

# SELF HEALING WITH QMETRY AUTOMATION STUDIO



QMETRY

# Table of Content

- 1 Self-healing with QMetry Automation Studio
- 2 How to focus on automating the right things
- 3 The case for Self-healing suites
- 4 How Self-healing works in QMetry Automation Studio (QAS)?
- 5 QAS Healing Mode
- 6 Conclusion

# Self-healing with QMetry Automation Studio



Our aim is to provide our customers with a test automation solution that is future ready as well as one that assures proven ROI. Considering that, our experts researched and evaluated other available tools in the market to understand — What are the pain points of the users? Where are the other automation tools failing to solve user problems?

Few of our key observations were

**1** Companies in their Digital Transformation journey need DevOps and Agile equipped tools

**2** A DevOps practice following the agile way of project delivery demands frequent and quick changes at all level

**3** It becomes difficult for inter dependent tools to support each other when there are frequent changes on a daily basis in the developed code. For instance, when the development is adding or modifying existing interface frequently. Your existing automation scripts can no longer support. This requires your created test cases and scripts to adapt quickly to this change.

So, what is the best way to accommodate these changes while testing?

To answer that we took a step towards self-diagnosis and then self-healing of objects.

# How to focus on automating the right things

A great deal of thought and effort is put into which tests ought to be automated and which shouldn't. Of all the applications of test automation, UI automation is probably one among the hardest. Tests are slower to write compared to other tests. They also need frequent updates and so requires continuous maintenance. To be efficient with your time and efforts and to keep the maintenance costs low, consider what you are writing these UI tests for.

Also, with additional functions and features being added to the System Under Test during successive cycles, automation scripts also need to be added, reviewed and maintained for each of these. Maintenance is necessary to improve the effectiveness of such Automation Scripts.

Test automation is ideal for the functions and features that are least likely to change. The downside is that such scripts, written for a stable code are not reviewed often. This leads to unexpected issues when the underlying code function is changed. It modifies the operation of a tested feature. We have a new problem on hand; the code has changed but the script has not and now there are false positives.

The error flags line up the results review and require constant and additional monitoring to ensure that the errors are truly from bogus tests.

When this goes on for a while, it throws the entire test automation process out of gear. Test automation is seen as an expensive, redundant activity and management will begin to doubt the efficacy of test automation.

What teams need is a great locator strategy for the system. This will reduce the maintenance efforts for the automation suite considerably. Fragile locators cause the maximum number of intermittently failing tests.



# The case for Self-healing suites



Test Automation has great potential and value for Agile and DevOps teams. However, due to the rapid velocity of changes and inability to log and manage these changes with the same speed in test scripts, automation may stop working.

Automation's promise of lowering cost, time and effort is offset by the need to manually troubleshoot, debug and fix these test automation scripts owing to their fragile nature.

