Degree Programme in Computer Engineering

First Cycle 15 credits

Migration from Manual to Automatic Regression Testing

Best practices for Salesforce Test Automation

REDVE AHMED

Migration from Manual to Automatic Regression Testing

Migration från manuell till automatisk regression testning

Best Practices for Salesforce Test Automation

Bästa praxis för Salesforce test automation

Redve Ahmed

Examensarbete inom Datateknik
Grundnivå, 15 hp
Handledare på KTH: Olof Forsberg
Examinator: Ibrahim Orhan
TRITA-CBH-GRU-2023:100
KTH
Skolan för kemi, bioteknologi och hälsa
141 52 Huddinge, Sverige

Sammanfattning

Målet med denna avhandling är att undersöka om det går att automatisera regression testning för en SaaS-applikation med hjälp av ett serverlöst tillvägagångsätt. Avhandlingen täcker grunderna inom molnkoncept, mjukvaruutveckling, olika typer av testramverk och SaaS-applikationer. Avhandlingen går även igenom gamla arbeten som har gjorts inom området. Rapporten behandlar val av verktyg med åtanke på vad företaget Polestar har för behov. Testramverket ska kunna utföra automatiska regressionstester på SaaS-applikationen Salesforce. Resultatet är ett testramverk som kan köra ett antal utvalda tester på Salesforce. Systemet driftsattes med hjälp av serverlösa Docker containrar på Amazon Web Services. Avhandlingen täcker även alternativa verktyg som kan användas för testautomation och även potentiella förbättringsmöjligheter.

Nyckelord

CI/CD, Selenium, Test Automation, Salesforce, SaaS

Abstract

The goal of this thesis is to explore the possibility on if it is possible to automate regression testing for a SaaS application with a serverless approach. The thesis covers the fundamentals of the software development lifecycle, cloud concepts, different types of testing frameworks, and SaaS applications. The report researches various testing tools that can be used in accordance with Polestar's needs. The testing framework must run the existing tests and deliver the results of the tests. The system must be able to coexist with the testing strategy that is in place today. The result is a testing framework that can run a number of selected tests on the SaaS application Salesforce. The system was deployed with serverless docker containers through Amazon Web Services. The report also covers what a future implementation can look like and potential improvements.

Keywords

CI/CD, Selenium, Test Automation, Salesforce, SaaS

Acknowledgment

I want to express my sincere thanks to Polestar for giving me the opportunity to conduct this study. I'm particularly grateful to Axel Sundqvist for supplying the necessary tools for this thesis.

My gratitude also extends to my colleagues who have been incredibly supportive, providing constructive feedback throughout this process.

I'd also like to extend my thanks to my supervisor at KTH, Olof Forsberg. His patience and assistance have been invaluable during this journey.

Redve Ahmed

Stockholm, May 2023

Table of contents

1 Introduction

As software continues to grow so does the scope of testing for the software. Testing can be done in many ways to suit the needs of the developers. A form of testing is regression testing. Regression testing aims to test all code changes in order to ensure that the existing functionality of the application works as intended. However, this approach to testing can be demanding and time-consuming due to the need to test all changes.

1.1 Problem description

Testing is a vital part of any Software Development Life cycle (SDLC). Testing is conducted to ensure that the software works as intended and to find potential bugs. Testing of software can be achieved in two ways and that is manual testing and automated tests. A big problem that many companies face is that Software as a Service (SaaS) applications do not come with adequate testing tools. This makes developing in-house solutions for testing SaaS applications a necessity for accomplishing efficiency and streamlined testing.

The company Polestar has already begun an investigation on which framework will be the most optimal for use with the SaaS application Salesforce. This thesis aims to make use of the chosen tools by the company to implement and demonstrate a testing framework that can be integrated with today's testing strategy.

1.2 Goals

The goals are the following:

- Examine which cloud providers offer serverless container solutions
- Establish if automatic regression testing can replace manual regression testing
- Examine whether a serverless approach is suitable for a SaaS application testing framework
- Determine which test automation tool fits Salesforce best according to Polestar's needs
- Develop and write tests with the chosen test automation tool in accordance with the provided scripts

1.3 Boundaries and delimitations

This thesis will be limited to only regression testing. Another boundary is the current testing strategy that is in place as this report aims to see if a new testing environment can be integrated with the current testing strategy. The thesis will only automate two test cases as most cases are similar when it comes to automating them. The thesis will only cover Selenium and Selenide thoroughly as these two test automation tools are within the scope of the company when it comes to test automation tools.

1.4 Method

The method used in this thesis is to do a literature study first in order to gain knowledge on the best practices for test automation. The literature study will also include what a SaaS application is and how the cloud works fundamentally. The approach will then be determined based on the findings made in the literature study. An implementation will be done with the selected tools and architecture. An evaluation will be done after the implementation.

1.5 The author's contribution to the thesis

The investigation, analysis, conclusion, and implementation of the system were done by the author of the thesis. All the resources such as software and guidance from Salesforce were provided by Polestar. This thesis is a collaboration between the author of the thesis and Polestar.

2 Theory and background

This chapter includes the necessary theory and background so that the reader can grasp the research topic. Section 2.1 chapter lays the foundation for what a SaaS application is and how they work. Section 2.2 will include the history of software testing and what software testing is while section 2.3 introduces different cloud concepts. Section 2.4 covers testing thoroughly and different kinds of testing, lastly section 2.5 covers related work that has been done within the topic.

2.1 Software as a service

Software as a Service (SaaS) has become a very attractive offering for many companies as it eliminates many blocks from traditional software development [1]. SaaS applications are provided through the internet. SaaS applications are maintained and updated by the vendor. SaaS applications are delivered through the internet and therefore only require a functioning internet connection and a web browser. The applications are accessed through a web browser. This frees users of complex software installations. SaaS applications are often subscription-based services as compared to traditional software.

2.1.1 SaaS infrastructure

Vendors handle the underlying infrastructure for instance the database and the servers [1]. This eliminates the need to constantly update and manage the underlying IT infrastructure. This allows the companies that adopt a SaaS application to focus solely on configuring the software to their preference. SaaS applications also eliminate the need to manage backups, as these features are included within the different license forms. Scalability is a crucial factor in modern software and SaaS vendors manage this aspect as the software operates on their infrastructure.

2.1.2 SaaS security

As the internet has gotten bigger and bigger so has the demand for security. Security is a crucial part of any software and can often be one of the most difficult parts to develop and maintain [2]. This is also another key area that the vendors take care of [1]. The vendors take care of the complex security such as hosting the application securely and making sure that the application follows today's standards regarding web application security. With that being said this does not eliminate every aspect of security from the customer as they are still responsible for customizing the right user privileges.

2.1.3 SaaS limitations

SaaS applications can still pose some limitations to organisations who are willing to adopt SaaS applications instead of developing in-house solutions. The main limitations are the vendors themselves as they are the ones who are in charge of what functions get implemented [3]. This is also one of the main boundaries to why some companies are not willing to adopt a SaaS approach as there can be missing functionality that can be critical to the business. The adaptation of a SaaS application involves giving up some control. Although this can streamline operations however, it also means relying on the SaaS provider to keep and provide the right functionality.

2.1.4 Salesforce
Salesforce has become one of the leading Customer relationship management (CRM) systems boasting over 19% in the CRM market share according to Forbes [4]. A CRM is somewhat of a business strategy that companies can adopt to keep track of customer interactions and establish new potential customers [5]. A CRM can be adopted by many different businesses as it helps streamline everything from marketing and sales. Companies choose to implement a CRM into their business plan as it is a perfect tool for combining technology and with sales.

2.2 Software development life cycle
The Software development life cycle (SDLC) is a process that is used by developers to define a systematic approach to software development [6]. This section will cover how testing is a vital part of the SDLC (section 2.2.1) and how development operations is a part of the SDLC (section 2.2.2).

2.2.1 Testing
Testing is a vital part of the SDLC and can arguably be one of the most import ones in the SDLC [7]. The primary goal of testing is to ensure quality and that the software works as intended. The scope of testing increases alongside the software, this means that testing can become very complex as the software evolves. This is due to needing to test more features that gets added to the software.

Software testing can be conducted in two ways and that is manual testing and automated testing. Manual software testing is when a human manually executes different test cases and interact with the software [8]. The tester follows predefined test scenarios to validate that the software works as intended. All the test scenarios have an expected outcome so that the tester can verify if the feature works as intended. Manual testing is labour-intensive as it requires a human to perform repetitive tasks. Automated testing involves using automation software to test software. Automated testing executes predefined test cases and is meant to replace manual testing.

2.2.1.1 Regression testing
Regression testing is conducted to verify that newly implemented features does not disrupt existing features [9]. Regression tests are designed to grow alongside the program, therefore modifications and iterations are essential to maintaining good software testing. This form of testing can be seen as one of the last steps before deployment, as regression testing tries to achieve as much code coverage as possible. This can make regression testing vital to some companies as this is what ensures them that their product works as intended. Regression testing can take a lot of time as the tests themselves can grow very large alongside the software that is being developed.

2.2.2 Development operations
Development Operations (DevOps) has gained very good momentum within the software engineering world [10]. DevOps aims to integrate Development and Operations into one continuous loop while still maintaining an agile workflow. DevOps also improves continuous delivery and continuous integrations this due to

always trying to keep everything within a closed working loop. The DevOps principle makes use of existing tools and software to achieve the desired workflow.

2.2.2.1 Jenkins

Jenkins is an open-source automation software that can be used to build, test, or deploy software [11]. Jenkins can be installed in various ways such as using the systems package manage and through Docker. Jenkins is a tool that allows for seamless integration with other services through remote triggers such as webhooks.

2.3 Cloud concepts

Understanding cloud concepts and what the cloud can provide has become a very important aspect of IT in general [12].

