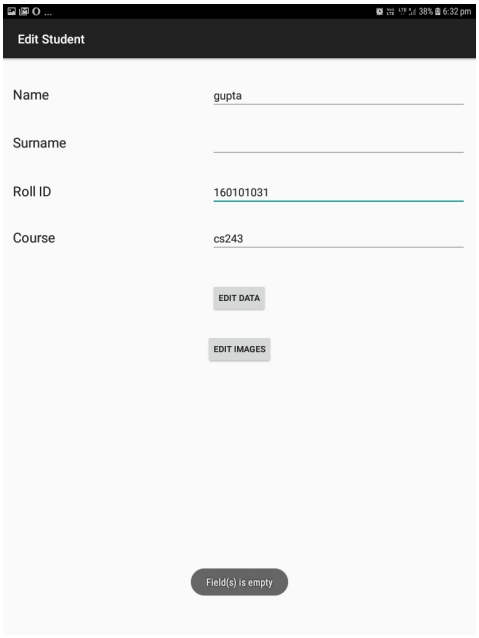


   Output: Displays "No such data found". Hence, we get our desired output.

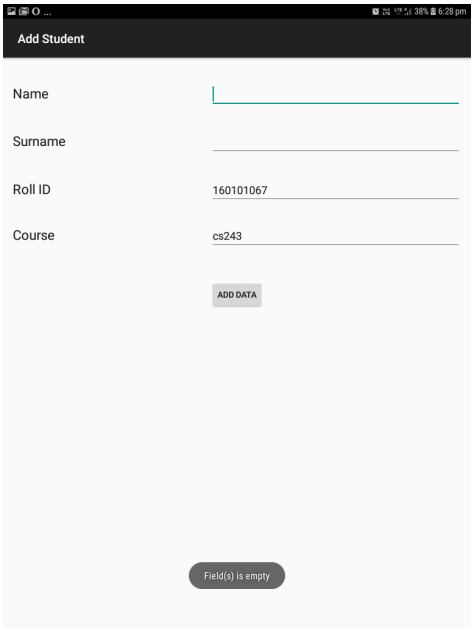
4. **Input**: If any one field is left empty
   Expected Output: Shows error on empty fields.



(a) Input with empty fields

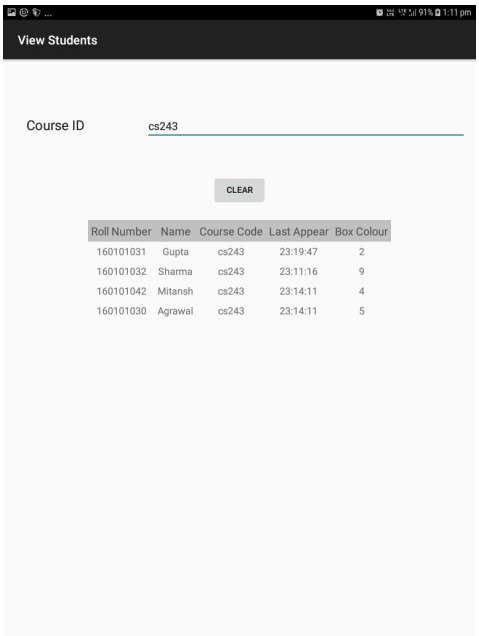


(b) Output

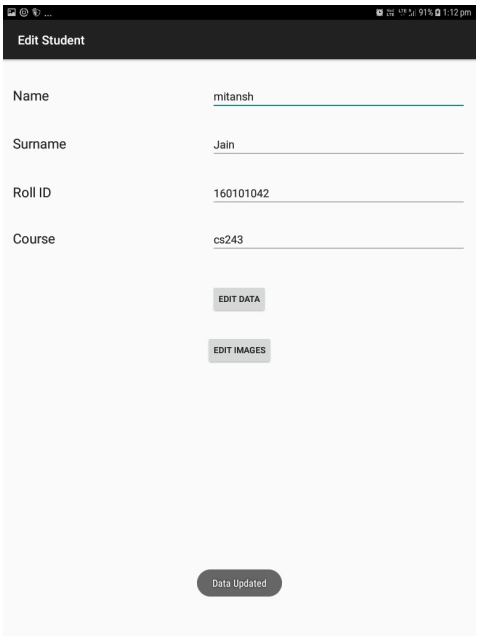


Output: Displays "Field(s) is empty".We get our desired output.

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".



(a) Student list



(b) Output

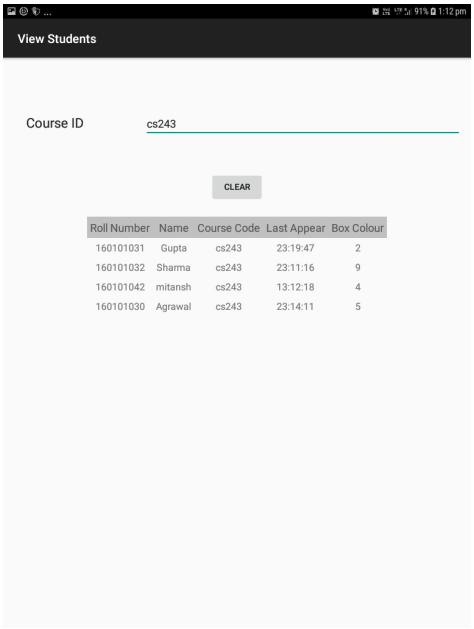
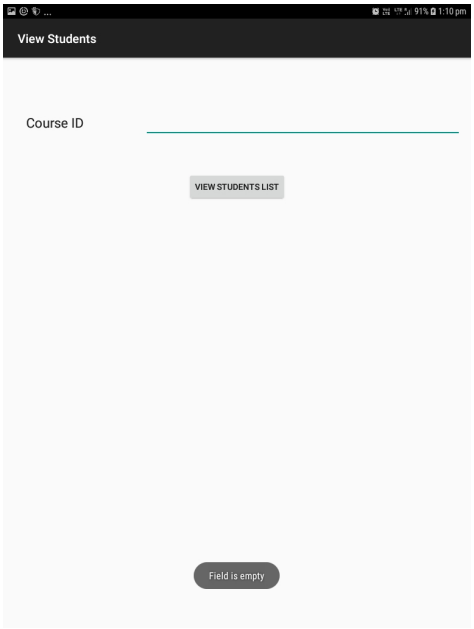


Figure 4.18: Updated List

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.

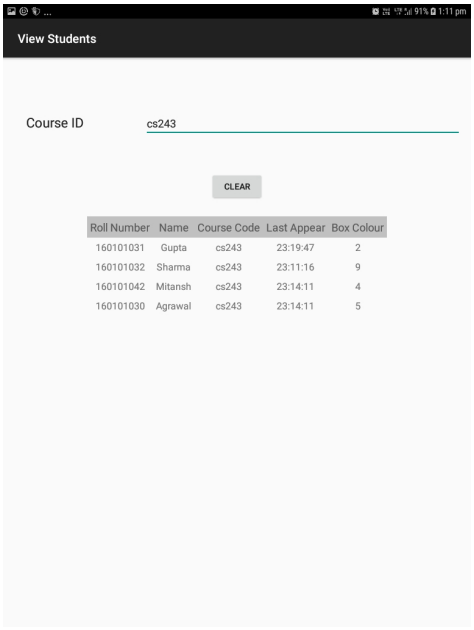
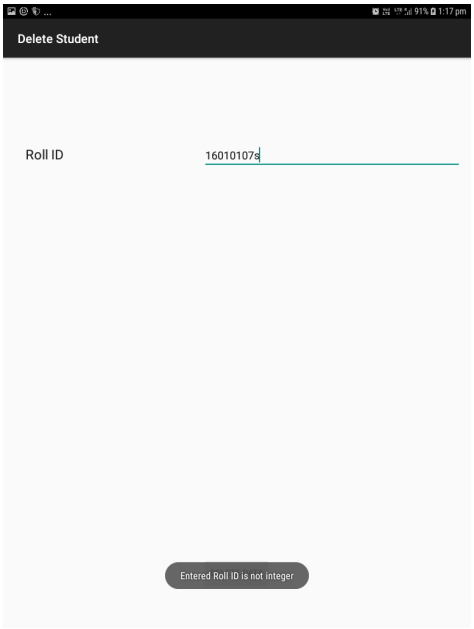


Figure 4.20: Input with corrected course id

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".