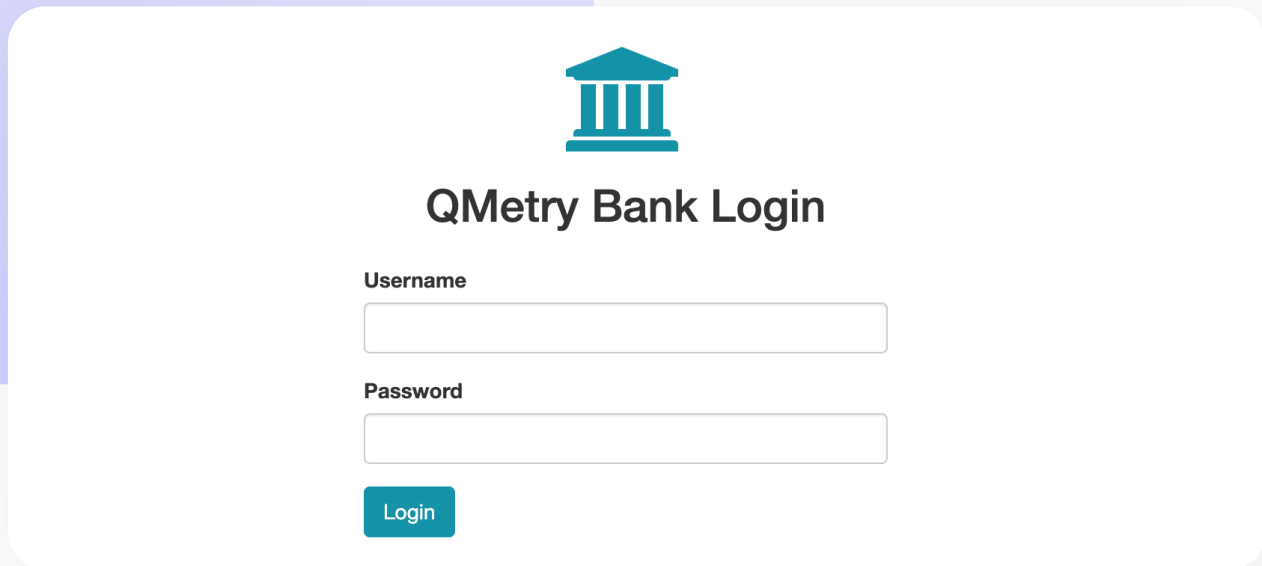
Manual identification takes a while whenever there is a break in the script. These breaks are often caused by changes in object properties. In larger projects, these aberrations tend to accumulate.

This inevitably slows down the development cycle and put a damper on the adoption of automation. And so, there is a need for systems that detect breaks and repair them automatically without manual intervention. Only then can you realize the full potential and ROI of test automation. Therefore, teams need to adopt Self-healing automation and tools that enable this capability.

To leverage the transformative power of test automation, therefore, developers need to embrace Self-healing automation tools.

