SE UNIT 4

Team Members:

1. Sushanth Prabhu – PES2UG20CS359
2. Sneha BT- PES2UG20CS340
3. Sonakshi Jamadhiar- PES2UG20CS344
4. Tanushree Mondal – PES2UG20CS366

Problem Statement 1: Unit Testing

TEST ID	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	RESULT
1	Correct name and password	Name:RAM Password:abc	Login Successful	Login Successful	Pass
2	Correct name and wrong password	Name:Ram Password:abcdfgh	Login unsuccessful	Login unsuccessful	Pass
3	Password and re-password matching	Password:abc Re-password:abc	Registration Successful	Registration Successful	Pass
4	Wrong Name and Password	Name:Shyam Password:abcdfgh	Login unsuccessful	Login unsuccessful	Pass
5	Password and Re-password not matching	Password:abc Re-passowrd:asfdg	Registration unsuccessful	Registration unsuccessful	Pass

Problem Statement 2

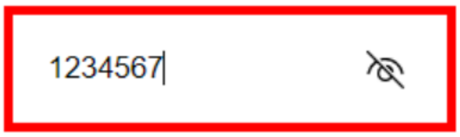
Dynamic Testing Dynamic testing involves execution of your code to analyse errors found during execution. Some common techniques are Boundary Value Analysis and Mutation Testing.

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?

Boundary Value Analysis is used to test boundary values because the input values near the boundary have higher chances of error. Boundary tests check for the input values near the boundary that have a higher chance of error. The values checked for are min-1, min, a value between min and max, max, max+1

Here the password field is considered for boundary value analysis. The number of characters in the password can range from 8 to 15. Hence, tests are done for min-1, min, a value between min and max, max+1

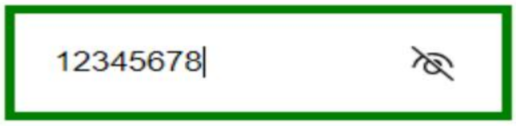
For min-1= 7 characters, an error is thrown asking for the password to be at least 8 characters



1234567

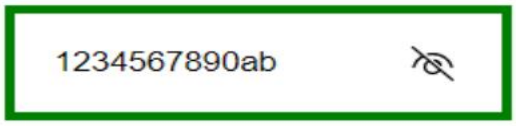
Password should contain atleast 8 characters.

For 8 characters, the input is valid as indicated by the green border



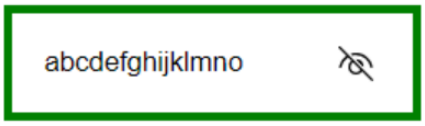
12345678

For length of password between 8 and 15, the input is valid

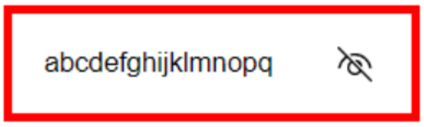


1234567890ab

For password length of max=15, the input is valid


abcdefghijklmno

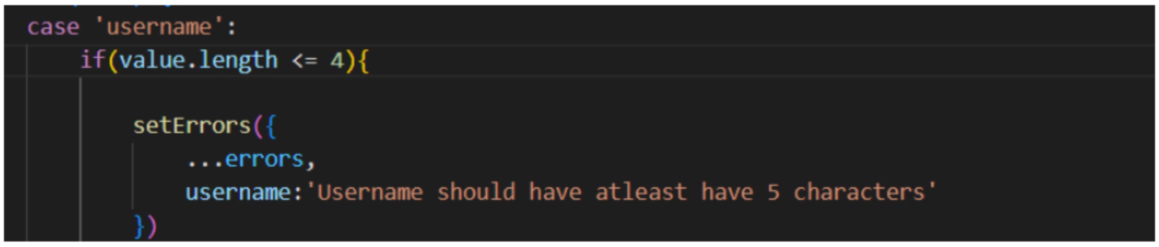
For password length greater than 15, the input is not valid as indicated by the red border around the box and an error message saying max length of password is 15 is displayed.