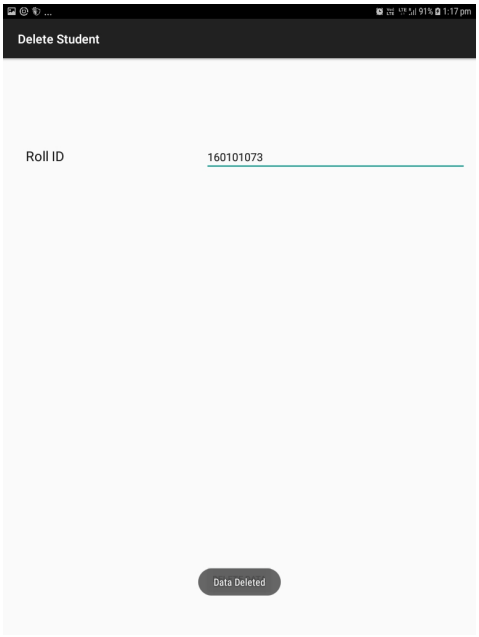
Output: Displays "Roll Number is not an integer".We get our desired output.

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.

(a) Student List

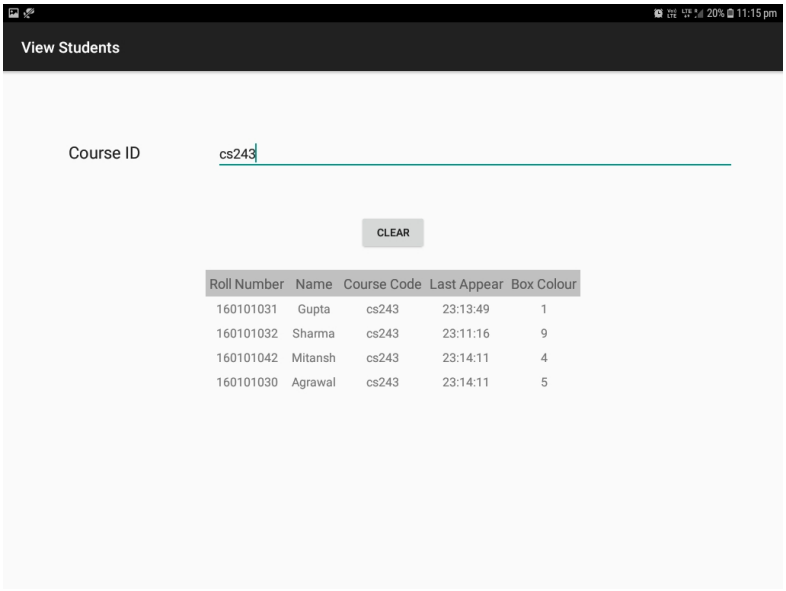


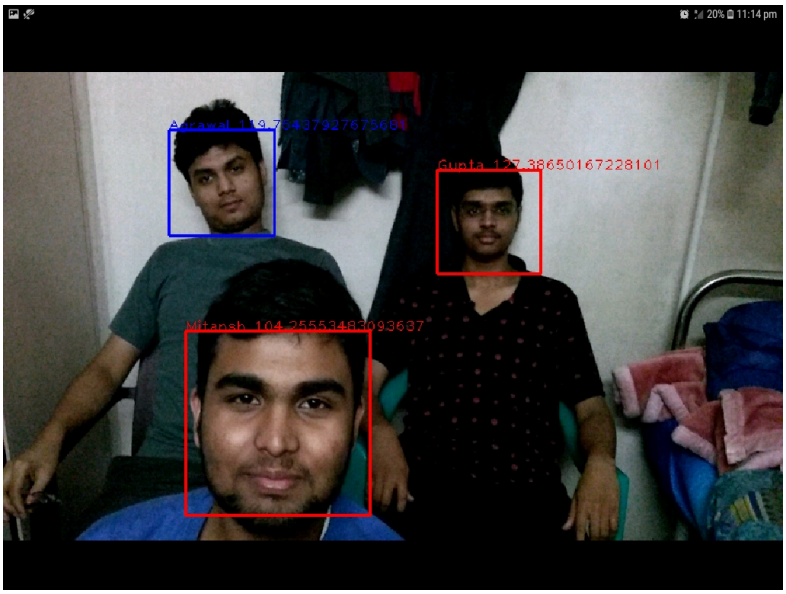
(b) Output

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

4.8 Module name: Camera Session and Face Recognition

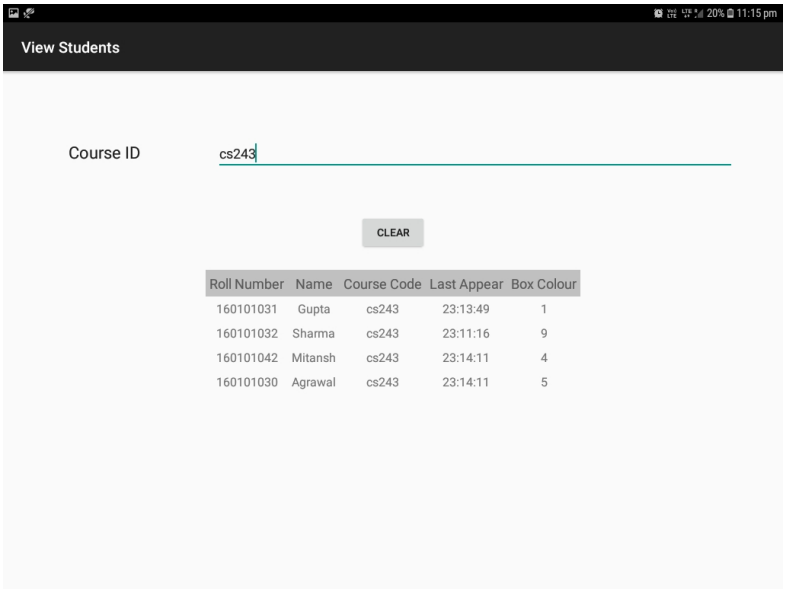
- **Equivalence Classes**:
  1. **Input**: State of student < 5.
     Expected Output: Bounding box color: red

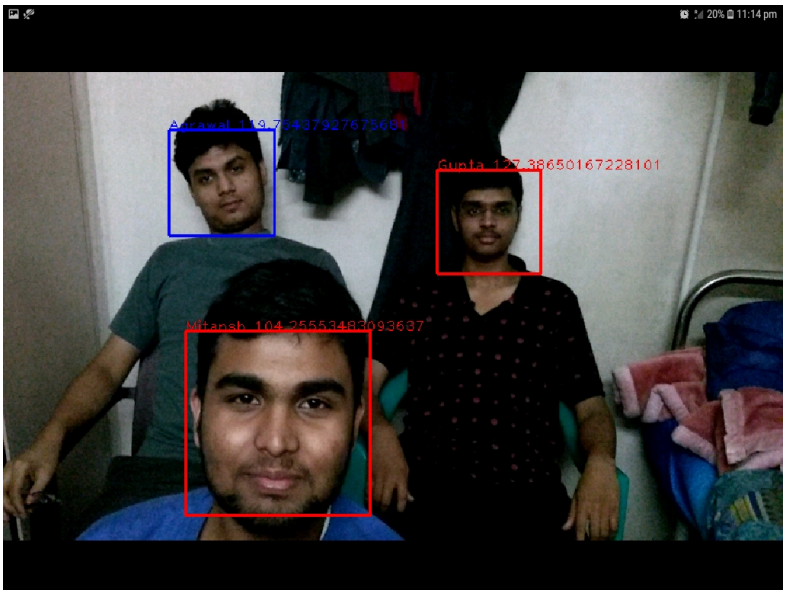




Output: Displays person recognised with corresponding bounding box.We get our desired output.

