



Implementing Automated Software Testing

Neha Kaul

AP | ARCLER
PRESS

Implementing Automated Software Testing

IMPLEMENTING AUTOMATED SOFTWARE TESTING

Neha Kaul



www.arclerpress.com

Implementing Automated Software Testing

Neha Kaul

Arcler Press

224 Shoreacres Road

Burlington, ON L7L 2H2

Canada

www.arclerpress.com

Email: orders@arclereducation.com

e-book Edition 2023

ISBN: 978-1-77469-608-8 (e-book)

This book contains information obtained from highly regarded resources. Reprinted material sources are indicated and copyright remains with the original owners. Copyright for images and other graphics remains with the original owners as indicated. A Wide variety of references are listed. Reasonable efforts have been made to publish reliable data. Authors or Editors or Publishers are not responsible for the accuracy of the information in the published chapters or consequences of their use. The publisher assumes no responsibility for any damage or grievance to the persons or property arising out of the use of any materials, instructions, methods or thoughts in the book. The authors or editors and the publisher have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission has not been obtained. If any copyright holder has not been acknowledged, please write to us so we may rectify.

Notice: Registered trademark of products or corporate names are used only for explanation and identification without intent of infringement.

© 2023 Arcler Press

ISBN: 978-1-77469-403-9 (Hardcover)

Arcler Press publishes wide variety of books and eBooks. For more information about Arcler Press and its products, visit our website at www.arclerpress.com

ABOUT THE AUTHOR



Neha Kaul is an experienced software consultant and technical author currently residing in Sydney, Australia. She is the author of five technical books: Object Oriented Programming with Java, Logging Frameworks in Java, Applications of Data Mining in Engineering, Management and Medicine, Software Security: Building Secure Software Applications and Analytic Methods of Systems and Software Testing. She received her double Master's Degree in Computer and Communication Networks and Information Technology from Telecom SudParis and University Paris-Saclay in 2016. She is a recipient of the prestigious Telecom Scholarship for Excellence provided by Fondation Telecom, France. She received the Bachelor of Engineering degree in Computer Engineering from the University of Pune, India in 2011. From 2011 to 2014, she was employed as a Software Engineer with Geometric Ltd, Pune, India. Her major avenues of research include Advanced Java/J2EE frameworks, Automation Testing, Software Quality, Software Security, Logging Frameworks, Project Management and Leadership.

TABLE OF CONTENTS

<i>List of Abbreviations</i>	<i>ix</i>
<i>Dedication</i>	<i>xi</i>
<i>Preface</i>	<i>xiii</i>
Chapter 1 Software Testing: Definition and Importance	1
1.1. What is Software Testing?	2
1.2. Importance of Software Testing	2
Chapter 2 Automated Software Testing	9
2.1. Introduction.....	10
2.2. Benefits of Automation Testing	10
2.3. Types of Automation Tests in Software	16
2.4. Different Automation Testing Software	23
Chapter 3 Katalon Studio	25
3.1. Introduction.....	26
3.2. Installation.....	26
3.3. Practical Implementations/Examples	38
Chapter 4 Watir	111
4.1. Important Watir Commands and Terminology	112
4.2. Watir Installation	114
4.3. Examples	127
Chapter 5 Ranorex Studio	191
5.1. Setup and Installation	192
5.2. Ranorex Studio Basics.....	199
5.3. Examples	201
Bibliography	255
Index	259

LIST OF ABBREVIATIONS

API	application programming interface
REPL	read-eval-print loop
TDD	test-driven development
UI	user interface

DEDICATION

To my wonderful husband, Sabin, you have been a source of constant support and encouragement.

To my parents, Virender and Anita, who always pushed for tenacity.

To my sister, Nidhi, for your wisdom and unconditional love.

PREFACE

Software systems are an essential component of our day-to-day lives. Software has morphed into one of our most basic necessities. We depend on software to accomplish routine tasks and activities in our lives. The impact that software has made in this world is enormous. Software applications are engaged on a large scale in both essential and non-essential sectors across the world. The immensity of the involvement and impact of software in the world is enormous. Software applications and systems help in completing complex tasks in an easier and cost-effective way which has improved the quality of life of millions of people.

The relevance of software will only increase in the future. Software will continue to increase in complexity as it is going to be used to solve the biggest problems faced by the world. It is a proven fact that as the complexity of software rises, the challenges associated with software testing and software maintenance will subsequently rise too. Complex software solutions created to solve complicated problems are destined to have an element of complexity in the process of testing as well. It is a known fact that there is a direct relation between the quality of a software system and testing. Effective testing is thought of as a measure of efficiency and quality of software. Testing is a vital non-skippable step in the software development lifecycle.

The field of software testing has grown considerably since its origin in the early 1900s. Testing of software helps instill confidence in the quality of the software to its users and stakeholders. It is advisable that software be evaluated and tested thoroughly as it is vulnerable to a range of potential attacks that may be of a malicious nature. Comprehensive testing and evaluation of any software solution is paramount to identifying vulnerabilities in the system and shield it from potential attacks. The lack of quality testing is one of prime reasons for the failure of software systems resulting in significant losses for the stakeholders, clients, and users. One can assume with certainty that software testing is a determining factor in the success of a software solution.

Over the last few decades, the field of software testing has grown exponentially. A branch of software testing namely automation testing has helped reshape the way in which testing is done. Automation testing is the exercise of executing tests automati-

cally in a repetitive manner. This branch of study concentrates on execution of repetitive tests, management of the test data and the utilization of the results to ameliorate and improve the quality of software. In this book, we will investigate the automation testing field within software testing. The advantages of this type of testing will be discussed in brief. Further, different software solutions that perform automation testing will be presented and examined in this book. The implementation of different automated software systems will be presented in detail. Detailed practical implementations of automated software applications covering different types of testing scenarios have been provided in this book. This book assumes that the audience is familiar with the basic concepts of software testing.

SOFTWARE TESTING: DEFINITION AND IMPORTANCE

CONTENTS

1.1. What is Software Testing?	2
1.2. Importance of Software Testing	2

In this chapter, we shall investigate the definition of software testing in brief and the importance of software testing in the SDLC lifecycle.

1.1. WHAT IS SOFTWARE TESTING?

Software Testing is an activity of evaluating a software system and its components to check if the software system meets its specifications. It can be described as the process of examining software with the intention of verifying whether the requirements have been met by the software under test (Beizer, 2003).

In elementary terms, it can be described as an activity or procedure of running/executing a software application or system with the purpose of pinpointing any errors, bugs, gaps, or missing/incomplete requirements that were not present in the initial requirements.

The definition of testing according to the ANSI/IEEE 1059 standard is (Singh and Singh, 2012):

“A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.”

The two main goals of the process of software testing are:

- Verification and validation of the requirements; and
- Bug detection to improve the quality of the software.

Based on the way in which tests are carried out and who/what carries out the test, software testing is divided into two broad categories: manual testing and automation testing.

1.2. IMPORTANCE OF SOFTWARE TESTING

Software Testing is considered to be an important part of the software development life cycle. The need for software testing originated from the desire to provide some proof that a software system works as required. The need to prove the validity of software led to the creation of Software Testing as a discipline. The initial implementation of software testing demonstrated the benefits of testing to the software community which made it very popular in the software community. In addition to detecting bugs/errors with the intention of improving the quality of the software, testing also helped with providing metrics regarding quality of the product.

Some of the advantages of software testing are:

- **Early Detection of Errors/Bugs:** Software testing helps find errors early in the software development process so that fewer errors or bugs are found once the product has been developed. When a software product is in the development phase, there is a possibility that one or more developers will make a mistake during the coding process. There could be several reasons for them making an error, such as lack of programming knowledge, lack of expertise in the programming language and syntax, improper understanding of the requirements, incorrect algorithmic implementation, insufficient experience in the subject matter, or just a simple human error. The software testing process helps locate simple or complex errors/omissions in the code. In the case of enterprises that adopt Agile and DevOps processes, the testing process is iterative, and hence bugs are identified and fixed promptly. This iterative process helps in effectively managing the software development and helps understand potential areas of complexity in the requirements, thereby improving the quality of the product.
- **Improved Product Maintenance and Product Performance:** After a software product has been released on to the market it goes into its maintenance phase. Defects that come up after the release of the product are handled by the maintenance teams.

In most cases, maintenance teams designed to handle service requests post-product deployment are small in number. Therefore, should there be a large number of defects reported after the deployment of the software product, the maintenance team would not be well-equipped to handle the workload and become saturated with maintenance-related work. If they are unable to address all the requests coming their way in a timely manner, customer satisfaction could be threatened. Higher number of defects found after the product is in the hands of the customer can further damage customer and stakeholder relations. Furthermore, a significant number of defects could necessitate use of additional resources to help fix the defects leading to monetary losses. By use of software testing methodologies, early detection of issues is possible which helps in the maintenance phase of the product. If the product has been well tested and is of good quality, the number of defects is therefore less, and the maintenance team's job is reduced. A well

tested software product is highly likely to have an exponential success rate and generate only trivial issues after it has been released in the market.

Along the same lines one can say that a product that has been thoroughly tested will generate good results. Needless to say, but to incur a gain on the financial markets, a product must be of good quality and perform exceptionally well which makes testing absolutely indispensable. A software product will do well on the market if it functions correctly and produces fewer defects (Mills, Dyer, and Linger, 1987). This can be achieved only through thorough testing. During the continuous examination by means of software testing, the product will undergo repeated testing throughout its lifecycle, weeding out unwanted bugs as and when they are detected. The product is validated and re-validated over time increasing its accuracy. More the software is tested; more it is understood and examined critically which leads to better performance. An oversimplification of this statement is – a software product that is well tested will perform well. This can only be attained by the help of software testing.

