SOFTWARE UNIT TESTING DOCUMENT

GROUP 22

ABHINAV MISHRA : 160101005
SAHIB KHAN : 160101057
SHIVAM KUMAR : 160101066

1.INTRODUCTION

This document contains the complete unit testing of our app Classroom Monitoring . 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 Class Room Monitoring.

2.CODE TESTING TEAM

1.TESTING TEAM PROFILE

The code 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 & Engineering. All of the members are currently in the sophomore year.

1. Kapil Goyal
2. Samyak Jain
3. Saurabh bazari

All members of the team are proficient in Java and have past experience in developing android applications.

2.BLACK BOX TESTING

2.1 PURPOSE OF BLACK BOX TESTING

The purpose of Black Box testing for the Classroom Monitoring App. Software planning is done as soon as the details of the project are ready. The main intent of this is that the developers can realize if their requirements will suffice and if the program does as based on the functional requirements.

2.3 FUNCTIONING OF BLACK BOX TESTING

Generally Black Box Testing attempts to find errors in the external behaviour of the code in the following categories:

- Incorrect or missing functionality
- Interface errors
- Errors in data structures used by interface
- Behaviour or performance errors
- Initialization and termination errors
- Concurrency and timing errors

Through this testing we can determine if the functions appear to work according to specification. It is suggested that the black box testing is performed by an actor who is not a developer as the tests are made to ensure the functions perform as needed by the customer. Black box testing is also called functional testing and behavioural testing.

2.4 EQUIVALENCE CLASS PARTITIONING

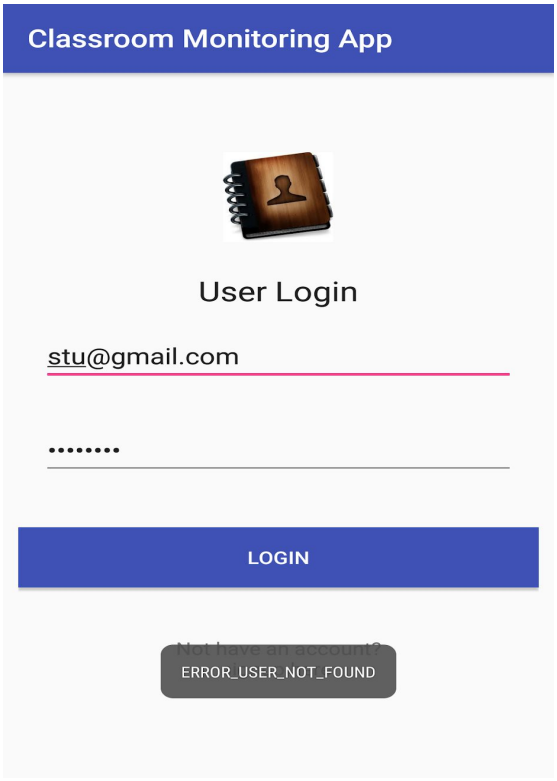
In practical life testing each test case is impossible.Equivalence Class Partitioning is done so that a single test case represents a range of test cases and if test that case we are able to test some common property in all the test cases.
In each of the module we will be dividing our test cases in equivalence class and will test for any one example from each of the equivalence class claiming that it will suffice for all the test cases. Boundary Analysis will also be done as it is generally found that error occurs at the boundary of two equivalence classes.

2.5 TESTING

2.5.1 MODULE : LOGIN

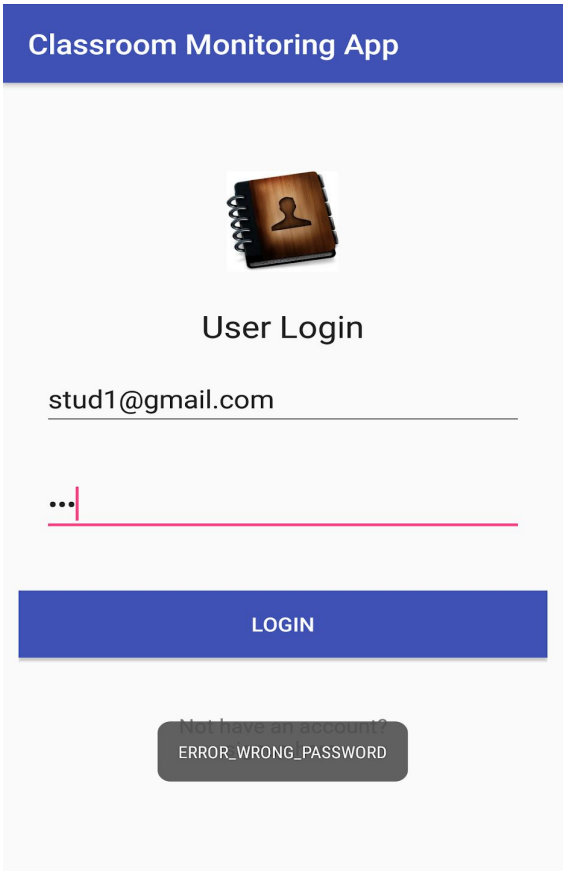
EQUIVALENCE CLASS

1. Description : Input given is **wrong Username**
   Input: Username entered is “stu@gmail.com” (no such user exists in database)
   Password entered “123456” (does not matter)
   Expected Result : Error message displaying username is not found
   Observed Result:



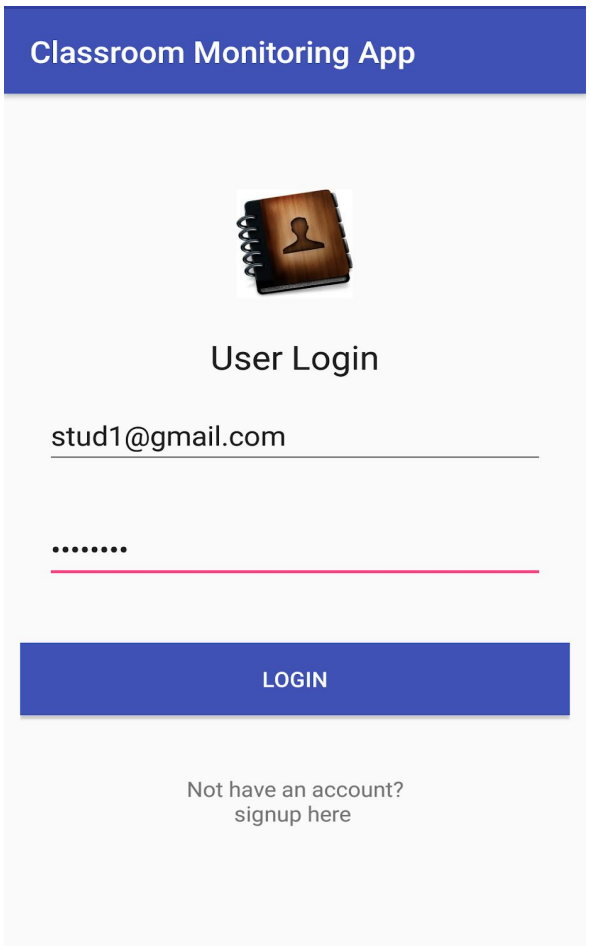
Conclusion: **CORRECT**

2. Description : Input given is **right Username** and **wrong Password**
   Input: Username entered is “stud1@gmail.com” (such user exists in database)
   Password entered “abc” (right password is “password”)
   Expected Result :Error message displaying password is wrong
   Observed Result:



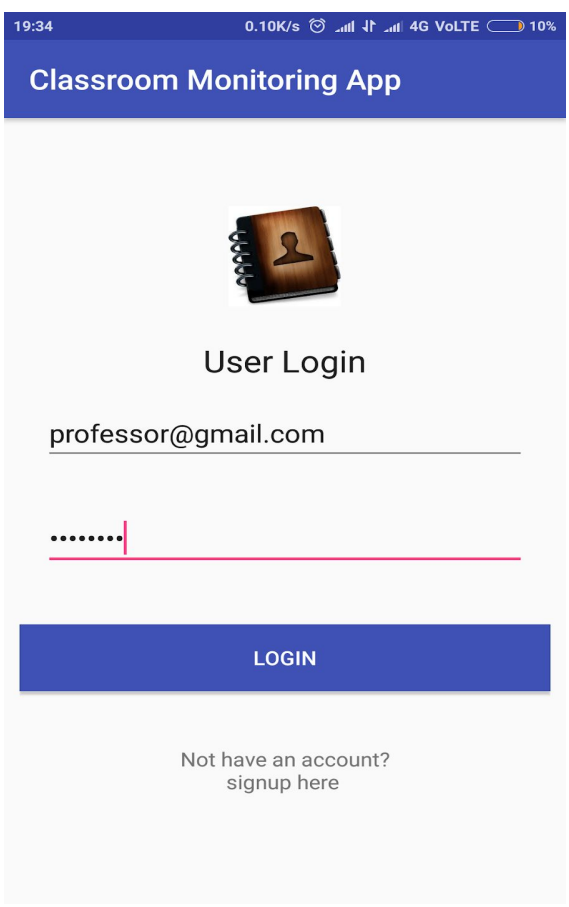
Conclusion: **CORRECT**

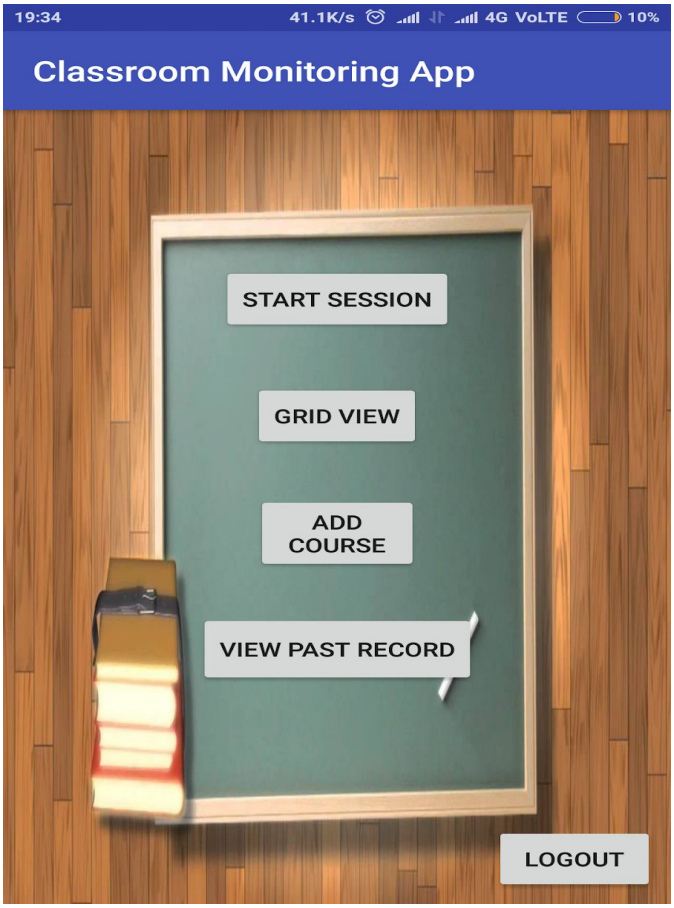
3. Description : Input given is **right Username** and **right Password** , profile is **student**
   Input: Username entered is “stud1@gmail.com” (such student user exists in database)
   Password entered is “password” (right password)
   Expected Result : Successful login Student Main Menu should be displayed
   Observed Result:



Conclusion: **CORRECT**

4. Description : Input given is **right Username** and **right Password** , profile is **professor**
   Input: Username entered is “professor@gmail.com”(such professor user exists)
   Password entered is “password” (right password)
   Expected Result : Successful login Professor Main Menu should be displayed
   Observed Result:





Conclusion: **CORRECT**

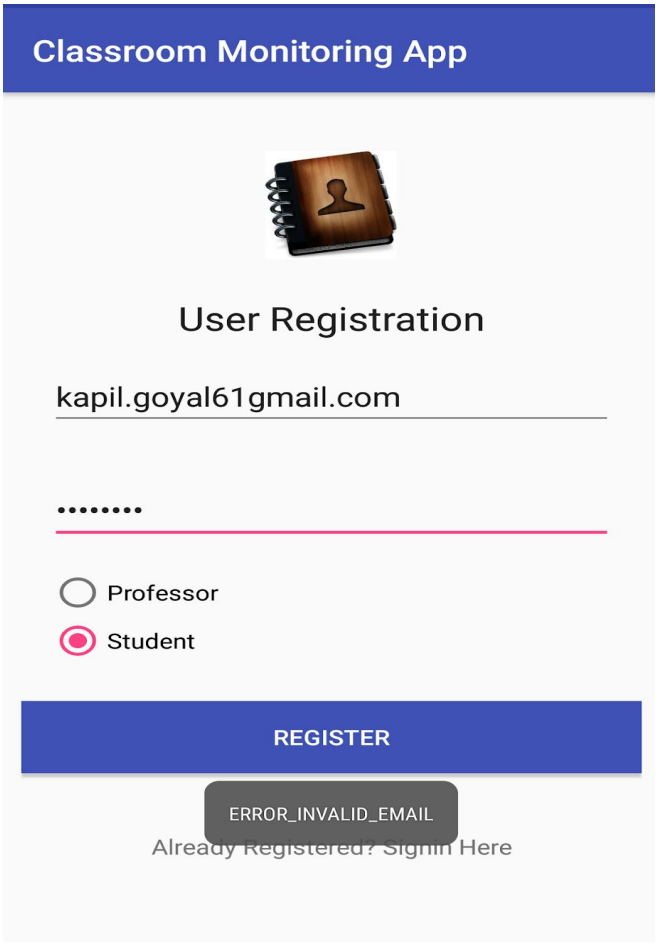
BOUNDARY ANALYSIS

As the elements of equivalence classes are discrete there is no scope for boundary analysis

2.5.2 MODULE : SIGN UP

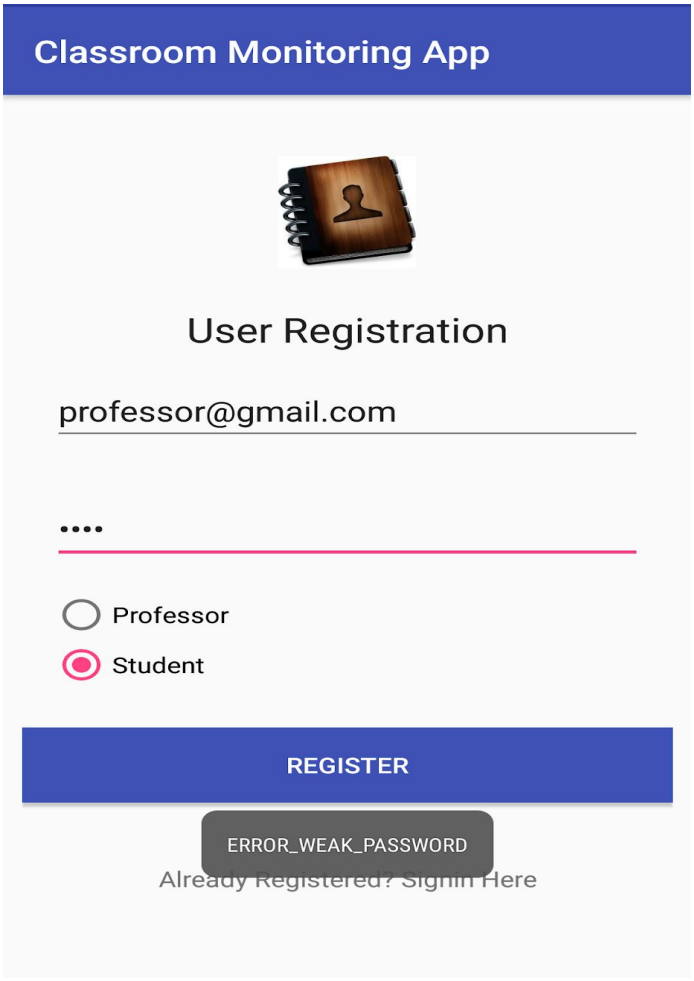
EQUIVALENCE CLASS

1. Description : Input given is **invalid Username**
   Input: Username entered is “kapil.goyal61.gmail.com”
   Password entered “password” (does not matter neither do selected profile)
   Selected profile “student” (does not matter)
   Expected Result : Error message displaying email is invalid
   Observed Result:



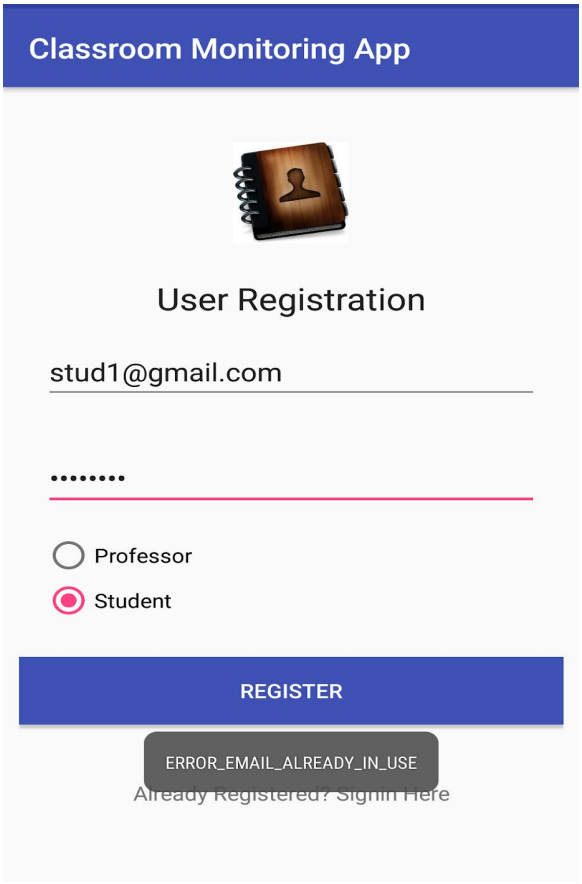
Conclusion: **CORRECT**

2. Description : Input given is **valid Username** and **invalid Password**
   Input: Username entered is “professor@gmail.com”
   Password entered “abc” (Too weak)
   Selected profile “student”(does not matter)
   Expected Result : Error displaying password is weak
   Observed Result:



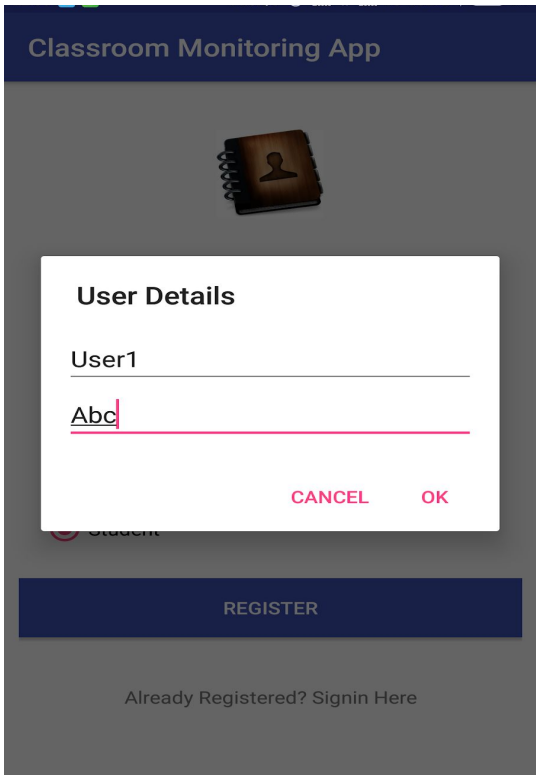
Conclusion: **CORRECT**

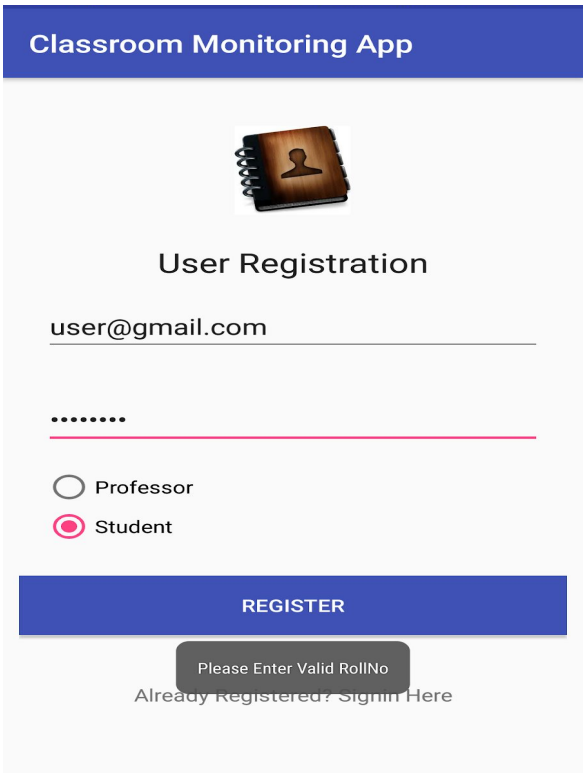
3. Description : Input given is **valid but existing Username** and **valid Password**
   Input: Username entered is “stud1@gmail.com”(already in use by another user)
   Password entered “password” (does not matter)
   Selected profile “student”(does not matter)
   Expected Result : Error displaying email is already in use
   Observed Result:



Conclusion: **CORRECT**

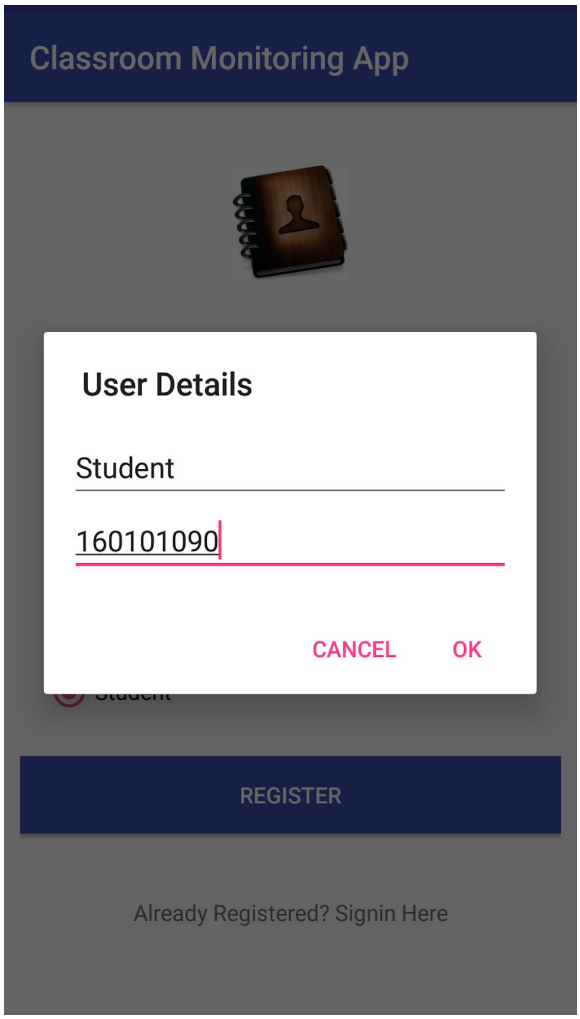
4. Description : Input given is **valid and new Username** and **valid Password** but while entering user details rollNo/ProfId entered is non iteger
   Input: Username entered is “user@gmail.com” (new username)
   Password entered “password” (valid password)
   Selected profile : student (does not matter)
   Name entered “user1” (does not matter)
   Roll No entered is “abc” (Non integer)
   Expected Result :Error message displaying roll no is not valid.
   Observed Result:

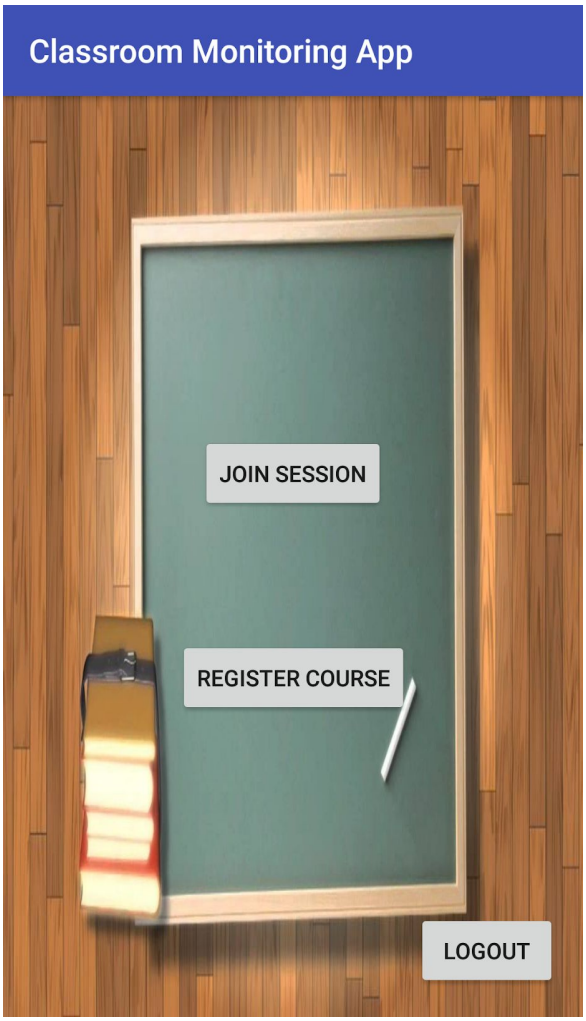




Conclusion: **CORRECT**

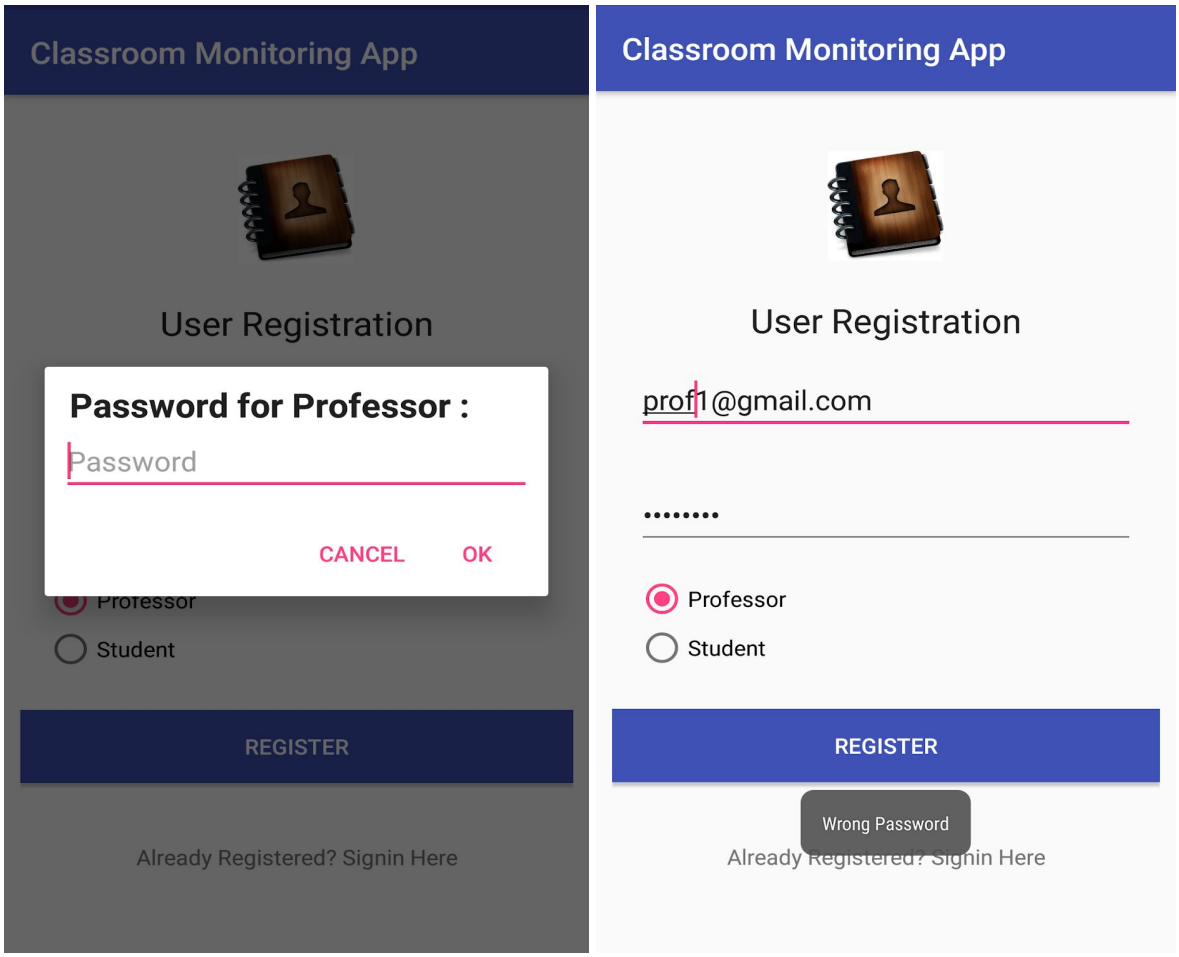
5. Description : Input given is **valid and new Username** and **valid Password** (**User Profile : Student**)
   Input: Username entered is “student@gmail.com” (new username)
   Password entered “password” (valid password)
   Selected profile student
   Expected Result : dialog opens for user details (name and roll no) registers the student and opens student main menu.
   Observed Result:





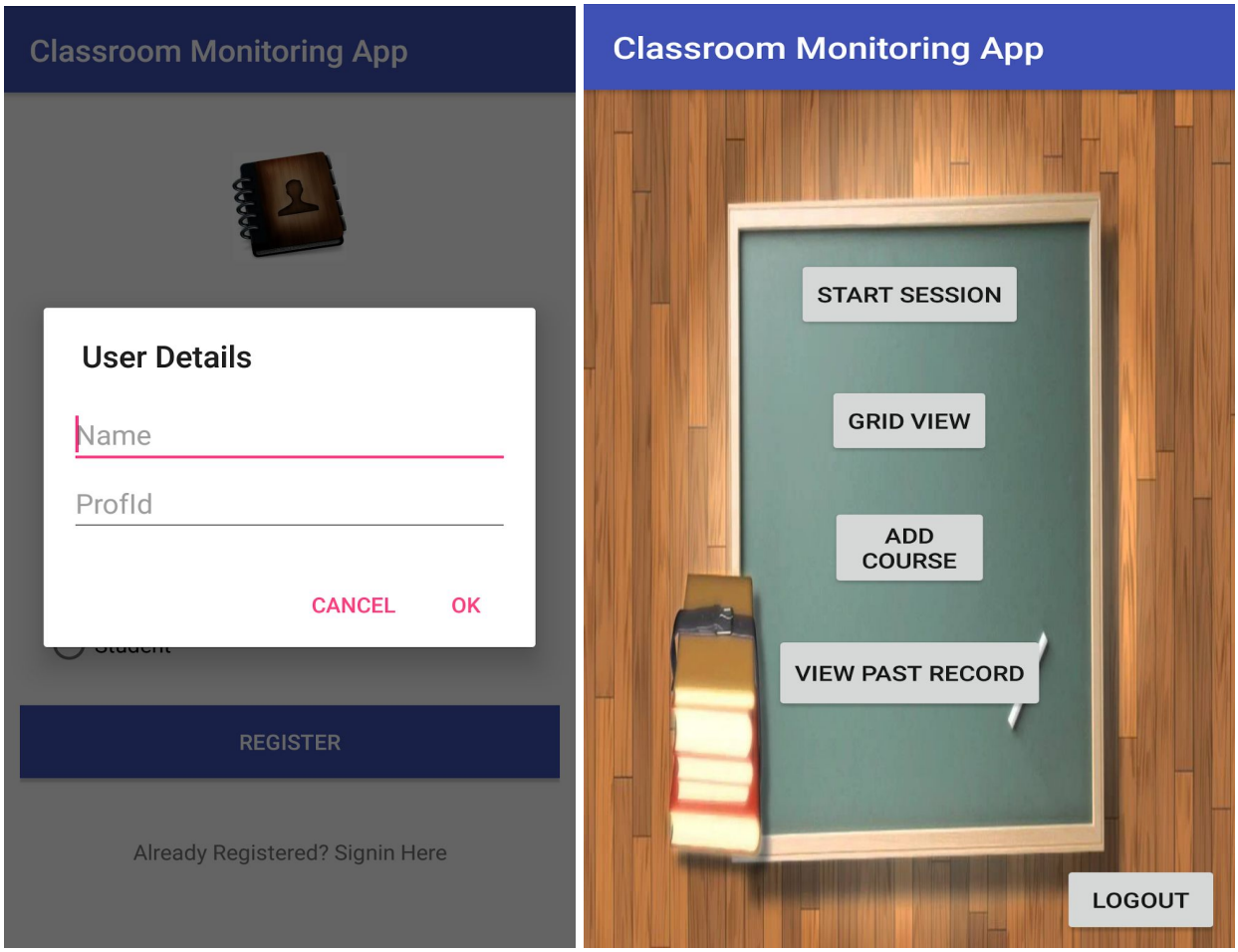
Conclusion: **CORRECT**

6. Description : Input given is **valid and new Username** and **valid Password** (**User Profile : Professor**) and **wrong Professor profile password**
   Input:: Username entered is “prof1@gmail.com”(already in use by another user)
   Password entered “password”
   Selected profile professor
   Professor profile password entered “abc” (wrong password)
   Expected Result : dialog opens for profile password after entering wrong password error message displaying password entered is wrong
   Observed Result:



Conclusion: **CORRECT**

7. Description : Input given is **valid and new Username** and **valid Password** (**User Profile : Professor**) and **right Professor profile password**
   Input: Username entered is “prof1@gmail.com”(already in use by another user)
   Password entered “password”
   Selected profile professor
   Professor profile password entered “123456” (right password)
   Expected Result : dialog opens for profile password after entering right password dialog opens for user details (name and professor id) registers professor and opens professor main menu.
   Observed Result:



Conclusion: **CORRECT**

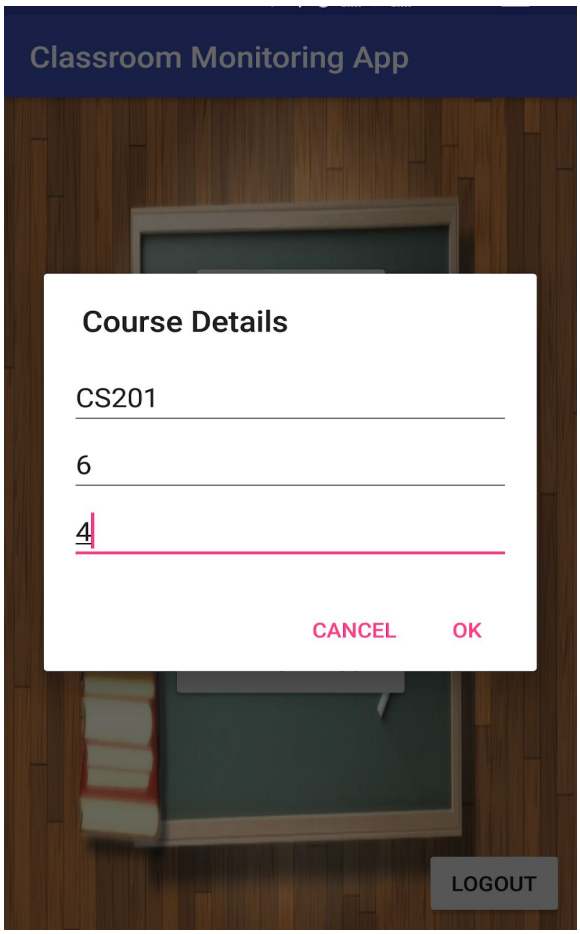
BOUNDARY ANALYSIS

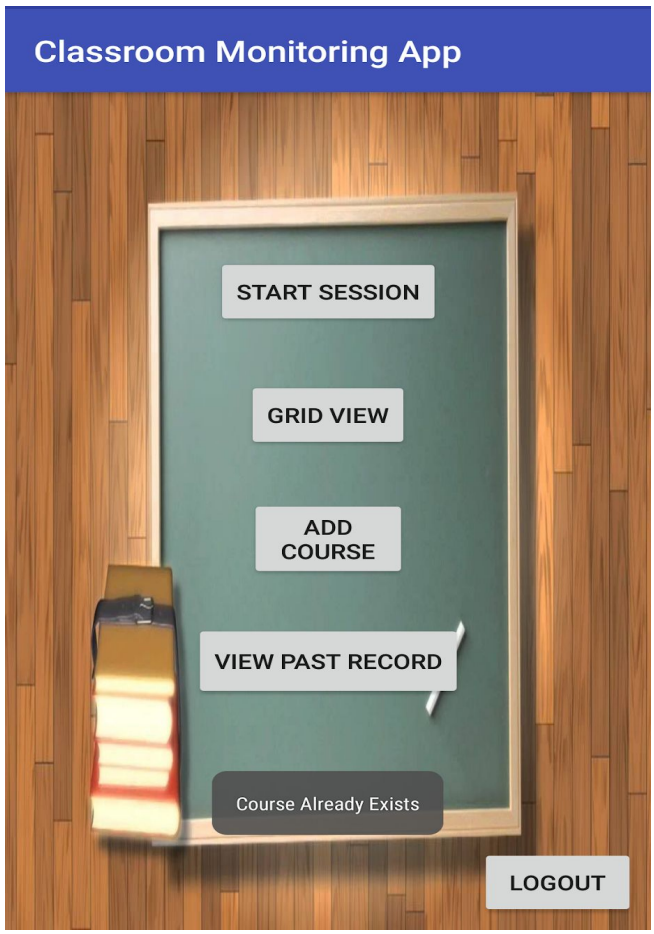
As the elements of equivalence classes are discrete there is no scope for boundary analysis

2.5.3 MODULE : ADD COURSE

EQUIVALENCE CLASS

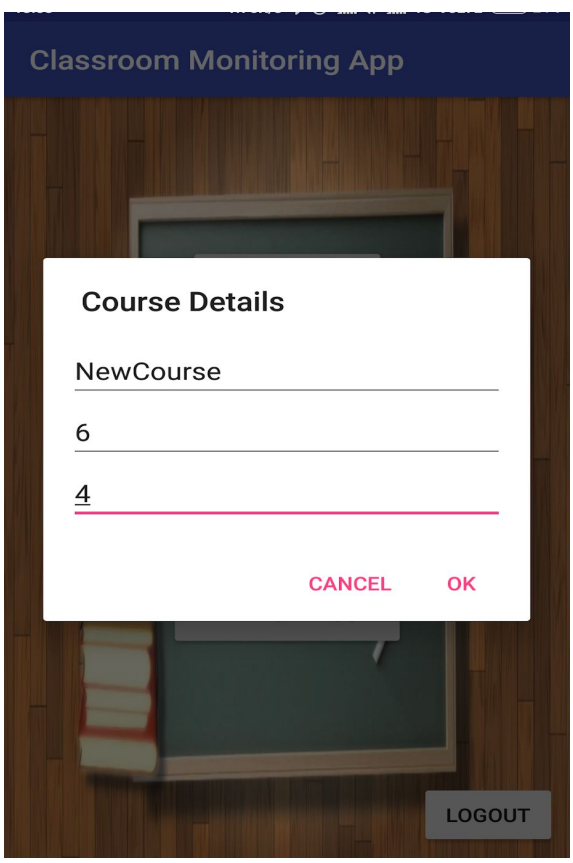
1. Description : Entered course **already exists**
   Input: Course name entered “CS201” (course already exists in database)
   No of rows entered 6
   No of columns entered 4
   Expected Result : Error message showing course already exists
   Observed Result:

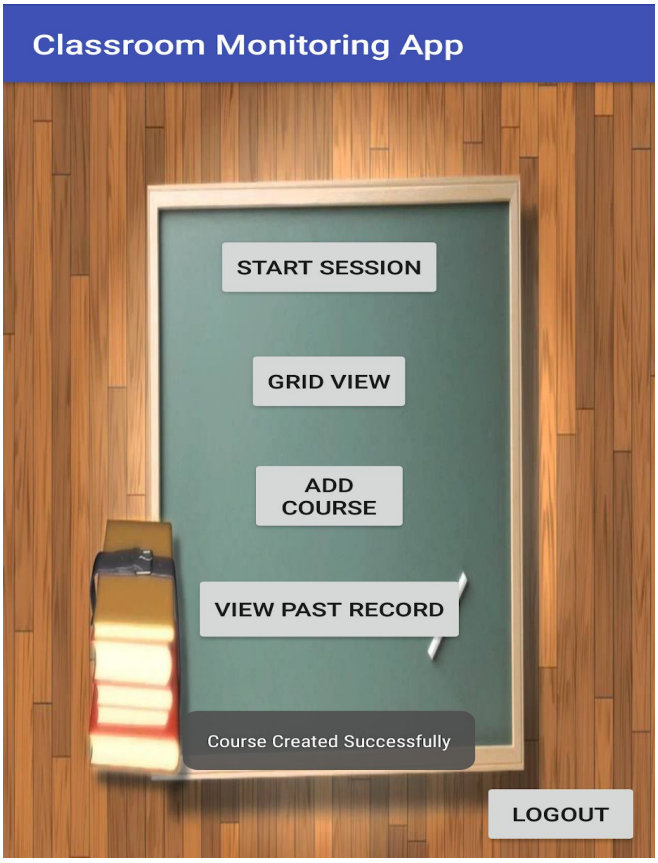




Conclusion: **CORRECT**

2. Description : Entered course **is a new course**
   Input: Course name entered “NewCourse” (no course with same exists in database)
   No of rows entered 6
   No of columns entered 4
   Expected Result : Toast displaying successful creation of course
   Observed Result:





Conclusion: **CORRECT**

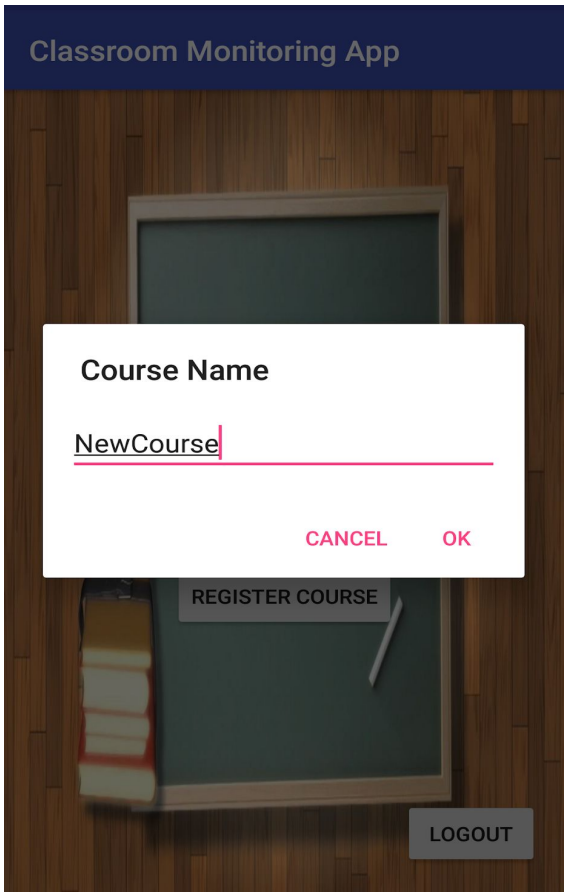
BOUNDARY ANALYSIS

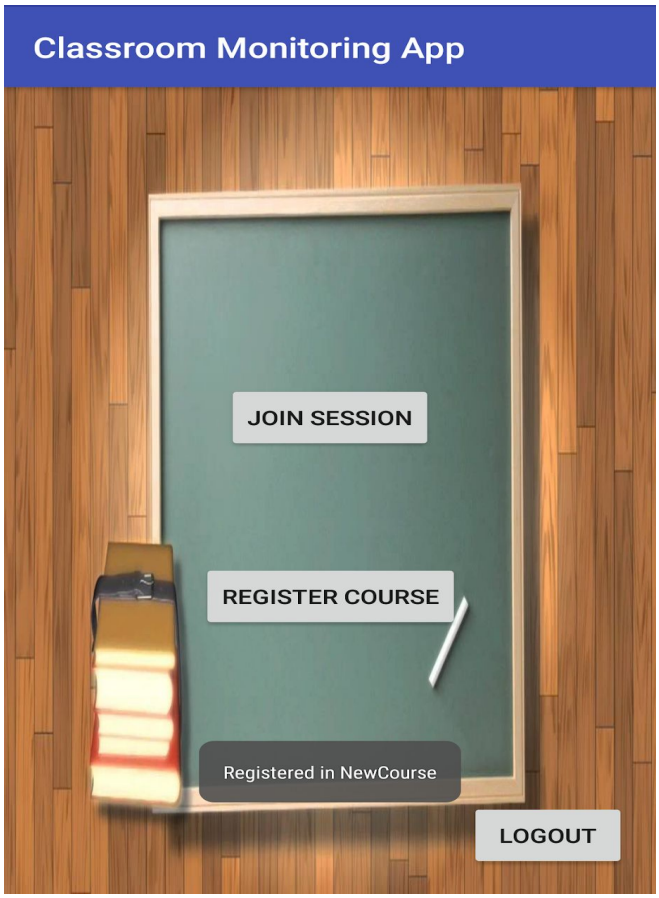
As the elements of equivalence classes are discrete there is no scope for boundary analysis

2.5.4 MODULE : REGISTER COURSE

EQUIVALENCE CLASS

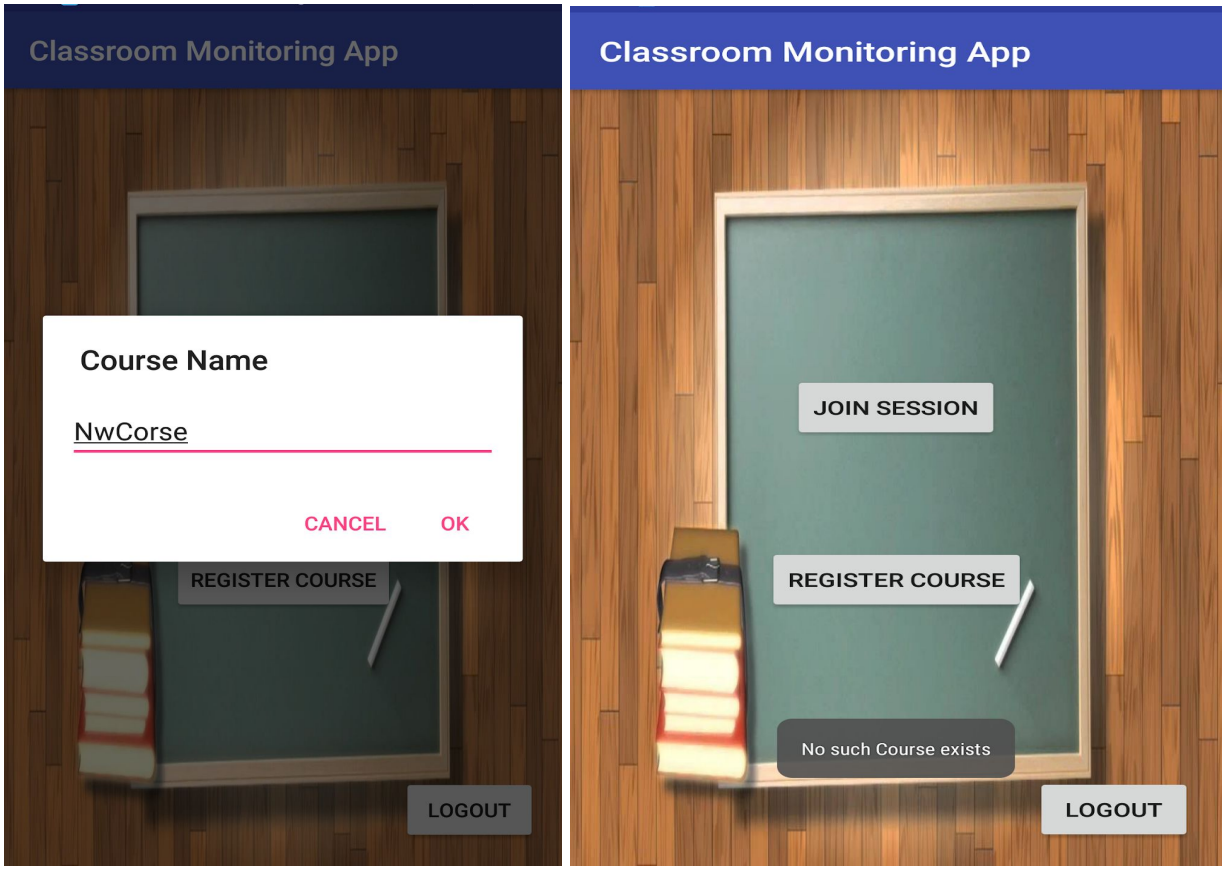
1. Description :Entered course **exists**
   Input: Course name entered “NewCourse” (course exists in database)
   Expected Result : Toast displaying registration is successful
   Observed Result:





Conclusion: **CORRECT**

2. Description :Entered course **does not exist**
   Input:Course name entered “NwCourse” (course does not exists in database)
   Expected Result :Error message displaying no such course exists
   Observed Result:



Conclusion: **CORRECT**

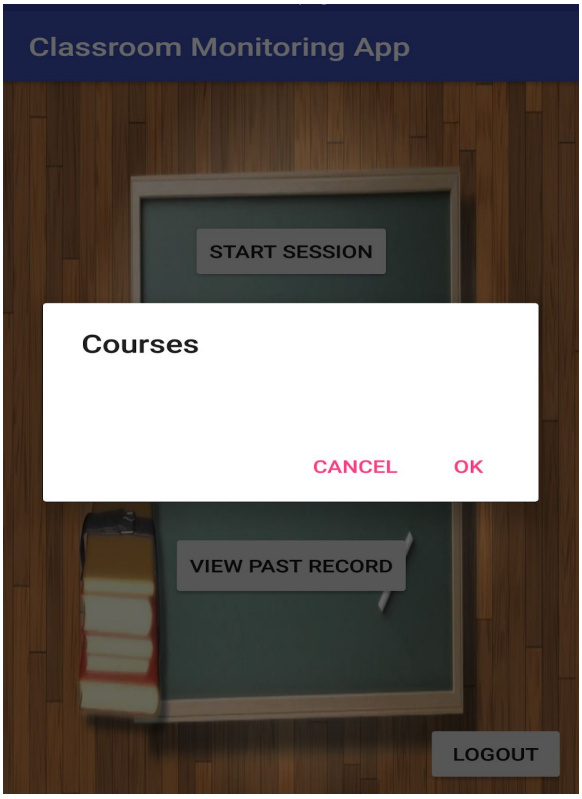
BOUNDARY ANALYSIS

As the elements of equivalence classes are discrete there is no scope for boundary analysis