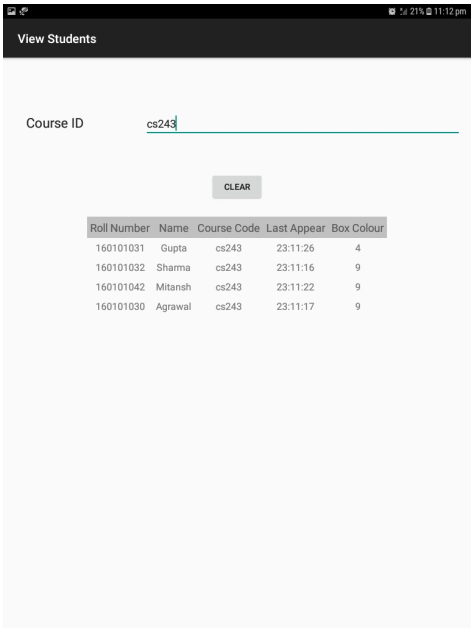
2. **Input**: 5 < State of student < 8
   Expected Output: Bounding box color: blue

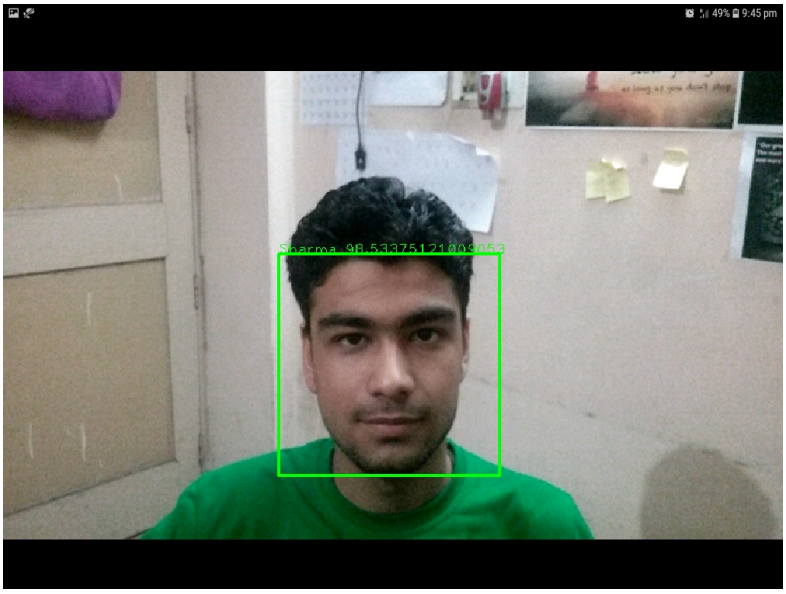




Output: Displays person recognised with corresponding bounding box.We get our desired output.

3. **Input**: State of student > 8
   Expected Output: Bounding box color: green





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.

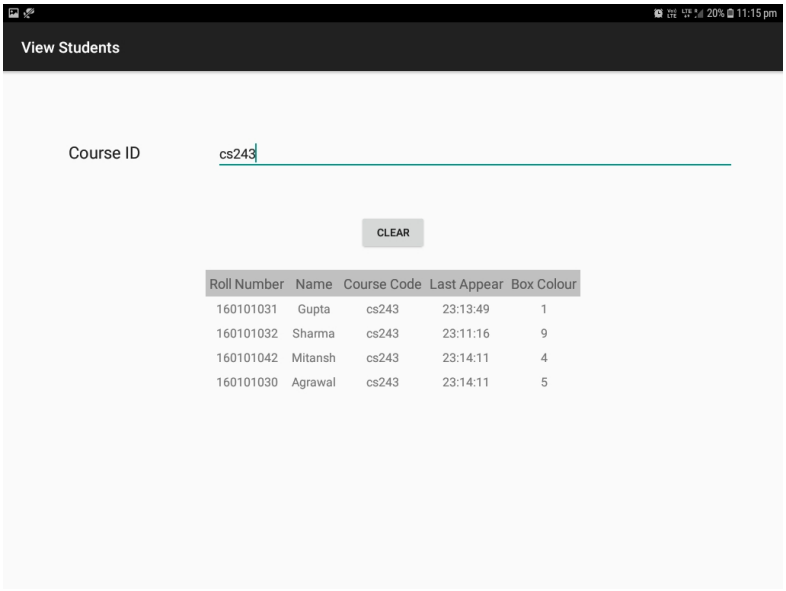


Figure 4.22: List of students in class

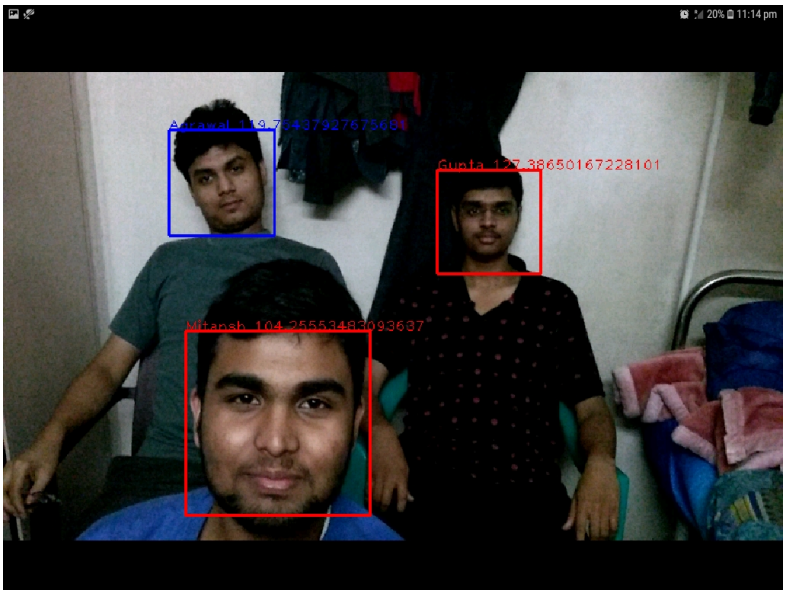
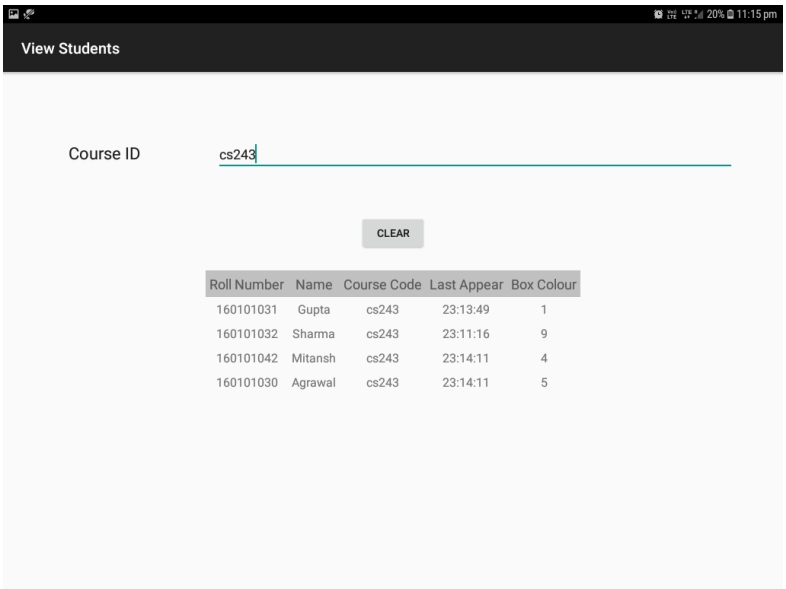
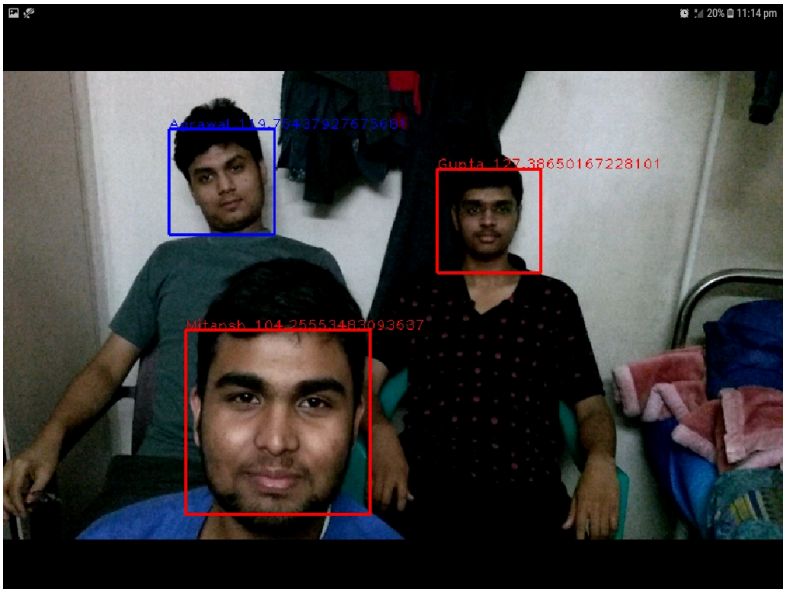