- **Enhanced Product Quality:** In order to realize the view of the software proposed by the client or stakeholders, it is highly desired that the system do what it is supposed to do in an efficient manner. The software development phase must correctly implement the software requirements correctly to attain the desired effect for the customer. In the end, a software product is mostly provided as a service to the customers and hence it is essential that the product delivers the value promised to the customer. This is where software quality comes into place.

Software testing is a precursor to software quality as it validates the requirements and the functioning of the software system. Tests performed in a repeated manner on the software system assure that the most common use case scenarios have been covered. This helps in providing a product that functions well and does not fail during execution. Rigorous testing ensures that scenarios of execution of the software with different inputs have been done and that the software matches its expectations. The more a software application is tested, the lesser the chances are of finding a major flaw or defect, thereby guaranteeing the quality of the product.

Furthermore, software testing can uncover hidden or tricky issues resulting in an added benefit to the overall health of the software product. Moreover, a software product that is sufficiently tested produces better results and is more reliable. Acquiring accurate results of execution is an

excellent measure of the quality of the software product. The better the product is tested, the better it performs consequently improving its quality.

- **Cost Effective:** Contrary to the popular belief, software testing is successful in reducing long terms costs of a software product. The development of a software solution is a complex process made up of several stages in its life cycle. If issues or bugs are detected during the initial phases of the development cycle, the cost of rectifying the mistakes is less. As the stages of the software development process move towards release it becomes harder and costlier to correct errors. An error found after the product has been released would incur maintenance costs and costs pertaining to releasing an updated version of the software. A similar defect found in the development phase would not cost the same. Testing done at an early stage in an iterative manner is helpful in identifying all possible defects before the product is handed off to the users.

Issues found after the software product has been released incur losses that are not just of a monetary nature. In addition to reduced sales of the product due to its poor quality, if an application has several defects that are present in the final product there could be loss in terms of reputation and customer satisfaction. If customers are not satisfied with the product, they could switch to using other products of a similar nature. They could lose trust in the company and be wary of entering into a professional relationship with the same firm in the future which is less than ideal for the company.

Software testing, on the other hand, is an excellent tool for avoiding such disastrous results. It serves as a pocket-friendly approach over time. It is an investment that will benefit the project in terms of its budget and quality. Additionally, proper testing uncovers more defects which in turn reduces the maintenance costs which ends up as an economical decision in the end. The price that is paid by implementing software testing practices is definitely worth the returns acquired owing to the success of the well-tested and accurately functioning software product.

- **Customer Satisfaction:** The primal intent of designing and releasing a software product is to deliver a product that meets the expectations of the customer. The goal of the entire process of development is to provide the best possible product to the customer and ensure that he is satisfied with the product. As the customer/stakeholder is the one paying for the product, his/her

satisfaction is primordial to its success. The company providing the software product, in turn, wishes to ensure this by providing the customer the best possible experience while using their product.

The experience the customers have with the product, directly and indirectly, dictate the value of the software in the market. If the product given to the customers is of a great caliber its value will only rise in the market. If the product quality is trusted by the clients using the product, the loyalty of the clients is gained. Additionally, if the product garners more customers its success increases as chances of getting term clients rises.

A software product does not function properly more than 40% of the time fails to gain the trust of the customer. A product that does not work at all times, that has unusual behavior for the same inputs, is considered to be unstable and unreliable. Unsatisfactory experiences with the product do not help gain the customer's confidence. It is important that user's or customer's experience with the product is a good one, else they might end up looking for other products in the market. Customer's confidence can be gained if the product performs well which can only be achieved through detailed testing. Detailed and thorough testing is the only way to ensure that a customer's experience with the product is a positive one. Repeated testing of different use cases that will be executed by the customer is a good way to check if the product works as designed for the customer. Testing done in an iterative manner which is also continuous is ideally a good way to uncover defects, fix them and retest them in the next iteration. It will help reduce defects, improve the accuracy of the software system thereby paving the way for a product of good quality which the customer can trust.

- **Better Business:** Any organization providing a software product which is supposed to be released in the market is in the market to provide a service to the customers and to make profit. The market to which any software products are released is a critical place where all new products are introduced to the public. Any company that produces software products is counting on the software to do well to make profits and stay afloat as a business. As discussed in the previous point, customers do not look too kindly on software systems that do not work correctly. They might be skeptical of adopting the product if they are not completely satisfied with its performance. If the product does not do well and is disliked by its customers, there is a chance that the product fails in the market. Failure on the part of the software could result in significant losses

financially. The company could end up failing and incurring a debt. If things do not go well, they might even have to shut down their operations. This usually happens in the case of start-ups or small enterprises, and financial loss is a dire outcome for such organizations.

An infallible way to assure that the company's product and in turn the company's business performs well is to test it thoroughly. This process would elevate the quality of the software product which in turn would gain the customer's support and appreciation which eventually would increase the product's global success. The only way of avoiding product failure is to ensure that the product performs well is to ensure its quality, its validity and establishing the client's trust. This can be achieved with the help of software testing. Testing the product in depth before introducing it would help deliver a system that is robust, of good quality and user friendly. This in turn, would help build long-lasting client relationships and retain the customers which would help the business stay afloat and generate profit.

AUTOMATED SOFTWARE TESTING

CONTENTS

2.1. Introduction.....	10
2.2. Benefits of Automation Testing	10
2.3. Types of Automation Tests in Software	16
2.4. Different Automation Testing Software	23

This chapter introduces the concept of automated software testing. We cover the benefits of automated software testing followed by its classification and descriptions of each type of classification.

2.1. INTRODUCTION

Automated testing is the implementation of software tools to automate the manual process of reviewing and validating a software product (Dustin, Rashka, and Paul, 1999). Test automation is used to perform automation of tasks that are repetitive in nature and perform testing tasks that are difficult to perform in a manual setting. Automation Testing is a technique of software testing which follows the basic principle of software testing which is to test the product and then compare the expected results with the actual outcomes. The difference is that in case of automated testing, this process is done by means of an automation testing tool or test scripts.

Automated software testing is in most cases done by using an automation tool. This is so that the quality assurance team can focus on other tests that require manual intervention and input. Doing so will increase the test coverage and render the project scalable. The automated type of software testing is usually well matched and compatible with large-scale software development projects that require testing of the same sections again and again and projects that have already covered an initial iteration of manual testing. In the next section, we look at some advantages of automated software testing.

2.2. BENEFITS OF AUTOMATION TESTING

One of the rudiments of the agile delivery process is the development of software using automation testing. The agile process aims to achieve continuous delivery of the software and automation testing is the only course of action to sustain such a continuous delivery model. Automated software testing serves to increase test coverage, reduce the testing costs, increase testing productivity, and achieve and sustain continuous delivery of the software (Collins et al., 2012).

The earliest origins of automation in the field of software testing can be linked to automation in the automotive industry in 1947 done by the company Ford that created an automation department (Groover, 2020). The automobile industry used the word ‘automation’ to expound the

augmented use of automatic devices and controls in production lines that were mechanized. The word ‘automation’ was conceived in 1946 by D.S. Harder who worked as an engineering manager the Ford Motor company at that point in time (Groover, 2020). Although this is a term that is widely used in the context of manufacturing, it is also employed in other domains to describe a wide range of systems that implement a considerable amount of substitution of either electrical, mechanical, or computerized action for human intelligence and/or action.

One of the earlier works in the field of automation testing was a paper written in the year 1962 titled ‘Automatic program testing’ by G. Renfer presented at the 3rd conference of the Computing and Data Processing Society. This paper described the creation/programming of testing packages that would be used for standardization of test procedures and promoted efficient use of machine’s time (Renfer, 1962).

In the late 1900s (1970s–1980s) the field of automation testing got more traction and test tools capable of automation soon became available (“Test automation,” 2021). The field of automation testing has grown exponentially since then. Now, it is a discipline, and the evolution of this field has given birth to exciting career opportunities for students studying software engineering and professionals alike.

Automated testing in software helps the process of software development become more robust, productive, and capable of delivery in fast iterative cycles. Let us look at some of the benefits of implementing automation in software projects:

- **Reduced Long-Term Costs:** The initial cost of investing in automation testing is a little steep which is a deterrent for most companies. But contrary to popular belief, automation in software testing has proven to be worth the primary payoff. It is proven to be more cost efficient as compared to manual testing. Analysis has shown that over a period this method of testing helps you break even.

Although manual testing is needed in some cases, it does not allow the execution of tests in a repetitive manner. Manually repeating tests is more expensive, arduous, and demanding. Over time, repeating tests in a manual fashion increases the cost of testing your software application.

On the other hand, in contrast to manual testing, automated software testing is inexpensive because once the scripts for testing have been

generated; they can be reused any time with incurring any supplementary costs. Despite the initial hurdles in terms of money for the adoption and implementation of automation testing, the pros of automation outweigh the cons, in this case, money. One must remember that the size and depth of adoption of automation testing determines the return on the initial investment in automation. The more one uses automation and creates tests, the higher the returns accrued. The higher the number of test cases and test suites, the better the returns on the money spent for adoption of automation.

What automation testing does is that it liberates time that could be used to focus on cases where manual testing is necessary. Larger and more challenging issues such as customer inputs, improvements, and functionality-related topics can be covered by the quality assurance team. Automation helps reduce the need for revising the code several times, and in due time, it pays for itself. Furthermore, every time the code of the software is changed, the automated tests can be re-executed as many times as desired with no supplementary overhead and cost. Another way to be cost-effective by using automated testing is to execute tests in a parallel manner. Instead of running the test cases or test suite or test scripts individually, one can use parallel testing, thereby allowing you to execute multiple tests that are automated at the same time. This way, considerable amount of time and effort is reduced. The sizeable reduction in terms of execution time of automated tests can then be put to good use to take care of other needs of the project.