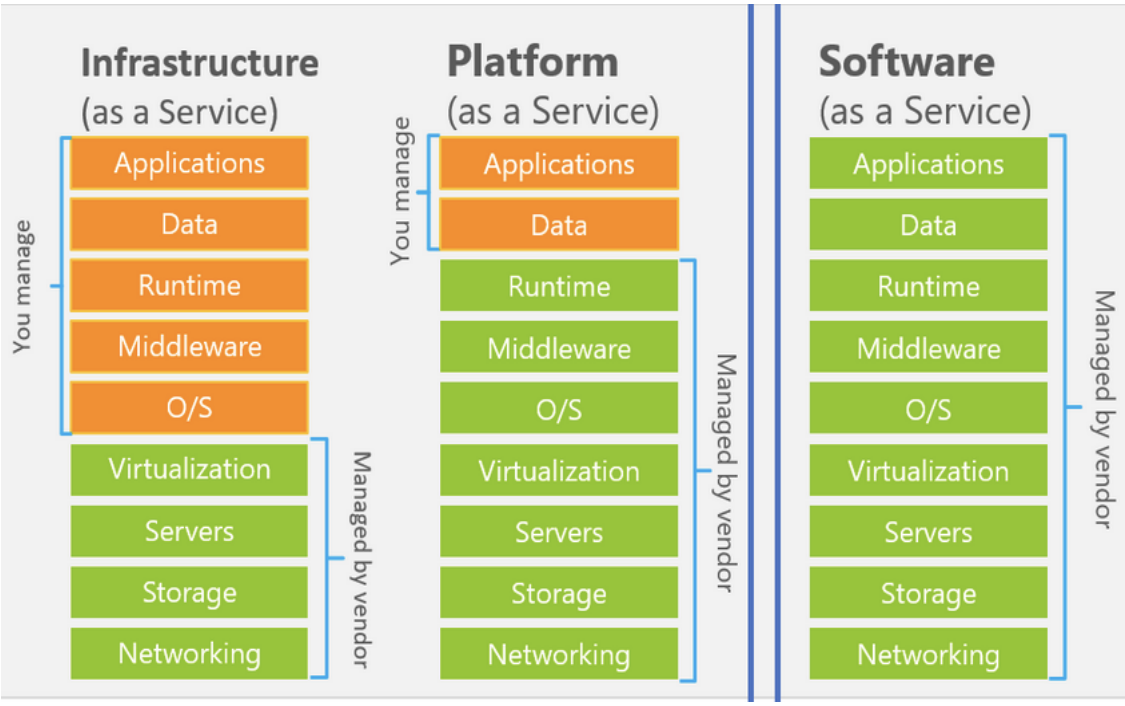

Figure 2.1 Illustration of as a Service offering (Microsoft, 2023)

The cloud introduces three key services to the users as depicted in figure 2.4. These three services are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). There are a handful of more services that has been introduced alongside the cloud such as Serverless, but these services are mainly a hybrid of the aforementioned services. All the services are charged accordingly to a pay-as-you-go model that is provided by the chosen cloud provider.

2.3.1 Infrastructure as a service

IaaS is the service that offers the most customization and control as depicted in figure 2.4. IaaS essentially replaces the need to buy IT hardware and makes the payment go to a pay-as-go [13]. This means that the people who are using the services are responsible for configuring the operating system and everything above. IaaS also helps the companies to scale their infrastructure accordingly as they can just scale down the unwanted resources.

2.3.2 Platform as a service
PaaS sits in between IaaS and SaaS and allows the users to get a preconfigured machine with OS and selected middleware installed [14]. A good example of a PaaS is a database management system as this is preconfigured from the vendors with the right configure for deployment. PaaS eliminates having to make the choice about which hardware to use, this can be seen a win compared to IaaS [13,14].

2.3.3 Software as a service
SaaS is mentioned previously in this report (See section 2.1) but this part will cover on where SaaS places in compared to the rest of three key services [15]. SaaS is the service that offers users the least amount of control and customization as depicted in figure 2.4. This service provides users with applications over internet. All the underlying aspects of this service is handled by the chosen provider.

2.3.4 Serverless
Serverless is a cloud service that was born from the cloud for the cloud [16]. A serverless approach allows the user to completely disregard the underlying hardware and infrastructure. This makes serverless a hybrid between PaaS and SaaS. Serverless does not offer any software it only offers runtime environments. Function as a Service (FaaS) is one of the more commonly known serverless approaches and is often the one that is referred to when mentioning Serverless. FaaS allows the developer to write code in selected language and deploy to the chosen cloud provider. This code is then managed by the cloud provider [17]. Containers can also be deployed serverless (see section 2.3.5.3).

2.3.5 Deploying to the cloud
Deployment of applications can be done in many ways with each public cloud provider offering their similar ways of deployment. Virtual machines and containers are the most used when deploying to the cloud.

2.3.5.1 Virtual machines
Virtual Machines (VM) are perhaps one of the most traditional ways of deploying applications besides barebone [18]. A VM is one of the more demanding forms of virtualization as it requires much more to be virtualized for each running instance of a virtual machine. A VM requires a hypervisor, followed by an installation of a guest operating system.

2.3.5.2 Container
A container on the other hand can be easily compared to a VM as the virtualization looks very similar for both the methods [18]. A container can be seen as a more lightweight option to a Virtual Machine as it doesn't require as much resources as a virtual machine. The container runtime engine is more lightweight when compared to a hypervisor. A container engine can intelligently distribute resources needed to each container and each container can be ran on the same engine [19]. Containers are mainly run as Linux containers, this allows the engine to have a shared operating system across each container. This makes containers more lightweight as they do not need to have a separate guest operating system for each container. The container engine only needs to have the right dependencies for the containerized application

to run. A containerized application is in general smaller than an application deployed on a virtual machine as the virtual machine needs to have a whole operating system per application. Containers allows for more separation between applications as they are so lightweight and can be managed very easily through applications such as docker.

2.3.5.3 Serverless containers

Containers can also be deployed as serverless containers through different cloud vendors such as Amazon Web Services (AWS), Google Cloud Platform (GCP) and Azure. Serverless containers removes the need to manage a server which has the container runtime engine [20]. The container runtime engine is handled by the chosen cloud provider. The computing recourses are set by the user but can easily be changed if needed. All the major cloud providers offer similar solutions to serverless Containers [20,21].

2.4 Testing frameworks

This section will focus on different testing frameworks for different languages and different applications this is to give context on why testing frameworks are needed for any modern software or languages. This section will also cover on what test automation is.

2.4.1 Testing frameworks with Java

Java is still one of the most used languages used according to a stack overflow survey done in 2022 [22]. There are many different testing frameworks for Java and each framework has different use cases. TestNG allows developers to have different configuration files for different test suites and allows for parameterised tests [23]. Frameworks such as TestNG also allow the developer to produce automated reports on how the various test cases executed.

JUnit is another popular testing framework within Java [24]. JUnit is a very simple testing framework. JUnit is similar to TestNG in terms of features. The main difference is that JUnit does not have as many annotations as TestNG does. Annotations in Java is used to supply the compiler with predefined information.

Both JUnit and TestNG provide test suites but are achieved in different ways. Both frameworks offer parameterised tests however, this is achieved very differently for both frameworks. TestNG makes use of XML files as configuration files while JUnit makes use of classes.

Assertion are often used when writing tests. An assertion is a sort of statement that is believed to be true [25]. Assertions play a key role in testing as this is used in order to make sure that the test that has been executed gets the desired outcome. The assertion method will throw an exception if the desired outcome is not present at the end of the execution thereby failing the test.

2.4.2 Testing within Salesforce

Salesforce includes a statically typed language which is called Apex, the language has close resemblance to Java [26]. Apex is mainly used for writing business logic and

database procedures. Test classes can be written with Apex however, this does not allow the developer to write tests that cover the whole codebase. This is due to Salesforce user interface being built by the vendor and not by the developer. The HTML and the styling for the website gets generated by Salesforce. This makes testing difficult as the developer has no control over how the site gets generated. This has led to various companies creating their own testing frameworks for Salesforce [27,28]. The main problem with the test frameworks that are on the market is that they are very expensive. This makes adopting open-source alternatives such as Selenium, Cypress and Playwright much desired [29].

2.4.3 Selenium web driver

Selenium was developed with the primary goal of automating web browser. Selenium allows developers to run various kinds of web browsers through the Selenium API [30,31]. Selenium can use various types of locators to find specific elements in the document object model and this makes it very suitable for all kinds of web application testing [27]. There are a variety of selectors in Selenium for instance the id selector, class selector and XPath selectors. It is up to the developer on which kind of selector to use. The selectors are very important as they are the one who locates and interacts with elements with the HTML.

Selenium is supported on a variety of different languages such as Java, C#, Python and JavaScript this makes it very accessible for any developer [27]. The web driver additionally supports the ability to wait for a web page to load, thereby emulating real usage as much as possible. This can be very important as some heavier web pages tend to take much longer to load then others. Selenium can be used in conjunction with other frameworks such as TestNG to create a fully-fledged testing framework.

2.4.3.1 Page object model pattern

Design patterns are very common within software engineering and test automation is no different to that as there are also design patterns here. A very common pattern is called Page Object Model (POM) and the goal of POM is to abstract different web pages into classes [32]. This can be compared to objects in object-oriented programming. POM allows for reusability of different methods across various tests. This in turn makes the code non repetitive. An example of this can be a login page, as the login page can be used in a variety of different tests.

2.4.4 Test automation

Test automation was briefly touched upon in this report, but this subsection will cover on how to transition to test automation through previous done work. It is important to establish how to make a smooth transition to automated tests.

Test automation is the practice of using automation tools in order to achieve testing without humans performing them [33]. Test automation aims to replace manual testing. Test automation can help improve testing speeds and reliability of tests. However, test automation still requires good precision and good planning in order to work properly. A poorly planned implementation of test automation can lead to more maintenance and more costs in the end. This makes it crucial to plan and implement test automation in the right way. It is important to do an assessment of

the current testing strategy in order to see what can be carried over to the new test automation framework. Test automation can be seen as very difficult in the beginning of the implementation as this is where the most time is need for it to function properly. Although this can in the end benefit a company greatly as it helps decrease the amount of time needed for testing. The tests themselves do need maintenance as they to evolve alongside the codebase.

Test automation can be seen as a must today as it helps companies to save time during their workload. Test automation can be well incorporated with today's scrum standards.

2.5 Related works

This section will present previous done work that are used in this thesis.

2.5.1 A journey from manual testing to automated test generation in an industry project

Claus Klammer and Rudolf Ramler [33] discusses on the challenges and difficulties of migrating from manual testing to automated testing. The authors of the paper makes an attempt at creating a test automation framework for a GUI application that is written in Java which ends up working properly. The authors put a heavy emphasize on that a proper plan is need in order to migrate over to test automation. This paper also covers the best way to run tests.

2.5.2 Testing using selenium web driver

Paruchari Ramya, Vemuri Sindhura and P Vidya Sagar published a paper [30] on how to automate web application testing with the help of Selenium. This paper goes through which tools is to be selected in combination with Selenium in order to achieve the best test automation framework. This paper discusses the key differences between test automation and manual testing and why test automation if done right can replace manual testing.

The implementation resulted in a transition to test automation. This was due to the benefits that came with the implementation of test automation such as reports for how tests went and a positive outcome in terms of accuracy for the tests.

2.5.3 Analysis and design of Selenium webdriver automation testing framework

Satish Gojare, Rahul Joshi and Dhanashree Gaigaware wrote a paper [31] that extensively researched the design of Selenium Webdriver. The authors claim that Selenium alone is a incomplete web testing tool and concluded that more features needed to be added for it to become a complete web testing tool. Hence the report investigated on which tools could be added for alongside Selenium in order to make it the complete web testing tool. This resulted the authors using TestNG alongside developing their own functions to make it complete. These tools were screenshot generation upon failed test as the authors concluded that this would help the developers to analyze why the test failed.