2.5.5 MODULE : START/END SESSION

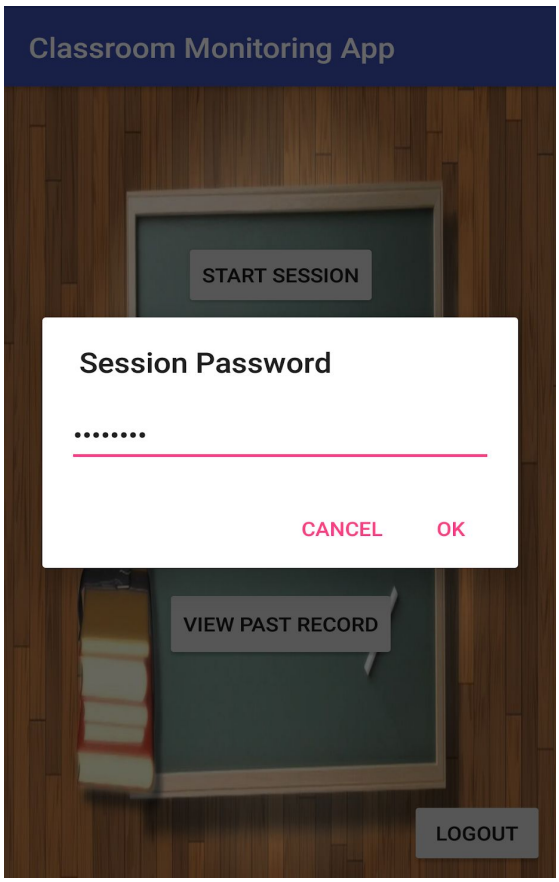
EQUIVALENCE CLASS

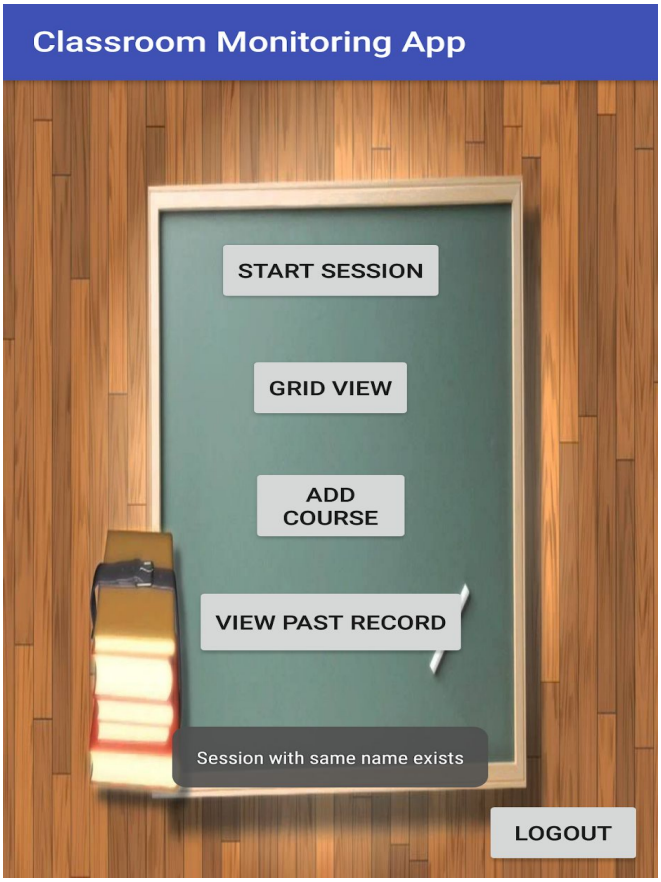
1. Description : Professor **has no course in course list**
   Input: Start Session button is pressed
   Expected Result :Dialog box displayed for selecting course is empty
   Observed Result:



Conclusion: **CORRECT**

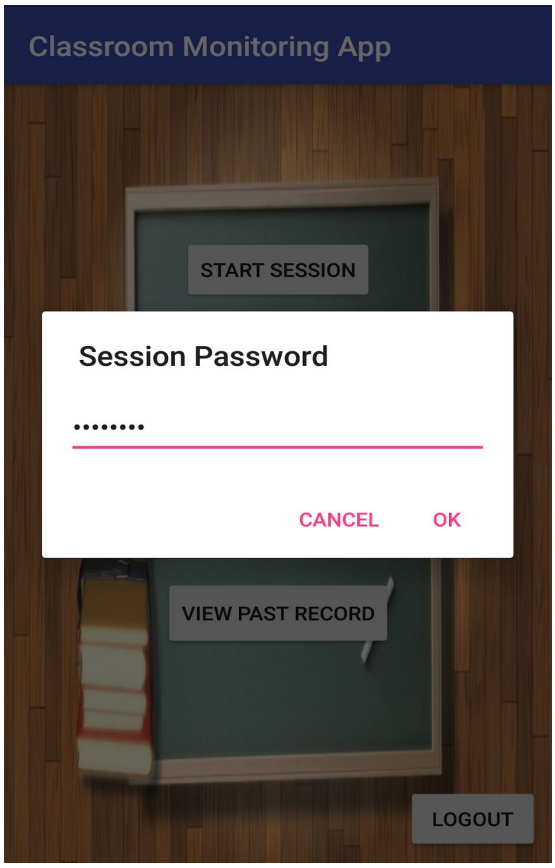
2. Description : Professor **selects a course in course list** but the **session name entered by professor has been used previously** in the same course
   Input: Selected Course is NewCourse
   Session name entered in “session2”(session with same name exists in database)
   Expected Result : Error message displaying session with same exists
   Observed Result:

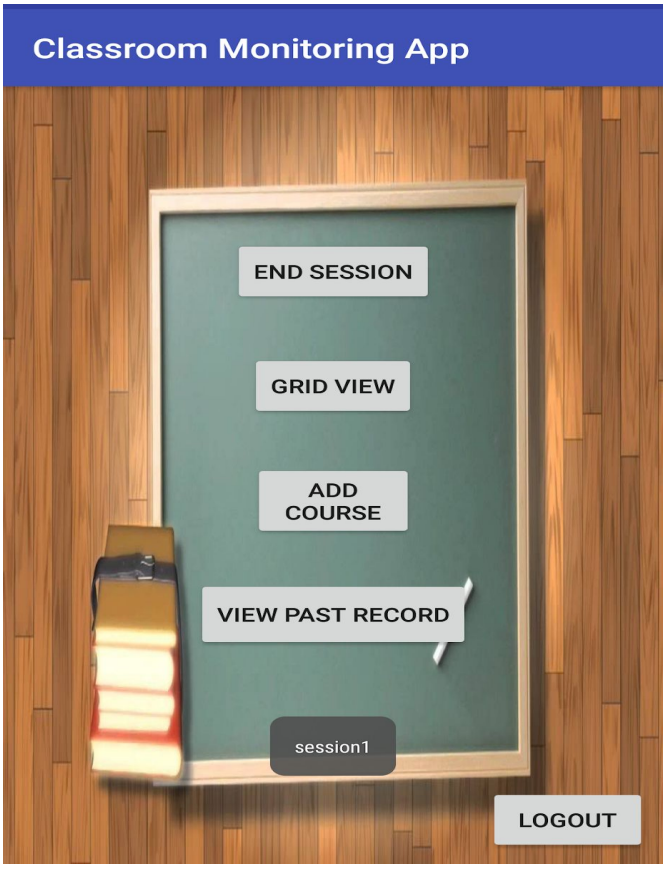




Conclusion: **CORRECT**

3. Description : Professor **selects a course in course list** and the **session name entered by professor has never been used previously** in the same course
   Input: Selected Course is NewCourse
   Session name entered in “session1”(session name is not in database)
   Expected Result : Toast displaying session name entered ,start session button changes to end session.
   Observed Result:





Conclusion: **CORRECT**

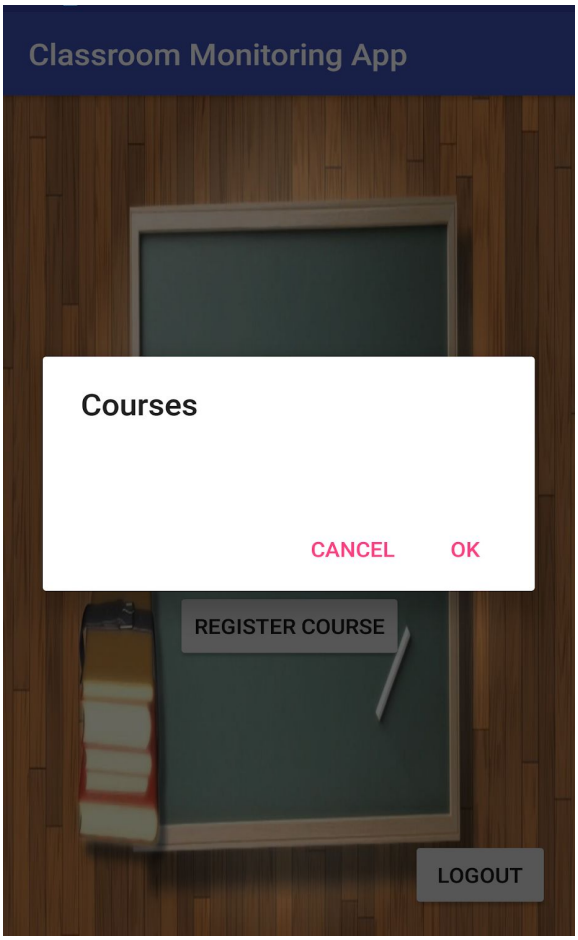
BOUNDARY ANALYSIS

As the elements of equivalence classes are discrete there is no scope for boundary analysis

2.5.6 MODULE : JOIN SESSION

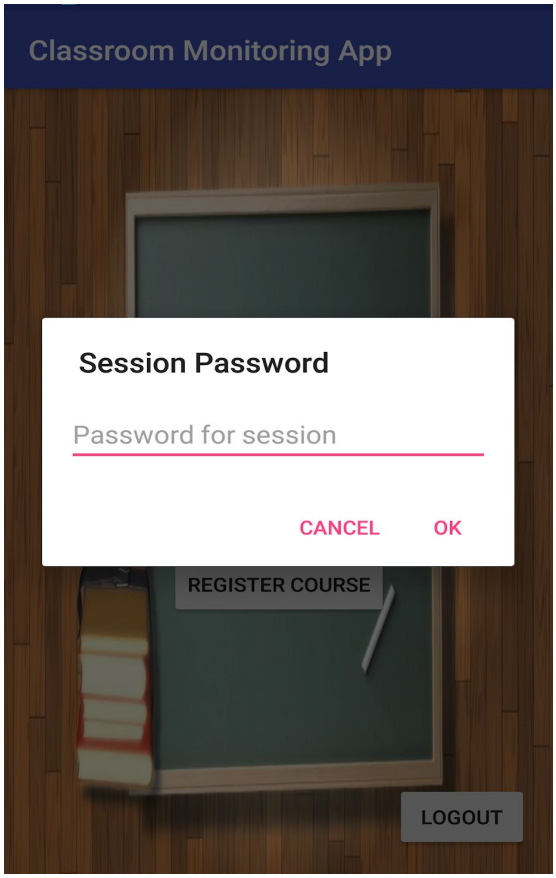
EQUIVALENCE CLASS

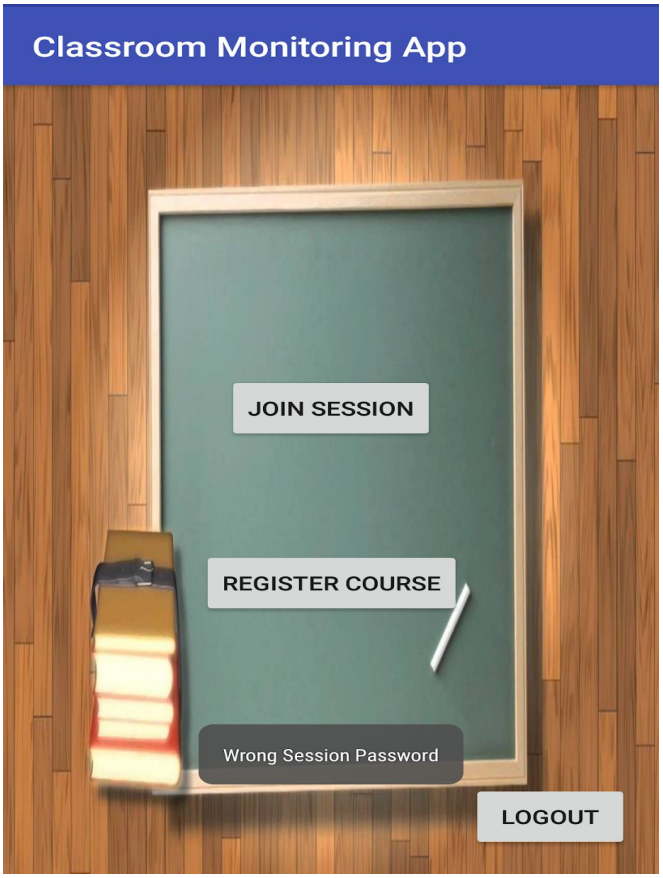
1. Description : Student is **not registered in any course**
   Input:Join Session button is pressed
   Expected Result :Dialog box displayed for selecting course is empty
   Observed Result:



Conclusion: **CORRECT**

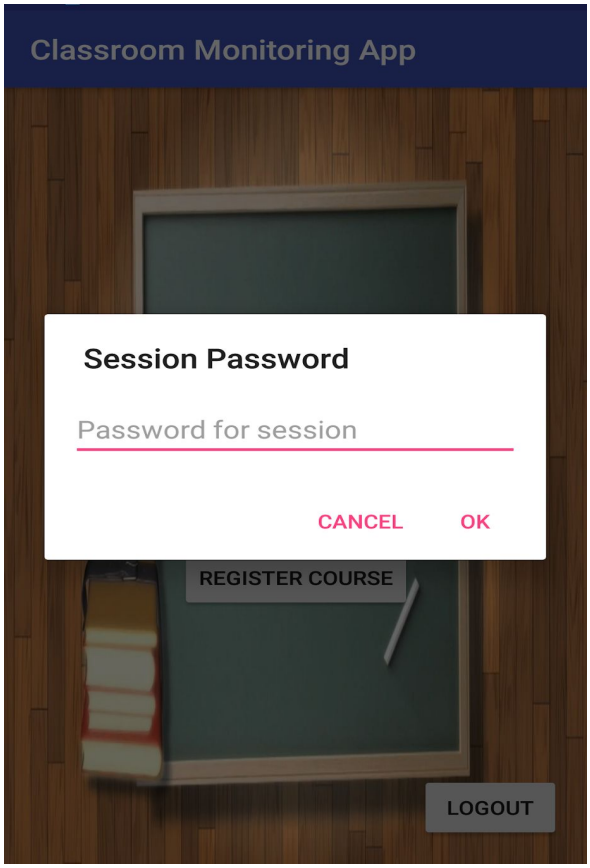
2. Description : Student **selects a registered course** but the **session name entered is not active**
   Input: Selected Course is NewCourse
   Entered session name is “abcde”(the active session name of course is“session2”)
   Expected Result : Error displaying entered session password is wrong
   Observed Result:

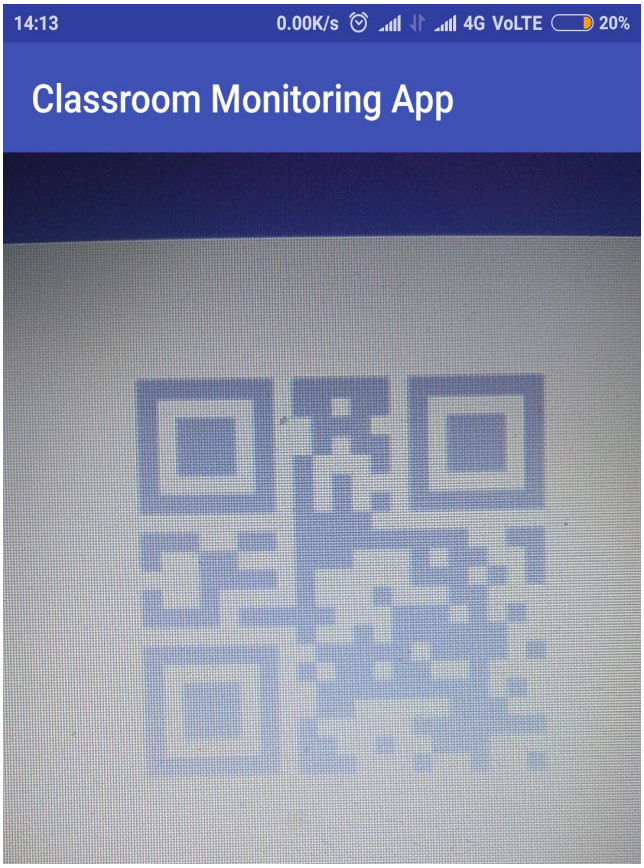




Conclusion: **CORRECT**

3. Description : Student **selects a registered course** and the **session name entered is the active session**
   Input: Selected Course is NewCourse
   Entered session name is “session2”(the active session name)
   Expected Result : QR code scanner opens for scanning the position
   Observed Result:





Conclusion: **CORRECT**

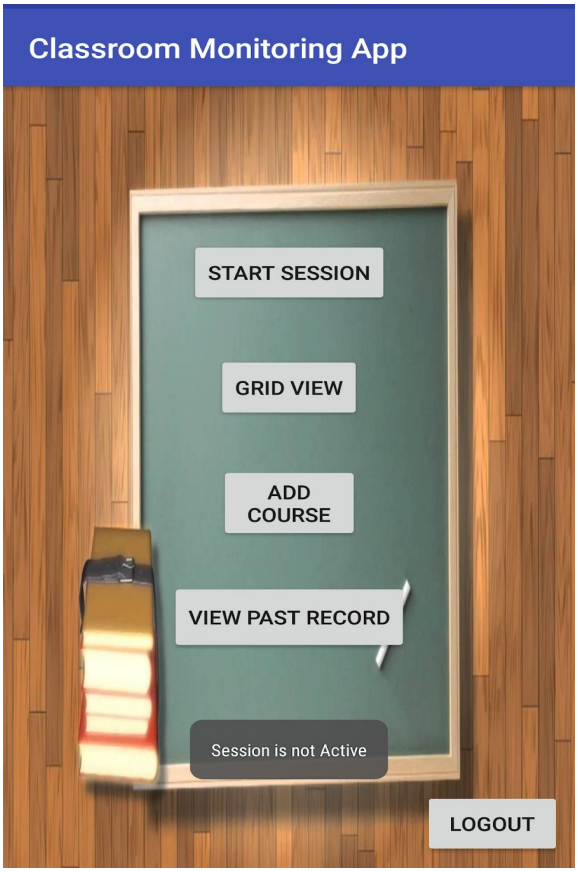
BOUNDARY ANALYSIS

As the elements of equivalence classes are discrete there is no scope for boundary analysis