# How Self-healing works in QMetry Automation Studio (QAS)?

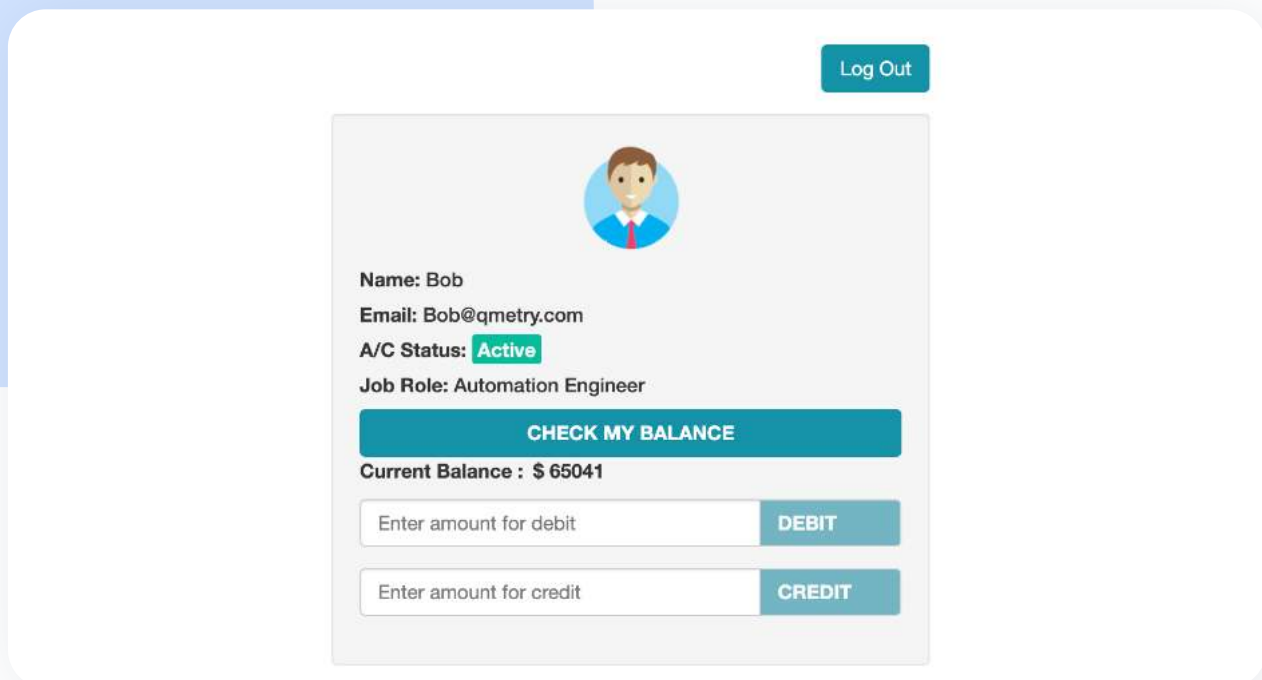
## Step 01



The login form features a teal icon of a classical building with four columns. Below the icon, the text "QMetry Bank Login" is centered. The form includes two input fields: "Username" and "Password". A teal "Login" button is positioned below the password field.

We take an example where there is an online Banking Website that allows user to debit and credit amount from their account. As a first step, user named Bob, will login to this bank application.

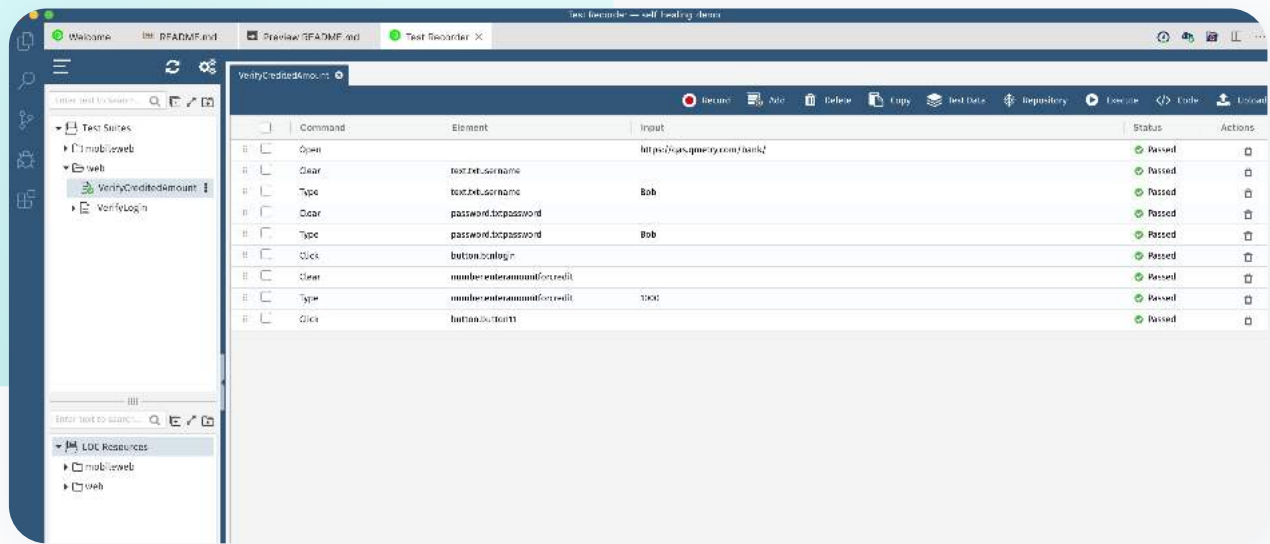
## Step 02



The interface shows a user profile for Bob. At the top right is a "Log Out" button. The profile card includes a circular avatar of a man in a blue suit. Below the avatar, the following information is displayed: "Name: Bob", "Email: Bob@qmetry.com", "A/C Status: Active" (with "Active" in a teal box), and "Job Role: Automation Engineer". A teal button labeled "CHECK MY BALANCE" is present. Below this, the "Current Balance : \$ 65041" is shown. There are two input fields: "Enter amount for debit" with a teal "DEBIT" button, and "Enter amount for credit" with a teal "CREDIT" button.