Figure 4.23: Students identitfied correctly

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

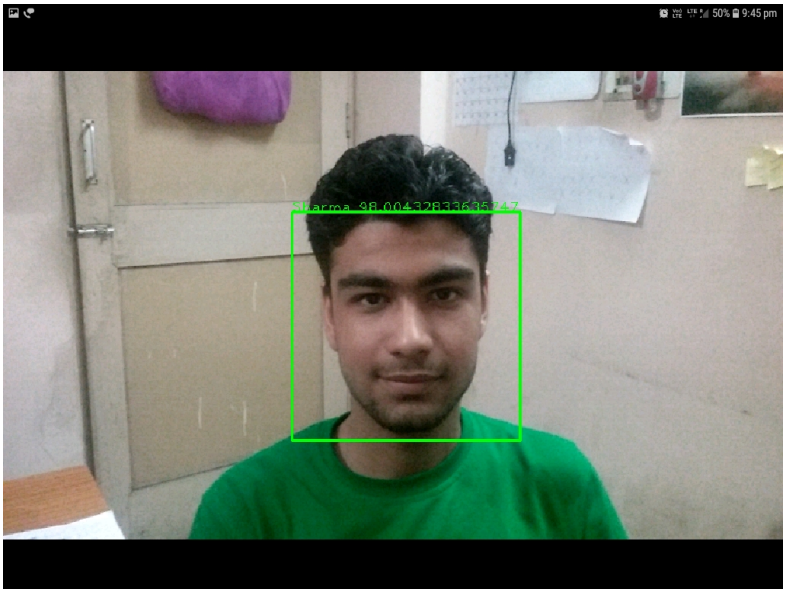
- **Boundary Cases**:
  1. **Input**: State of student = 5.
     Expected Output: Bounding box color: blue





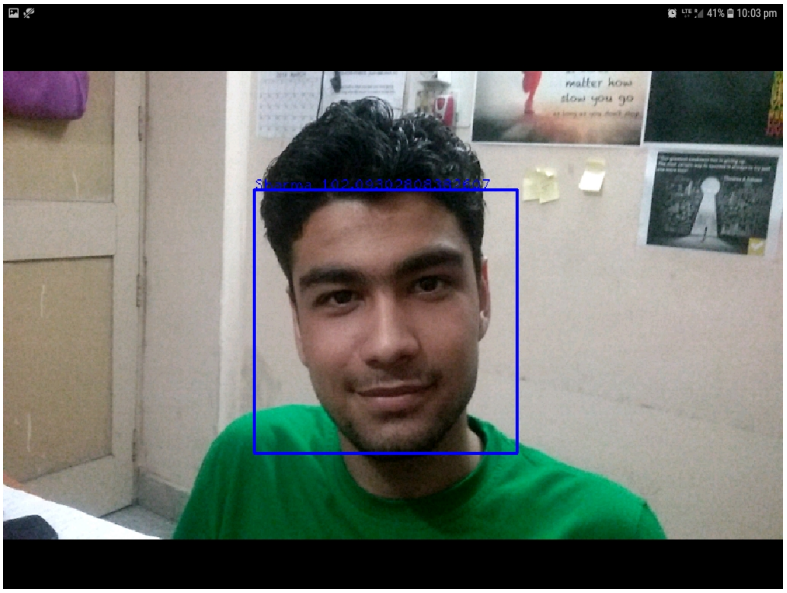
Output: Displays correct bounding box color for state=5.We get our desired output.

  2. **Input**: State of student = 8
     Expected Output: Bounding box color: green



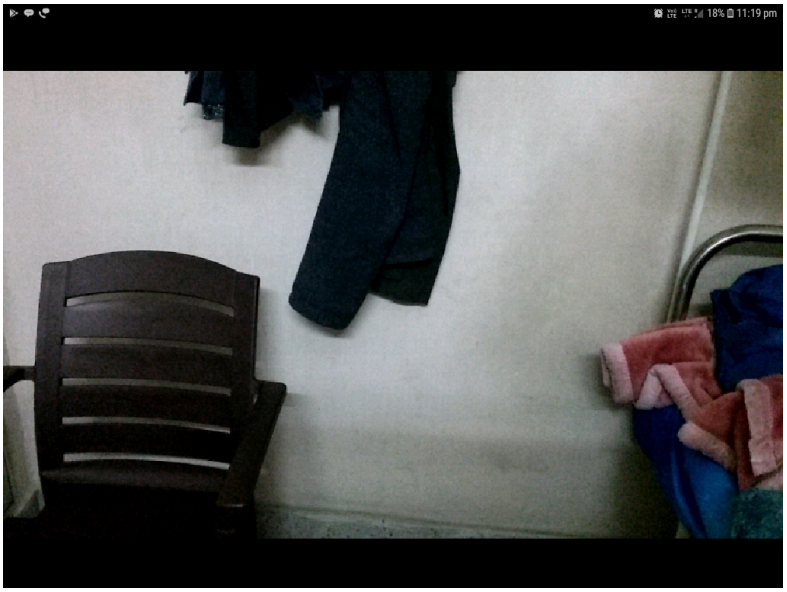
Output: Displays correct bounding box color for state=8.We get our desired output.

3. **Input**: Number of known students = 1
   Expected Output: Student should be correctly identified.



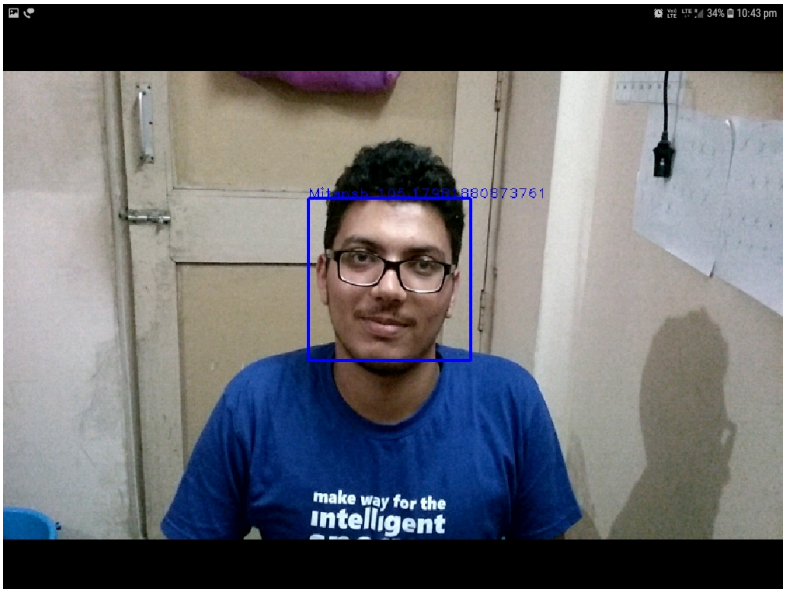
Output: Displays correctly identified person with its bounding box.We get our desired output.