2.5.4 A survey of the Selenium ecosystem

Boni Garcia, Micael Gallego, Francisco Gortazar and Mario Munoz-Orgarnero released an article [29] in which they did a survey on the Selenium ecosystem. This survey covered popular tools that are used alongside Selenium such as Jenkins and Selenide. This survey also covers the most popular languages that are being used for Selenium, the most popular language according to the survey was Java. This paper was used when deciding on which frameworks to use.

3 Methods and results

This chapter of the report will cover the implementation of the testing environment and an assessment of today's testing strategies. The result will also be presented in this chapter.

3.1 Preliminary studies

This thesis work started with outlining a plan for how to limit the thesis for it to fit the time span of the project. This allowed the author to define clear goals for this thesis project. The thesis work started with doing research around what a SaaS application is, and what limitations there is with a SaaS product. The articles used to find information was found through IEEE Xplore, ResearchGate, Google scholar, Diva and ScienceDirect. Testing frameworks and understanding of how the Software lifecycle was done in order to create an architecture of the testing environment that fits the current testing strategy.

It was concluded through the literature study that a migration from manual testing to test automation is possible [30,31,33].

3.2 Requirements from the company

Based on the requirements provided by the company Polestar, the corresponding test automation framework that is to be developed should provide the following features:

- Run automated tests in Salesforce
- Provide results upon completion of tests and send email notifications
- Be able to rerun the tests if desired
- Tests should be easy to maintain
- The test framework should be simple and easy to maintain
- Provide some sort of help to developers if a test case fails

The tests must be written in Java as it closely resembles the programming language Apex. The tests must be written using the Selenium or a similar tool that is still within the Java atmosphere. The testing framework should be able to run the current tests which are being performed manually. The last criteria is that the testing framework should complement the current working procedures which is scrum.

The implementation should be as cheap as possible as this framework aims to cut costs as compared to the tools that are available (see section 2.4.2).

3.3 Assessment of current testing strategy

An assessment of the company's current testing strategy is needed to decide which tools are going to be used in the test automation. According to documentation provided by the company, the testing strategy for today's regression testing is that they perform all the tests manually after each sprint. Each sprint lasts approximately two weeks, which means that the regression tests get performed manually two times a month if nothing goes wrong. The regression tests are performed again if

something does go wrong, this is to ensure quality and that the final product meets expectations. There are in total 15 different tests today and each test differ very much in how long time it takes to execute the tests. However, all of the tests do involve the same actions and these actions are to press certain elements in Salesforce and check to see if the expected results are present. The test cases that exist today have a clear expected outcome of every step in the test script. This makes the transition to automated test scripts smooth as there is no need to make modifications to the current test scripts.

It is important to outline the current testing procedures and how changes are shipped to production. A developer first makes changes locally and tests them locally in order to understand if the changes made gave the desired outcome. The updated code then gets merged with a testing branch which everybody has access to. This is where QA tests the changes to see if the changes does what they are supposed to do. These tests are unit tests and are only meant to test each feature individually. All the changes then gets merged into the staging branch, this is the last branch before the production branch. This is where the regression testing is conducted before merging all the changes into production.

However, the tests that exist today are somewhat limited as adding more tests that are preformed manually would increase the time required for testing drastically. This makes the desire to try and adopt test automation even more demanding.

3.4 Selection of tools

The available tools need to be assessed in order for them to be used in the implementation. This assessment is done in this section of the chapter. The chosen tools should fulfil the requirements outlined in section 3.2.

3.4.1 The use of Selenium

Selenium was developed for web automation, enabling developers to automate interactions within a web browser. Actions such as filling out forms, clicking buttons and extracting data are among numerous actions that can be done with Selenium [26]. This has enabled developers to use Selenium for testing both static and dynamic websites. Selenium is widely supported by most browsers making it one of the industry standards when it comes to web automation. Selenium has a wide community making it easy to find information regarding Selenium.

Advantages of using Selenium:

- Selenium supports all the programming languages. This allows developers to write tests with the language they are comfortable with [26].
- Selenium has a large community of developers which are constantly contributing to its improvement [26].

Drawbacks of using Selenium:

- Selenium requires complex setup as it is the developers who has to download the browser driver [26].

- No standardised way of waiting for an element to appear or disappear [26].

3.4.2 The use of Selenide

Selenide is a web automation tool that is built on top of Selenium. Selenide was created in order to overcome the difficulties that come with using selenium. This can be shown through the implementation of automatic waits in Selenide [34]. Selenide makes code more readable as it is very concise.

Advantages of Selenide:

- Selenide has built in support for screenshots. Screenshots are taking upon a test failing [34].
- Less code is required as the Selenide library abstracts away parts of the code [34].

Drawbacks of Selenide:

- This tool only has support for Java and other languages that run on the Java Virtual Machine (JVM) such as Kotlin [34].
- Developers choosing to opt for Selenide has less control as the library abstracts parts away from the developer [34].

3.4.3 Comparison between Selenide and Selenium

Both Selenide and Selenium are great options for web automation. Hence making them both very viable for any type of web automation project. But the tools do have some key differences. Selenide in comparison to Selenium does not require the developer to download browser binary as this is handled by the Selenide library the first time a test is ran. Many of Selenium features still carry over to Selenide as Selenide is a wrapper that is built on top of Selenium. Selenium offers very good community engagement which makes it easier to find solutions to problems that may occur when trying to write test script. Selenium offers a variety of languages compared to Selenide which only offers Java and those languages that can on the JVM.

3.4.4 Choosing between Selenide and Selenium

The choice of Selenide and Selenium ultimately comes down to the developer and their needs. The company does have certain needs when it comes to the tools as outlined in section 3.2. One of these being that the chosen tool for web test automation must be in Java. This makes Selenide and Selenium still viable as both have support for Java. There are two key outlining's in section 3.2 that heavily favours Selenide and these are that tests should be easily maintainable. Tests written in Selenide requires less code than Selenium thus lowering the complexity of the code [34]. This is due to Selenide having more built-in functions as suppose Selenium. Developers are less prone to having to write their own methods, an example of this is waiting for elements to load this is due to Selenide having built in wait methods.

The second is that the tool should have some sort of support for developers if the tests fail. Selenide excels in this area since it has built-in support for screenshots if a test fails. This can be a big help to developers who are debugging the code as they can exactly where the test failed. A screenshot can also help when going through old bugs and can help greatly with bugs that get detected in the future. Selenium's lack of a screenshot feature was highlighted in "Analysis and Design of Selenium WebDriver Automation Testing Framework" [31]. This led the authors to develop their own screenshot feature that works alongside Selenium. The result of the implementation resulted developers needing less time to identify errors in the code ultimately lowering maintenance cost.

A common issue with Selenium is the flakiness of the tests [29]. A flaky test case is when a test gives inconsistent results despite having the same prerequisite. Flakiness in Selenium is mainly caused by there being different wait strategies. This can be mitigated by using a common wait strategy however, this is easier said than done. As this can lead to more maintenance of the test code due to developers needing to check each other's test code. This is where Selenide has a big advantage as Selenide has default wait methods. A lowered number of flaky tests can also lead to less time spent on maintenance [34]. This will allow developers to continue develop new tests instead of spending time on tests that are flaky. Selenide was chosen due to its promise of reducing flaky tests.

3.4.5 Choosing of TestNG and Jenkins

Earlier work has concluded that TestNG can be great in order to create reports after tests has been executed [31]. This was also one of the requirements outlined in section 3.2. The work also concluded that TestNG was designed to overcome the limitations of JUnit [31]. TestNG has support for parameterised testing which allows the developer to run the same tests but with different parameters through xml files. This can be of great use as this enables developers to define pre-set values using different xml files. TestNG can cover all form of tests where JUnit only is meant to cover unit tests.

Jenkins was used as it has great support for TestNG and the reports that it produces. The test code does not need to be pushed to Jenkins, as Jenkins is able to get the through Git and Github remotely. This makes it simpler as we do not need to introduce yet another tool. Jenkins also has a feature which delivers the report that gets generated by TestNG upon completion in the form of an email to selected receivers. Jenkins is also the most used CI/CD platform for Selenium according to a paper done 2020 [29].

An alternative solution would be to have tests run locally instead of having them run in a separate Jenkins instance. This is the most popular choice among developers who do not opt to run tests in a Jenkins instance [29]. This alternative will not fit this sort of implementation. Because there is no way to integrate it with the current DevOps solution which is one of the goals outlined in section 1.2.

3.5 Assessment of cloud providers

It is important to choose the right cloud provider when deploying to the cloud. Different cloud providers offer different services, and each cloud provider has different payment plans. The maturity of cloud provider can play a huge role for many companies as they want to opt for the more established cloud provider. AWS, Google Cloud Platform and Azure has been researched in this report.

This is where it can be difficult to choose cloud provider as all the major providers all offer very similar services only with different names. All the aforementioned cloud providers also have servers in Sweden except for GCP. The closest servers to Sweden are in Finland in GCP's case.

The system needs to be easily maintainable as outlined in section 2.3. This greatly enhances the appeal of selecting a serverless container. This is because serverless containers abstract away the underlying infrastructure. The requirements for what the cloud provider is supposed to offer is not that complex for this implementation. The only service that is needed in this case are serverless containers. A more complex implementation would have taken other services into consideration.

The offered services can differ from cloud provider to cloud provider. This makes it crucial to understand what kind of services is needed [35]. Serverless containers was looked at for this implementation as this would ultimately make development easier. This is due to serverless containers not needing any provisioning of the underlying infrastructure (See section 2.3). Serverless containers can be used for this particular implementation as the main objective of this framework is to be able to do regression tests on a SaaS application. The use of serverless containers fits perfectly into one of the requirements from the company which was outlined in section 3.2. The requirement is that the test framework should be easy to maintain. All the researched cloud providers do offer a solution for serverless containers [20,21,36].

The security of a cloud provider plays a crucial role when selecting a cloud provider as stated by the NCSC [37]. The NCSC highlights that it is important to check if the considered cloud provider works against selected frameworks such as CSA star. All the aforementioned cloud providers have achieved a level 2 CSA star [38]. A level 2 star indicates that the audit was done by a trusted third party. It is important to look at what the specific use cases are. This implementation is aimed to towards testing a SaaS application. It is because of this that the security of the application is not taken into consideration as this is handled by the vendor. The tests are going to be executed in a testing branch which is completely isolated from the production branch.

All the aforementioned cloud providers do fit perfectly with this sort of implementation as all of them do offer the right service for this implementation. All the cloud providers do also work towards the same frameworks and have the same certifications when it comes to security [38]. This makes the choice ultimately come down to price. This is due to all the aforementioned cloud providers offering the service needed for this implementation and follow the same security frameworks. Cost does play a significant for this implementation as the goal is to develop an

inhouse framework that has less maintenance cost than the existing ones (see chapter 1).