2.5.7 MODULE : GRID VIEW

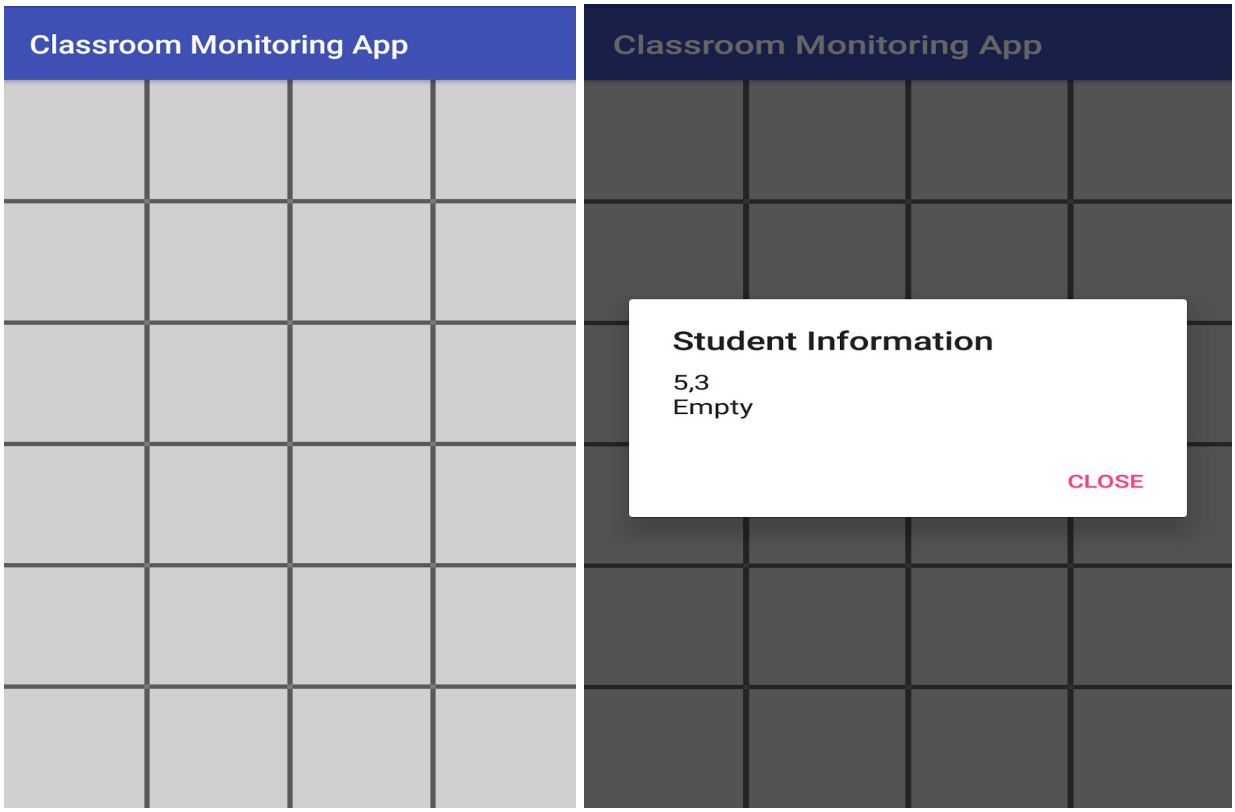
EQUIVALENCE CLASS

1. Description : Professor **has no active session**
   Input:No session is active but Grid view button is pressed
   Expected Result :Error message displaying no session is active
   Observed Result:



Conclusion: **CORRECT**

2. Description : Professor **has an active session** number of **students present is zero**
   Input: A session is active and Grid View button is pressed but no students has joined
   Information about seat(5,3) is requested
   Expected Result : 2D view of class divided in box according to number of rows and columns but clicking on any seat shows the position empty
   Observed Result:



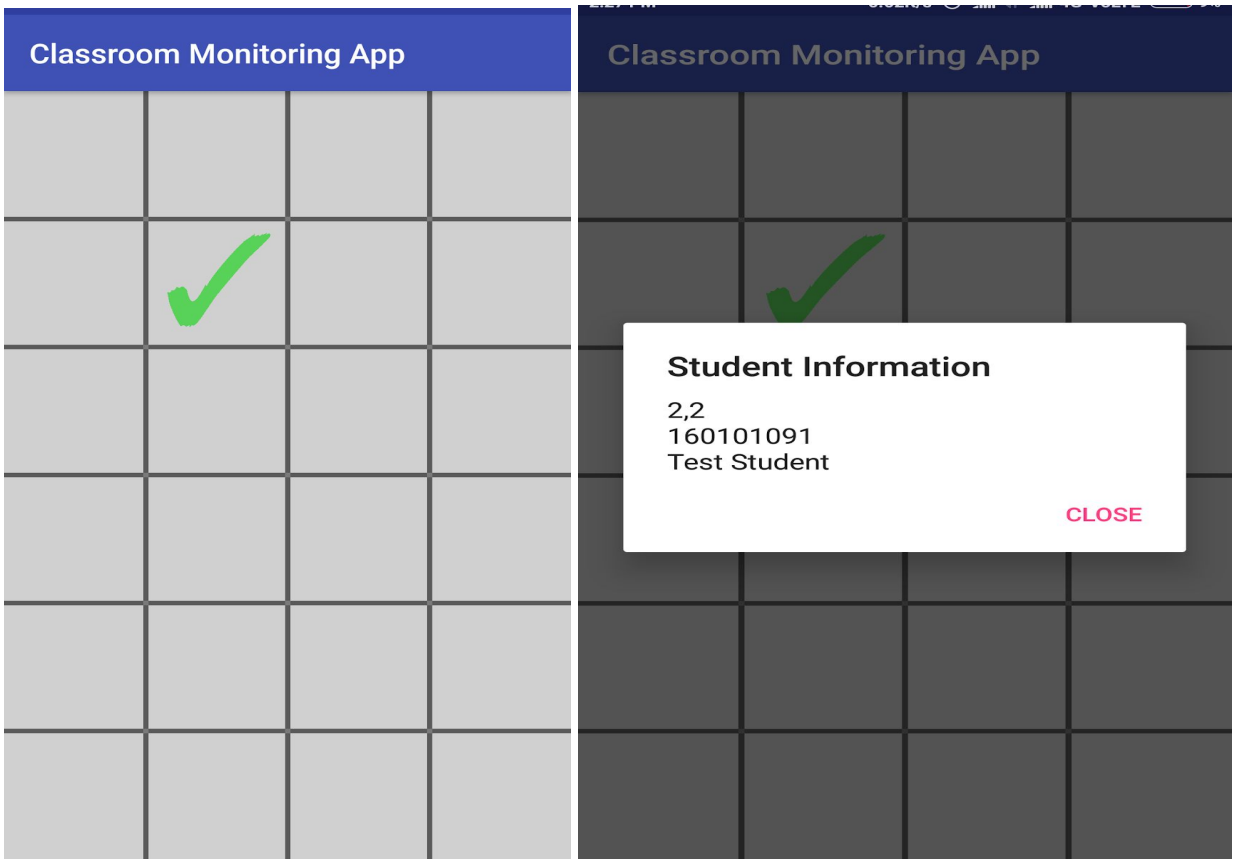
Conclusion: **CORRECT**

3. Description : Professor **has an active session** number of **students present is non zero**

   Input:A session is active and Grid View button is pressed one student has joined at (2,2)
   Details are requested for seat (2,2).

   Expected Result : 2D view of class divided in box according to number of rows and columns but clicking on seat of students shows student information sitting on the position .

   Observed Result:



   Conclusion: **CORRECT**

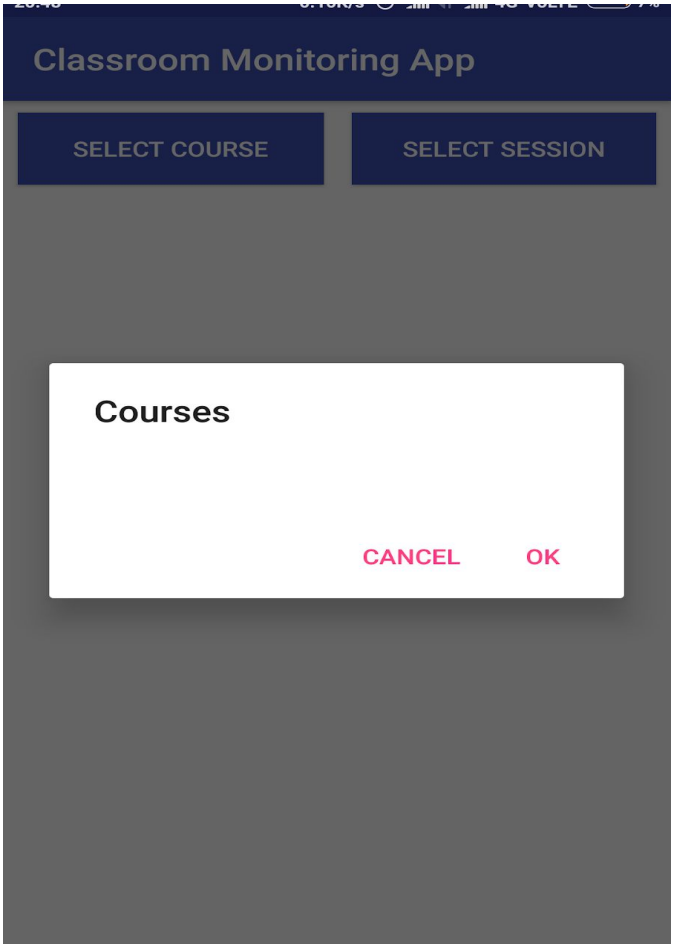
BOUNDARY ANALYSIS

Boundary analysis is done at number of attending student = 1.

2.5.8 MODULE : VIEW PAST ATTENDANCE

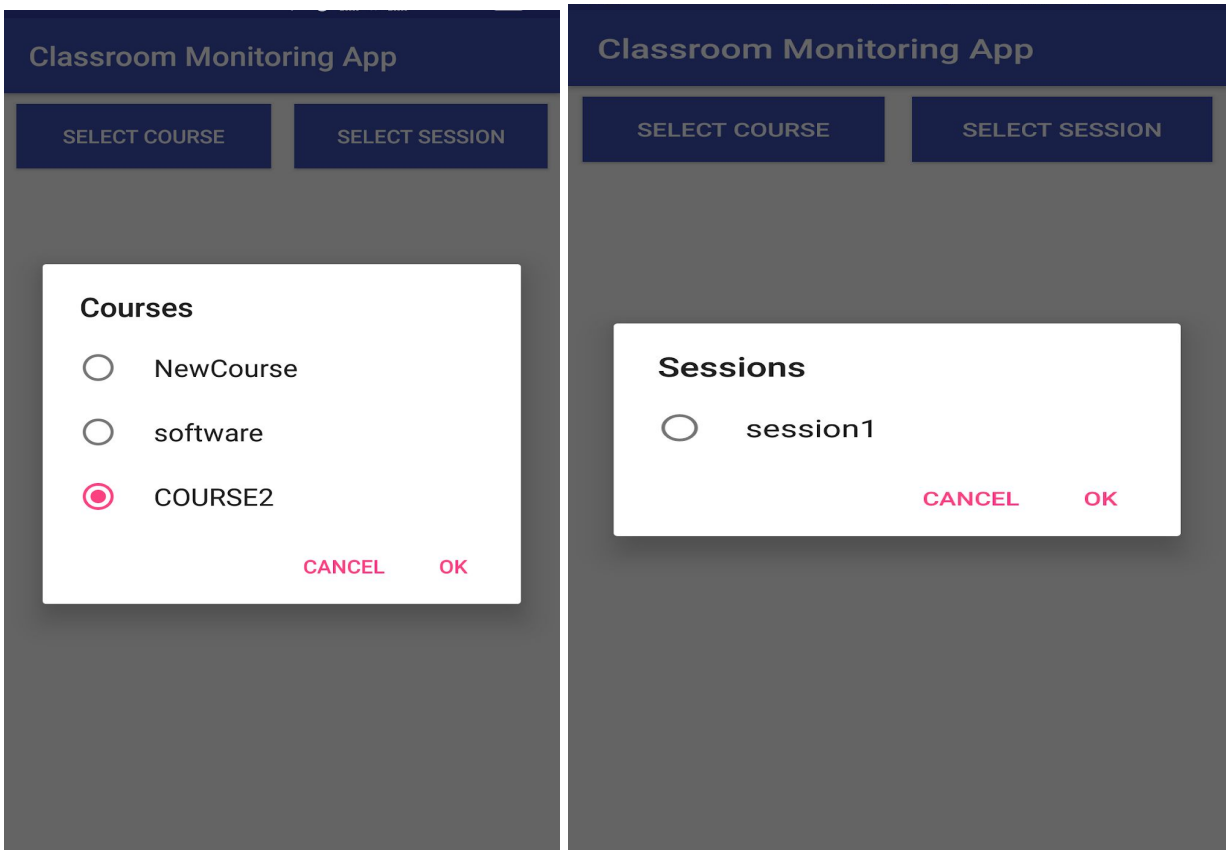
EQUIVALENCE CLASS

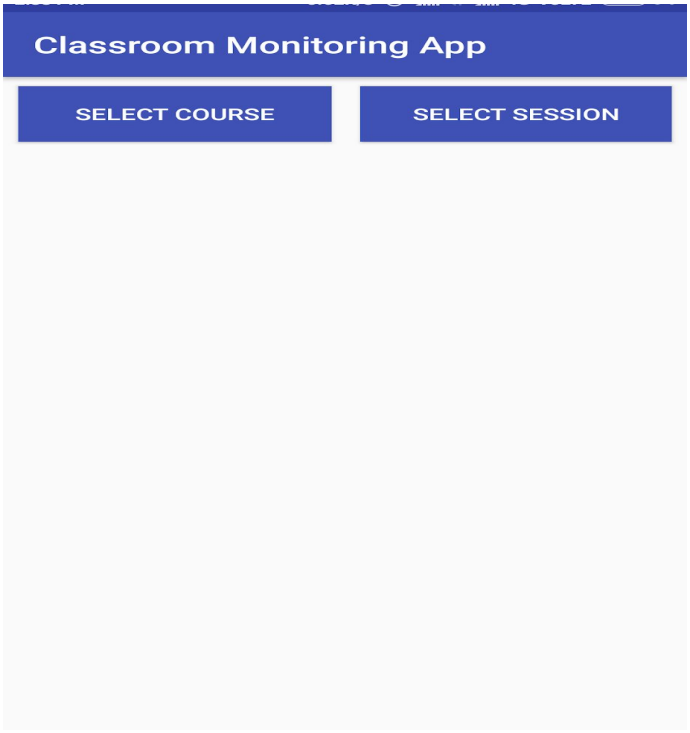
1. Description : Professor **has no course in course list**
   Input: Select Course Button is pressed
   Expected Result : Dialog box showing list of courses is empty
   Observed Result:



Conclusion: **CORRECT**

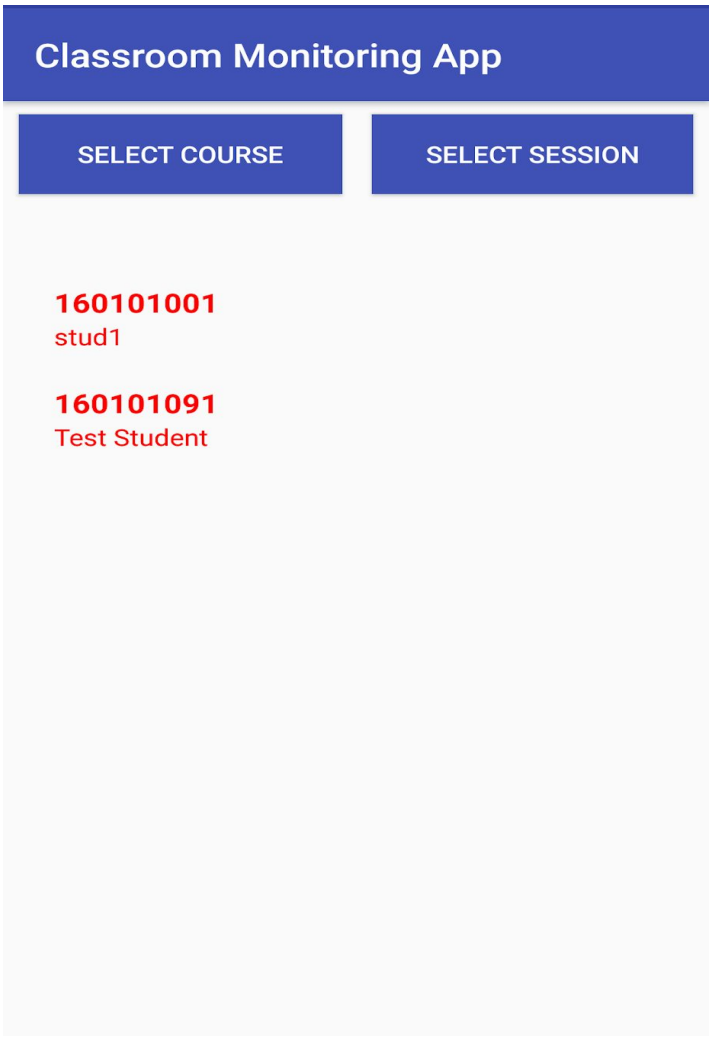
2. Description : Professor **selects a course in course list** and selects a **session** but **number of students registered in the course is zero**
   Input: Selected course is “COURSE2” (registered student is zero)
   Selected session is “session1”
   Expected Result : no student is displayed
   Observed Result:





Conclusion: **CORRECT**

3. Description : Professor **selects a course in course list** with **non zero session** and **number of students registered in the course is non zero** but **zero student present**
   Input: Selected course is “NewCourse” (2 students are registered in the course)
   Selected session is “session1” (no students attended the session)
   Expected Result : Both students will be displayed red
   Observed Result:



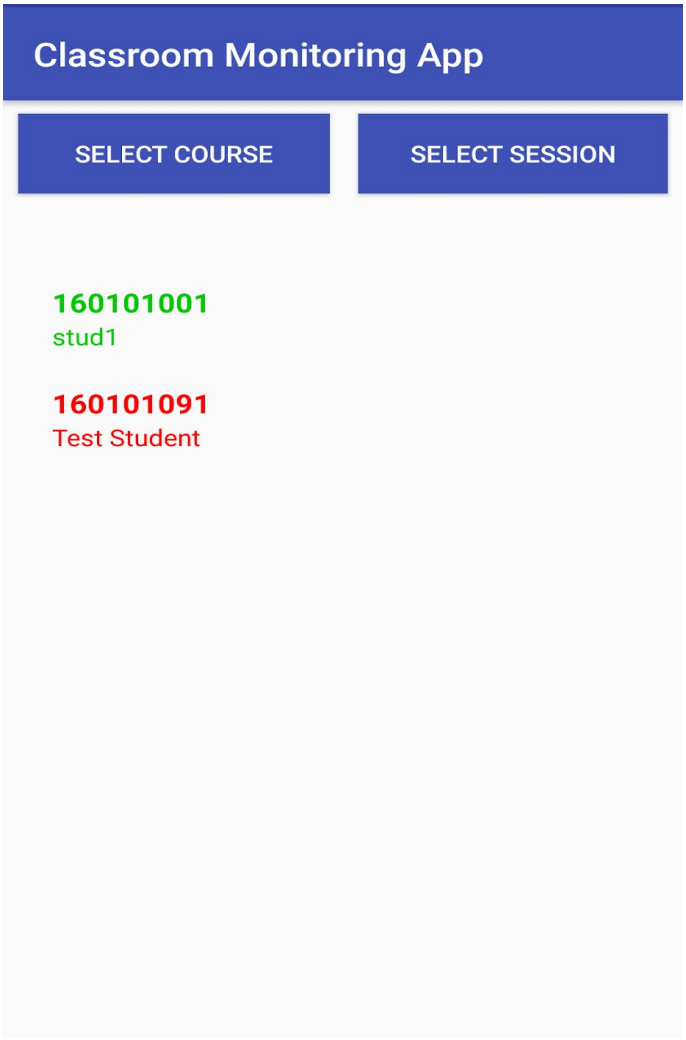
Conclusion: **CORRECT**

4. Description : Professor **selects a course in course list** with **non zero session** and **number of students registered in the course is non zero** with **non zero student present**

   Input: Selected course is “NewCourse” (2 students are registered in the course)
   Selected session is “session2” (1 students attended the session)

   Expected Result :The present student will be displayed green the absent red

   Observed Result:



   Conclusion: **CORRECT**

BOUNDARY ANALYSIS

Boundary analysis is done at number of attending student = 1.

3.WHITE BOX TESTING

White-box testing is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The White Box Testing basically consists of four strategies to the test the source code statement coverage :

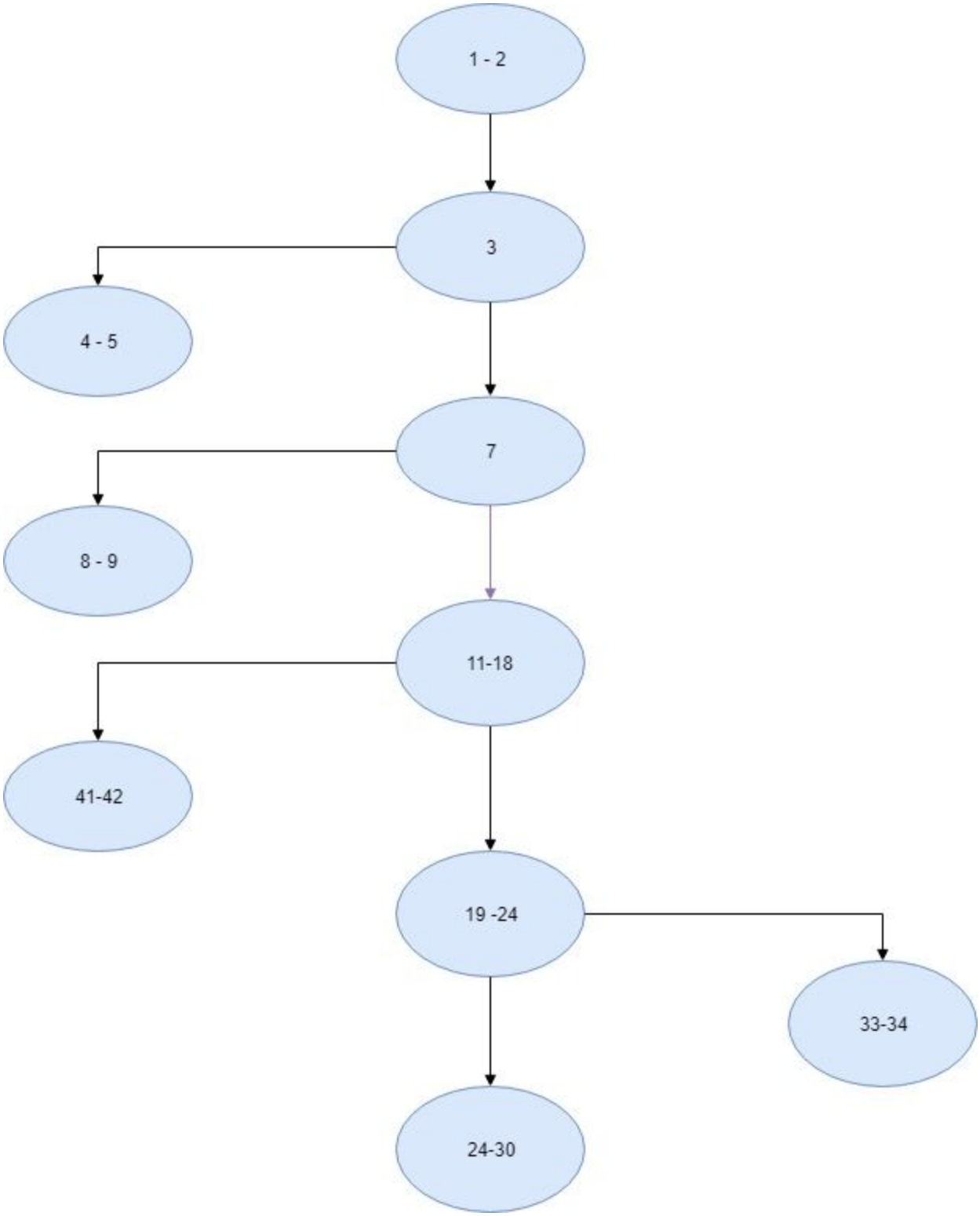
1. Branch coverage
2. Condition coverage
3. Path coverage

The following modules present in the source code are tested using path coverage as follows:

3.1 MODULE : LOGIN

FUNCTIONS: userLogin()

```
String email = editTextEmail.getText().toString().trim();
String password  = editTextPassword.getText().toString().trim();
if(TextUtils.isEmpty(email)){
    Toast.makeText(this,"Please enter email",Toast.LENGTH_LONG).show();
    return;
}
if(TextUtils.isEmpty(password)){
    Toast.makeText(this,"Please enter password",Toast.LENGTH_LONG).show();
    return;
}
progressDialog.setMessage("Logging you in Please Wait...");
progressDialog.show();
mFirebaseAuthenticationDatabase.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                progressDialog.dismiss();
                if(task.isSuccessful()){
                    mUserDatabaseReference.addValueEventListener(new ValueEventListener() {
                        @Override
                        public void onDataChange(DataSnapshot dataSnapshot) {
                            try {
                                User user = dataSnapshot.getValue(User.class);
                                if(!user.isPermission()){
                                    finish();
                                    Intent i;
                                    intent =new Intent(MainActivity.this, StudentMainActivity.class);
                                    intent.putExtra("rollNo",Integer.toString(user.getRoll()));
                                    intent.putExtra("name",user.getName());
                                    startActivity(i);
                                }
                                else {
                                    finish();
                                    startActivity(new Intent(MainActivity.this,ProfessorActivity.class));
                                }
                            }
                            catch (Exception exception){
                                exception.printStackTrace();
                } } });}
                else {
                    error_message = ((FirebaseAuthException) task.getException()).getErrorCode();
                    Toast.makeText(MainActivity.this,error_message,Toast.LENGTH_SHORT).show();
                }}});
```


1 - 2
3
4 - 5
7
8 - 9
11-18
41-42
19 -24
33-34
24-30

Total No of Linearly Independent Paths : 5

Path 1: 1-2 => 3 => 4- 5
Description : This path is executed if email field is left empty
Input :
Email - “”
Password- “abcdefg” (does not matter)
Output :Toast with message “ Please Enter Email ” appears on screen

Path 2: 1-2 => 3 => 7 => 8- 9
Description : This path is executed if password field is left empty
Input :
Email - “prof@gmail.com” (valid)
Password- “”
Output :Toast with message “ Please Enter Password ” appears on screen

Path 3: 1- 2 => 3 => 7 => 11-18 => 41-42
Description : This path is executed if the user credentials are invalid
Input :
Email -”prof@gmail.com” (valid user exists in database)
Password - “12456” (incorrect password)
Output : Toast with message “ERROR_WRONG_PASSWORD” appears on screen

Path 4: 1 => 2 => 3 => 7 => 11-18 => 19-24 => 24-30
Description : This path is followed when the user who is logging in is a student
Input :
Email -”stud@gmail.com” (valid user exists in database)
Password - “12456” (correct password)
Output : the user is logged into the app and is redirected to StudentMainActivity

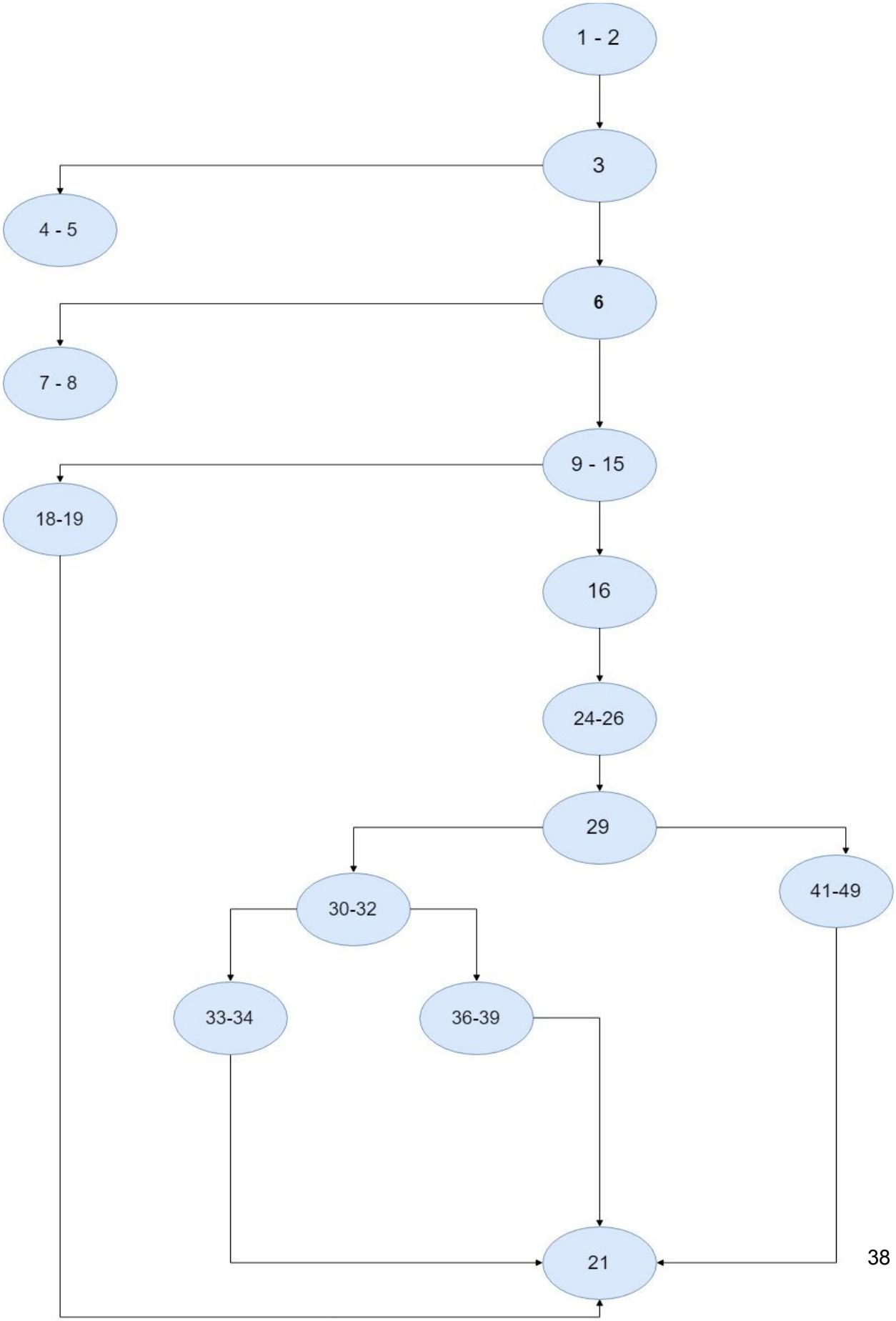
Path 5: 1 => 2 => 3 => 7 => 11-18 => 19-24 => 33-34
Description : This path is followed when the user who is logging in is a professor
Input :
Email -”prof@gmail.com” (valid user exists in database)
Password - “12456” (correct password)
Output : the user is logged into the app and is redirected to ProfessorMainActivity

3.2 MODULE : SIGN UP

FUNCTIONS:registerUser()

```
String email = editTextEmail.getText().toString().trim();
String password  = editTextPassword.getText().toString().trim();
if(TextUtils.isEmpty(email)){
    Toast.makeText(this,"Please enter email",Toast.LENGTH_LONG).show();
    return; }
if(TextUtils.isEmpty(password)){
    Toast.makeText(this,"Please enter password",Toast.LENGTH_LONG).show();
    return; }
progressDialog.setMessage("Registering Please Wait...");
progressDialog.show();
firebaseAuth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if(task.isSuccessful()){
                    getUserDetails();
                }else{
                    String error_message=((FirebaseAuthException) task.getException()).getErrorCode();
                    Toast.makeText(SignupActivity.this,error_message,Toast.LENGTH_SHORT).show();
                }
                progressDialog.dismiss();
            }});
public void getUserDetails(){
    SignupDialog object = new SignupDialog();
    object.setPermission(isProfessor);
    object.show(getSupportFragmentManager(),"dialogue");
}
public void applyText(String name,String roll) {
    try {
        String user_id=mFirebaseAuth.getCurrentUser().getUid().toString();
        mDatabaseReference.child(user_id).setValue(new User(isProfessor, Integer.parseInt(roll),name));
```

```
        if (isProfessor) {
            startActivity(new
            Intent(this, ProfessorActivity.class));
        } else {
            Intent intent = new Intent(this, StudentMainActivity.class);
            intent.putExtra("rollNo", roll);
            intent.putExtra("name", name);
            startActivity(intent);
        }
    } catch (NumberFormatException e) {
        final FirebaseUser user = firebaseAuth.getCurrentUser();
        AuthCredential credential = EmailAuthProvider.getCredential("user@example.com", "password1234");
        user.reauthenticate(credential).addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                user.delete().addOnCompleteListener(new OnCompleteListener<Void>() {
                    });}});
        Toast.makeText(this,"Please Enter Valid RollNo",Toast.LENGTH_SHORT).show();
    }}
```


1 - 2
3
4 - 5
6
7 - 8
9 - 15
18-19
16
24-26
29
30-32
41-49
33-34
36-39
21

Total No of Linearly Independent Paths : 6

Path 1: 1-2=> 3 => 4-5

Description : This path is executed if the email field is left empty

Input :

Email -""

Password - "12456"

Output : Toast with message " Please Enter Email " appears on screen

Path 2: 1-2 => 3 => 6 => 7-8

Description : This path is executed if password field is left empty

Input :

Email - "prof@gmail.com" (valid)

Password- ""

Output :Toast with message " Please Enter Password " appears on screen

Path 3: 1- 2 => 3 => 6 => 9-15 => 18-19 =>21

Description : This path is executed if user enters a used/existing email

Input :

Email - "prof1@gmail.com" (already existing in database)

Password- "123456"

Output :Toast with message " ERROR_INVALID_EMAIL " appears on screen

Path 4: 1- 2 => 3 => 6 => 9-15 =>16 => 24-26 => 29 => 30-32 => 33-34 =>21

Description : This path is executed if user enters a valid new credentials and is a professor and user details entered are valid (Id is an integer)

Input :

Email - "prof1@gmail.com" (new email not in database)

Professor profile password - "123456" (to be professor this must be the input)

Password- "password"

Name - "Professor1"

ProfID - "1001"

Output :ProfessorMainActivity opens

Path 5: 1- 2 => 3 => 6 => 9-15 =>16 => 24-26 => 29 => 30-32 => 36-39 =>21

Description : This path is executed if user enters a valid new credentials and is a professor and user details entered are valid (Id is an integer)

Input :

Email - "stud1@gmail.com" (new email not in database)

Password- "password"