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 marked with unknown.



Output: Recognises the person to be "Mitansh" although he is a unknown student. It may be due to light conditions and face orientation as openCV is very sensitive to these aspects. If any unkwown 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 Funtion: signup()

```
public void signup() {
    EditTextEmptyHolder = false;
    if (!validate()) {
        onSignupFailed();
        return;
    }
    EditTextEmptyHolder = true;
    _signupButton.setEnabled(false);
    final ProgressDialog progressDialog = new ProgressDialog(SignupActivity.this,
            R.style.AppTheme_Dark_Dialog);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage("Creating Account...");
    progressDialog.show();
    name = _nameText.getText().toString();
    email = _emailText.getText().toString();
    password = _passwordText.getText().toString();
    reEnterPassword = _reEnterPasswordText.getText().toString();
    SQLiteDataBaseBuild();
    SQLiteTableBuild();
    EmptyEditTextAfterDataInsert();
    new android.os.Handler().postDelayed(
            new Runnable() {
                public void run() {

                    CheckingEmailALreadyExistsOrNot();
                    progressDialog.dismiss();
                }
            }, 3000);
}

```

Figure 5.1: Code for signup() function

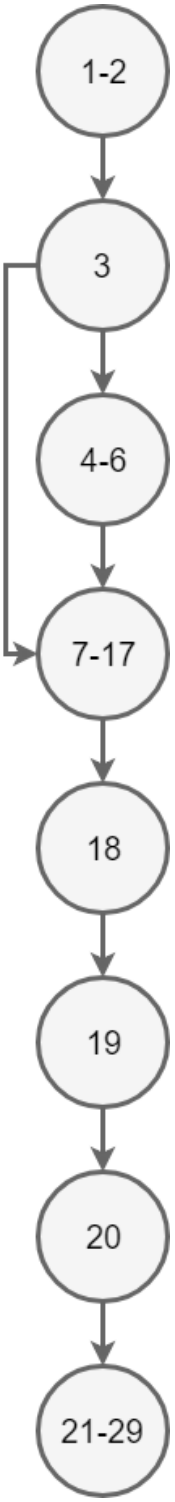


Figure 5.2: CFG for signup() function

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 Funtion: 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

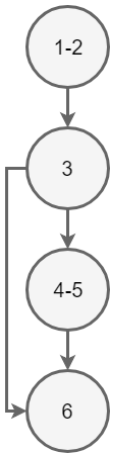


Figure 5.4: CFG for CheckEmailAlreadyExists() function

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 Funtion: 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

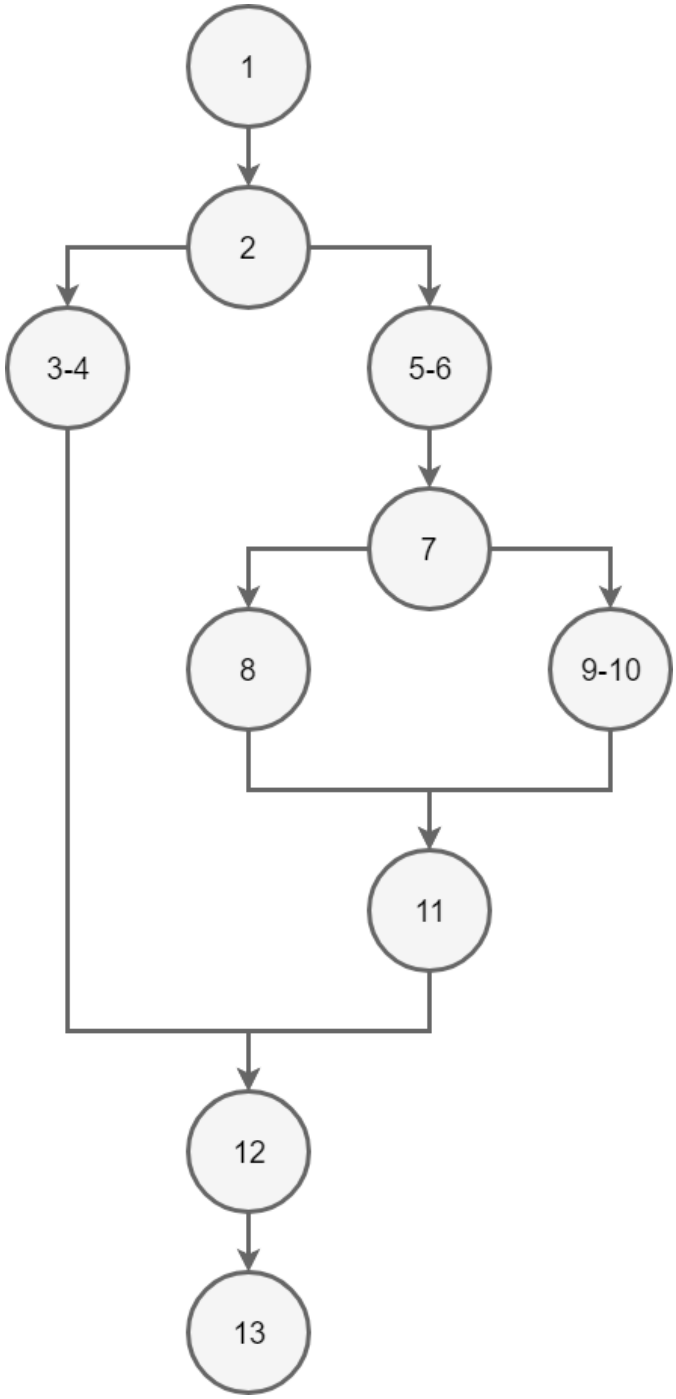


Figure 5.6: CFG for checkFinalResult() function

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 Funtion: validate()

```
public boolean validate() {
    boolean valid = true;
    String name = _nameText.getText().toString();
    String email = _emailText.getText().toString();
    String password = _passwordText.getText().toString();
    String reEnterPassword = _reEnterPasswordText.getText().toString();
    if (name.isEmpty() || name.length() < 3) {
        _nameText.setError("at least 3 characters");
        valid = false;
    } else {
        _nameText.setError(null);
    }
    if (email.isEmpty() || !android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
        _emailText.setError("enter a valid email address");
        valid = false;
    } else {
        _emailText.setError(null);
    }
    if (password.isEmpty() || password.length() < 4 || password.length() > 10) {
        _passwordText.setError("between 4 and 10 alphanumeric characters");
        valid = false;
    } else {
        _passwordText.setError(null);
    }
    if (reEnterPassword.isEmpty() || reEnterPassword.length() < 4 || reEnterPassword.length() > 10 || !(reEnterPassword.equals(password))) {
        _reEnterPasswordText.setError("Password Do not match");
        valid = false;
    } else {
        _reEnterPasswordText.setError(null);
    }
    return valid;
}
```