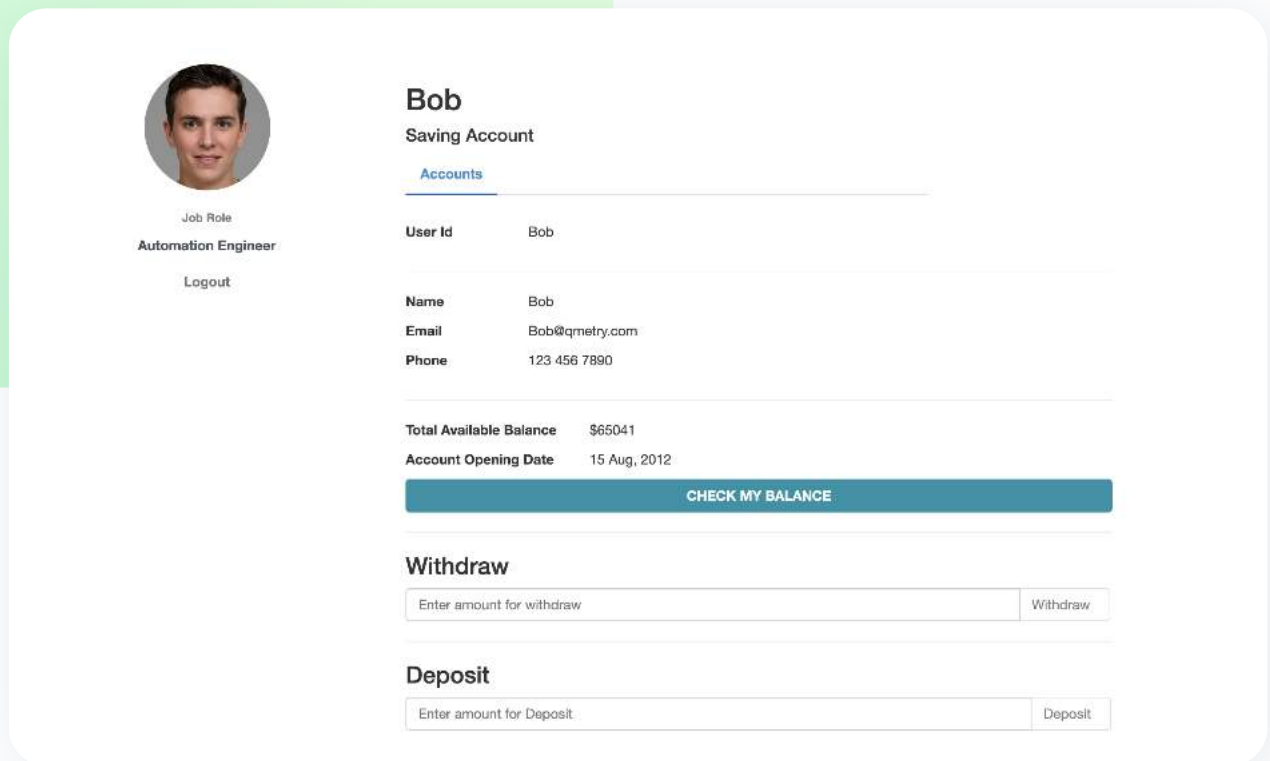
User wants to withdraw \$100 from his account. So, the user who is Bob will enter amount for debit in the Debit Field. After entering amount, Bob will click on "Debit" button.

## Step 03



Now, using QAS, you have scripted the above test case for it to work as automated script. And your test execution within QAS is successful.