	AWS	GCP	Azure
vCPU per hour	0.0445 $	0.0648 $	0.04660 $
Gbit per hour	0.0049 $	0.0072 $	0.00511 $

Figure 3.1 Serverless container cost for AWS, GCP and Azure in northern Europe [21,22,39]

Figure 3.1 shows that AWS has the lowest cost for serverless containers. This makes AWS the best choice for this implementation. It is important to keep in mind that this choice is due to all the cloud providers satisfying the needs. So either one would have worked for this implementation.

3.6 Testing environment architecture

The architecture of the system was made around Bitbucket webhooks as this was going to play a vital role in the system. This meant that the system was going to need some sort of way to handle an HTTP request to invoke the automatic tests.

This made going with a solution using containers the way to go as containers can still be considered serverless if deployed in a certain way. AWS Fargate was the service used to achieve a serverless workflow for the containers as it allows the developer to completely disregard managing an external server.

The architecture was made to showcase how the system would work. The system proposed is relatively simple, as this only covers the testing portion of the DevOps cycle [11]. A workflow sketch was made to get an overview of the desired system.

The proposed flow:

1. Changes are pushed to the regression branch from the QA branch. A branch is an separate version. This is due to the current testing strategy as all changes need to go through unit tests before being merged in to the QA org.
2. Bitbucket then picks up that there has been changes in the regression, this in turn makes Bitbucket send a http request to trigger the tests.
3. The tests get pulled from a github repository.
4. The tests then gets executed.
5. An email notification gets sent to selected receivers with the results.

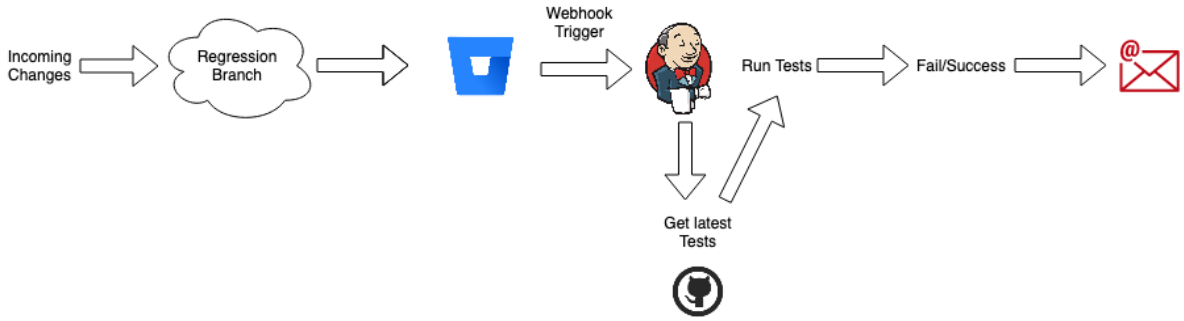

Figure 3.2 Workflow for testing framework

The system proposed is depicted in Figure 3.2. This flow shows how the tests get triggered and what happens when a test fails or succeeds. This flow aims to mirror the manual process as much as possible was mentioned in section 3.3.

Figure 3.2 was inspired by a workflow that was depicted in "DevOps: A Historical Review and Future Works" [11]. However, the workflow depicted in "DevOps: A Historical Review and Future Works" was made for traditional web applications. Traditional web applications involves stages such as deployment and building the whole web application. Changes needed to made as SaaS applications do not need any build or deployment this is due to how SaaS applications works.

The Jenkins instance was deployed as a container through AWS Fargate. Persistent storage was not configured as this was used as a demo, to showcase how a complete testing framework can look like for Salesforce with a serverless approach. Configuring persistent storage is recommended for saving configs and saving files in general if the container needs to be redeployed. The Jenkins instance will first be run locally to build and run the tests. This is a crucial step as the tests need to behave in the expected way.

3.7 Writing tests

The test scripts were provided by the company. Only two of the available tests were chosen to showcase how the final implementation can look like. This is due to the tests involving very similar steps. This made it more viable to only write two tests instead all of them. This was also done in order to fit the timeframe of the project. However, the choice of only writing two out of the available tests may lead to inconclusive results.

The choice of web browser is important as highlighted in "A Survey of the Selenium Ecosystem" [30]. Although this can be disregarded as it is the vendors that handle all the underlying infrastructure for the web application, this includes web browser compatibility.

The two chosen tests is among the 15 regression tests that get manually performed every two weeks. Testcase #1 aims to test if support staff and admins can create client accounts. As well as sending an invitation to the created account. The desired

outcome is that the text “YOUR FLOW HAS FINISHED” should appear after sending the invitation to the user. This test is deemed unsuccessful if the text “YOUR FLOW HAS FINISHED” does not appear after sending an invite. Test case #2 aims to test if two accounts can be merged. Test case #2 can be broken down into the following steps:

1. Create two new accounts in the Salesforce environment.
2. Create one order that are related to each account.
3. Move one of the orders from one account to the other.
4. Merge the two accounts and check if both orders are found within the merged accounts.

The expected outcome if this test case is that the new merged account should have both the orders. This test is deemed unsuccessful if the merged account does not have the two orders.

It is important to first perform the test script manually before developing automatic scripts for them. The knowledge of which buttons that needs to be clicked is important as this is the basis to automating the tests [27]. The developer console in Google Chrome was utilized in order to find the right html elements to click on.

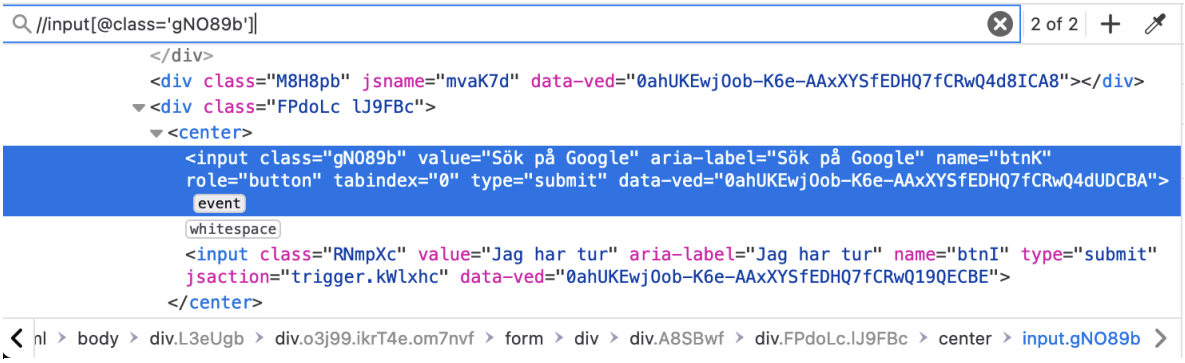

Figure 3.3 Example of XPath using Google Chrome

Figure 3.3 shows an example of how to use the developer console to find the right HTML element. An XPath is one of the many locators that can be used when searching for elements in an HTML document. The XPath locator is the most complex locator that can be used with HTML [35]. This makes the use of other locators much more desirable as they are not as complex as the XPath locator. However, other locators such as ID and classes cannot be used for this implementation. This is due to how Salesforce generates their HTML. The developer of the platform does not have any control over how the HTML gets generated. This raises the complexity of locators for Salesforce.

Both test cases were performed manually first in order to understand what the tests actually did and to identify which buttons to press. This also helped with identifying common pages that both the test cases used. The login page to the Salesforce platform was used by both test cases. Each induvial click needs a corresponding XPath, it is therefore important to test the XPaths in the developer console while developing the tests. It is therefore important to breakdown the test scripts into different steps. A developer then writes the corresponding XPath in Selenide with the help of the XPath locator function. It is then up to the developer on what action to perform.

```
        $x("//span[@title='New Account']").click();
        $x("//span[text()='Next']").click();
        $x("//input[@placeholder='Last Name']").setValue(random1);
        $x("//span[text()='Person Country']//parent::span//following-
sibling::div//div[@class='uiPopupTrigger']").click();
        $x("//a[@title='Sweden']").click();
```

Figure 3.4 Example of XPath in Selenide

Figure 3.4 is an example of how the XPaths are used in Selenide. The writing of the tests involves writing different XPaths. Figure 3.4 also showcases different actions such as clicking on an element and setting a value in an input field. The different XPaths are shown in Figure 3.4. Each line in Figure 3.4 is a separate click. Both the test cases are written in a similar manner. The main problem with writing tests using any sort of web automation tool is the time it takes to write the test [27]. This problem also gets enlarged when trying to write tests for Salesforce due to the nature of how Salesforce sites gets generated. The tests should be periodically executed, this is done in order to assure that the tests do what the developer wants it to do.

The login page was made into a separate class following the POM design pattern. This enhances the code's reusability, as it can be used across various test cases [32]. The code for the login page can be found in appendix A. This code was used in both test case #1 and #2.

The Selenide code for both test cases had to be constantly executed under development. Claus Klammer and Rudolf Ramler highlighted the need to test the test code during development this was to ensure the correct expected results [33].

The code for the two test cases can be found in appendix C and D. It is important to highlight the frequent checks that are made within the test code. This is to minimalize tests that fail due to inconsistencies such as browser timeouts. The test lastly gets validated by inspecting if the desired element exists in the HTML. The test is deemed unsuccessful if the element does not exist. It is important to note that the test code only works with the company's Salesforce platform. However, the test code can be used as inspiration when trying to automate similar SaaS applications such as Salesforce. The tests are ran with the command "mvn clean test" this starts the chosen browser and run the test accordingly to the code.

A problem was encountered during the development of the chosen test cases. The problem is that Salesforce forces multi factor authentication (MFA) when logging on

from an unknown Ip address. Two solutions were established, one of them being to access the email that has been sent through the automated test. However, this can be somewhat complicated as email providers tends to also force MFA, meaning that this outcome is not ideal for this situation. Another solution is to whitelist the IP address in the Salesforce application before running the test. This is a much better way to approach this as the docker container can be set a static Ip address. Although this restriction does not allow for something like Github Actions or Circle CI unless selfhosted runners are deployed. As it would be quite inefficient to whitelist all Github or Circle CI's public IP addresses, this also creates a security risk the IP addresses sometimes changes, and this would also allow for unwanted users to access the application.

3.8 Building and deployment of system

Docker was used to create an image that could be used when deploying to AWS. Docker was also used to test the system locally before deployment.

```
FROM jenkins/jenkins:jdk11
USER root
RUN apt-get update && apt-get install -y wget
RUN wget -q -O - https://dl-
ssl.google.com/linux/linux_signing_key.pub | apt-key add -
RUN echo "deb http://dl.google.com/linux/chrome/deb/ stable
main" >> /etc/apt/sources.list.d/google.list
RUN apt-get update && apt-get install -y google-chrome-stable
```

Figure 3.5 Dockerfile for Jenkins with Google Chrome

The Dockerfile that was created for deployment is depicted in Figure 3.5. This base image is pulled from Jenkins. After that Google Chrome gets installed as this is the chosen web browser.