Figure 5.7: Code for validate() function

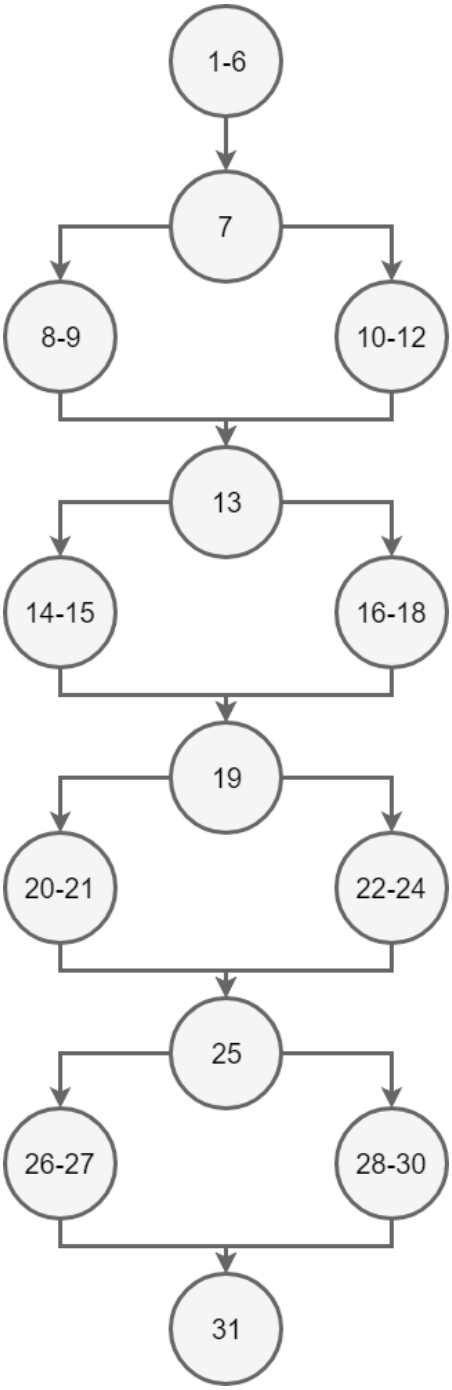


Figure 5.8: CFG for validate() function

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 Funtion: AddData()

```
public void AddData() {
    btnAddData.setOnClickListener(
            new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    boolean isInteger = false;
                    String courseID = editCourse.getText().toString();
                    String name = editName.getText().toString();
                    String surname = editSurname.getText().toString();
                    String rollID = editTextId.getText().toString();
                    int roll_number=0;
                    try{
                        roll_number = Integer.parseInt(editTextId.getText().toString());
                        if (roll_number > 0) {
                            isInteger = true;
                        }
                    }
                    catch ( NumberFormatException e ) {
                        isInteger = false;
                    }
                    if (courseID.isEmpty() || surname.isEmpty() || name.isEmpty() || rollID.isEmpty()) {
                        Toast.makeText(AddStudentActivity.this, "Field(s) is empty", Toast.LENGTH_LONG).show();
                    } else if(isInteger) {
                        if(myDb.ifStudentExists(roll_number, courseID) == Boolean.FALSE) {
                            boolean isInserted = myDb.insertData( roll_number,
                                    editName.getText().toString(),
                                    editSurname.getText().toString(),
                                    editCourse.getText().toString() );
                            if(isInserted == true) {
                                Toast.makeText(AddStudentActivity.this, "Data Inserted", Toast.LENGTH_LONG).show();
                                Intent intent = new Intent(AddStudentActivity.this, AddImages.class);
                                intent.putExtra("id", editTextId.getText().toString());
                                startActivity(intent);
                                finish();
                            } else {
                                Toast.makeText(AddStudentActivity.this,"Data not Inserted",Toast.LENGTH_LONG).show();
                            }
                        } else {
                            Toast.makeText(AddStudentActivity.this, "Roll Number is not integer", Toast.LENGTH_LONG).show();
                        }
                    } else {
                        Toast.makeText(AddStudentActivity.this,"Roll Number is not integer",Toast.LENGTH_LONG).show();
                    }

                }
            }
    );
}
```

Figure 5.9: Code for addData() function

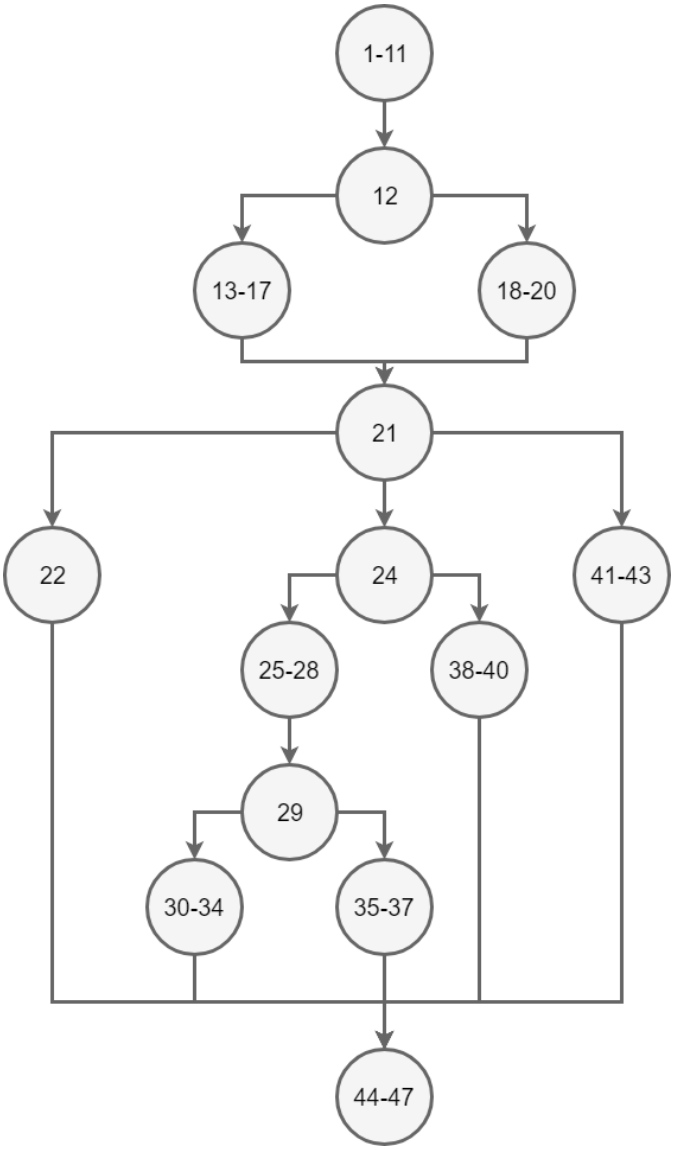


Figure 5.10: CFG for addData() function

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"

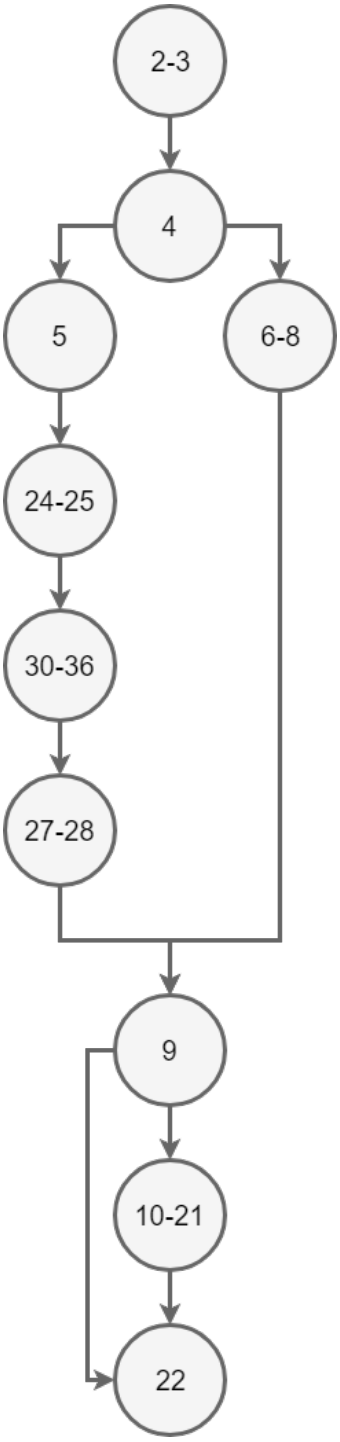
5.3 Module: AddImages

5.3.1 Funtions: onResume(), addimage(), btnTakePhotoClicker.OnClickListener()

```
private static final int IMG_COUNT = 15;
public void onResume() {
    super.onResume();
    if (tempCounter != IMG_COUNT){
        addImage();
    } else {
        button.setText("Return to Main Menu");
    }
    if (tempCounter == IMG_COUNT) {
        button.setOnClickListener(
            new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Intent nextPage = new Intent(AddImages.this,
                            MainActivity.class);
                    startActivity(nextPage);
                    finish();
                }
            }
        );
    }
}

public void addImage() {
    button.setOnClickListener(
            new AddImages.btnTakePhotoClicker()
    );
}

class btnTakePhotoClicker implements  Button.OnClickListener {
    @Override
    public void onClick(View view) {
        Intent isPhotoAcceptable = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(isPhotoAcceptable, CAM_REQUEST);
    }
}
```