- **Faster Software Development and Delivery:** Automation of testing in software allows one to execute tests that can be executed repeatedly and with speed. This will save a lot of time that would have been spent on manual testing. The testing team would not have to wait for longer periods of time to obtain the results of execution of the tests. They would obtain the results in a few hours which would help them give feedback, if any, to the development team rapidly so that they could correct the defects. Once the code is changed, rerunning the tests would incur no overhead; one must simply execute the scripts again. This not only reduces time, but all reduces the development cycle and helps release the product frequently. Owing to the rapid execution of test cases and the repeatable quality of the tests, automation easily augments the speed in which the software is developed.

Key benefits of shorter testing times by the use of automation are as follows:

- i. **Short Cycle of Development:** As testing is faster compared to manual testing, issues are found and fixed faster, which reduces the time of a regular iteration of software development.
 - ii. **Frequent Software Releases:** As testing is done at a faster rate, an iteration of the development cycle is completed in lesser time, which in turn helps release more versions of the software, which is desirable (Mantyla et al., 2015).
 - iii. **Prompt Changes to the Application:** Owing to the reduced testing time, any modifications or updates to the software can be brought about in a rapid manner. Also, as the development cycle is shorter with automated testing, any changes that are implemented can also be released quickly.
 - iv. **Rapid Time-to-Market:** As the development and testing life cycle is shortened by use of automation in the software testing process, it is no surprise that the time required to deliver the product is reduced as well (Limaye, 2019). Thanks to automated testing, the product can be delivered to market in a shorter time frame which works in the favor of the company. The use of automation in testing will boost the development process and reduce the overall waiting time to release the product into the market.
1. **Growth in the Productivity of the Team:** A distinct gain offered by automating the tests is that no individual intervention by a person is needed to execute the automated tests. This means that the tests can be executed at night in bulk or in parallel and the results can be repeated the next morning. As automated tests execute on their own and in a repeatable manner, nobody from the quality assurance team needs to be present in person and run them and monitor them. This helps plan the testing process so that automated tests are carried out at night, the results monitored early next morning, and feedback given to the development team immediately upon harvesting the results. This reduces the time spent between execution of the tests and fee-

dback given to the development team. The developers do not have to spend time testing extensively thereby saving their time and allowing them to focus on other development focused tasks. Same goes for the quality assurance engineers who can spend less time on testing and focus on tasks that need human intervention. In short, the team in general now has time to concentrate on critical tasks rather than spend their time on testing which gives their productivity and motivation a boost.

2. **Test Accuracy:** Automation testing reduces the human interaction and intervention into repeatable tests to a minimum. This eliminates the chances of human error that could be made by any person during the testing an application repeatedly. One must remember that executing and repeating the same tests repeatedly can become monotonous for a quality assurance engineer. The engineer can lose interest or become tired of doing the same thing over and over again which could result in him making mistakes and missing things. This can be eliminated by use of automation in software testing. With automation, there are likely to be no human errors during the process of testing as a machine would be executing the tests. The test cases that are generated by an automation tool are more precise and ideally cover all the possible scenarios which will reduce the risk of failures. In general, the tests conducted by use of automation have a high chance of being accurate. Reduced error in the testing process will lead to product releases that are free of defects.

This can also help the quality assurance team as it frees up their time to focus on critical issues. They can switch their attention from doing a repetitive and mind-numbing task to interesting critical issues such as exploratory testing, customer needs, generation of additional automation scripts and so on. The quality assurance team is free to perform manual testing on areas they feel need more inspection and attention. The benefit of using automation is that the time gained by the test engineers can be used to draw insights from the tests that might not be noticed by test automation.

3. **Higher Product Quality and Performance:** As automating testing gives us the capability of generating and executing thousands of test cases in parallel at the same time, the software application is tested extensively. The possibility of developing automated test cases increases the number of tests that can be executed thereby granting extensive coverage of the application.

As more than thousands of tests can be executed simultaneously and on different platforms and different devices, in-depth testing can be guaranteed which would result in the software being a better-quality product. Chances are that with increase in test coverage, most of the potential issues would be uncovered which means that the product would perform well. Different hardware and operating system configurations on which the software can work will be covered with ease by means of software testing. Many tests, including complex ones, can be created in a short time which leads to producing a system that is of high caliber.

4. **Rapid Feedback Loop:** Yet another reward of using automation in software testing is that due to the rapidity of executions of tests, the reports are obtained sooner. This means the developers receive the reports and the feedback instantly and they can then quickly set about correcting any issues found during testing. They can immediately start working on issues raised by the iteration of automation testing. They can react more quickly to the report rather than in a few weeks in case of manual testing. They do not have to wait and recall what they coded a week back as the results are received not long after the code was written and therefore it is fresh in their minds. This helps the developers maintain the context under which they wrote the code.

This is primarily of use when the software system is already released to the market. In such cases, any defects found post-deployment need to be fixed quickly and testing such defects and their fixes manually will only slow down the response time. Comparatively, when test automation is used, any changes or modifications to the system will be done quickly and the application will be released in a short span of time. This will help maintain the customer's trust. Thus, owing to the rapid feedback loop the productivity and responsiveness of the team increases, which further provides better user experience resulting in a satisfied customer.

5. **Continuous Integration/Continuous Delivery and DevOps (Development and Operations):** Automation testing is one of the pillars for the process of DevOps and continuous delivery of software products. Currently, most software projects implement the Continuous Integration/Continuous Delivery and DevOps practices. In a delivery pipeline that is programed to perform continuous integration and delivery of the software product,

there is a need to test every little code modification (or code commit) done which is not possible unless one has automated testing. It is not feasible to test every code modification quickly and in an efficient manner when a continuous delivery pipeline is implemented where a software release is delivered several times in a week. To test the product effectively in the given time frame is next to impossible which why is automation comes in to save the day. Automation and CI/CD methodologies go hand in hand as automation helps implement development cycles that are short by allowing the quality assurance team to test each delivery/version of the product rapidly. This way the product is tested after every change in an efficient and fast-paced manner. By using automation testing the implementation of CI/CD and DevOps practices becomes effortless.

2.3. TYPES OF AUTOMATION TESTS IN SOFTWARE

To decide the automation test suite that you wish to implement in a particular project, it is important to know the type of the automation test that you wish to include in your project. The type of automation test helps define the test suite that the project will implement. Depending on your software application there are different types of testing that could be potentially automated.

Automation of tests can be classified in different ways based on the type of functional testing, the type of testing (unit testing, regression testing, etc.), or the phase of software development life cycle.

2.3.1. Automation Based on Type of Functional Testing

1. **Functional Test Automation:** Automated tests can be designed based on the functionality of the application under test (Polo et al., 2013). Functional testing is used to test the business logic of the software system (Beizer, 1995). The logic with which the system is implemented is tested using functional testing which means that this type of testing can be automated as well. Automation of functional tests would entail the writing of scripts that validate the business logic of the system. The test script meant for automation must also be capable of validating the functionality that is expected from the application.
2. **Non-Functional Test Automation:** Another way in which one can decide the automated test suite is to base the tests on

non-functional aspects of the project. Non-functional tests are used to test the non-functional parameters or requirements of the software project. Non-functional parameters consist of parameters such as performance, database, scalability, security, etc. These requirements are static in nature which makes it easier for them to be included in automated tests. Such requirements can remain static or constant for the test suites or they can also be scaled depending on the size of the software application and the hardware and/or operating system requirements.

2.3.2. Automation Based on Type of Testing

There exist several types of testing such as unit testing, regression testing, smoke testing, integration testing, security testing, etc. (Beizer, 2003).

Some of these types of tests that can be automated are:

- **Unit Tests:** It is one of the most common tests used and run by developers. A unit is nothing but a test that is used to test a particular part or particular functionality implemented by a certain part of code (Pugh and Ayewah, 2007). In short, a unit can be anything from a small method to a function. The gist is that the unit performs a single functionality that needs to be tested. A unit test is a script that tests a specific code by means of initializing it, calling the different methods and functions in the code, and checking the values returned by the method or function.

Unit tests are generally written by developers/programers locally as a part of the standard TDD (test driven development) practices. Most of the unit test are built to test the code of the software application and are usually in built in the code of the software itself. Unit tests are written by the developers but in recent times automation engineers or testers may be required to write them as well. Such tests are used by developers to check if the particular code behaves as it is supposed to without errors. Executing a unit test and obtaining a positive result (no defects) means that the code compiles well, that it does not have issues and that it is working as designed. Unit tests do not target complicated and highly functional aspects of the software application because they usually target just the code. This makes it easier for one to automate these tests. The developer can just write the tests and execute them whenever he wants.

- **Integration Tests:** This is the test that verifies whether all the modules function correctly once integrated with one another. This

type of testing is used to test the software system by combining all the modules or integrating all the modules and sub-modules and checking whether the functionality desired is achieved after the amalgamation (Beizer, 2003). An integration test, in most cases, is a code level script which means that when writing such a test we do not run the user interface (UI). An integration test tests the entire process that involves several objects that interact with one another.

Integration testing can be done through different means such as through API testing or by use of mocking. An integration test would require a little more work for its creation as the test team has to set up a lot of data and consider complex processes and how different modules interact with one another. This test might also require a mock database. Although the initial configuration for an integration test is a little high, once automated, it is very easy to add new modules to the software and test their integration. Therefore, an integration test is considered to be good candidate for automation. Additionally, to eliminate the initial effort required, one can always make use of automation frameworks that reduce the initial workload.

- **Smoke Tests:** These are tests that are performed to ensure the stability of the software application. It is used to guarantee that the core functionality provided by the software is working as expected (Chauhan and Kumar, 2014). This is a type of a functional test that covers the important and crucial functions of the software. This type of test is carried out to guarantee that the application can be tested additionally does not ‘catch fire’ which is where the name ‘Smoke Test’ comes from. Smoke tests are generally tests that are run after the application build is done (Antunes and Vieira, 2012). This means that after a full build is done, smoke tests are run to check if the application still works at its core before any more tests are run. This helps determine issues in the application early in the development cycle.

