SOFTWARE ENGINEERING

GROUP NUMBER – 7

Team Details:

SL.NO	NAME	SRN
1	Nilkant	PES2UG20CS530
2	Pradhyumna	PES2UG20CS533
3	Shreepathi	PES2UG20CS553
4	Manik	PES2UG20CS576

Problem Statement – 1: Unit Testing

A unit is the smallest block of code that functions individually. The first level of testing is Unit testing and this problem statement is geared towards the same.

• Discuss with your teammates and demarcate units in your code base o Note: discuss why the code snippet you have chosen can be classified as a unit

Login Module :

```
package com.example.homely

import android.os.Bundle import
androidx.fragment.app.Fragment import
android.view.LayoutInflater import
android.view.View import
android.view.ViewGroup import
android.widget.Toast
import androidx.navigation.fragment.findNavController import
com.example.homely.databinding.FragmentLoginBinding import
com.google.firebase.auth.FirebaseAuth
```

```
class LoginFragment : Fragment() {

lateinit var firebaseAuth:FirebaseAuth     private var
binding_ : FragmentLoginBinding?= null     private val
binding get() = binding_!!

override fun onCreateView(        inflater: LayoutInflater,
container: ViewGroup?,        savedInstanceState:
Bundle?
                ): View {


binding_ = FragmentLoginBinding.inflate(layoutInflater,container,false)        firebaseAuth =
FirebaseAuth.getInstance()        binding.apply {

btnLogin.setOnClickListener {
          login()
}

tvSignup.setOnClickListener {
findNavController().navigate(R.id.action_loginFragment_to_signInFragment)
}
}
return binding.root
}

 private fun login(){
             binding.apply{

val email = etEmail.text.toString()            val
password = etPassword.toString()

//Test case possible
             if(email.isBlank() || password.isBlank()){
Toast.makeText(requireContext(),"Email/Password cannotempty",
Toast.LENGTH_LONG).show()
return
}
```

// Test case possible

```
    If(password.length()<6 || password.length()>15){
Toast.makeText(requireContext(),"Password should be between 6-15 characters",
Toast.LENGTH_LONG).show()
return
}

firebaseAuth.signInWithEmailAndPassword(email,password).addOnCompleteListener(re
quireActivity()){

if(it.isSuccessful){

        findNavController().navigate(R.id.action_loginFragment_to_mainFragment
}
 else{

        Toast.makeText(requireContext(),it.exception?.localizedMessage.toString(),Toast.LE
        NGT H_LONG).show
}
 } } }
```

We have chosen the **Login Module** for Unit Testing.

In our Project, we are using **Email** and **Password** for logging into the application/system software.

Testcase ID	Email	Password	Expected output	Actual output	Result
1	Valid	Valid	Valid	Valid	Pass
2	Valid	Invalid	Invalid	Invalid	Pass
3	Invalid	Valid	Invalid	Invalid	Fail
4	Invalid	Invalid	Valid	Valid	Fail
5	Blank/Empty	Invalid	Invalid	Invalid	Fail
6	Blank/Empty	Valid	Invalid	Invalid	Pass
7	Invalid	Blank/Empty	Invalid	Invalid	Pass
8	Valid	Blank/Empty	Invalid	Invalid	Fail
9	Blank/Empty	Blank/Empty	Invalid	Invalid	Pass

~Credits Shreepathi : PES2UG20CS553

- **Ideate how you could further modularize larger blocks of code into compact units with your teammates**

  - Modularization is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each module contains everything necessary to execute only one aspect of the desired functionality.
  - We can modularize larger blocks of the code into compact units by using nested functions to encapsulate the blocks of code.
  - To keep the functions atomic in nature so that it performs a specific task with or without least interaction with the other modules
  - We should try to avoid repetitions and consolidate repeated code in loops or functions.
  - Try to use fewer (i.e., less than three) arguments per function.
  - Try to minimize the number of classes, functions, modules, variables etc. for effective re-usability and readability.

~ Credits Shreepathi : PES2UG20CS553

2.Dynamic Testing

Dynamic testing involves execution of your code to analyse errors found during execution. Some common techniques are Boundary Value Analysis and Mutation Testing.

2.a Boundary Value Analysis

When it comes to finding errors in your code base, they are often found at locations where a condition is being tested. Due to this, developers often use Boundary Value tests to reduce defect density.