2-3
4
5
6-8
24-25
30-36
27-28
9
10-21
22

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 Funtion: deleteStudent()

```
public void DeleteData() {
    btnDelete.setOnClickListener(
        new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                boolean isInteger = false;
                String rollID = editTextId.getText().toString();
                int roll_number=0;
                try
                {
                    roll_number = Integer.parseInt(editTextId.getText().toString());
                    if (roll_number > 0) {
                        isInteger = true;
                    }
                } catch ( NumberFormatException e ) {
                    isInteger = false;
                }
                if (rollID.isEmpty()) {
                    Toast.makeText(DeleteStudentActivity.this, "Field(s) is empty", Toast.LENGTH_LONG).show();
                } else if(isInteger) {
                    Integer deletedRows = myDb.deleteData(roll_number);
                    Integer deletedImg = imDb.deleteImg(roll_number);
                    if (deletedRows > 0 && deletedImg > 0){
                        Toast.makeText(DeleteStudentActivity.this, "Data Deleted", Toast.LENGTH_LONG).show();
                    } else{
                        Toast.makeText(DeleteStudentActivity.this, "Roll ID not present in database", Toast.LENGTH_LONG).show();
                    }
                } else {
                    Toast.makeText(DeleteStudentActivity.this,"Entered Roll ID is not integer",Toast.LENGTH_LONG).show();
                }
            }
        }
    );
}
```

Figure 5.11: Code for deleteStudent() function

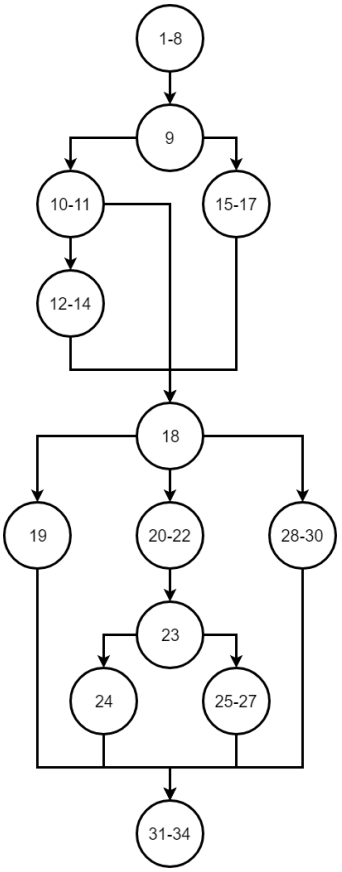


Figure 5.12: CFG for deleteStudent() function

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 Funtion: UpdateData()

```
public void UpdateData() {
    btnviewUpdate.setOnClickListener(
            new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    boolean isInteger = false;
                    int roll_number=0;
                    try
                    {
                        roll_number = Integer.parseInt(editTextId.getText().toString());
                        isInteger = true;
                    } catch ( NumberFormatException e ) {
                        isInteger = false;
                    }
                    if(isInteger) {
                        boolean isUpdate = myDb.updateData(roll_number,
                                editName.getText().toString(),
                                editSurname.getText().toString(), editCourse.getText().toString());
                        if (isUpdate == true) {
                            Toast.makeText(EditStudentActivity.this, "Data Updated", Toast.LENGTH_LONG).show();
                        } else {
                            Toast.makeText(EditStudentActivity.this, "Data not Updated", Toast.LENGTH_LONG).show();
                        }
                    } else {
                        Toast.makeText(EditStudentActivity.this,"Roll Number is not integer",Toast.LENGTH_LONG).show();
                    }
                }
            }
    );
}
```

Figure 5.13: Code for UpdateData() function

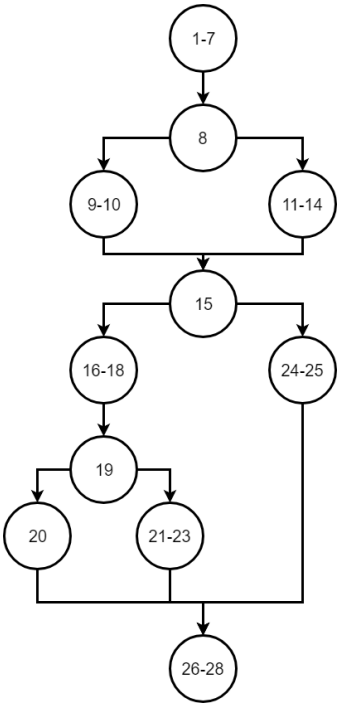


Figure 5.14: CFG for UpdateData() function

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 Funtion: UpdateImages()

```
public void UpdateImage() {
    button.setOnClickListener(
        new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                boolean isInteger = false;
                String courseID = editCourse.getText().toString();
                String name = editName.getText().toString();
                String surname = editSurname.getText().toString();
                String rollID = editTextId.getText().toString();
                int roll_number=0;
                try
                {
                    roll_number = Integer.parseInt(editTextId.getText().toString());
                    isInteger = true;
                } catch ( NumberFormatException e ) {
                    isInteger = false;
                }
                if (courseID.isEmpty() || surname.isEmpty() || name.isEmpty() || rollID.isEmpty()) {
                    Toast.makeText(EditStudentActivity.this, "Field(s) is empty", Toast.LENGTH_LONG).show();
                } else if(isInteger) {
                    if ( myDb.ifStudentExists(roll_number, courseID) ) {
                        Integer deletedImg = imDb.deleteImg(roll_number);
                        if (deletedImg > 0) {

                            Toast.makeText(EditStudentActivity.this, "Images Deleted", Toast.LENGTH_LONG).show();

                            Intent intent = new Intent(EditStudentActivity.this, AddImages.class);

                            intent.putExtra("id", editTextId.getText().toString());
                            startActivity(intent);
                            finish();
                        } else {
                            Toast.makeText(EditStudentActivity.this, "Roll ID not present in database", Toast.LENGTH_LONG).show();
                        }
                    } else {
                        Toast.makeText(EditStudentActivity.this, "Roll ID with given course not present", Toast.LENGTH_LONG).show();
                    }
                } else {
                    Toast.makeText(EditStudentActivity.this,"Entered Roll ID is not integer",Toast.LENGTH_LONG).show();
                }
            }
        }
    );
}
```

Figure 5.15: Code for UpdateImages() function

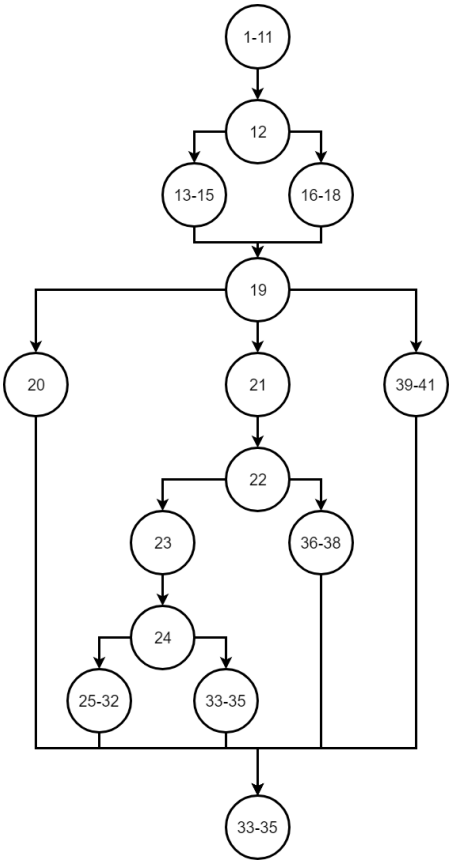


Figure 5.16: CFG for UpdateImages() function

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 corse not in database"