Usually, smoke tests are not large in number which means that the test suite would be small as well because only the core functionalities are tested. But, as smoke tests are executed after every build, they are executed several times. Hence, it is another prime candidate for automation. Since it is repetitive in nature, automation will reduce the manual labor required to perform the smoke tests. It makes sense to automate smoke tests as they are a precursor to any further testing that needs to be performed by the quality assurance team. Smoke tests are functional in nature and based on the type

of software application being developed, critical features and paths can be outlined to be tested and automated by the quality assurance team (Chauhan and Kumar, 2014).

- **Regression Tests:** It is the test that is done when a new module/component is added to check that the new module has not impacted the existing modules in any way. It is a test that makes sure that the new addition to the software has not affected the existing modules and their features in any way (Beizer, 2003). In other words, this test ensures that the old code works exactly the way it worked before without any issues or ‘regressions.’ This test is usually performed to ensure the validity of the application after new changes have been added to it.

Regression tests are usually performed once a new module has been added to the software system and been tested on its own. Unlike integration testing which is carried out to check the functionality of different modules with each other, regression testing is performed to check if old defects have been reintroduced into the software application after some code changes have taken place. This test is ordinarily done after the new modules have been tested on its own which means that it is executed after every iteration of the test process. As regression tests are performed quite often, they are one of tests that can be easily automated. As regression tests are repeated in each iteration where the main tests stay the same and a few tests are added after each new module is tested, it is usually automated. Due to frequency of execution of regression tests, most quality assurance teams try to automate them.

- **Security Tests:** These are usually functional and non-functional type of tests that check the software system for any vulnerabilities. Security tests are done to uncover any weaknesses in the system. They reveal potential areas of weakness in the software application that can be exploited by someone with a malicious intent. In most cases, security is critical to the acceptance of the software application. It is done to ensure that that software application being built is robust. This type of testing covers a wide range of parameters such as authorization, authentication, safety protocols, etc., that are used to test the system and help enhance its security and increase robustness. The goal is to uncover signs of weakness in the system.

The security tests may vary from application to application, but based on the system being built, a set of security tests tailored for the system under consideration can be proposed and executed. As such tests are usually static in nature and would not change regularly, they can be easily automated. The ways in which one tests the basic authentication and authorization of a software application is a testing standard and hence security tests can be automated with ease (Potter and McGraw, 2004).

- **Performance Tests:** It is a non-functional type of test that aims to test the response of the software under test. This testing is one of the most frequently implemented testing techniques amongst all the different non-functional testing. A performance test aims to observe the way the software application responds under normal conditions. The intent is to check if the expected network load is handled well by the system and that the performance falls under acceptable standards. For instance, the response time to open a website should ideally not be longer than a minute under normal circumstances. If it takes more than a minute, it means that it does not respond well to the load it receives. The load that can be handled by the system is tested using performance tests. Responsiveness and stability of the software to handle its regular influx of requests is measured and evaluated by performance tests.

In most cases, the expected system load and expected responsiveness of the software is already determined before the development has started. Performance tests determine whether the system responds the way it is supposed to respond under regular working conditions. The parameters are usually pre-defined and do not vary that much making performance tests a good pick for automation.

- **Acceptance Tests:** These are tests that evaluate the software system based on its functionalities, its behavior, and capabilities. In short, these tests inspect the software system for its readiness to be deployed and released to the market. This test checks whether the business needs of the client are satisfied by the system. Acceptance testing implies that the system is ready to be evaluated by the potential users of the system (Humble and Farley, 2010). It entails the testing of the acceptability of the system by the end-user or the business user. The software application is tested and evaluated by means of acceptance tests, and it is decided by the stakeholders/clients/customers whether

the software application is acceptable for the purpose of the final delivery or not. Acceptance testing is additionally used to establish the conformity of the terms of the contract and legal requirements by the software system. Whether the software system satisfies its legal and regulatory needs/standards is checked at this point of testing, and an informed decision is made. A variety of things are tested here, such as the end user's acceptance, the contractual or legal acceptance, the operational acceptance, and in some cases, alpha and beta acceptance tests are carried out too.

The validity of the software system for real-time uses by a real-world user in a target or simulated environment is performed in the case of acceptance tests. The idea behind executing these tests is to assure that the software application works the way it is supposed to and that it meets the needs of the user. This type of test serves to ensure that the user can execute the business processes seamlessly without complications. The intent is to guarantee that the end-user/customer can interact with the software system in a hassle-free and consistent manner. The software system is tested for acceptance and validation of its requirements and to ensure that it works as mentioned in the contract. The legal and contractual testing focuses on verifying that the software system complies to the agreed-upon criteria of acceptance as documented in the contract. Acceptance test focuses on the software and makes sure that the ensure that regulatory standards at regional and national level have been met by the software.

In simpler terms, acceptance tests are highly functional tests that determine how acceptable the software is to the end-users (Humble and Farley, 2010). This test is usually the final step in the testing process before the software can be released. As acceptance criteria is pre-defined in the contract, these tests can be thought of and determined quite early in the software development phase. This makes it a good test for automation. As the criteria for acceptance will already be known prior to the start of development, the test cases and the test suite for acceptance can be prepared in advance and hence easily automated which would reduce the load on the quality assurance team.

2.3.3. Phase of Testing

Another way or type of automation is automation that is performed based on the phase of software testing. The primary phases where testing can be automated are:

1. **Unit Testing Phase:** As can be inferred from the name ‘unit testing,’ this phase of testing focuses on testing and evaluating individual units or components of a software application (Beizer, 2003). Unit tests are executed during the course of development, ideally by the development team before they hand over the system to the quality assurance for testing. Unit testing is usually done manually by the development team (Beizer, 2003). This phase of testing is ordinarily the first phase of testing which is done mostly manually. But, with a little planning and discussion, it can also be automated.
2. **Application Programming Interface (API) Testing Phase:** It tests the business layer of the software application under test. Software applications that are built on APIs and that support API architecture usually employ API testing (Bangare et al., 2012). Any software system that is based on API architecture can implement API testing. This type of testing is meant to validate the business layer of the software system by inspecting the response time of the APIs used in the application for all the requests. Several test combinations of request-response for various APIs that the application is built on are executed. APIs are akin to middlemen that connects different systems used by the software application in a smooth manner. Due to its nature, API testing is usually done after the software development is complete to guarantee that a smooth integration within the systems and the software under test. API testing is a flexible process which could be conducted prior or after the UI Testing phase. As API tests are of a typically fixed nature that follows the theme of request-response, it is an excellent contender for automation. As API testing is like integration testing, some teams include API testing within their integration tests.
3. **User Interface (UI) Testing Phase:** This phase is the last phase of testing in case of a large group of software projects. The UI is the what the customers/users interact with and hence is an important part of the testing process. UI testing is carried out by quality assurance engineers or testers after the application has been deemed to be stable by the other tests (McGraw and Hovemeyer, 1996). This phase focuses on examining the front-end of the software, its functionality and the look and feel of the application. This phase is focused on replicating the most

authentic user experience in a test environment and finding any issues when a potential user interacts with the system. UI testing is a functional test where the business logic is examined and evaluated. The functionality and the UI elements or the frontend of software system are tested tests (McGraw and Hovemeyer, 1996). The idea is to mimic a real-life scenario of use of the software system and find any underlying issues in the working of the software.

The UI tests can be automated as the scenarios are known to the quality assurance team. Based on the specifications of the software system, UI acceptance criteria can be defined at the time of validation of the software design and updated after each iteration of the development cycle in case of an agile process. An agile process can be useful when UI tests need to be automated. Currently, automation of the UI tests is a practice that is commonplace due to advanced software practices that help automation tests.

Now that we have covered different types of automation testing, we shall delve in different software tools that can be used for the process of automation testing in the next section.

2.4. DIFFERENT AUTOMATION TESTING SOFTWARE

In this section we list some of the commonly used software for automation testing.

Name	Brief Description
Selenium	It is an open-source automation testing platform that is recommended for automation testing of web projects.
Katalon Studio	It is a commercial automation testing tool that can be used for testing API, web applications and mobile applications. Access to the studio is free for individuals but the enterprise version is priced.
Ranorex	It is a commercial automation testing tool that helps automate web applications, standalone applications, and mobile applications.
HP UFT (QTP)	UFT (previously known as QTP) is a licensed, paid automation testing tool that can automate Web, Desktop, Mobile, Oracle, SAP, and Java applications. This tool is known for its cross-browser testing capabilities.
Watir	This is an open-source library developed in Ruby language which can be used to automate web applications.

Telerik Test Studio	Telerik Test Studio is a licensed, paid automation tool which is used to test desktop, web, and mobile applications.
Appium	It is an open-source test automation framework that can be used for testing native applications, mobile applications, and hybrid applications.
Serenity	It is a free open-source library that aids in writing automated acceptance tests.
TestComplete	TestComplete is an automation test tool for GUI testing and is capable of testing desktop, web, and mobile applications (<i>TestComplete</i> , 2022).
Silk Test	Silk test is a licensed automation tool for functional and regression testing developed by microfocus which supports cross-browser testing and can test desktop applications, web applications, mobile applications, web applications, enterprise applications and rich client applications.

Over the next chapters, we will study the following three automation testing tools: Katalon Studio, Watir, and Ranorex Studio.

KATALON STUDIO

CONTENTS

3.1. Introduction.....	26
3.2. Installation.....	26
3.3. Practical Implementations/Examples.....	38

In this chapter, we cover the automation tool Katalon Studio. This chapter introduces Katalon Studio, covers its installation process, and then demonstrates its use.