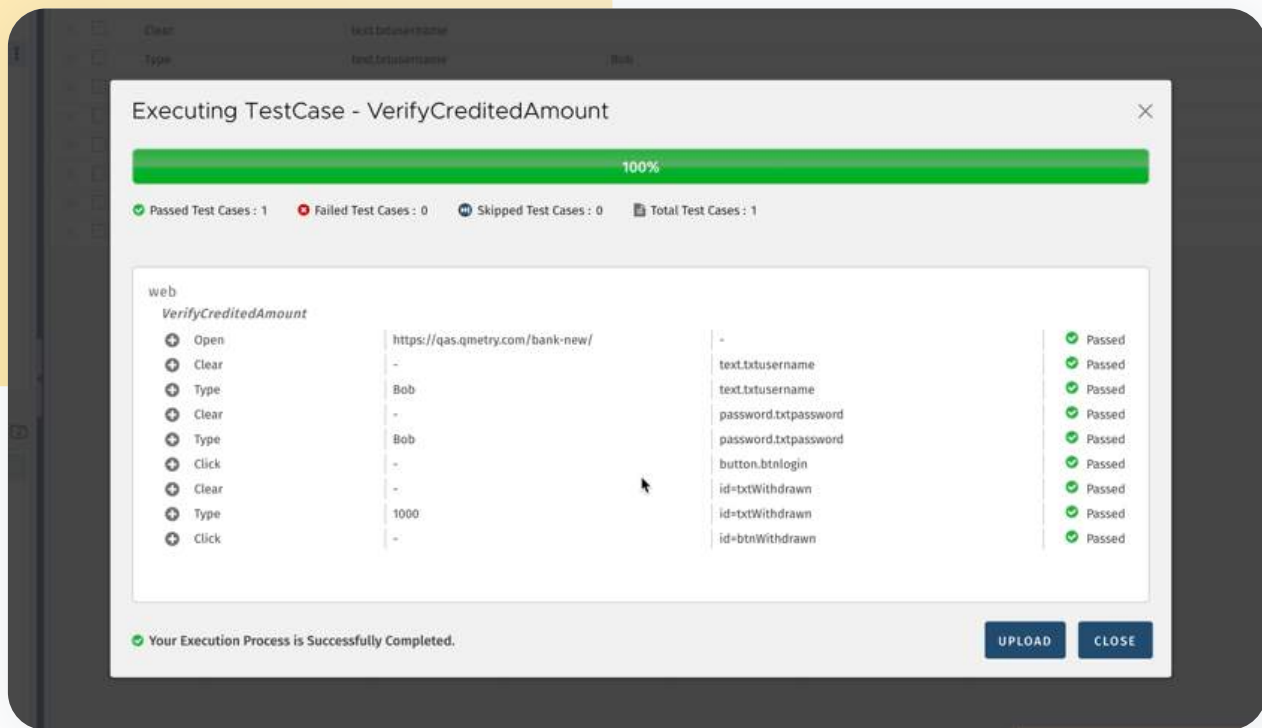
## Step 04



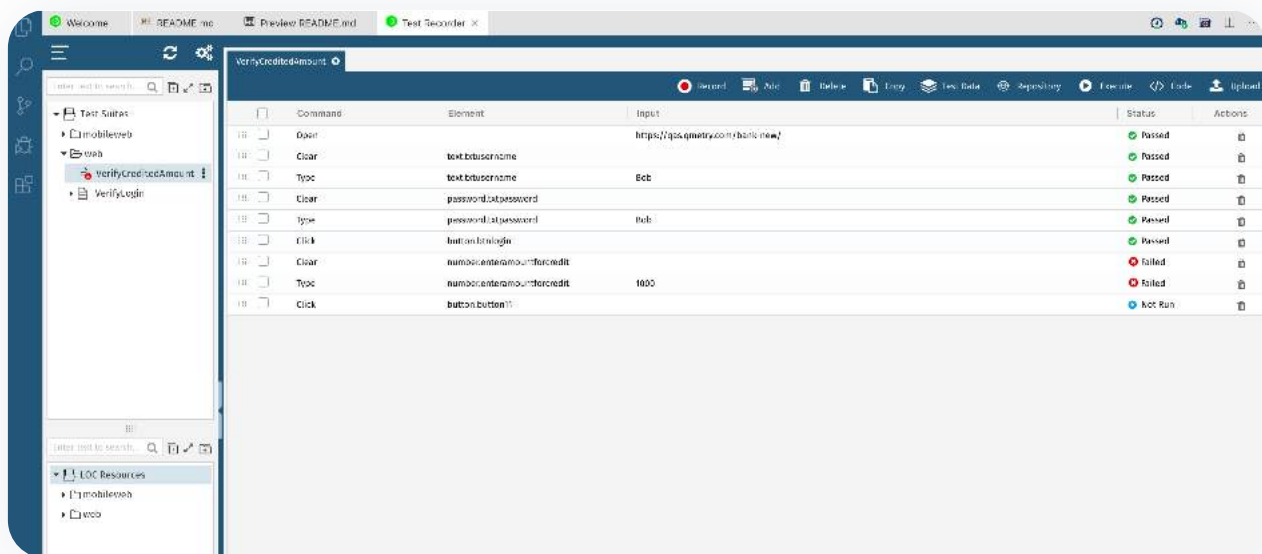
Let's assume there is a requirement for the UI of this application to be changed. And developer changed the design and position of UI elements.