How would you define a boundary test?
Note: Simple relational conditions are a basic example

Boundary value analysis is Functional based testing/Black box testing often used to check for errors at the boundary conditions

For boundary testing we have selected the **phone number field** from the owner form module. For 10 digit phone number - As we do not see any other constraints

1. 00000 00000
2. 10000 00000
3. 55555 55555
4. 99999 99999
5. 1 00000 00000

Total number of boundary values =4n+1= 4*1+1 = 5 boundary values

Build your boundary test cases and execute them

Test Cases	Input (Phone number)	Output
T1	09999 99999	Invalid
T2	10000 00000	Valid
T3	55555 55555	Valid
T4	99999 99999	Valid
T5	1 00000 00000	Invalid

2.b Mutation Testing

Using your isolated units from the first problem statement, ideate with your team mates on how to mutate the code

The condition is:

if **(input_length(password)>=6 and input_length(password)<=15)** {

output –> valid password

}

• Develop at least 3 mutants of the functioning code and test all 4 code bases using the test case from the first problem statement

In our project the min length of password is and maximum of 15 characters

The possible Mutants are :

M1 :

IF(INPUT_LENGTH>=6 AND INPUT_LENGTH<=15)

M2 :

IF(INPUT_LENGTH>6 AND INPUT_LENGTH>15)

M3

IF(INPUT_LENGTH<6 AND INPUT_LENGTH<15)

M4

IF(INPUT_LENGTH<6 AND INPUT_LENGTH>15)

Test Cases	Mutant	Expected output	Actual output	Result
T1	M1	Valid	Valid	**Alive**
T2	M2	Valid	Invalid	Killed
T3	M3	Valid	Invalid	Killed
T4	M4	Valid	Invalid	Killed

Mutants M2, M3, M4 got killed but mutant M1 is still alive.