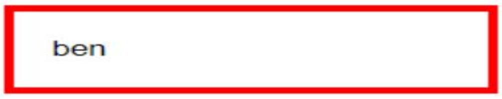

abcdefghijklmnopq

Problem Statement 3- Mutant code

Original code

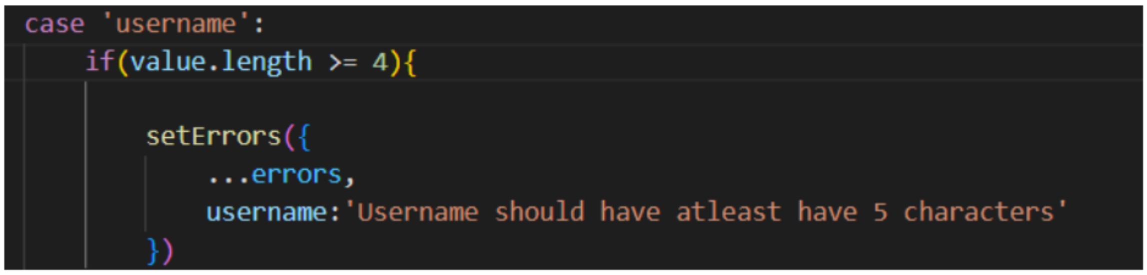
If the length of the username is less than 4, it is an invalid input.


```
case 'username':
    if(value.length <= 4){

        setErrors({
            ...errors,
            username:'Username should have atleast have 5 characters'
        })
```

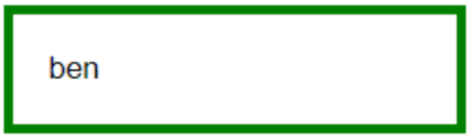

ben

Username should have atleast have 5 characters

Mutant code:



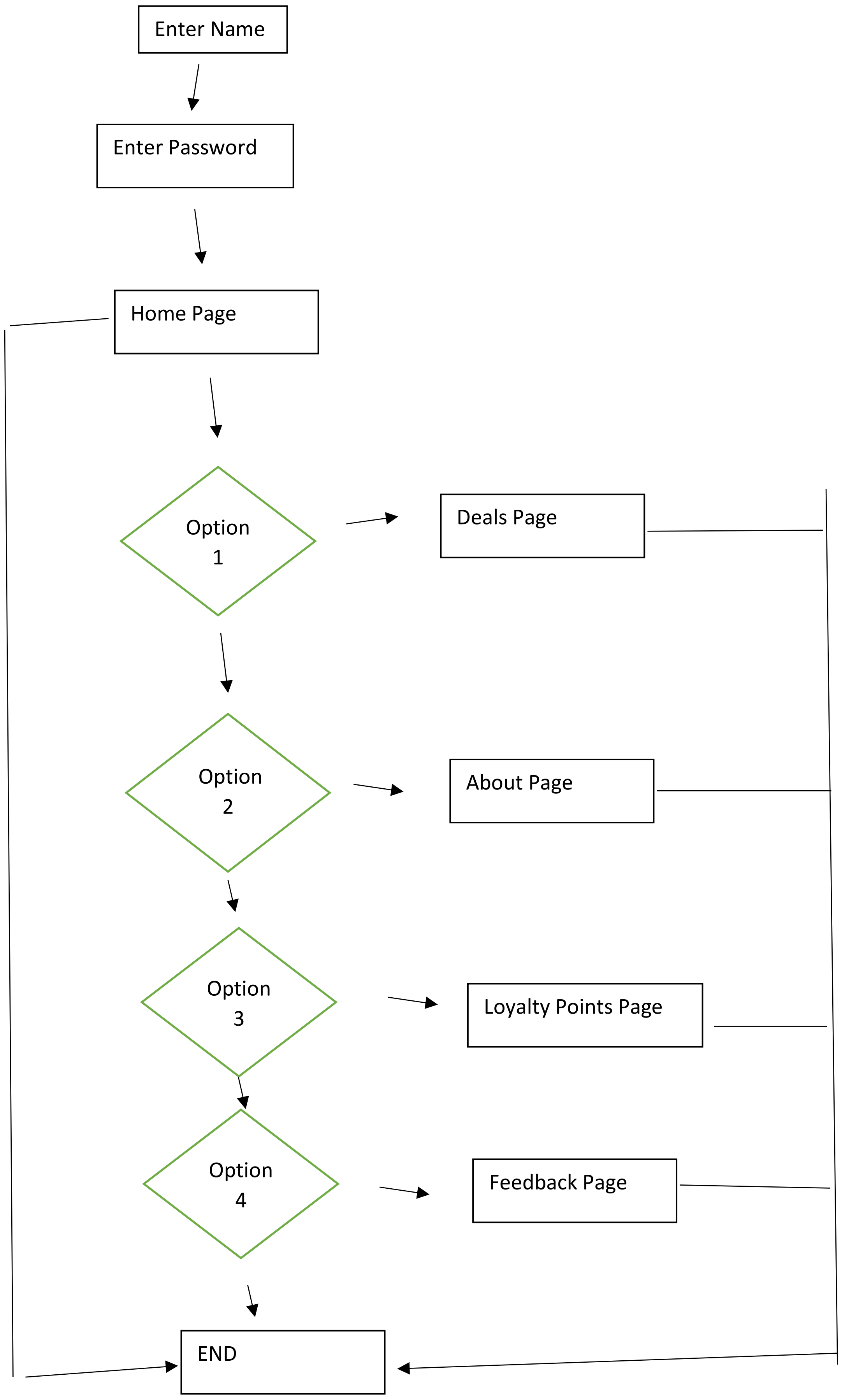
```
case 'username':
    if(value.length >= 4){

        setErrors({
            ...errors,
            username:'Username should have atleast have 5 characters'
        })
```