3.1. INTRODUCTION

Katalon Studio is a robust and easy-to-use automation tool which was initially released in January 2015. It was first released with an engine that was selenium-based. It is an automation testing tool that was designed to create scripts for automation without having to code. The platform is a simple and straightforward tool for automation that does not require in-depth knowledge of programming. It is an excellent automation tool that supports testing on multiple platforms and allows execution of automated GUI tests. Additionally, this platform permits execution of tests on a variety of operating systems (Eriez, 2009).

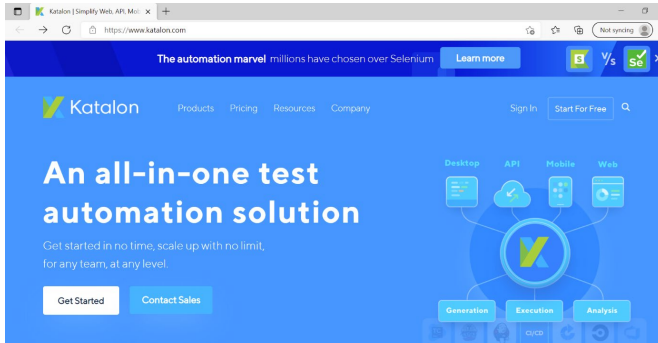
Katalon studio offers desktop, API, Mobile, and Web solutions. Katalon studio offers an excellent IDE that is easy to use and generate automated test scripts (Eriez, 2009). It was initially released as a free solution but now boasts of an Enterprise version that was introduced to provide additional options. But the basic Katalon Studio version that was intended for individual users still is free of charge (Katalon | Simplify Web, API, Mobile, Desktop Automated Tests, 2022). In the upcoming section, we shall look at the process to install Katalon Studio on your machine.

3.2. INSTALLATION

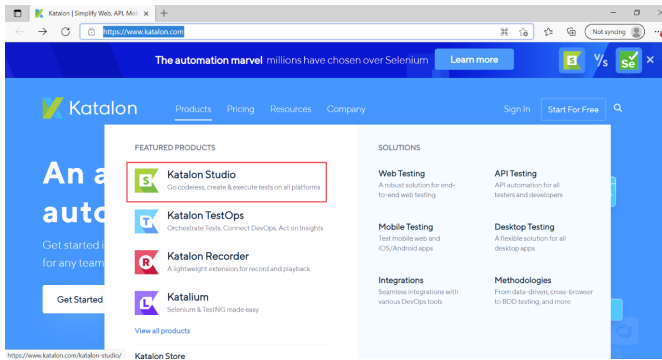
Katalon Studio is very easy to install on any machine of your choice. Here, we install the version designed for individual users on a 64-bit Windows 10 operating system.

The steps followed are as follows:

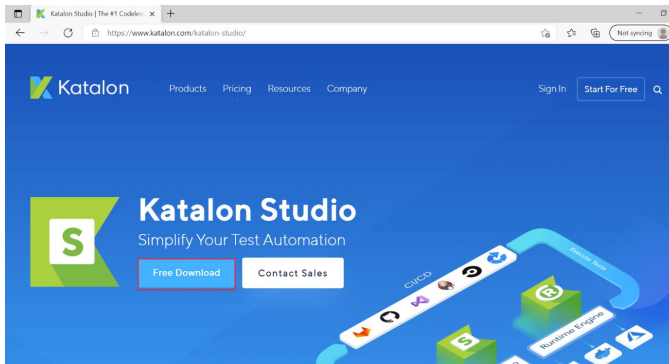
- First, navigate to the Katalon studio website: Katalon | Simplify Web, API, Mobile, Desktop Automated Tests.



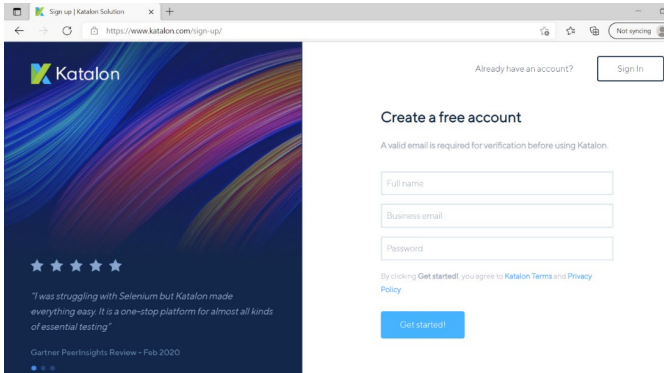
- Go to Products and select the product 'Katalon Studio'



- Select the free download option.

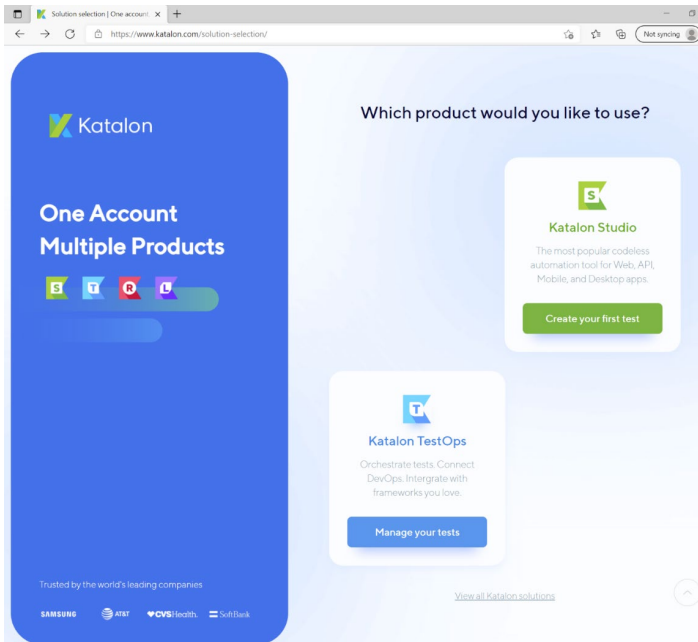


- The site will prompt you to create a free account.



We create an account and click on the ‘Get Started!’ button. You will receive an email asking you to verify your account. Once your account is verified, you can log into your account.

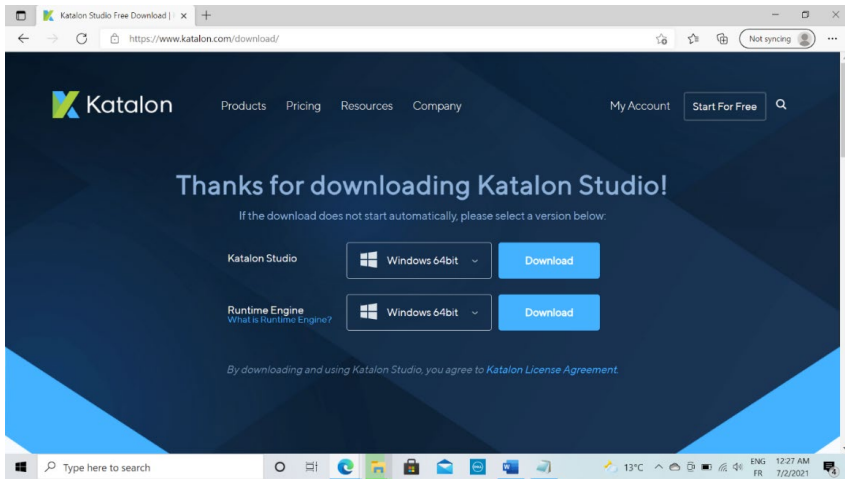
Once you log into your account you are directed to the following page:



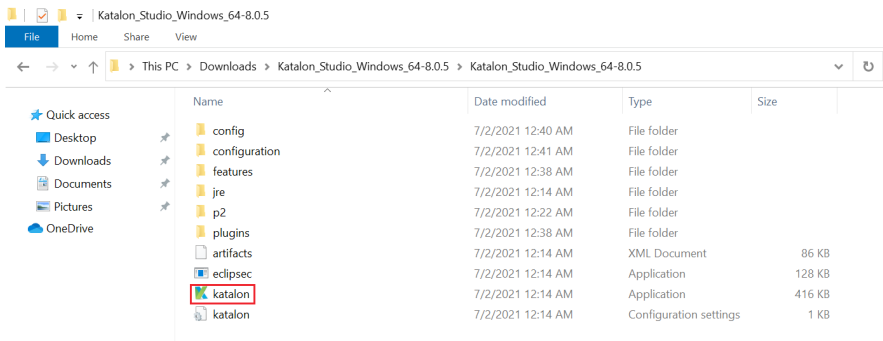
Two options are provided namely Katalon Studio and Katalon TestOps.

We will be focusing on Katalon Studio in this book. Hence, we click on the link that says, ‘Create your first test’ (‘Create Your First Test,’ 2022).

This will redirect us to the download page where the download will begin instantly.



Once the download terminates, extract the package, and start the installation by clicking on 'katalon.exe' which will launch the tool.



To activate Katalon studio, you will need to enter the credentials that you used when signing up for the free account (Eriez, 2019).

