Test Plan

C++ Code Streaming Project

Kendall Vargas

QA Software Tester

October 23rd, 2024.

Contents

Project Overview:

The C++ code consists of a short system that calculates the total amount to be paid based on hours watched on a streaming platform.

The main menu consists of 3 options:

1. First, enter the data and information so that the system calculates the amount.

2. The second, show on a receipt all the required data based on what was entered on menu 1.

3. The third, leave the system completely.

The user is asked for these details:

- Client ID
- Full Name
- Invoice number
- Number of hours watched on the platform
- If it is exempt from taxes

Requirements:

- The menu repeats until a correct option is entered (invalid: incorrect letters or numbers).
- Client ID must not allow letters or numbers other than 9.
- Invoice number must not allow letters or numbers other than 6.
- Total hours viewed must not allow letters or numbers for less than 0 or greater than 720.
- The tax menu repeats until a correct letter is entered.
- The return to the main menu repeats until a correct letter is entered.

- Important: You cannot access option 2 without having gone through option 1.

Business Logic:

There are already 2 fixed payments each month, which are:
fixedFee: 10 USD
contentProtection: 5 USD

The streaming hour has a fixed price of 2 USD per hour. For the additional hourly charge, it works as follows:

- The first 10 hours have no additional charges.
- After 10 to 40 hours consumed, an additional 10% is applied to the final price.
- From 51 to 100 hours, 20% is applied to the final price.
- And above 100 hours, 30% is applied to the final price.

For the tax price, it is 13% of the final total price of the payment. This is if the client is not exempt from taxes, if they are exempt, this additional price is not added.

Objectives:

For the correct delivery of this C++ project, I need to test all the code and validate it works correctly based on the project requirements.

Since this project does not have a GUI as such, it can be said that it is more related to a static and white box analysis and testing, however, in the data entry, I used some black box testing.

In Scope:

- "White box":
  - We need to perform sentence coverage and decision coverage, since we know the structure of the code internally. This can be performed by going through the different decisions of the code to validate each output is correct and represents the respective path.
  - Check that there's no errors on the IDE or variables not used.
  - Dead code, or paths that are not reachable as a normal user.

- Black Box:

With the output, a small screen is received to insert data and then present the receipt result. This can be considered as our "front-end" to validate the code is working as expected before implementing this to a real page:

- Verifying all menus are working as expected.

- Apply testing techniques in the input of data (int, string, float).
- Validate the response of the receipt includes expected data.
- Calculate manually if the amount given is correct.

❖ Experienced based techniques:
- Apply positive and negative scenarios across menus and all different inputs.
- Validate the correct structure of menus, receipt, and grammar issues.

Out of Scope:

Database: This project does not include a database, due to this, there's no way to store an ID to a specific object when entering the name and the ID, or matching the information with an invoice ID.

Test Strategy:

Step 1: The creation of test cases for the different features and calculations will be covered by some testing techniques:

❖ Boundary Value Analysis
❖ Equivalence Class Partition
❖ Sentence coverage and Decision coverage

Step 2:

❖ Exploratory Testing: Perform exploratory testing besides execution of the Test Cases.
❖ E2E Testing: Test the normal E2E flow covering user actions and responses.

Closing:

Even though this was a very short code, I took some time to build this small test plan.

I consider creating test plans is an important practice even in short projects, since it allows us to keep some documentation about all the features being tested and in the current scope, having the possibility to look back and have some kind of guidance along the way.

In summary, it plays a crucial role in maintaining quality, documenting key decisions, and providing peace of mind that everything is covered and detailed correctly.