Name - "Student1"

RollNo - "160101001"

Output :StudentMainActivity opens

Path 6: 1- 2 => 3 => 6 => 9-15 =>16 => 24-26 => 29 => 41-49 =>21
Description : This path is executed if user enters a valid new credentials but user credentials are not valid RollNo/ProfId is not an integer.
Input :

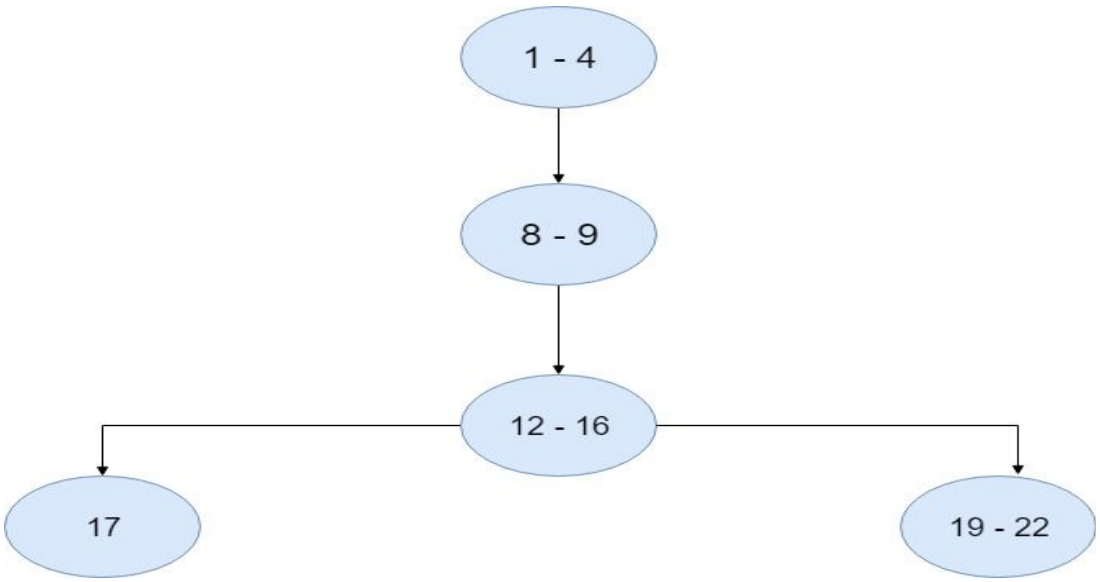
Email - “stud1@gmail.com” (new email not in database)
Password- “password”
Name - “Student1”
RollNo/ProfID - “abc”

Output :Toast with message “ Please Enter Valid RollNo ” appears on screen

3.3 MODULE : ADD COURSE

FUNCTIONS: onClick() , openDialoge() , saveData()

```
1  buttonAddCourse.setOnClickListener(new View.OnClickListener() {
2      @Override
3      public void onClick(View v) {
4          openDialogetoAddCourse();
5      }
6  });
7    public void openDialoge(){
8        dialogue object = new dialogue();
9          object.show(getSupportFragmentManager(),"dialogue");
10     }
11 public void saveData(final String course, final int row, final int column) {
12     final int exist;
13     mAllCourseReference.addListenerForSingleValueEvent(new ValueEventListener() {
14         @Override
15         public void onDataChange(DataSnapshot dataSnapshot) {
16             if(dataSnapshot.hasChild(course)){
17              Toast.makeText(ProfessorActivity.this,"Course Already Exists",Toast.LENGTH_SHORT).show();}
18             else {
19              Toast.makeText(ProfessorActivity.this,"Course Created Successfully",Toast.LENGTH_SHORT).show();
20              String user_id = mFirebaseAuthenticationDatabase.getCurrentUser().getUid().toString();
21              mCourseDatabaseReference.child(user_id).push().setValue(new CourseDetail(course,row,column));
22              mAllCourseReference.child(course).setValue(course);
23             }
24         }
25     };
26 }
```



Total no. of linearly independent paths : 2

Path 1: 1-4=> 8-9 => 12-16 => 17

Description : This path is executed if the course added already exists
Input :

Course Name-"CS202" (Course already exist)
Row :10
Column :10

Output : Toast with message "Course Already Exist" appears on screen.

Path 2: 1-4=> 8-9 => 12-16 => 19- 22
Description : This path is executed if the course does not exist and all fields are filled properly
Input :

Course Name-"CS203" (Course is not in database)
Row :10
Column :10

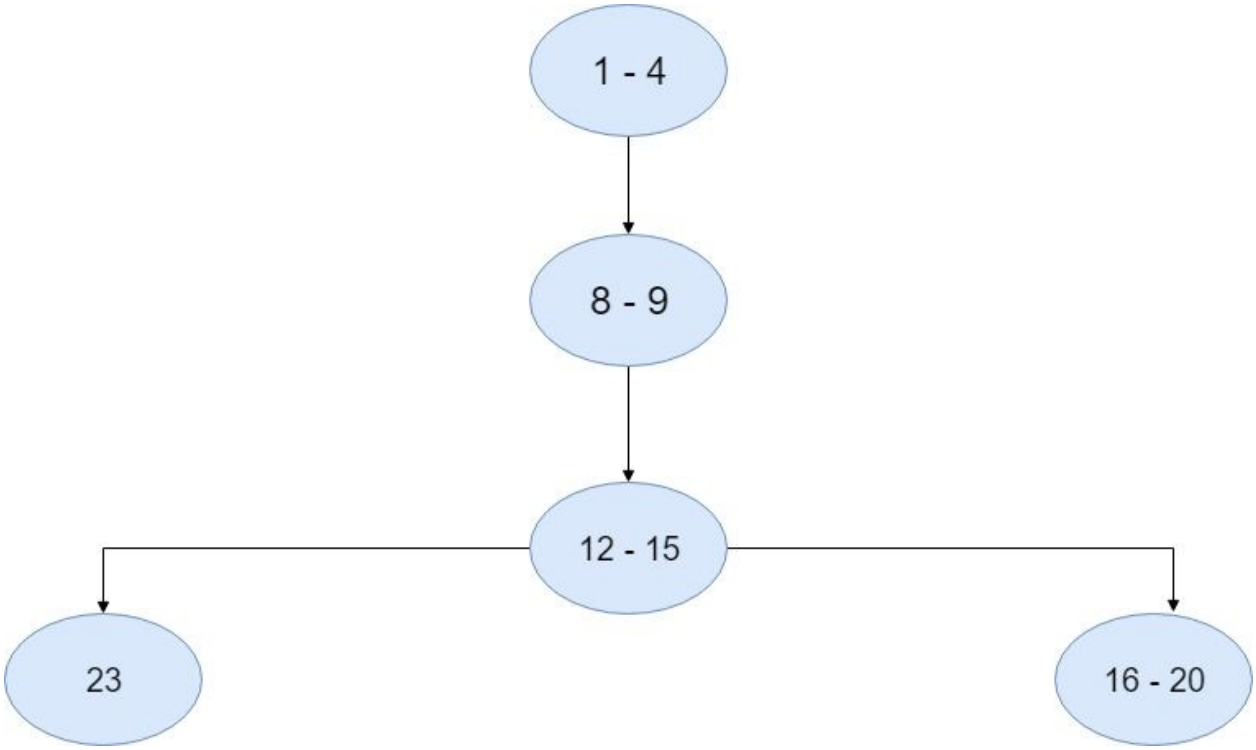
Output : Toast with message "Course Created Successfully" appears on screen.

3.4 MODULE : REGISTER COURSE

FUNCTIONS:onClick() , openRegisterCourse() , applyTextRegisterCourse()

```
buttonRegisterCourse.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                openRegisterCourse();
            }
        });
public void openRegisterCourse(){
    RegisterCourseDialog dialogeForRegisterobject = new RegisterCourseDialog();
    dialogeForRegisterobject.show(getSupportFragmentManager(),"dialog");
}
public void applyTextRegisterCourse(final String courseName) {
    mAllExitisngCourseReference.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            if (dataSnapshot.hasChild(courseName)){
             newCourseName= courseName;
             String user_id=mFirebaseAuth.getCurrentUser().getUid().toString();
             mCourseDatabaseReference.child(user_id).push().setValue(newCourseName);
             mRegisteredStudentDatabase.child(newCourseName).child(myRollno).setValue(myRollno+"_"+myName);
             Toast.makeText(StudentMainActivity.this,"Registered in"+ courseName,Toast.LENGTH_SHORT).show();
            }
            else {
                Toast.makeText(StudentMainActivity.this,"No such Course exists",Toast.LENGTH_SHORT).show();
            }
        }
    }

}
```



Total no. of linearly independent paths : 2
Path 1: 1-4=> 8-9 => 12-15 => 233
Description : This path is followed the student enter a course name which is not in database
Input :

User presses REGISTER COURSE Button
Course - “MA505” (non existing course)

Output :Toast with message “No such Course Exists ” is displayed on screen

Path 2: 1-4=> 8-9 => 12-15 => 16-20
Description : This path is followed when student enters a course name (valid) to register in that course
Input :

User presses REGISTER COURSE Button
Course - “CS223” (Existing Course)

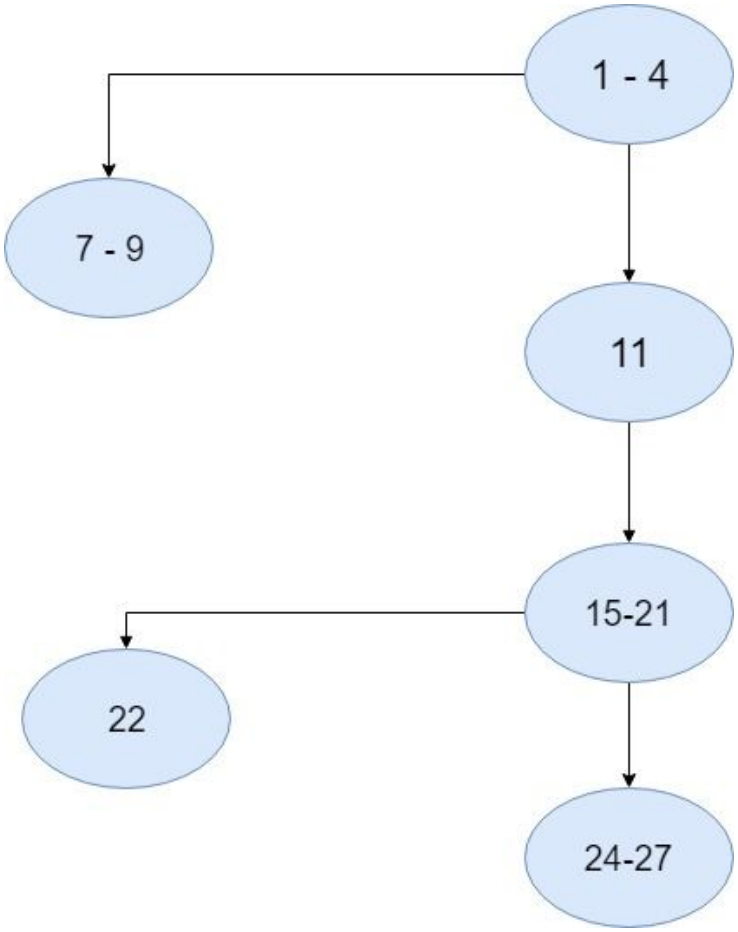
Output :Toast with message “Registered in CS223 ” is displayed on screen and CS223 is added in database in student course list and student’s name and roll no is saved in database in CS223 course.

3.5 MODULE : START/END SESSION

FUNCTIONS:onClick() , selectCourse() ,applyTexts()

```
startSessionButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (!sessionIsActive) {
                    selectCourse(v);                }
                else {
                    sessionIsActive=false;
                    String selected_course_name=CourseListOfProfessor.get(selectedCourseNo).getCourseName();
                    mCourseLectureReference.child(selected_course_name).child(getSessionName()).setValue(null);
                    startSessionButton.setText("START SESSION");  }}});
public void selectCourse(View view){
        showDialogForStartSession();
        }
public void applyTexts(final String sessionname) {
    setSessionName(sessionname);
    Toast.makeText(this, sessionname, Toast.LENGTH_SHORT).show();
    String selected_course_name=CourseListOfProfessor.get(selectedCourseNo).getCourseName();
    mFirebaseDatabase.getReference().child(selected_course_name).addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            if (dataSnapshot.hasChild(sessionname)){
                Toast.makeText(ProfessorActivity.this,"Session with same name exists",Toast.LENGTH_SHORT).show();           }
            else {
                mCourseLectureReference.child(selected_course_name).child(sessionname).setValue(sessionname);
                mSessionsReference.child(selected_course_name).child(sessionname).setValue(sessionname);
                sessionIsActive=true;
                startSessionButton.setText("END SESSION");  }}});}

```



Total no. of linearly independent paths : 3

Path 1: 1-4=> 7-9

Description : This path is followed if a session is active and user presses END SESSION button

Input :

User presses the END SESSION Button

Output :Session is dismissed and END SESSION Button changes to START SESSION

Path 2: 1-4=> 11 => 15-21 => 24-27

Description : This path is used to start session. User presses START SESSION Button, selects a course and enter session key for new session

Input :

Course - “CS243” (Existing course)

Session key - “pass” (new session password)

Output :A new session is created and START SESSION Button changes to END SESSON Button

Path 3: 1-4=> 11 => 15-21 => 22

Description : This path is executed when a session with same session key exists

Input :

Course - “CS243” (Existing course)

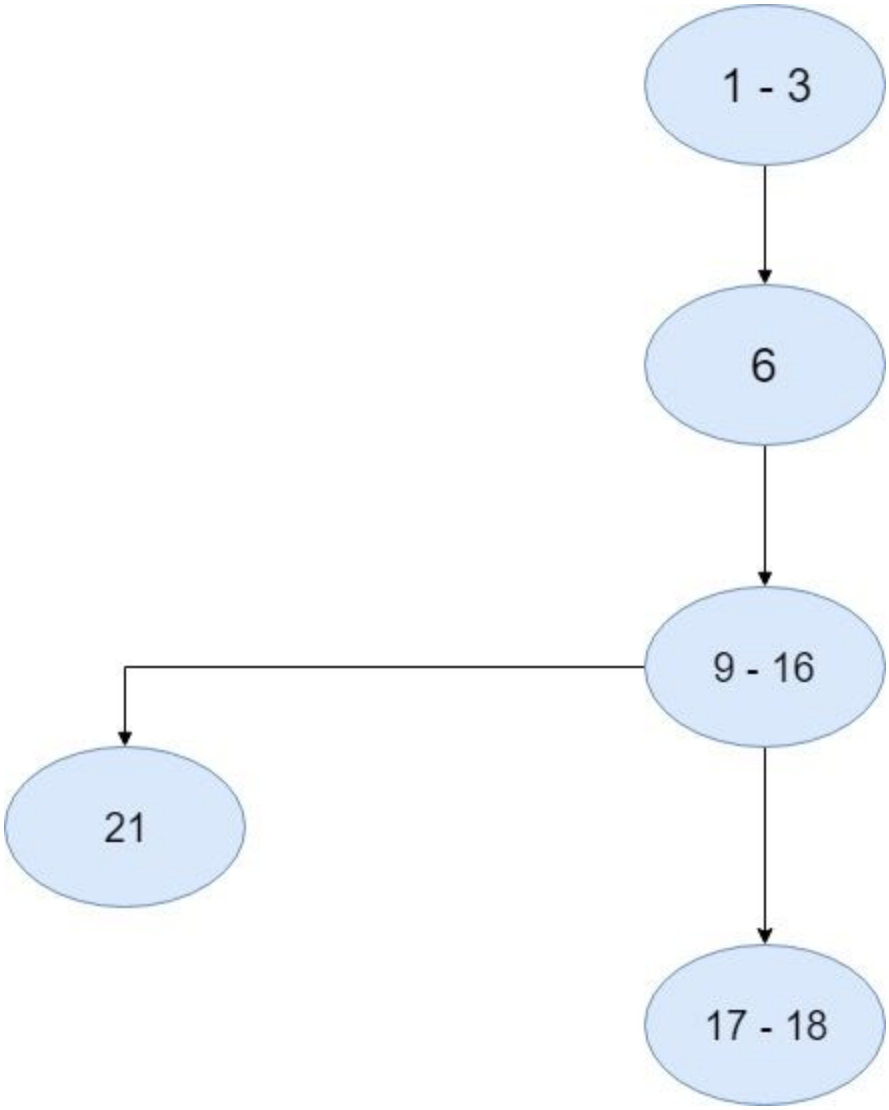
Session key - “pass” (existing session password)

Output :A Toast with message “Session with same name exists” is displayed.