Product Activation

Email

Password

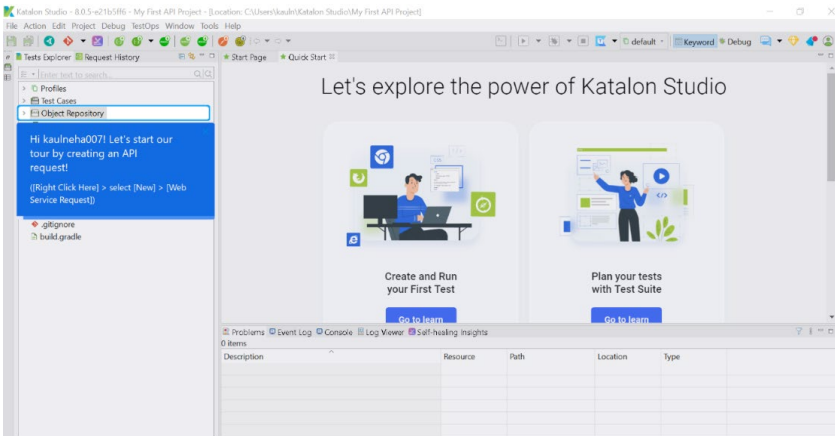
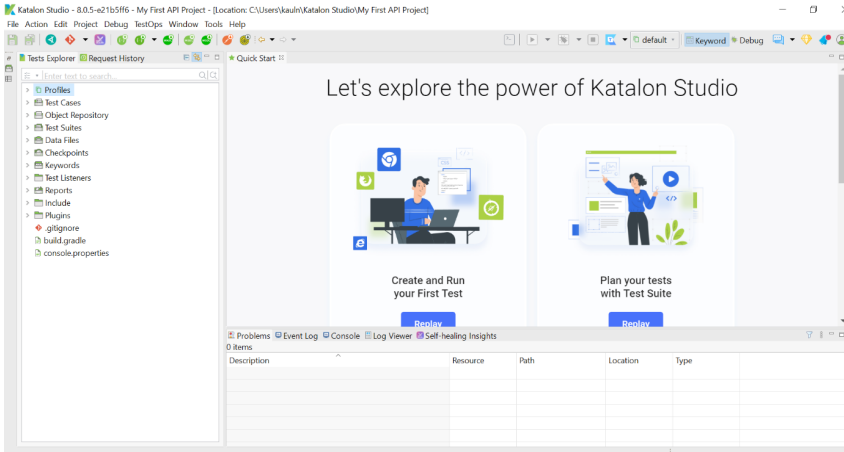
Please use your registered email and password from Katalon website to activate.

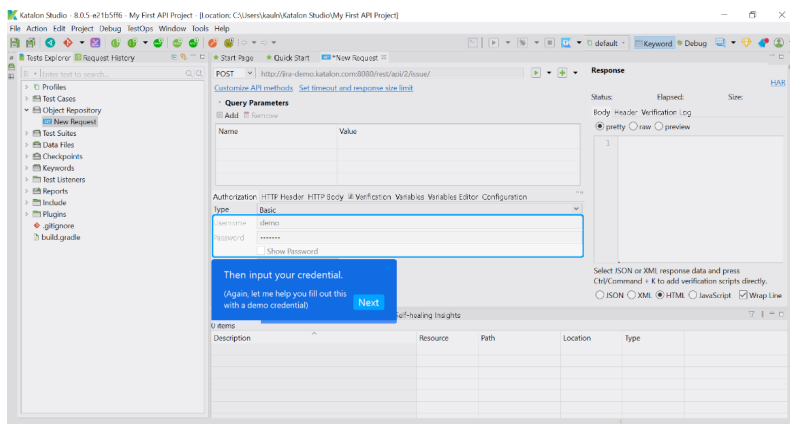
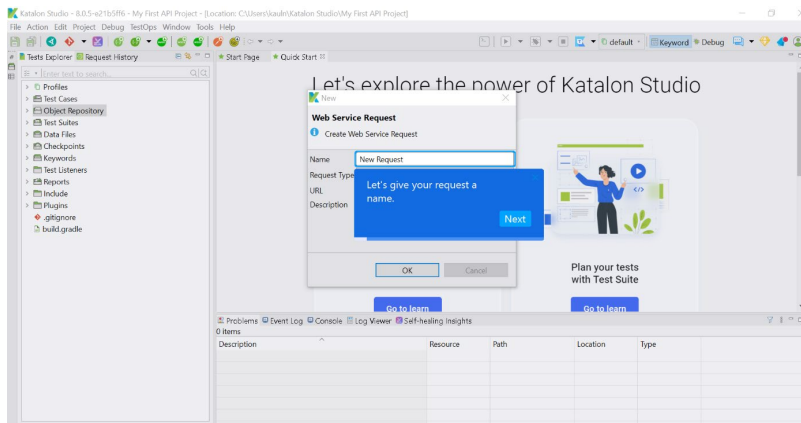
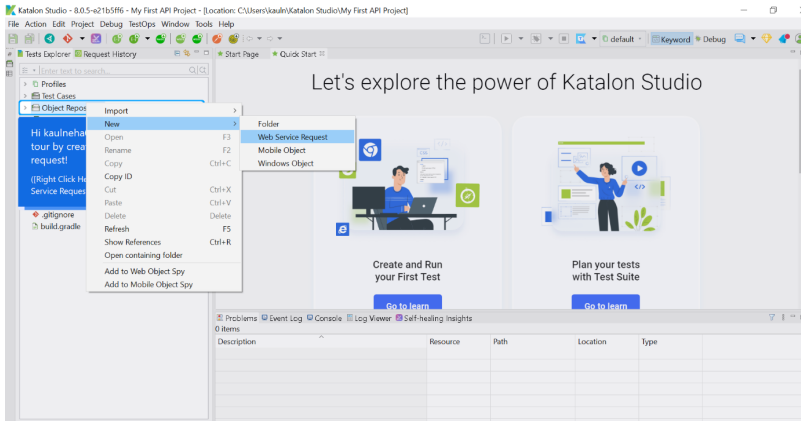
[Forgot Password?](#) |
 [Register](#) |
 [Offline Activation](#) |
 [Config Proxy](#)

Once the product has been activated, the **Quick Guide** screen is displayed which guides you through the significant features.

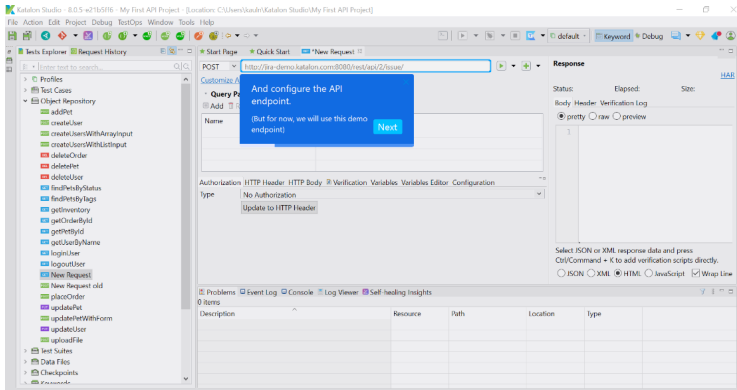
At the end of the introduction, Katalon Studio suggests creating a new project which we follow. Then, we proceed to creating and running a test case as prompted by the quick guide by clicking on ‘Create and Run your First Test.’

The steps followed are:

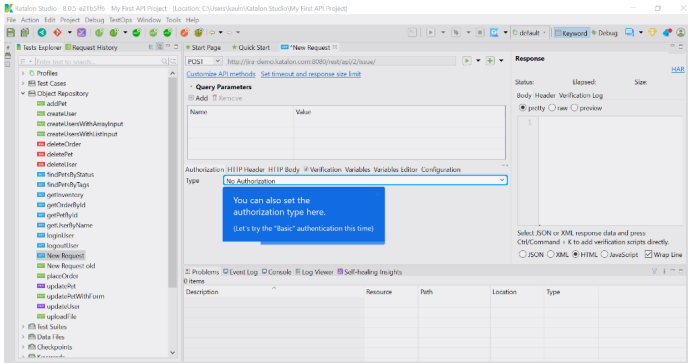




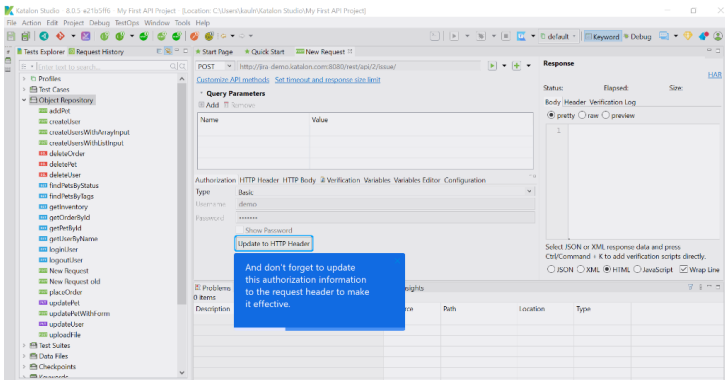
You are prompted to configure the endpoint. A simple POST request is configured.



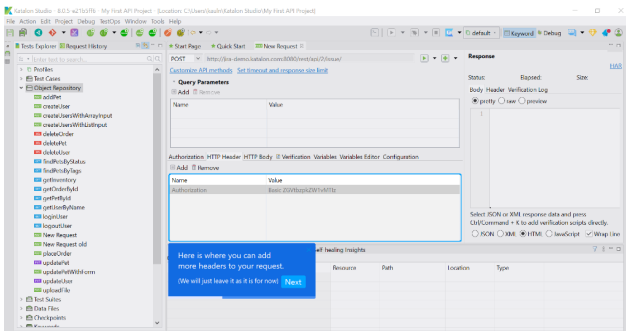
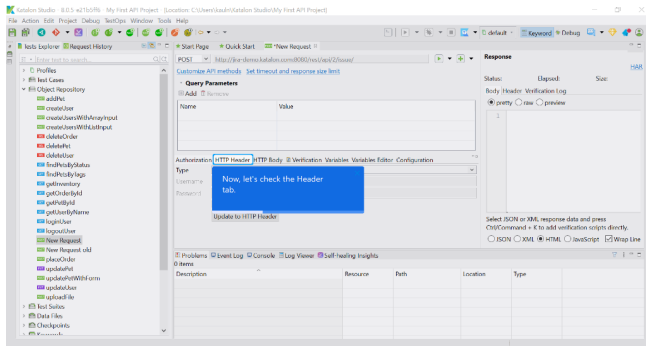
Further, we can set up authorization for granting access to the request. We choose basic type of authorization.