It is important to expose port 8080 when running the image as this is the port that Jenkins is listening to when receiving requests. The test code was uploaded to Github so that it could be accessed remotely with Jenkins.

An access token to Github was created as the repo was private. The access token only needs read rights as Jenkins is not going to be performing any changes in the test code. The docker container was ran in order to set everything up in Jenkins. Maven was used to build and run the tests. The commands used were "mvn clean test" was used for triggering the tests. A report is generated when the tests have been executed which are then picked up by Jenkins to showcase how tests went.

3.8.1 Deployment

The deployment was made through AWS elastic container service (ECS) as deploying the tests as a FaaS was not suitable for this type of implementation. The first part of the deployment was to create a container repository in AWS so that the container image could be uploaded to AWS. This could also be achieved using any other

repository for hosting container images although this would require some sort of authentication to that repository.

The container image was pushed to the container repository which was set up using the AWS command line interface (CLI). Certain cloud providers have different approaches on how to deploy containers serverless, this report will cover on how the deployment is done through AWS.

A ECS cluster needs to be defined first, a cluster can be seen as a group that groups together all services, tasks and configurations. This is also where the network is defined. The default virtual private cloud (VPC) network was chosen as the AWS account had no prior deployments. A task definition needs to be created. A task definition is used as a template for configuration of the Fargate instance.

An elastic load balancer was created in order to capture all application traffic and distributes them to the desired location. This is needed to send requests to the container instance. This is also where HTTPS was setup, by creating a self signed certificate. This means that elastic load balancer will listen to everything on port 443 and then redistribute them to port 8080 within our container instance. The elastic load balancer gets a randomly generated CNAME which is the only point of internet facing source. This makes it quite easy to achieve encrypted traffic as all traffic within AWS does not need to be taken account for.

The container was deployed using the task definition that was created. The load balancer was put on the cluster so that all traffic on port 443 gets redistributed to port 8080 on the container. The Jenkins instance could then be accessed through a web browser by going to the URL that was provided by the load balancer.

3.9 Results

This section will showcase the deployed framework as well assess if the goals outlined in section 1.2 were fulfilled.

3.9.1 Implementation of framework

The deployed framework followed the architecture that was proposed in section 3.6. Figure 3.2 was the workflow that was implemented. This workflow follows the exact same steps as the workflow in section 3.6. Bitbuckets webhook function was used to trigger the tests [40].

The implemented flow did as following:

1. Changes are pushed to the regression branch from the QA branch. A branch is an separate version. This is due to the current testing strategy as all changes need to go through unit tests before being merged into the QA org.
2. Bitbucket then picks up that there has been changes in the regression, this in turn makes Bitbucket send a http request to trigger the tests.
3. The tests get pulled from a github repository.
4. The tests then gets executed.

5. An email notification gets sent to selected receivers with the results.

AWS was chosen as the cloud provider however, GCP and Azure could have been used as well. The choice ultimately came down to pricing as all the aforementioned provided a service for serverless containers. Although it is important to carefully research on what services may be needed in the future. This specific implementation only needed to utilize serverless containers, this is the reason as to why other services were disregarded. Cost can be considered as one of the main reasons as to why companies are not willing to adopt Salesforce automation tools that are already out on the market. This makes AWS the best choice for this specific implementation as it also follows the company's needs outlined in section 3.2.

Two out of 15 tests were chosen for this implementation, this was due to all the tests involving similar steps in terms of developing automized tests for them. The test code can be found in the appendix section of the report. Selenide proved to be a very nice tool for test automation as it had very good built-in support for waits. The tests were very robust in terms of not failing due to browser timeout. A report gets generated once every test has been executed. The report can be viewed in Jenkins, a separate HTML report also gets generated. The report is generated with the help of TestNG, parameters are also listed in the report. Figure 3.6 is a generated report for test case #1.

Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
Suite							
My Test	5	0	0	0	30 049		

Class	Method	Start	Time (ms)
Suite			
My Test — passed			
Tests.psidInvite	Login	1692827138717	2182
	createAccount	1692827140904	6694
	loginAsTest	1692827150519	8813
	sendInviteAdmin	1692827147599	2918
	testInviteAsTest	1692827159334	5244

My Test

Tests.psidInvite#Login

Parameter #1	Parameter #2
[redacted]	[redacted]
Messages	
Logging In Logged In	

back to summary

Figure 3.6 Generated report for test case #1

Figure 3.6 shows an example of a report that is generated for a successful test. The test report generated also shows which parameters were used for the different

methods. The report generated shows that test case #1 takes approximately 30 seconds this is much faster as supposed to manual testing. Test case #1 takes approximately 15 minutes to perform manually, this was discovered when performing the tests. However, all the fifteen tests were not automated. This does lead to an inconclusive result as all the tests were not automized. This was due to the timeframe of the project. As it can take quite some time to write robust tests and stable tests. This is ultimately why only two out of 15 tests were chosen for this thesis project. The test coverage can also expand with the help of a testing framework, tests are limited today due to how long time they to perform manually. Higher test coverage can ultimately help the development team save money and time.

A stack trace gets shown every time a test fails however, the stack traces can be somewhat difficult to understand. This is where screenshots upon failure come in as this can help developers and staff to identify the error a lot quicker.

The testing framework achieves consistency across the two tests. The tests themselves will always run in a similar manner most of the time. Selenide did not eliminate flakiness of the tests completely. Tests do still fail from time to time, this can be due to many different reasons. However, the two test cases takes about 30 seconds each to execute. So, a developer can rerun the tests quickly if a test fails due to flakiness. This is one of the bottlenecks that was discovered after writing the tests.

3.9.2 Serverless implementation

The framework was deployed using a serverless approach. AWS Fargate was chosen as it follows a serverless principle (see section 2.3.5.3). Salesforce as a website does have some limitations when trying to access it from a new IP address. This can make it quite difficult when trying to deploy a testing framework using a serverless approach. However, this was fixed by deploying the serverless framework with use of an static IP address. The service used for this is called Elastic Load Balancing. An IP address can be whitelisted in the Salesforce platform. This will allow for incoming traffic from that specific IP address without triggering the MFA.

3.9.3 Evaluation of test framework

The result of this implementation is a testing framework that can run two out 15 test cases. The implementation takes a serverless approach as the framework was deployed with Fargate through AWS. The tests themselves were written with Selenide which gives the developer the ability to emulate human interactions with a web browser. This testing framework can be used and integrated with the company's current testing strategy. However, all the current tests needs to be automized for it to truly replace the current manual workflow. This implementation is a beginning to fully automized regression tests for the company's workflow.

The test framework allows developers and quality assurance personnel to quickly identify bugs and unwanted features. This is made through the use of screenshots. The tests themselves are performed much quicker when they are automized as compared to manual testing.

The framework can also be used to store historical data of the test results. Every report can get saved within AWS. This can enable developers to quickly identify solutions to errors that has occurred previously.

4 Analysis and discussion

This chapter evaluates the outcome and discusses if a serverless approach is best suited for this type of implementation. Alternative methods of deployment will also be discussed. Section 4.3 and 4.4 evaluates the social and ethical impact of the implementation as well as the environmental impact.

4.1 Analysis of test framework

It is important to carefully plan and analyse which parts of the development process that is going to be affected by changes when considering test automation. This is one of the reasons as to why regression testing was explored. Regression testing gets done every two weeks however, it is very time demanding due to it being of the most crucial steps in the development process.

One of the biggest challenges that was discovered was the amount of time that needed to be spent on writing the tests. Salesforce is a very difficult platform to write tests for as it generates very complex HTML code. The use of tools for writing locators was used however, it generated very unreadable code due to the complex structure of the HTML. The outcome of this implementation shows that one of the biggest drawbacks with test automation, is the amount of time and preparation is needed to implement a proper test framework. One of the most demanding parts in test automation is the actual writing of the test scripts. The automated tests can slow down the development process if written they are written poorly. One of the highlighted ways to write good tests is to enforce constant checks such as checking if certain elements on the web page has disappeared.

The choice of only automating two out of the 15 available tests does however, leave to an inconclusive result. The result could have been different if all 15 were automized. However, this could have led to unstable tests as the tests themselves do take very long time to write. This was ultimately done to fit the time span of the project.

The biggest advantage of test automation is that if done correctly can save tremendous amount of time in the long run. An example of this is test case #1 as it takes 15 minutes to perform it manually and 30 seconds to run it with Selenide. The results indicate that the workforce needs to shift towards maintenance of the test code if test automation is desired. Although regression tests does not need to be changed every iteration of the code as the main point of regression testing is to test if existing functionalities works as intended (see section 2.2.1.1). This means that the test code does not need to be updated often. This is one of the strong advantages of adopting test automation for regression testing.

Another advantage is that automatic tests ensures the developers consistency across the tests. The human factor gets taken away with automatic testing. However, manual testing can still be preferred over automatic testing sometimes. Manual testing is very useful for testing functionality that needs to get tested only once. It takes more time to develop a automatized test rather than simply doing a single test

once. Test automation is useful when the developer wants to rerun the same tests multiple time. However, manual testing does not get completely obsolete as manual testing is still needed in order to develop automized tests.

Manual testing in compared to automatic testing does not need as much preparation. Manual testing can be heavily favourable for applications that are not going to be so long lived. It can also be preferable during early stages of development, as this will allow the developers to develop that they want first.

The choice of test case #1 and #2 does set a good example that test automation can be achieved within Salesforce. However, it also does showcase some poorly written XPaths. Some of the XPaths could have been shorter and this would in turn have led to more readable code. This is a result of inexperience as a developer does get better over time and starts to develop their own style of writing tests.

The main problem with automating a SaaS application is that developers do not have any control over how the HTML is generated. This makes for quite a challenge when writing scripts for Salesforce. The use of id as locators is rendered useless as the ids in the HTML get generated each time when accessing the sites. This can make for quite complex XPath locators which are hard to read. This requires the developer to write locators that are sustainable. Sustainable locators are locators that are not susceptible to page changes. Locators mainly get affected by Salesforce updates that change the underlying HTML code. A workaround for making the code more readable is to follow the POM pattern. This allows for more readable code for developers who are not used to reading web automation code. There are XPath generators that a develop can use. This was tested in during implementation however, the XPaths that were generated by the generator were very complex and unreadable. This due to how complex the Salesforce HTML code is. XPath generators can be used when the HTML is not that complex. The time needed to develop an automized test would have gone down drastically if the generators would have produced good XPath.