The new UI looks like this. Major changes are - UI is changed and the button name has been changed to "Withdraw" and "Deposit" instead of "Debit" and "Credit".

## Step 05



However, the automated test scripts are not changed as per the new requirement. Hence, the same automated tests run but now for new design. Due to this change, these automated tests fail.

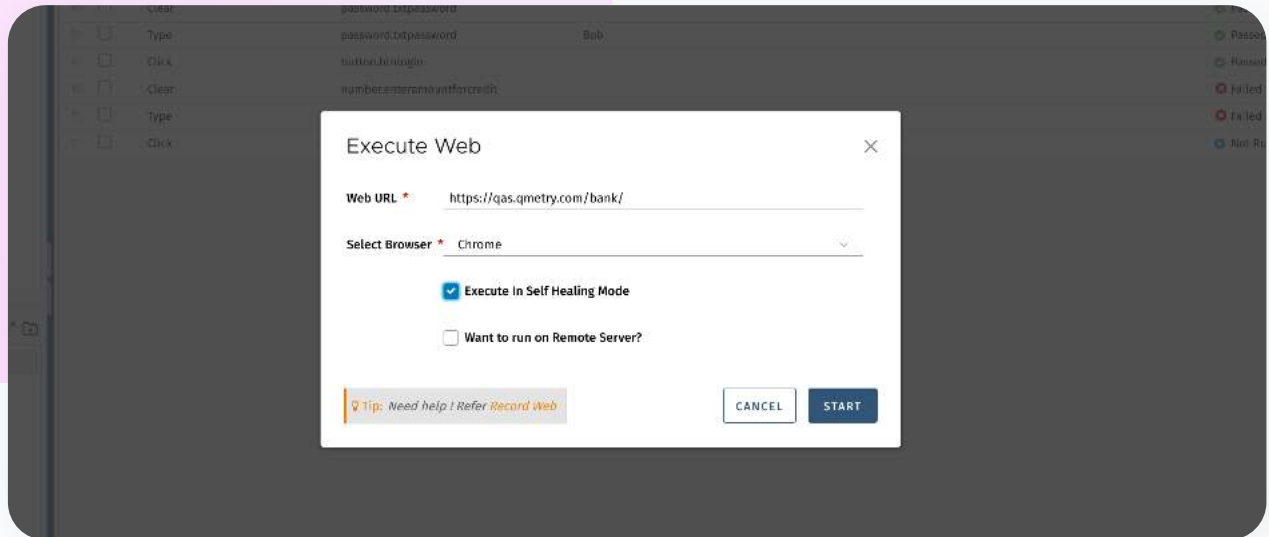


This is the point where automation engineer's time is spent the most in fixing these flaky tests. They have to find out what has actually happened that caused these tests to fail. They have to go to recorder manually and find the new element and verify that, are the tests failing due to change in the UI or are these tests genuinely failing? These increases maintenance time and costs to a large extent.