We proceed to update the authorization information to the header.

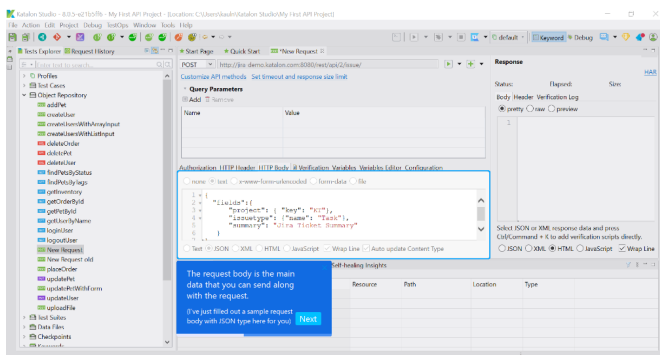


We click on the HTTP header tab to check if the header has been updated.

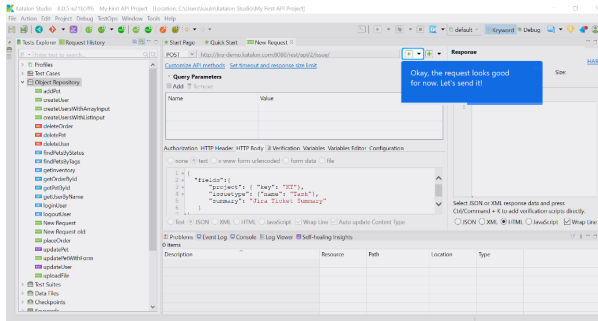


The headers tab can be used to add additional headers. Once the header is verified, we are prompted to check the request body.

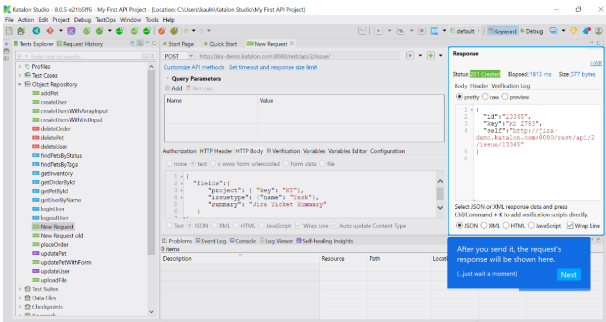
As shown in the screenshot below, the request body is in a simple JSON form.



Now that the basic parameters of the request are in order, we save the request, and we can simply execute the API to test if it works. We click on the play button at the top right corner of the request screen to execute the request.

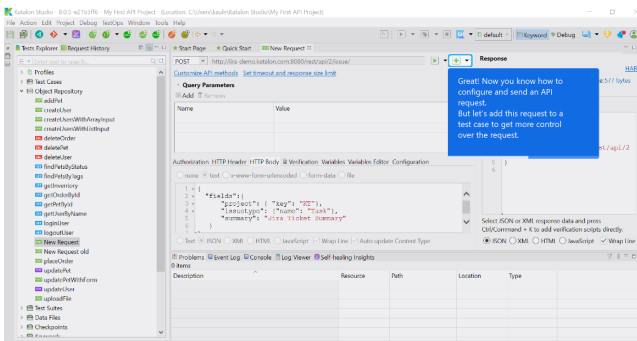


The response is shown in the response tab located right next to the request tab on its right.



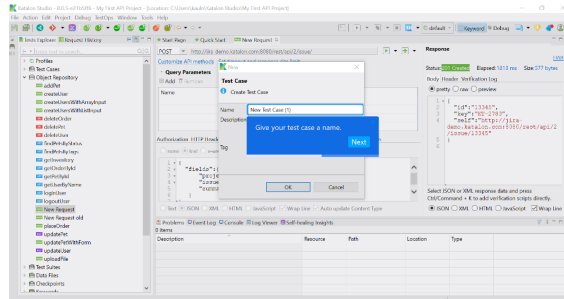
As we can see, we receive a proper response to the request.

We are now further prompted to add this test case to a test suite. Adding to a test suite allows us more control over the test cases.

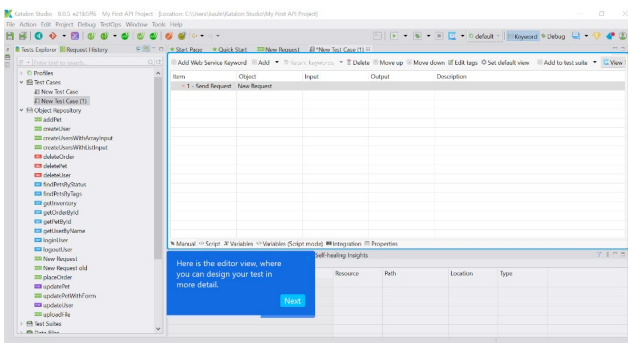


We click on the '+' button to create a new test suite and add this test case to the test suite.

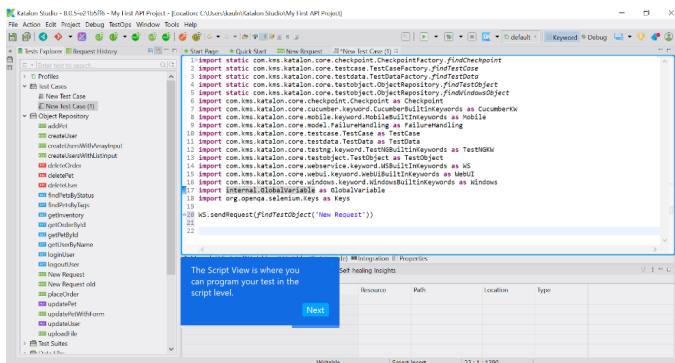
We provide a name for the test suite and proceed.



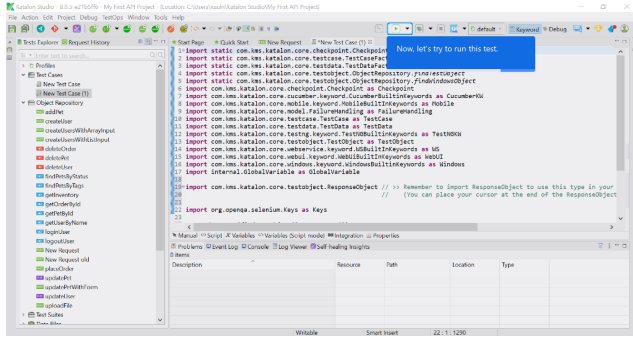
The editor view of the test suite is presented, which allows to configure the test suite.



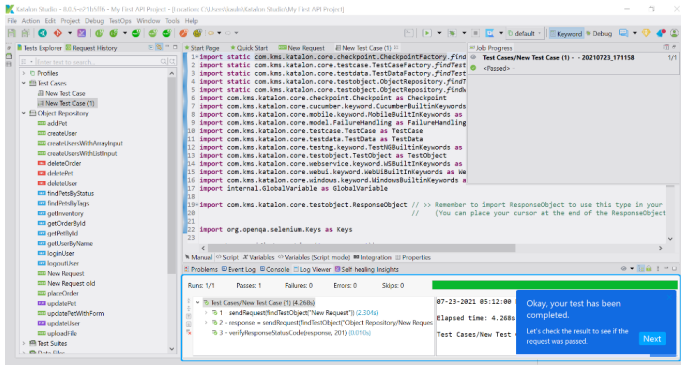
We are directed to check the script view of the test suite. Here, we can see the code that is generated when a test suite is created. If you wish to program the suite in a specific way, it can be done from the script view.



We do not make any changes to the script and save the test suite. We now proceed to execute the test suite by clicking on the play button at the top as shown in the screenshot below:

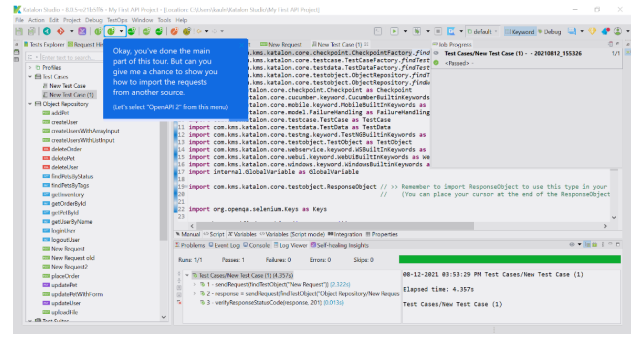


The results of the test suite are shown in the log viewer section at the bottom of the screen.

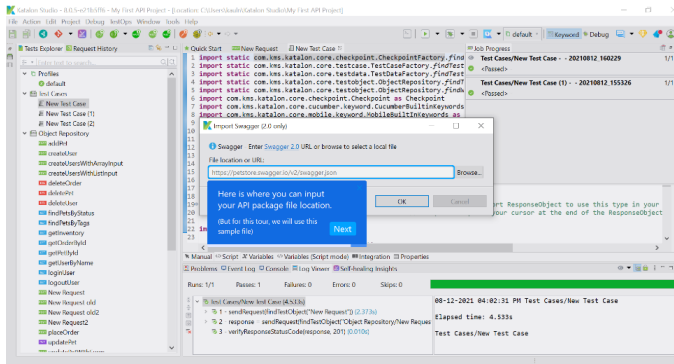


Once the test has been completed, we check if the request was sent correctly by looking at test case results. We can see that the request was sent correctly, and that the status code of the response was 201(OK).