ben

In the mutant code, <= is changed to >= , so if the length is less than 4, an error is not thrown and it is considered as a valid input.

Problem Statement 4 – Acceptance Testing



Cycolmatic Complexity=E -N +2P= 13-12+2 =3

Using the cyclomatic complexity as an indicator, the code already has the lowest complexity and cannot be optimised further.

Problem Statement – 5: Acceptance Testing

Since our project is in a rudimentary stage right now, it is not possible to conduct acceptance testing. However, once the project is ready, the clients will be able to test out the functionalities. The client will be able to use the product and find bugs in the task scheduling system or the tracking procedure. The addition, updation and deletion of tasks needs to be verified. Tests will also have to be performed on the statistics and analytics provided for the tasks completed as there would be a lot of mathematical operations and there are chances of logical errors. The timeline feature would also have to be tested rigorously so that faulty timelines are not displayed in the final product. Further, the client could suggest changes to the UI and aesthetics of the product. The client would also have to be satisfied with the overall experience and usability of the product.

Problem Statement 6- Maintenance Activities

Refactoring: Improving or updating the code without changing the software's functionality or external behavior of the application is known as code refactoring.This helps us in reducing the cost and making the code more efficient and maintainable. an un-refactored may contain a lot of confusion and cluster in code such as duplicate code , unhealthy dependencies between classes,too many responsibilities per class,etc. There are many different ways of refactoring a code like:

➢ Red-green refactoring:follows test first approach to design and implementation

➢ Refactoring by abstraction:technique reduces the redundancy

➢ Composing method:used when long methods are used

Time Complexity: The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input

Space Complexity: The space complexity of an algorithm quantifies the amount of space taken by an algorithm to run as a function of the length of the input. Refactoring of a code reduces both time and space complexity by reducing redundancy and keeping less responsibilities for a class

Reverse Engineering: Reverse engineering is the process of recovering specification and design information about the software system from its source code. Over time, an application may have been corrected, adapted and enhanced. As a result, the application can become complex, unstable with changes having unexpected and serious side effects and reverse engineering addresses it.Reverse engineering is a passive technique that is used to understand a piece of software prior to re-engineering.

Re-engineering: Re-engineering is the process of modifying the software to make it easier to understand, change and extend. This could be for improving maintainability of a software system at reasonable cost. Re-engineering may involve refactoring.