  Observed Output: Toast.text="Roll ID with given corse 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 Funtion: login()

```
public void login() {
    if (!validate()) {
        onLoginFailed();
        return;
    }
    _loginButton.setEnabled(false);
    final ProgressDialog progressDialog = new ProgressDialog(LoginActivity1.this,
            R.style.AppTheme_Dark_Dialog);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage("Authenticating...");
    progressDialog.show();
    String email = _emailText.getText().toString();
    final String password = _passwordText.getText().toString();
    Cursor result = sqLiteHelper.getPassword(email);
    if(result.getCount() == 0)
    {
        new android.os.Handler().postDelayed(
                new Runnable() {
                    public void run() {
                        onLoginFailed();
                        progressDialog.dismiss();
                    }
                }, 3000);
        return;
    }
    result.moveToFirst();
    TempPassword = result.getString(3);
    new android.os.Handler().postDelayed(
            new Runnable() {
                public void run() {
                    if(TempPassword.equals(password)) {
                        onLoginSuccess();
                    }
                    else {
                        onLoginFailed();
                    }
                    progressDialog.dismiss();
                }
            }, 3000);
}
```

Figure 5.17: Code for login() function

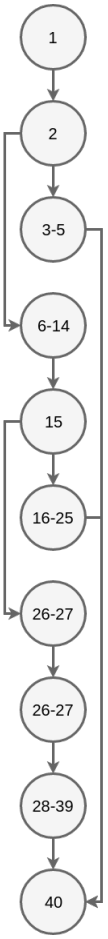


Figure 5.18: CFG for login() function

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 Funtion: validate()

```
public boolean validate() {
    boolean valid = true;
    String email = _emailText.getText().toString();
    String password = _passwordText.getText().toString();
    if (email.isEmpty() || !android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
        _emailText.setError("enter a valid email address");
        valid = false;
    } else {
        _emailText.setError(null);
    }
    if (password.isEmpty() || password.length() < 4 || password.length() > 10) {
        _passwordText.setError("between 4 and 10 alphanumeric characters");
        valid = false;
    } else {
        _passwordText.setError(null);
    }
    return valid;
}
```

Figure 5.19: Code for Validate() function

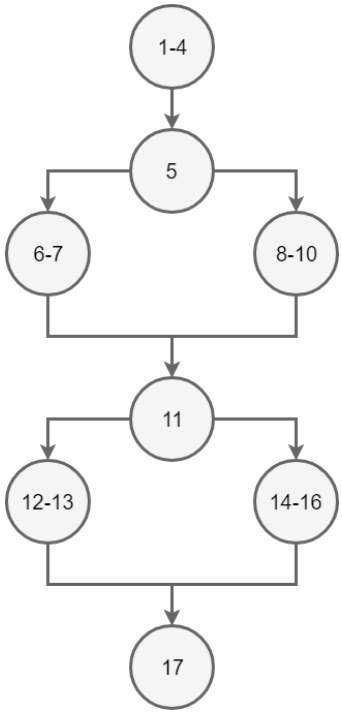


Figure 5.20: CFG for Validate() function

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 Funtion: faceDetect()

```
public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
    mRgba = inputFrame.rgba();
    mGray = inputFrame.gray();
    if(!isTrainComplete){
        training();
    }
    String globalTime = getTime();
    MatOfRect faces = new MatOfRect();
    mJavaDetector.detectMultiScale(mGray, faces, 1.1, 2, 2,
                new Size(mAbsoluteFaceSize, mAbsoluteFaceSize), new Size());
    int x=0,y=0;
    Rect[] facesArray = faces.toArray();
    for (int i = 0; i < facesArray.length; i++){
        x = facesArray[i].x;
        y = facesArray[i].y;
        Rect roi = new Rect(facesArray[i].tl(), facesArray[i].br());
        Mat cropped = new Mat(mGray, roi);
        int[] label = new int[1];
        double[] conf = new double[1];
        try{
            faceRecognizer.predict(cropped, label, conf);
        } catch (Exception e) {
            Log.d(TAG, e.toString());
        }
        Random randomVal = new Random();
        if(label[0] < 0 || conf[0] > 150.0) {
            Imgproc.rectangle(mRgba, facesArray[i].tl(), facesArray[i].br(), new Scalar(255,255,255), 2);
            Imgproc.putText(mRgba,"Unknown "+conf[0],new Point(x,y),Core.FONT_HERSHEY_PLAIN, 1.0, new Scalar(255,255,255));
        } else {
            int roll = personIndexList.get(label[0]);
            String playerTime = lastModifiedTime.get(roll);
            int boxColor = boundaryBoxColor.get(roll);
            if(getTimeDiff(globalTime, playerTime) > 60000){
                int newValue = randomVal.nextInt(9) + 1;
                boxColor = newValue;
                lastModifiedTime.put(roll, globalTime);
                boundaryBoxColor.put(roll, newValue);
            }
            Imgproc.rectangle(mRgba, facesArray[i].tl(), facesArray[i].br(), getBoxColor(boxColor), 2);
            Imgproc.putText(mRgba,personName.get(roll)+" "+conf[0],new Point(x,y),Core.FONT_HERSHEY_PLAIN, 1.0, getBoxColor(boxColor));
        }
    }
    return mRgba;
}
```

Figure 5.21: Code for faceDetect() function

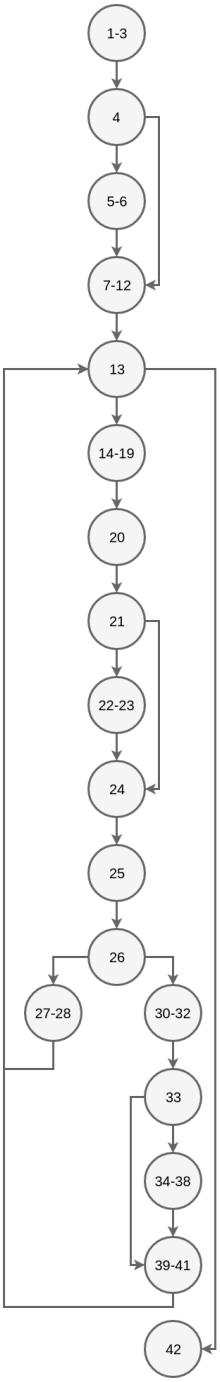


Figure 5.22: CFG for faceDetect() function

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 Funtion: 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

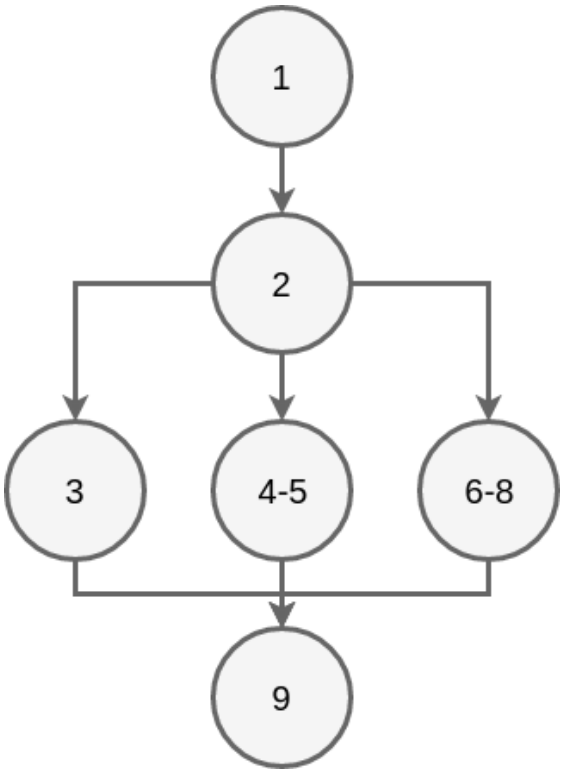


Figure 5.24: CFG for getboxColor() function

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.