3.6 MODULE : JOIN SESSION

FUNCTIONS:onClick() , selectCourse() , applyTexts()

```
buttonJoinSession.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                selectCourse(v);      }
        });
     public void selectCourse(View view){
        showDialogForJoinSession();
    }
public void applyTexts(String password) {
    sessionName = password;
    mRegisteredCourseListDatabaseReference = mFirebaseDatabase.getReference().child(CourseListOfStudent.get(selectedCourseNo));
    sessionDatabaseReference = mRegisteredCourseListDatabaseReference.child(sessionName);
    mAttendanceDatabase.child(newCourseName).child(sessionName).child(myRollno).setValue(myRollno+"_"+myName);
    mCourseSessionReference.child(CourseListOfStudent.get(selectedCourseNo)).addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            if (!sessionName.equals("") && dataSnapshot.hasChild(sessionName)){
                Intent intent = new Intent(StudentMainActivity.this, ScanActivity.class);
                startActivityForResult(intent,REQUEST_CODE);
            }
            else {
                Toast.makeText(StudentMainActivity.this,"Wrong Session Password",Toast.LENGTH_SHORT).show();
        }} }); }
```



Total no. of linearly independent paths : 2

Path 1: 1-3=> 6=>9 -16=>21

Description : This path is executed if user enters wrong session password/enter key for a session that is inactive

Input :

First user selects a course from list of registered course and then a prompt is displayed asking for session key

Course- “CS202” (Existing course)

Session password - “pass123” (Wrong Session key)

Output :Toast with message “ Wrong Session Password ” appears on screen

Path 2: 1-3=> 6=>9 -16=>17-18

Description : This path is entered if user enters correct session key for a session that is currently active

Input :

First user selects a course from list of registered course and then a prompt is displayed asking for session key

Course- “CS202” (Existing Course)

Session password - “pass@123” (Correct Session password)

Output :The user is redirected to ScanActivity to scan the Qr Code

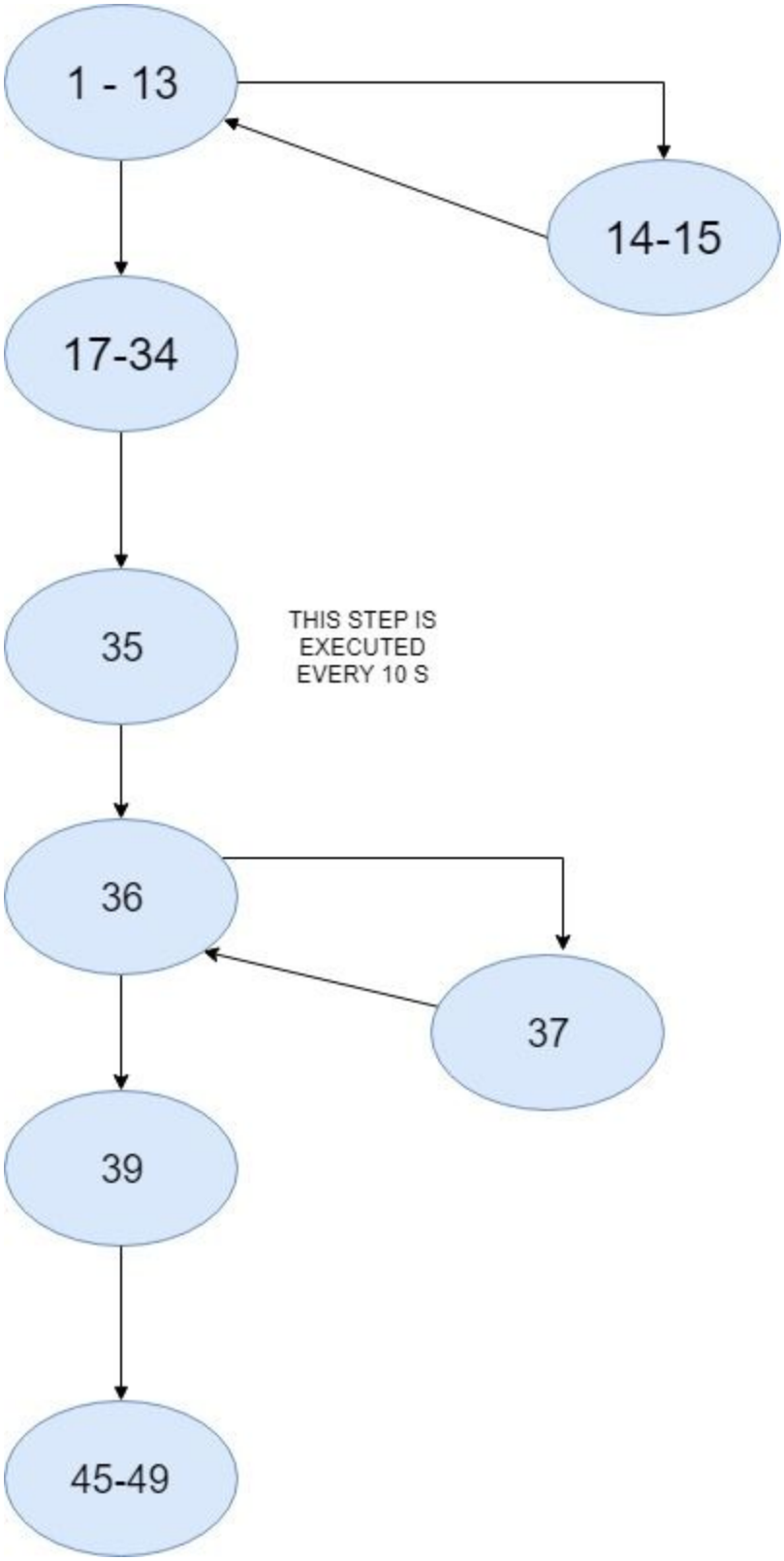
3.7 MODULE : GRID VIEW

FUNCTIONS:onCreate() , callCanvas()

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mFirebaseDatabase = FirebaseDatabase.getInstance();
    Bundle extras = getIntent().getExtras();
    if (extras != null) {
        nameofSession = extras.getString("sessionname");
        courseName = extras.getString("coursename");
        noOfRowsInClass =extras.getInt("row");
        noOfColumnsInClass =extras.getInt("col");
    }
    mappedStudentPositionarray =new String[noOfColumnsInClass * noOfRowsInClass];
    mappedStudentNamearray =new String[noOfColumnsInClass * noOfRowsInClass];
    for (int i = 0; i< noOfColumnsInClass * noOfRowsInClass; i++){
        mappedStudentPositionarray[i]="Empty";
        mappedStudentNamearray[i]="";
    }
    setContentView(R.layout.activity_grid);
    callCanvas(StudentPositionList,StudentStateList);
    mCourseSessionDatabaseReference = mFirebaseDatabase.getReference().child(courseName).child(nameofSession);
    mCourseSessionDatabaseReference.addChildEventListener(new ChildEventListener() {
        @Override
        public void onChildAdded(DataSnapshot dataSnapshot, String s) {
            Position classStudent = dataSnapshot.getValue(Position.class);
            StudentPositionList.add(classStudent);
            StudentStateList.add(randamValueGeneratorObject.nextInt(10) );
            int mapped_student_position_index=(classStudent.getRow()-1)* noOfColumnsInClass +classStudent.getColumn()-1;
            mappedStudentPositionarray[mapped_student_position_index]=Integer.toString(classStudent.getRoll());
            mappedStudentNamearray[mapped_student_position_index]=classStudent.getName();
            callCanvas(StudentPositionList,StudentStateList);

        }});
        new CountDownTimer(300000, 1000) {
        public void onTick(long millisUntilFinished) {
            Long remainingSec = millisUntilFinished/1000;
            if (remainingSec % 10 == 0 && StudentPositionList.size()>0) {
                for (int i=0;i<StudentStateList.size();i++){
```

```
                    StudentStateList.set(i, randamValueGeneratorObject.nextInt(10));
                }
                callCanvas(StudentPositionList,StudentStateList);
            }
        }
    }.start();
    }
public void callCanvas(ArrayList<Position> Slist,ArrayList<Integer>statelist){
    pixelGrid = new PixelGridView(this);
    pixelGrid.setNumberOfColumns(noOfColumnsInClass);
    pixelGrid.setNumberOfRows(noOfRowsInClass);
    pixelGrid.setListofstuds(Slist,statelist);
    setContentView(pixelGrid);
}
```


1 - 13
14-15
17-34
35
THIS STEP IS EXECUTED EVERY 10 S
36
37
39
45-49

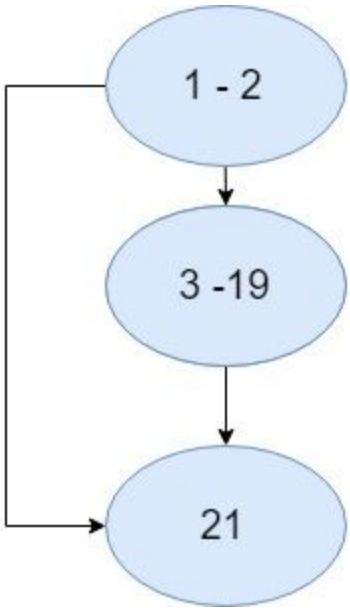
Total no. of linearly independent paths : 1

Path 1: 1-13 => 14-15 =>13=>17-34 => 35 => 36=>37=>36=>39=>45-49
Description :This is path in grid view .In this path first the mapped student list is initialised as empty (first for loop) then in next for loop the student are given random state values and then finally callCanvas is executed so that the map view is generated . Here the line 35 is executed every 10 sec to change value of student states at regular interval .So every 10 s callCanvas is called and view is refreshed on the screen.
Input :User presses GRID VIEW Button
Output: Class view with state augmented on student position

After loop termination the sequence of statements followed is same so loop is not leading to change in number of paths.

FUNCTION: onTouchEvent()

```
1  public boolean onTouchEvent(MotionEvent event) {
2      if (event.getAction() == MotionEvent.ACTION_DOWN) {
3          studentCellWidth=pixelGrid.getCellWidth();
4          studentCellHeight=pixelGrid.getCellHeight();
5          Log.d("w,h",studentCellWidth + ","+studentCellHeight);
6          int column = (int)(event.getX() / studentCellWidth)+1;
7          int row = (int)((event.getY()-250) /studentCellHeight)+1;
8          String rollNotoDisplay = mappedStudentPositionarray[(row-1)* noOfColumnsInClass +(column-1)];
9          String nameOfStudent = mappedStudentNamearray[(row-1)* noOfColumnsInClass +(column-1)];
10         AlertDialog.Builder builder = new AlertDialog.Builder(this);
11         builder.setTitle("Student Information")
12                 .setMessage(row +","+column +"\n"+rollNotoDisplay+"\n"+nameOfStudent)
13                 .setCancelable(false)
14                 .setNegativeButton("Close",new DialogInterface.OnClickListener() {
15                     public void onClick(DialogInterface dialog, int id) {
16                         dialog.cancel();
17                     }});
18         AlertDialog alert = builder.create();
19         alert.show();
20     }
21     return true;
22 }
23
24
25
```



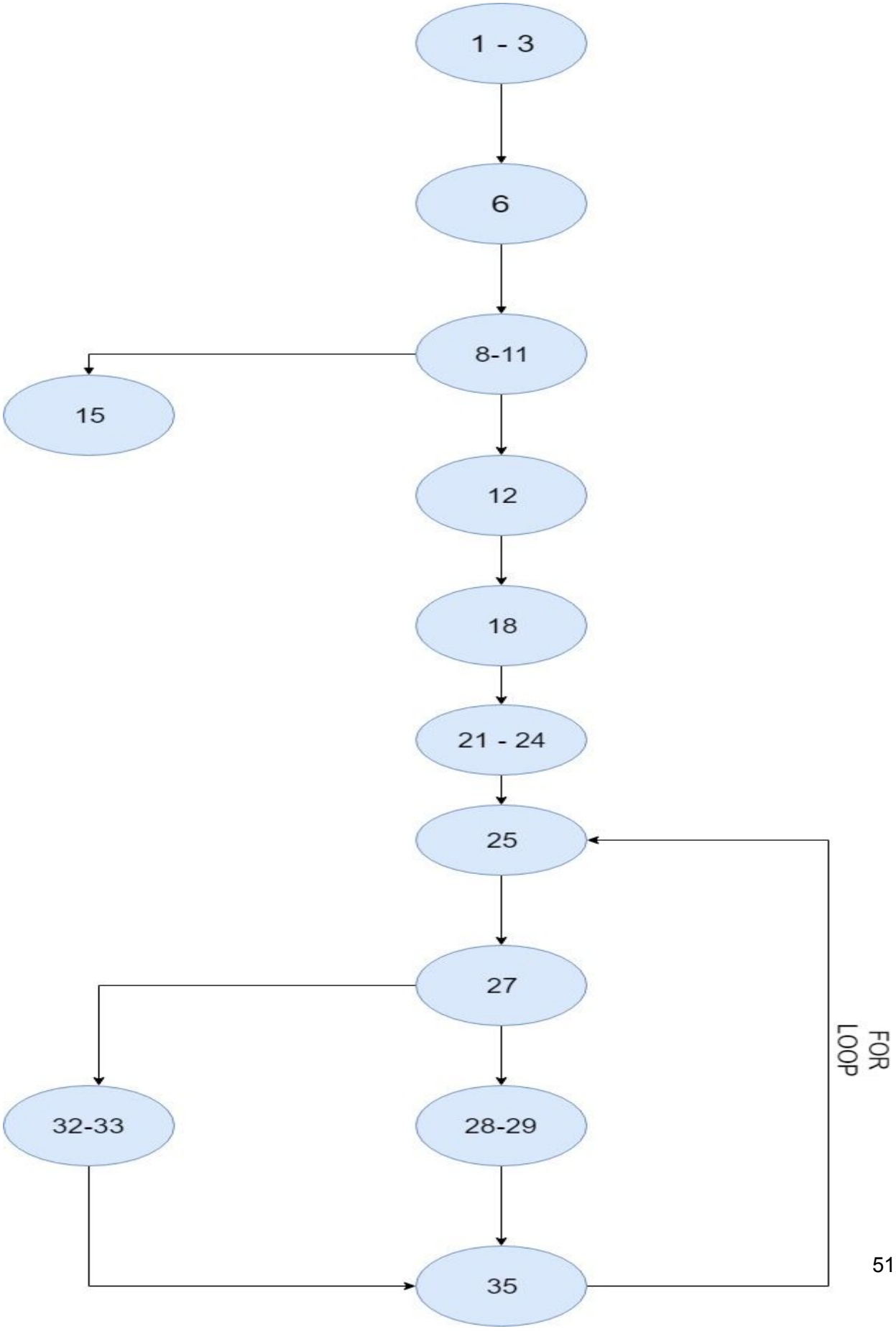
Total no. of linearly independent paths : 2
Path 1: 1-2 => 3-19 => 21
Description :This is path is followed when the professor clicks at any seat in the classroom 2D representation to know about the occupant
Input :
Touch at seat (2,2)
Output :
Dialog containing student details if occupied roll no and name else empty is displayed
Path 2: 1-2 => 21
Description :This is path is followed when there is no touch event so there is no input or output

After loop termination the sequence of statements followed is same so loop is not leading to change in number of paths.

3.8 MODULE : VIEW PAST ATTENDANCE

FUNCTIONS:(buttonSelectCourse)onClick() , selectCourse() , (buttonSelectSession)onClick() , selectSession() , setAdapter()

```
buttonSelectCourse.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                selectCourse(v);
            } });
public void selectCourse(View view){
    showDialogForSelectCourse();;
}
buttonSelectSession.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (isButtonPressed){
                selectSession(v);
            }
            else {
                Toast.makeText(ViewPastRecordActivity.this,"First Select Course",Toast.LENGTH_SHORT).show();
            }}});
public void selectSession(View view){
    showDialogForSelectSession();
}
public void setAdapter(){
        String selected_course_name=CourseListOfProfessor.get(selectedCourseNo).getCourseName();
        mAttendanceDatabase.child(selected_course_name).child(selectedSession).addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                for (int i=0;i < AllStudentList.size();i++){
                    String detail[]= AllStudentList.get(i).split("_");
                    if(dataSnapshot!=null && dataSnapshot.hasChild(detail[0])){
                        Student student = new Student(AllStudentList.get(i), "Present");
                        studentList.add(student);
                    }
                    else {
                        Student student = new Student(AllStudentList.get(i), "Absent");
                        studentList.add(student);
                    }
                    mAdapter.notifyDataSetChanged();
                } } }); }
```


1 - 3
6
8-11
15
12
18
21 - 24
25
27
32-33
28-29
35
FOR LOOP

Total no. of linearly independent paths : **3**
Path 1: 1-3=> 6=>8 -11=>15
Description : This path is executed if professor presses the SELECT SESSION Button before selecting any course
Input :

Professor presses the SELECT SESSION BUTTON without selecting a course

Output :Toast with message " Please select course " appears on screen

Description for Path 2 and Path 3 :
For these path when professor has successfully selected a course and then a session is also selected then the list of all registered student is accessed and then for each student the database is accessed and to check whether that student was present in selected session or not and if student was PRESENT then Path 2 is executed otherwise Path 3 is executed. And finally, when all student are checked then the Student list with their attendance is shown on screen

Path 2: 1-3 => 6 => 8-11 =>12 =>18 =>21-24 => 25 =>27 =>28-29=> 35
Input :

First user presses SELECT COURSE Button Followed by SELECT SESSION Button
Course- "CS202" (Existing course)
Session - "lecture1" (existing session key)

Output :Student list with their attendance is shown on screen with absent student is in color RED and present student is in color GREEN

Path 3: 1-3 => 6 => 8-11 =>12 =>18 =>21-24 => 25 =>27 => 32-33=> 35
Input :

First user presses SELECT COURSE Button Followed by SELECT SESSION Button
Course- "CS202" (Existing course)
Session - "lecture1" (existing session key)

Output :Student list with their attendance is shown on screen with absent student is in color RED and present student is in color GREEN

After loop termination the sequence of statements followed is same so loop is not leading to change in number of paths.

4. CONCLUSION

The white-box testing and the black-box testing were successful to a great extent.
In black-box testing we performed unit testing on different modules , it was taken into consideration to cover at least one case from each of the equivalence classes of each module so that overall all types of test cases can be said to be covered.The boundary analysis was also implemented although most of our module had equivalence classes with discrete values .
In white-box, various test cases were selected to cover different independent paths in the control flow graphs of all the functions in the module , It was found that every statement was covered for at least some of the test cases.