A potential improve that can be done to a similar project is not focusing on the deployment of testing framework. Instead, only focusing on first writing all the tests and potentially creating new ones. This can result in a more complete evaluation of the current testing strategy that is in place today. This would also allow for a more optimal testing strategi that is aimed towards test automation.

A serverless approach was still adopted, even though FaaS was not a suitable candidate for this type of implementation. The results indicates that a serverless approach using containers is very good as it reduces maintenance and labour of IT hardware. In comparison to deploying the test framework via a VM as this need more resources and configuration of a container runtime environment.

The serverless approach can be benefitable economically compared to a standard approach. This is because the users get charged hourly and for time of use. The report concludes that a serverless approach works perfectly for this sort of load balance. This due to the fact that the tests will only need to be around 3 times a month.

Selenide does work well with Salesforce if the developer writes tests that are stable. A drawback of Selenide or any other web automation tool for that matter is that writing tests can be very time consuming. The tests themselves are also very dependent on how the developer writes them as there no standardized way to write tests.

Some key takeaways from this specific implementation is that planning is needed for adaptation of test automation. Time and effort shift towards development of automized tests instead of manual testing. This can be seen as one of the drawbacks as testing framework introduces yet another code base to take care of, apart from the application. The testing framework needs to be taken care as the application grows larger. However, small changes are needed to be made to test code as regression tests do not change drastically.

Implementing a testing framework can be challenging and complex due to how the tests are written. This was showcased in the test code in the appendix, some of the locators do tend to become quite complex. The testing framework does work well in terms of being able to rerun tests on demand. This is one of the benefits of adopting test automation. This allows the developers to fix errors that may occur during the regression testing and then rerun the tests much faster than doing it manually.

4.2 Analysis of chosen tools and cloud provider

It is important to consider all tools of testing for the selected SaaS application. This is because different SaaS applications have different testing frameworks that are already on the market. Salesforce just happen to not have so many compelling ones and this makes it more desired to build an inhouse solution for test automation. The implementation used Selenide which is one of many options. An alternative strategy may be to utilize dedicated testing software such as Provar, that is specifically designed for Salesforce. Alternatively, the adoption of a JavaScript framework like Cypress could also be a viable solution. Selenide fits in very well with the current developers as it built with Java. This played a key role as Java is very similar to Salesforce own language which is Apex. Java also allows for integration with testing frameworks such as TestNG and Junit.

The use of Selenide helps us to write more concise tests. The tests tend to be more readable when going with Selenide compared to Selenium. Selenide also provides browser management, which makes it easier to maintain the tests. As it always runs on the browser version that is installed on the machine. This in compared to Selenium where you must supply a browser binary. Selenide also allows to set a global timeout which is very useful instead of having to specify each time that a method is called with Selenium. This makes for less repetitive code as well. Selenide has built in support for screenshot which can be very useful when having to debug what went wrong. This can be especially helpful as developers are not able to see tests being performed.

Flakiness of the tests still remain a problem despite using Selenide. Flakiness cannot be totally avoided when developing automized tests for web applications as there many factors that can affect the tests. This is something to take consideration when

adopting test automation. However, flakiness was reduced after deploying the testing framework to the cloud. Flakiness was most prevalent when developing the tests locally this can be due to hardware being limited. This does make some tests a bit more uncertain. The advantage with test automation is that it takes significantly shorter time to run them, so the solution is to just rerun the test that failed again. The difficult part about flakiness is that it can be difficult to find out if the test failed due to flakiness. However, a developer can quickly identify this with the screenshots that gets taken if a test failed.

The choice of cloud provider can be very critical and important to a company. This thesis did not cover every aspect of how to choose the best cloud provider. The choice of cloud provider can often lead to vendor lock in if the choice is not made right. A good example of this if a company wants to adopt an Active Directory later down the line. As Active Directory is a service that is only available on Azure [36]. This specific implementation only needed a service for serverless containers. This makes all the cloud providers very optimal for this sort of implementation. However, it is important to look at what could be needed down the line in terms of services.

4.3 Environmental impact

The environmental impact can be significantly reduced when opting to go for a serverless approach. The serverless approach allows the developers to opt for a more dynamic allocation of resources instead of having to occupy an x number of resources always. The resources can be dynamically scaled, meaning that the application or the system will not suffer from insufficient amount of resources if maintained properly. The developer can always choose to have a minimum number of cores and ram for the application. The serverless approach also helps us to disregard the need to take care of servers. This means that the maintenance is handled by the cloud providers such as GCP, Azure and AWS. Adopting a serverless approach is more environmentally sustainable compared to the traditional methods. The cloud providers have centralized management for their own server meaning that they are the ones who are responsible for disposing their servers. Servers still become e-waste by the end of their lifespan, so e-waste does not necessarily go away just because serverless was the chosen method of deployment. But the larger companies do have the resources and recycling tactics to dispose of the e-waste correctly. Compared to individuals and smaller companies who does not have the adequate resources. It was concluded that deploying to the cloud is more environmentally friendly through the findings in the report.

4.4 Social and ethical impact

The implementation of test automation can vastly increase productivity and put end to repetitive tasks regarding testing. There are various amount of tests that are performed manually with these being the regression tests. This leaves much to be desired in terms of automation as these tests do not get changed all too often. The implementation of automated tests can help ensure the developer that the tests are performed as intended. The human factor does not come in to play when opting to adopt automated tests. A big factor to consider here is that the tests need to be

written properly to get the desired outcome. Automated tests can help with detecting issues with releases much faster than manual testing as it takes less time to test. This in term can lead to less mistakes happening when deploying and reducing the level of stress when trying to complete a deadline. The scope of quality assurance generally goes up if the test automation is implemented correctly.

Although one question still surfaces when doing any sort of automation implementation and that is who is to blame if anything goes wrong? With the help of the report and through studies it can concluded that this depends on the situation. It is important to set up routines for when a test fails for example. The key point is to identify why the test failed. The test can fail due to flakiness, or changes made in the system that breaks the test. Some failures can be related to the test developer and some failures can relate the system developer. Establishing comprehensive guidelines in advance is imperative to effectively address any issues that may arise with the implementation.

4.5 Economic impact

Test automation as supposed to manual testing does have some changes when it comes to the economical aspect. Perhaps the biggest change is that test automation requires more upfront costs compared to manual testing. This is due to the need to develop a new testing strategy that is aimed towards test automation and developing the tests themselves. Test automation does require some time before being able to properly utilize it. The upfront costs are mostly the development and implementation of the testing framework. Infrastructure costs can vary depending on how the company implements the testing framework. An on-premises implementation requires more upfront cost as there will be a need to buy the hardware. Manual testing only requires test scripts and personnel to perform the tests scripts. Manual testing shifts the cost towards personnel costs as it requires personnel to perform the manual tests. Whereas test automation requires the company to hire personnel who put the testing framework in place and maintain it. Maintenance is needed for the test automation framework to work overtime. However, maintenance of the testing framework is rather low once the test code is in place. This is due to the nature of regression testing as regression tests alter slightly from release to release. Test automation can in the long run help companies financially if they are willing to adopt test automation. The testing scope of the software can easily become much larger thus ensuring that no bugs are found in production. This results in developers being able to focus on new features instead of fixing broken features. This also allows quality assurance personnel to work with other tasks rather than testing.

4.6 Alternative approaches

An alternative approach to automatic regression testing can be to automate unit testing. This sort of implementation can have varied outcome depending on who is implementing it. This is because unit tests are often smaller and do not require as much line of code. Although unit testing with Selenium or Selenide can be achieved. However, this is not recommended as unit tests tends to be performed 2 times at most. This is not the same for regression testing as the same tests gets used over and over with small iterations [31].

Another approach can be to use a JavaScript framework aimed for web testing instead of a tool that is based on Selenium. Although the major JavaScript frameworks tends to suffer browser compatibility issues according the authors who did a survey 2020 [29]. Although this would not be a problem for Salesforce test automation as it is the vendors who are doing the browser compatibility tests and not the users. The JavaScript frameworks works very similar to Selenium and Selenide. Both frameworks use the same sort of locators. The use of JavaScript frameworks does not eliminate the hassle of writing good locators in Salesforce.

5 Conclusion

As the SaaS industry continues to grow so does the implementations of SaaS products. This makes testing a vital part of any SaaS product implementation. This thesis aim was to research the best practices for doing test automation within a specific SaaS application. The thesis furthermore covers if a serverless approach is viable for a testing framework that is designed for SaaS applications.

5.1 Validation of goals

This section covers if the goals presented in section 1.2 were met. The goals were the following:

- Examine on which cloud providers offers good serverless solutions
- Establish if automatic regression testing can replace manual regression testing.
- Come to a conclusion on if a serverless approach is suitable for a SaaS application testing framework.
- Determine which test automation tool fits Salesforce best according to Polestar's needs.
- Develop and write tests with the chosen test automation in accordance with provided scripts.

All the goals were met except "Establish if automatic regression testing can replace manual regression testing". This is because only two of 15 tests were chosen. This is because of the time frame for the project and that there was only one author who did this thesis work. However, the tests for Salesforce are very much alike this is because they mainly involve triggering buttons on the website and waiting for a result. Although all the tests should be automated in order to get a more accurate result. However, the code that was developed for this implementation can be used as inspiration for other test automation projects.

5.2 Future work

Due to there not being any testing strategy for automation testing, the project could not gather how much time was going to be saved with implementing a test automation framework. This can be quite interesting as getting data on how the testing time affects the current development procedures today. The project could have greatly benefited from a testing strategy that was aimed towards automation testing. A testing strategy should be in place for future projects to succeed. Another great point is to discover the SaaS applications API more and try to integrate it with the testing framework. This would allow for database queries which could help reduce the time needed for tests.

FaaS is still not completely ruled out for this sort of implementation although it would look very different. Openfaas can be seen as a good way for this sort of implementation. As Openfaas allows for complete control on how long the functions

can run. This in turn can help to solve the biggest hurdle with FaaS for this type of implementation. Openfaas would although require significantly more configuration and it would look completely different to the project that was covered in this thesis. This solution would also allow the functions to bind to a specific IP address.

Bibliography

[1] S. Aleem, F. Ahmed, R. Batool and A. Khattak, Empirical Investigation of Key Factors for SaaS Architecture, in IEEE Transactions on Cloud Computing, vol. 9, no. 3, pp. 1037-1049, 1 July-Sept. 2021, doi: 10.1109/TCC.2019.2906299.

[2] T. Yacob, Securing Sensitive Data in the Cloud: A New Era of Security Through Zero Trust Principles, Diva-portal.org, 2023. Available from: https://kth.diva-portal.org/smash/get/diva2:1739157/FULLTEXT01.pdf

[3] S. Bibi, D. Katsaros and P. Bozanis, Business Application Acquisition: On-Premise or SaaS-Based Solutions?, in *IEEE Software*, vol. 29, no. 3, pp. 86-93, May-June 2012, doi: 10.1109/MS.2011.119.