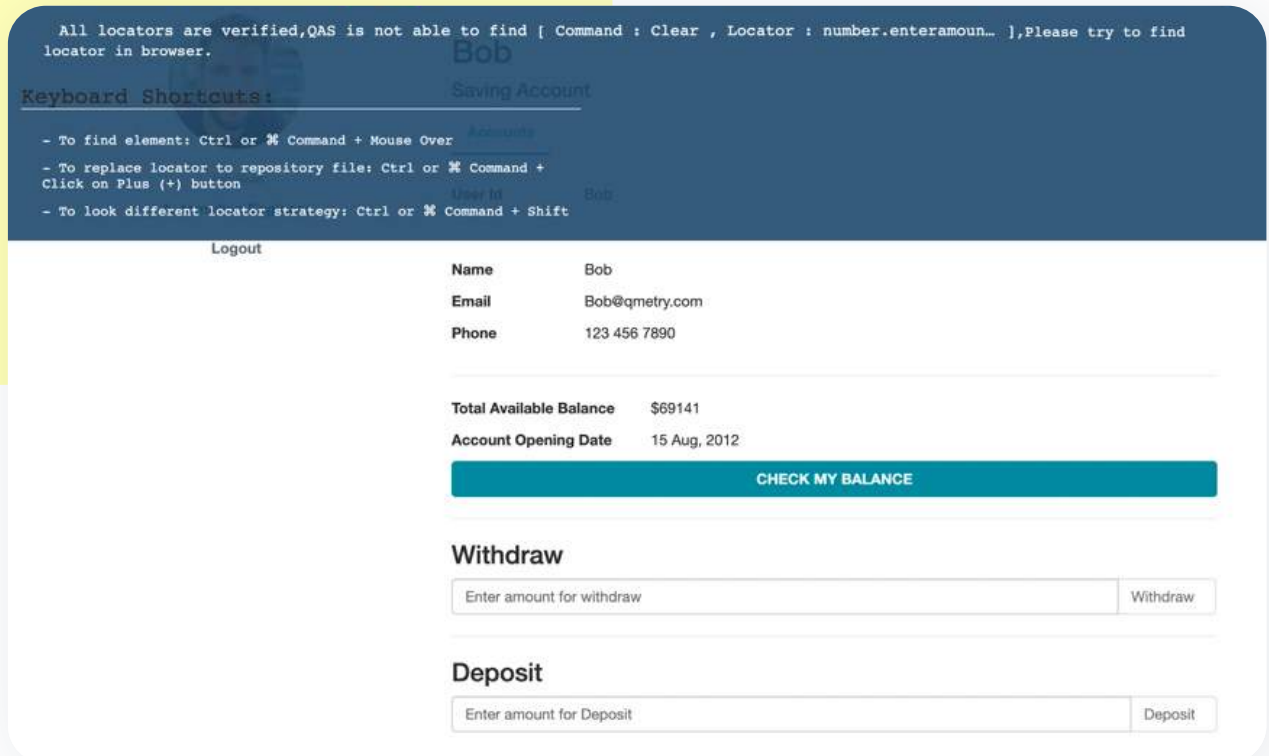
# QAS Self-healing mode

## Step 06



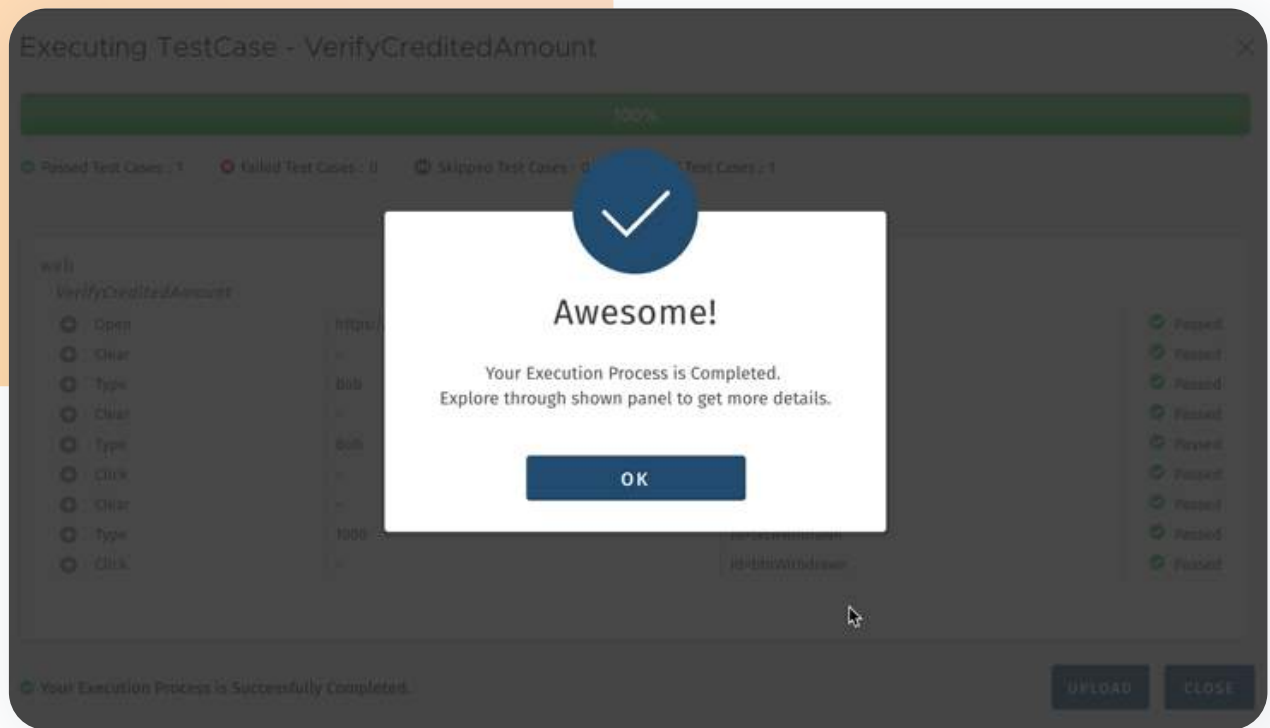
At this stage, I would like to show how QAS's Self-healing mode works. While executing the tests, the engineer would select "Self-healing" mode.

## Step 07



In QAS, Self-healing of scripts would automatically detect altered elements to find out the locators without any manual intervention from testers. QAS will re-test the script and notify the user whether the suggested solution has worked or not. In case any of the alternate elements are unable to get the locators, QAS will give an option to users to fetch the correct locators at run-time.

## Step 08



It also helps in reducing duplicate test cases. As QAS itself detects changes and creates alternates, it reduces the chances of creating a completely new test case for the same use case. With Self-healing, you just alter the current test case and not add a new test case.



# Conclusion

The Self-healing feature is useful when the web pages, on which the test case is recorded, keep changing and so their locators are. If the Self-healing Mode is enabled, QAS does its best to automatically replace locators to run the test successfully. Even if the test fails, QAS gives the user an opportunity to select the suitable locator on the web browser and add it to the script directly. The test will run again automatically from where it failed which will make the test case "Pass".

Self-healing covers the following use cases:

During execution, if a test step locator fails to be detected by its default locator value, the other locator strategies in the list will be automatically applied without any manual intervention from the tester. The execution will continue as if no failures happened.

During execution, if a test step locator is failed and can't be auto detected using any of the other locator strategies, QAS will pause the execution, allow users to select the relevant element and continue the execution. The new locator strategy will be auto updated for the next execution.

Start your 30 days free trial of  
QMetry Automation Studio

[Register Now](#)

Know more about the capabilities of QMetry  
Automation Studio from the factsheet

[Download Now](#)



# QMetry