MUTATION SCORE = (*KILLED MUTANTS / TOTAL MUTANTS) X 100*

= (3/4) X 100

= 75%

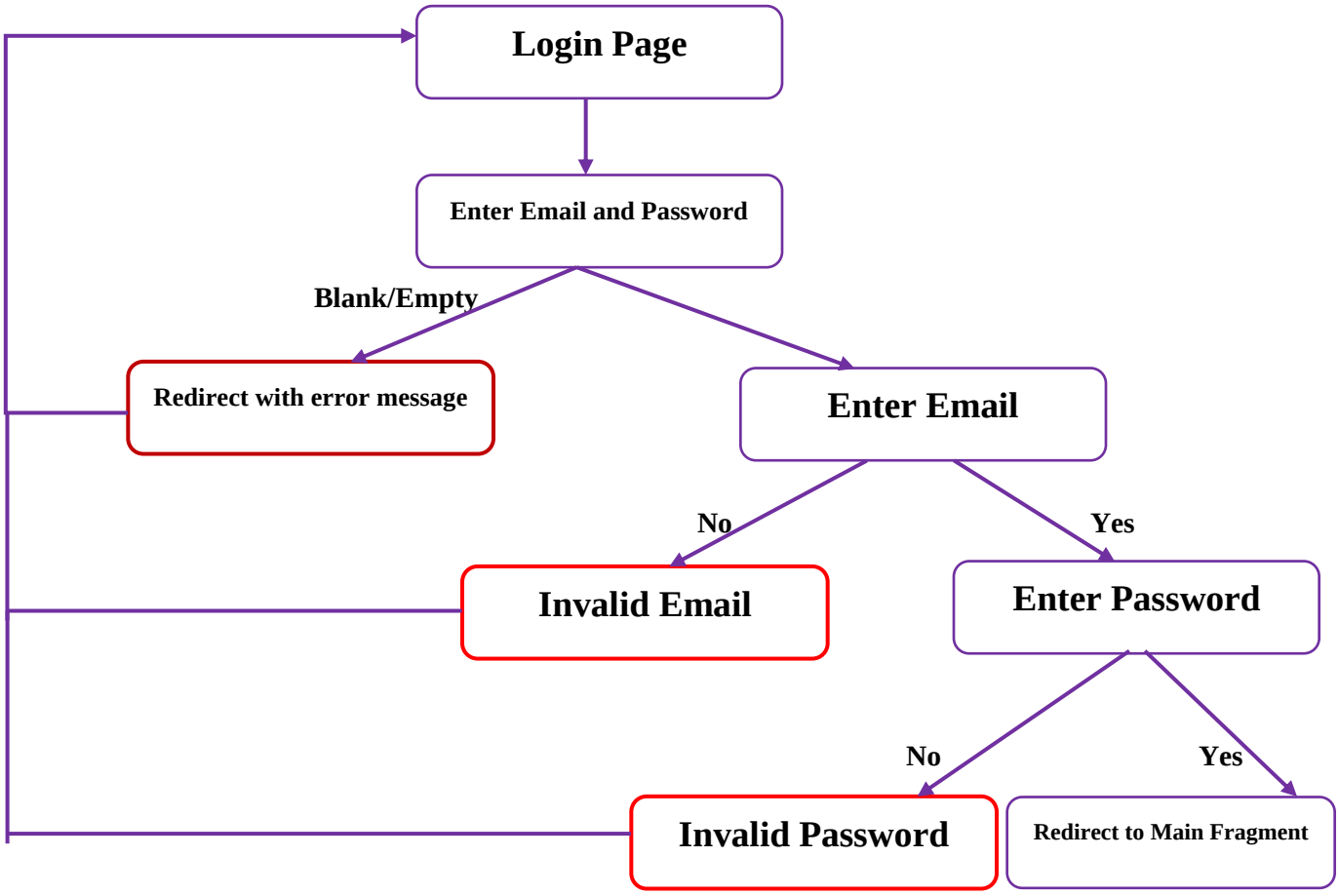
~ Credits Manik : PES2UG20CS576

Problem Statement – 3: Static Testing

Static testing involves validating your code without any execution. Under this problem statement, you will be expected to analyse and calculate the cyclomatic complexity of your code.

- **Using the unit you selected in the first problem statement as an example, develop the control flow graph of your problem statement.**

CONTROL FLOW DIAGRAM FOR LOGIN MODULE



- **Using the Control flow graph, calculate the cyclomatic complexity of your code.**
- **Using the cyclomatic complexity as an indicator, Ideate and code your unit again to reduce complexity**

$$\textit{Cyclomatic Complexity} = E - N + 2P$$

Where,

M = Cyclomatic Complexity

E = Number of Edges in the Control Flow Graph

N = Number of Nodes in the Control Flow Graph

P = Number of Connected Components

In Our Control Flow Graph,
E = 9
N = 7
P = 1
M = 9 – 7 + 2(1)
M = 4

Therefore, the Cyclomatic Complexity of our CFG is 4

The code written is efficient in its performance which reduces time and space complexity

~ Credits Nilkant : PES2UG20CS530

Problem Statement – 4: Acceptance Testing

- **Assume your neighbouring team is the client for your code.Give them an idea of what your product is and the software requirements for the product**
- **Exchange your code base and test each others projects to see if it meets user requirements**

In our project, the user interfaces and modules have been built in such a way that they are user-friendly. Modules built in this project are:

- **Registration Module**
    - Login Module
    - Sign in Module
- **Owner Module**
    - Owner form
    - Owner dashboard
    - Add stay
    - Edit stay
- **Tenant Module**
    - Tenant dashboard
    - View stay module
    - Integration of third party services

Some Bugs have been identified in the conventions of username and password, but majority of test cases along with boundary test cases are passing. **Searching for stays, and the basic CRUD functionality has been implemented really well**.

Analyzing the neighbouring team's code the following are the software requirements

- Customer has to login to avail services
- They have built a Website through which customers can order their food
- Various classifications such as Veg-NonVeg
- Using service oriented approach to integrate services like PayPal for payment

- **If you identify a bug in the project you are testing,inform the opposing team of the bug**

At this point of time there are no major bugs
They may face certain issues ahead which may be

- Unable to generate accurate billing amount.
- While registrating, there should be certain limit for the number of characters typed in

- **As a team, based in clients experience, ideate modifications to the existing project that could improve client experience**

The following are the suggested improvements as a client

- They can add feature where they could suggest rental rooms at reasonable amount
- They could have added Reviews/Ratings for food and the restaurant

~ Credits Pradhyumna : PES2UG20CS533

Problem Statement – 5: Maintenance Activities

Once a product is completed, it is handed off to a service based company to ensure all maintenance activities are performed without the added expenditure of skilled developers. However, a few tasks are performed by the maintenance team to gauge the product better. In this problem statement, you will be asked to experiment with your code.

- **Exchange code bases with your neighboring teams and reverse engineer a block of code in order to understand it's functionality**
- **After understanding the code block, Re-Engineer the code o Ideate how to refactor the code and the portion of the code base you would have to change o Discuss how the new changes would impact the time and space complexity of the project during execution**
- **After Reverse Engineering and Re-Engineering the code, perform acceptance testing between the teams**

- Code refactoring is defined as a process of restructuring the computer code without affecting the core functionality
- Refactoring improves the code readability and reduces complexity
- Can be performed after the product is deployed before adding updates and new features to the existing code

Reverse engineering can deconstruct the software to extract their design information.

- It allows us to identify how the developer design a particular part of code so that we can recreate and create a replacement part of the product the code the engineers copy or mimic the a design without the original blueprint
- The process of restructuring existing code changes the factoring without changing its external behaviour.
- Change the code without breaking current functionalities

~ Credits Pradhyumna : PES2UG20CS533