[4] Columbus L. “Salesforce now has over 19% of the CRM market [Internet]”. Forbes. Forbes Magazine; 2019 Available from: https://www.forbes.com/sites/louiscolumbus/2019/06/22/salesforce-now-has-over-19-of-the-crm-market/?sh=5ab874fe333a
[cited 2023 Mar 24].

[5] Anshari, Muhammad & Almunawar, Mohammad Nabil & Lim, Syamimi & Al-Mudimigh, Abdullah. (2018). Customer Relationship Management and Big Data Enabled: Personalization & Customization of Services. Applied Computing and Informatics. 15. 10.1016/j.aci.2018.05.004.

[6] R. Kneuper, Sixty Years of Software Development Life Cycle Models, in *IEEE Annals of the History of Computing*, vol. 39, no. 3, pp. 41-54, 2017, doi: 10.1109/MAHC.2017.3481346.

[7] R. Silva, P. Perera, I. Perera and K. Samarasinghe, Effective use of test types for software development, *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, Colombo, Sri Lanka, 2017, pp. 1-6, doi: 10.1109/ICTER.2017.8257795.

[8] C. Klammer and R. Ramler, A Journey from Manual Testing to Automated Test Generation in an Industry Project, *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Prague, Czech Republic, 2017, pp. 591-592, doi: 10.1109/QRS-C.2017.108.

[9] W. E. Wong, J. R. Horgan, S. London and H. Agrawal, A study of effective regression testing in practice, *Proceedings The Eighth International Symposium on Software Reliability Engineering*, Albuquerque, NM, USA, 1997, pp. 264-274, doi: 10.1109/ISSRE.1997.630875.

[10] M. Gokarna and R. Singh, DevOps: A Historical Review and Future Works, *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, Greater Noida, India, 2021, pp. 366-371, doi: 10.1109/ICCCIS51004.2021.9397235.

[11] Jenkins Documentation [Internet]. Jenkins. [cited 2023 Mar 30] Available from: https://www.jenkins.io/doc/.

[12] What is Cloud Computing? Cloud Computing Dictionary [Internet]. Microsoft Azure; [cited 2023 Mar 30].Available from: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing#cloud-computing-models

[13] What is IaaS? Cloud Computing Dictionary [Internet]. Microsoft Azure; [cited 2023 Mar 30]. Available from: https://azure.microsoft.com/en-ca/resources/cloud-computing-dictionary/what-is-iaas/

[14] What is PaaS? [Internet]. Microsoft Azure; [cited 2023 April 2]. Available from: https://azure.microsoft.com/en-ca/resources/cloud-computing-dictionary/what-is-paas/

[15] What is SaaS? [Internet]. Microsoft Azure; [cited 2023 May 24]. Available from: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-saas/

[16] What is serverless? [Internet]. Red Hat; [cited 2023 April 2]. Available from: https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless

[17] AWS Lambda. [Internet]. Amazon Web Services; [cited 2023 April 4]. Available from: https://aws.amazon.com/lambda/

[18] H. Eslahi-Kelorazi, F. Movahedi, and H. Eslahi-Kelorazi, Identification of some factors affecting the adoption of cloud computing in the construction industry, 2016 3rd International Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, 2016, pp. 127-132. doi: 10.1109/KBEI.2016.7899408.

[19] What is a Container? [Internet]. Docker; [cited 2023 Apr 10].Available from: https://www.docker.com/resources/what-container/

[20] AWS Fargate [Internet]. AWS; [cited 2023 May 24]. Available from: https://aws.amazon.com/fargate/

[21] Container Instance [Internet]. Azure; [cited 2023 May 24]. Available from: https://azure.microsoft.com/en-us/products/container-instances/

[22] Developer Survey 2022 [Internet]. Stackoverflow; [cited 2023 Apr 27]. Available from: https://survey.stackoverflow.co/2022/#technology

[23] TestNG Documentation [Internet]. TestNG; [cited 2023 Apr 27]. Available from: https://testng.org/doc/

[24] JUnit – Overview [Internet]. Tutorialspoint; [cited 2023 May 24]. Available from: https://www.tutorialspoint.com/junit/junit_overview.htm/

[25] Assertions in Java [Internet]. Geeksforgeeks; [cited 2023 May 24]. Available from: https://www.geeksforgeeks.org/assertions-in-java/

[26] Apex Developer Guide - Introduction to Apex [Internet]. Salesforce; [cited 2023 Apr 27]. Available from: https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro_what_is_apex.html

[27] Copado - The #1 Native DevOps Platform for Salesforce [Internet]. Copado; [cited 2023 Apr 27]. Available from: https://copado.com/

[28] Provar Testing [Internet]. Provar Testing; [cited 2023 Apr 27]. Available from: https://www.provartesting.com/

[29] García B, Gallego M, Gortázar F, Munoz-Organero M. A Survey of the Selenium Ecosystem. *Electronics*. 2020; 9(7):1067. Available from: https://doi.org/10.3390/electronics9071067

[30] Ramya, P., Sindhura, V., & Sagar, P. V. (2017). Testing using selenium web driver. In 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT) (pp. 1-7). Coimbatore, India. doi: 10.1109/ICECCT.2017.8117878.

[31] Gojare S, Joshi R, Gaigaware D. Analysis and Design of Selenium WebDriver Automation Testing Framework. Procedia Computer Science. 2015;50:341-346. doi: 10.1016/j.procs.2015.04.038.

[32] M. Leotta, D. Clerissi, F. Ricca and C. Spadaro, Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study, *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, Luxembourg, Luxembourg, 2013, pp. 108-113, doi: 10.1109/ICSTW.2013.19.

[33] C. Klammer and R. Ramler, A Journey from Manual Testing to Automated Test Generation in an Industry Project, *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Prague, Czech Republic, 2017, pp. 591-592, doi: 10.1109/QRS-C.2017.108.

[34] Davit Danelia, Selenium or Selenide [Internet]. Medium; [Cited 15/08/2023] Available from: https://medium.com/tbc-engineering/selenium-or-selenide-focf8221cb61/

[35] Choosing a cloud a provider [Internet]. NCSC; [Cited 17/08/2023] Available from: https://www.ncsc.gov.uk/collection/cloud/choosing-a-cloud-provider

[36] CSA Star Registry [Internet]. Cloud Security Alliance; [Cited 17/08/2023] Available from: https://cloudsecurityalliance.org/star/registry/

[37] Advantages and Disadvantages of Cloud Computing [Internet]. Google Cloud Platform; [Cited 17/08/2023] Available from: https://cloud.google.com/learn/advantages-of-cloud-computing

[38] Cloud Run [Internet]. Google Cloud Platform; [Cited 17/08/2023] Available from: https://cloud.google.com/run

[39] Manage Webhooks [Internet]. Atlassian; [Cited 18/08/2023] Available from: https://support.atlassian.com/bitbucket-cloud/docs/manage-webhooks

Appendix

Appendix A – login page code

```
package Pages;

import com.codeborne.selenide.Configuration;
import com.codeborne.selenide.Selenide;

import static com.codeborne.selenide.Condition.disappear;
import static com.codeborne.selenide.Selenide.$x;


public class LoginPage  {
    private final static String url = "";
    public LoginPage(){
        Configuration.timeout = 20000;
    }

    public static void open() {
        Selenide.open(url);
    }

    public static void setUsername(String username) {
        $x("/html//input[@id='username']").click();
        $x("/html//input[@id='username']").setValue(username);
    }
    public static void setPassword(String password) {
        $x("/html//input[@id='password']").click();
        $x("/html//input[@id='password']").setValue(password);
    }
    public static void clickLogin() {
        $x("/html//input[@id='Login']").click();
        $x("/html//input[@id='Login']").should(disappear);
    }
    public static boolean isLoginSuccessful(String SandboxName) {
        return $x("//header[@id='oneHeader']/div[1]/div/span[.=' Sandbox:
"+SandboxName+"']").exists();
    }
}
```

Appendix B – test case #1

```
package Tests;

import Pages.CreateAccount;
import Pages.LoginPage;
import com.codeborne.selenide.Configuration;
import com.codeborne.selenide.WebDriverRunner;
import com.codeborne.selenide.testng.TextReport;
import org.testng.Assert;
import org.testng.Reporter;
import org.testng.annotations.*;

import static com.codeborne.selenide.Condition.disappear;
```

```
import static com.codeborne.selenide.Condition.exist;
import static com.codeborne.selenide.Configuration.baseUrl;
import static com.codeborne.selenide.Configuration.timeout;
import static com.codeborne.selenide.Selenide.*;


@Listeners({ TextReport.class})
@Test
public class    psidInvite {

    String accountLink;

    @BeforeClass
    public static void setup() throws InterruptedException {

        Configuration.headless = false;
        Configuration.browser = "chrome";
        Configuration.reportsFolder = "reports/Build#" +
System.getenv("BUILD_NUMBER");
        timeout = 20000;
        baseUrl = "";
        open("/");
    }

    @AfterClass
    public static void logout() {
        closeWebDriver();
    }

    @Parameters({"userName", "password"})
    @Test
    public static void Login(String userName, String password) throws
InterruptedException {
        Reporter.log("Logging In");
        LoginPage.open();
        LoginPage.setUsername(userName);
        LoginPage.setPassword(password);
        LoginPage.clickLogin();
        Assert.assertTrue(LoginPage.isLoginSuccessful("Redve"));
        Reporter.log("Logged In");
    }

    @Parameters({"lastName", "country"})
    @Test(dependsOnMethods = "Login")
    public void createAccount(String lastName,String country) throws
InterruptedException {
        Reporter.log("Creating Account");
        CreateAccount.open();
        CreateAccount.clickNew();
        CreateAccount.setLastName(lastName);
        CreateAccount.setCountry(country);
        boolean check = CreateAccount.clickSave();
        Reporter.log("Created Account");
        accountLink = WebDriverRunner.getWebDriver().getCurrentUrl();
        Assert.assertFalse(check);
    }

    @Test(dependsOnMethods = "createAccount")
    public void sendInviteAdmin() throws InterruptedException{
```

```
        Reporter.log("Sending Invite");
        $x("//body[@class='desktop']/div[4]//lightning-formatted-
name[.='Test']").should(exist);
        $x("//lightning-button-menu[@data-target-
reveals='sfdc:QuickAction.Account.Submit_PSID_Invite,sfdc:StandardButton.
Account.Share,sfdc:StandardButton.Account.Delete,sfdc:StandardButton.Acco
unt.PrintableView,sfdc:QuickAction.Account.Clean_Old_Document_Versions,sf
dc:QuickAction.Account.Suggest_Duplicate_Acount']/button").should(exist);
        $x("//lightning-button-menu[@data-target-
reveals='sfdc:QuickAction.Account.Submit_PSID_Invite,sfdc:StandardButton.
Account.Share,sfdc:StandardButton.Account.Delete,sfdc:StandardButton.Acco
unt.PrintableView,sfdc:QuickAction.Account.Clean_Old_Document_Versions,sf
dc:QuickAction.Account.Suggest_Duplicate_Acount']/button").click();
        $x("//runtime_platform_actions-action-
renderer[@apiname='Account.Submit_PSID_Invite']//a").click();
        $x("//span[.='YOUR FLOW FINISHED']").should(exist);
        Reporter.log("Sent Invite");
    }

    @Test(dependsOnMethods = "sendInviteAdmin")
    public void loginAsTest() throws InterruptedException {
        Reporter.log("Logging In as Test");
        open("");
        $x("//span[text()='Logged in as RegTestAutoPSIDInvite
(regtestpsid@regtest.com) |  Sandbox: Redve | ']").should(exist);
        Reporter.log("Logged In as Test");
    }

    @Test(dependsOnMethods = "loginAsTest")
    public void testInviteAsTest() throws InterruptedException {
        open(accountLink);
        Reporter.log("Testing Invite as Test User");
        $x("//body/div[4]//slot[@name='primaryField']/lightning-
formatted-name[.='Test']").should(exist);
        $x("//div[@class='viewport'//button/lightning-primitive-
icon").click();
        $x("//runtime_platform_actions-action-
renderer[@apiname='Account.Submit_PSID_Invite']//a").click();
        $x("//span[.='YOUR FLOW FINISHED']").should(exist);
        Assert.assertEquals($x("//span[.='YOUR FLOW
FINISHED']").getText(), "YOUR FLOW FINISHED");
        Reporter.log("Tested Invite as Test User");
    }
}
```

Appendix C – test case #2

```
package Tests;

import com.codeborne.selenide.Configuration;
import com.codeborne.selenide.WebDriverRunner;
import com.codeborne.selenide.testng.TextReport;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebElement;
import org.testng.annotations.*;
```

```
import java.security.Key;
import java.util.List;
import java.util.UUID;

import static com.codeborne.selenide.Condition.disappear;
import static com.codeborne.selenide.Condition.exist;
import static com.codeborne.selenide.Configuration.*;
import static com.codeborne.selenide.Selenide.*;

@Listeners({ TextReport.class})
@Test
public class ReAssign {
    String random1;
    String random2;
    String Order1;
    String Order2;
    String Account2;
    @BeforeClass
    public void setup() throws InterruptedException {
        Configuration.headless = true;
        timeout = 20000;
        baseUrl = "";
        Configuration.reportsFolder = "reports/" +
System.getenv("BUILD_NUMBER");
        browser = "chrome";
        random1 = UUID.randomUUID()
                .toString()
                .substring(0,10);
        random2 = UUID.randomUUID()
                .toString()
                .substring(0,10);
        open("/");
    }

    @AfterClass
    public static void logout() {
        closeWebDriver();
    }

    @Parameters({"userName", "password"})
    @Test
    public static void Login(String userName, String password) throws
InterruptedException {
        $x("/html//input[@id='username']").setValue(userName);
        $x("/html//input[@id='password']").setValue(password);
        $x("/html//input[@id='Login']").click();
        $x("/html//input[@id='Login']").should(disappear);
        $x("//header[@id='oneHeader']/div[1]/div/span[.=' Sandbox:
Redve']").should(exist);
    }

    @Test(dependsOnMethods = "Login")
    public void  createOrder1AndAccount1() throws  InterruptedException{
        open("");
        executeJavaScript("arguments[0].click();", $x("//li[@data-target-
selection-name='sfdc:StandardButton.Order.New']/child::a/child::div"));
        executeJavaScript("arguments[0].click();",
$x("//span//span[text()='Order Type']//parent::span//following-
sibling::div//a"));
```

```
        $x("//a[@title='remarketed']").should(exist);
        executeJavaScript("arguments[0].click();",
$x("//a[@title='remarketed']"));
        $x("//span[text()='Account Name']//parent::label//following-
sibling::div//input").click();
        $x("//span[@title='New Account']").click();
        $x("//span[text()='Next']").click();
        $x("//input[@placeholder='Last Name']").setValue(random1);
        $x("//span[text()='Person Country']//parent::span//following-
sibling::div//div[@class='uiPopupTrigger']").click();
        $x("//a[@title='Sweden']").click();

$x("//div[2]/div[@role='dialog']//button[@title='Save']").click();

$x("//div[2]/div[@role='dialog']//button[@title='Save']").should(disappea
r);
        $x("//span[text()='Driver']//parent::label//following-
sibling::div//input").sendKeys(random1);
        $x("//div[@title='"+random1+"']").click();
        $x("//span[text()='Accepted Handover
Date']//parent::legend//following-sibling::div//div/a[@class='datePicker-
openIcon display']").sendKeys("2025-01-01");
        $x("//span//span[text()='Status']//parent::span//following-
sibling::div//a").click();
        $x("//a[@title='Delivery Planning']").click();
        $x("//span[text()='Master Order Id']//parent::label//following-
sibling::input").setValue(UUID.randomUUID().toString().substring(0,10));
        $x("//span[text()='Order Start Date']//parent::label//following-
sibling::div/input").setValue("2025-01-01");
        $x("//div[@class='modal-body scrollable slds-modal__content slds-
p-around--medium']//button[@title='Save']/span[.='Save']").click();

$x("//div[2]/div[2]/div/div[2]/span/div/a[text()='"+random1+"']").should(
exist);
        Order1 = WebDriverRunner.getWebDriver().getCurrentUrl();
        System.out.println(Order1);
        open(Order1);
    }

    @Test(dependsOnMethods = "createOrder1AndAccount1")
    public void createOrder2() throws  InterruptedException{
        open("");
        executeJavaScript("arguments[0].click();", $x("//li[@data-target-
selection-name='sfdc:StandardButton.Order.New']/child::a/child::div"));
        executeJavaScript("arguments[0].click();",
$x("//span//span[text()='Order Type']//parent::span//following-
sibling::div//a"));
        $x("//a[@title='remarketed']").should(exist);
        executeJavaScript("arguments[0].click();",
$x("//a[@title='remarketed']"));
        $x("//span[text()='Account Name']//parent::label//following-
sibling::div//input").click();
        $x("//span[@title='New Account']").click();
        $x("//span[text()='Next']").click();
        $x("//input[@placeholder='Last Name']").setValue(random2);
        $x("//span[text()='Person Country']//parent::span//following-
sibling::div//div[@class='uiPopupTrigger']").click();
        $x("//a[@title='Sweden']").click();
```

```
$x("//div[2]/div[@role='dialog']//button[@title='Save']").click();

$x("//div[2]/div[@role='dialog']//button[@title='Save']").should(disappea
r);
        $x("//span[text()='Accepted Handover
Date']//parent::legend//following-sibling::div//div/a[@class='datePicker-
openIcon display']").sendKeys("2025-01-01");
        $x("//span//span[text()='Status']//parent::span//following-
sibling::div//a").click();
        $x("//a[@title='Delivery Planning']").click();
        $x("//span[text()='Driver']//parent::label//following-
sibling::div//input").sendKeys(random2);
        $x("//div[@title='"+random2+"']").click();
        $x("//span[text()='Master Order Id']//parent::label//following-
sibling::input").setValue(UUID.randomUUID().toString().substring(0,10));
        $x("//span[text()='Order Start Date']//parent::label//following-
sibling::div/input").setValue("2025-01-01");
        $x("//div[@class='modal-body scrollable slds-modal__content slds-
p-around--medium']//button[@title='Save']/span[.='Save']").click();

$x("//div[2]/div[2]/div/div[2]/span/div/a[text()='"+random2+"']").should(
exist);
        Order2 = WebDriverRunner.getWebDriver().getCurrentUrl();
        System.out.println(Order2);
        open(Order2);
    }

    @Test(dependsOnMethods = "createOrder2")
    public void moveOrder() throws InterruptedException{
        open(Order1);

$x("//div[@role='group']/ul/li[4]//div[@class='uiPopupTrigger']//a[@role=
'button']").click();
        $x("//a[@data-target-selection-
name='sfdc:StandardButton.Order.Edit']").click();
        var deleteButtons = $$x("//span[@class='deleteIcon']");
        deleteButtons.get(0).click();
        $x("//span[text()='Account Name']//parent::label//following-
sibling::div//input[@title='Search Accounts']").sendKeys(random2);
        $x("//div[@title='"+random2+"']").click();
        $x("//div[@class='modal-body scrollable slds-modal__content slds-
p-around--medium']//button[@title='Save']/span[.='Save']").click();

$x("//div[2]/div[2]/div/div[2]/span/div/a[text()='"+random2+"']").should(
exist);

$x("//div[2]/div[2]/div/div[2]/span/div/a[text()='"+random2+"']").click()
;
        $x("//lightning-formatted-
name[text()='"+random2+"'][@slot='primaryField']").should(exist);
        Account2 = WebDriverRunner.getWebDriver().getCurrentUrl();
        System.out.println(Account2);

    }

    @Test(dependsOnMethods = "moveOrder")
    public void checkIfMoved() throws InterruptedException{
        open(Account2);
        $x("//lightning-formatted-
```

```
name[text()='"+random2+"'][@slot='primaryField']").should(exist);
        $x("//span[text()='Orders']//following-
sibling::span[text()='(2)']").should(exist);
        $x("//lightning-button-menu[@class='menu-button-item slds-
dropdown-trigger slds-dropdown-trigger_click']").click();
        $x("//a[@name='Account.Suggest_Duplicate_Acount']").click();
        var inputs = $$x("//lightning-base-combobox[@class='slds-
combobox_container']//input[@role='combobox']");
        inputs.get(0).sendKeys(random1);
        inputs.get(0).sendKeys(Keys.BACK_SPACE);
        $x("//lightning-base-combobox-formatted-
text[@title='"+random1+"']").should(exist);
        $x("//lightning-base-combobox-formatted-
text[@title='"+random1+"']").click();
        $x("//button[@class='slds-button slds-button_brand']").click();
        $x("//button[@class='slds-button slds-button_brand']").click();
        open(Account2);
        $x("//div[@title='View Duplicates']").click();
        $x("//span[text()='Select Item 2']//parent::label[@class='slds-
checkbox__label']").click();
        $x("//button[@type='button'][text()='Next']").click();
        $x("//span[text()='"+random2+"']").should(exist);
        $x("//button[@type='button'][text()='Next']").click();
        $x("//button[@type='button'][text()='Merge']").click();
        Assert.assertTrue($x("//span[text()='Orders']//following-
sibling::span[text()='(2)']").exists());
    }
}
```