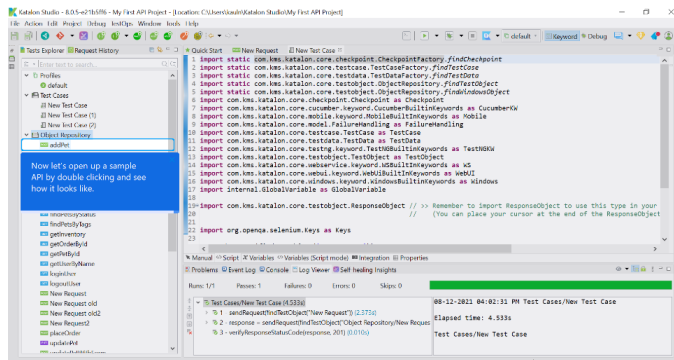
We proceed to importing multiple requests from another source. We go to the 'Import API' tab and click on 'OpenApi2' to open a sample API from the sample source files.



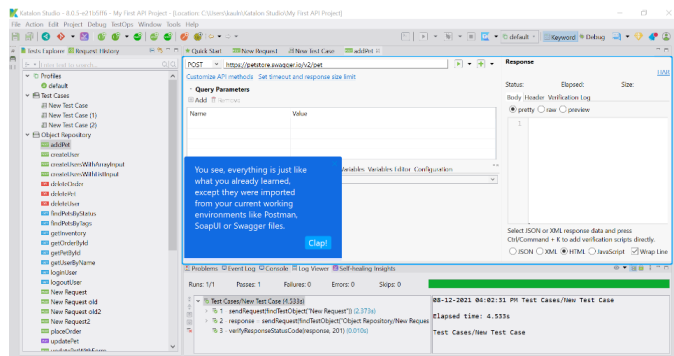
We are led to a window that asks for the file location to import the API. A pre-defined link is provided for the purpose of the demonstration. We click on OK to import the API.



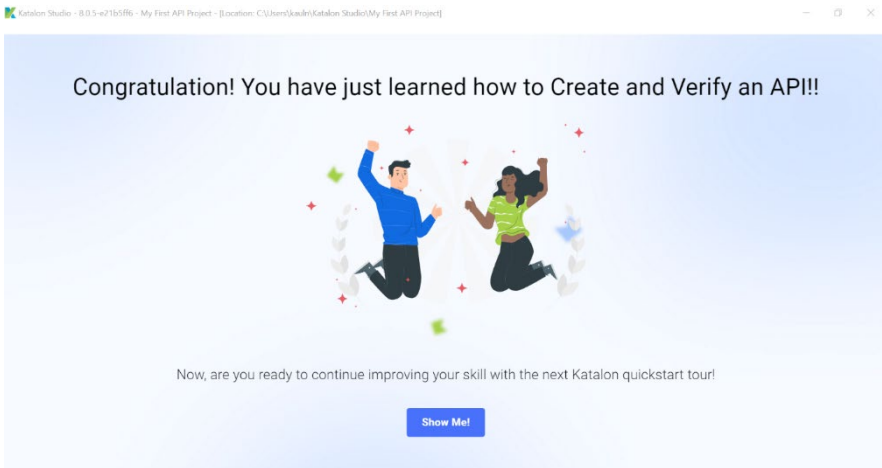
The addPet API is imported successfully into our repository. We double-click to open the sample API we just imported.



As we can see from the screenshot below, the newly imported API looks like the API we created and ran previously.



This ends the demonstration, and we can click on the ‘Clap!’ button to advance to the end.



Additional tours are provided by Katalon that can be followed to learn how to create tests for different types of solutions such as web, API, and mobile.

The Quick Guide menu can be accessed again later from the Help menu (Help → Quick Start Page). In the next section, we shall look at a few examples that demonstrate how to use Katalon studio for developing test cases and test suites for web services.

3.3. PRACTICAL IMPLEMENTATIONS/EXAMPLES

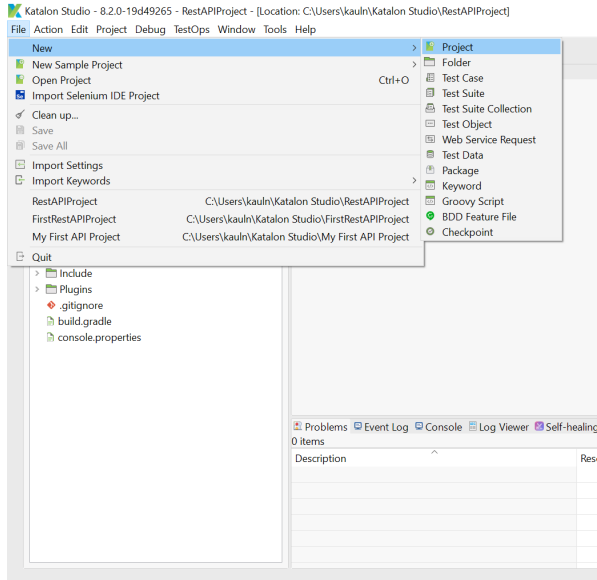
In this section, we learn how to use Katalon Studio with the help of some examples.

- **Example 1: Rest API Testing Using Katalon Studio:** In this example, we learn how to automate testing of restful web services using Katalon Studio. We use a dummy rest API that is freely available – JSONPlaceholder – Free Fake REST API (typicode.com) (JSONPlaceholder – Free Fake REST API, 2022).

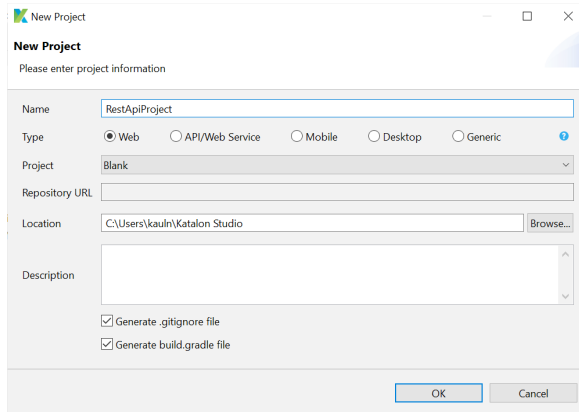
It is an online REST API provider that one can use whenever test/dummy data is needed in your project (JSONPlaceholder – Free Fake REST API, 2022).

The steps to test restful web services are as follows:

1. Open Katalon studio and create a new project.

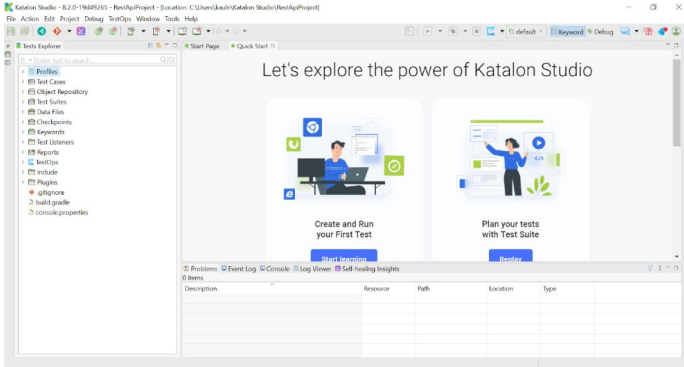


2. Create a new web project.

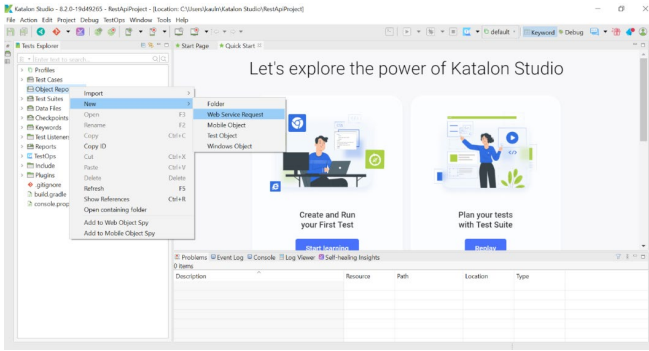


Enter a name for the project and click on the OK button.

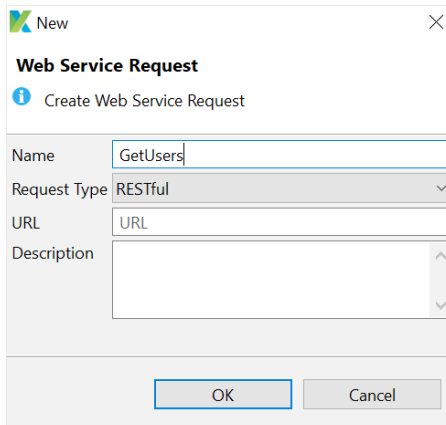
The project is created as shown below:



- 3. We now add the rest API to the project. To add the restful API to this project, right click on object repository, then go to New → Web service request.



- 4. The following window prompt for entering the details of the Web service request pops up.

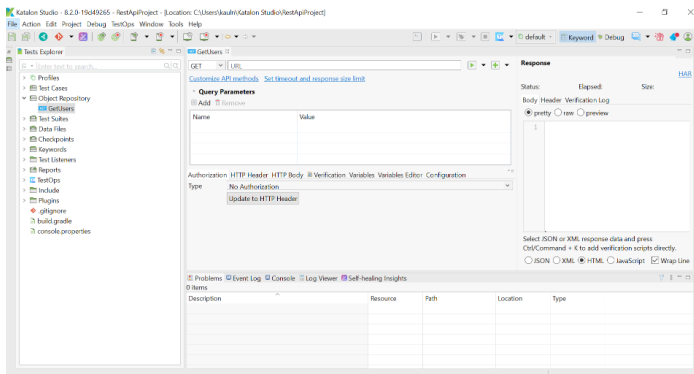


In this example, we will be using the ‘users’ resource from the online rest API provider JSONPlaceholder. Hence, we give the name ‘GetUsers.’

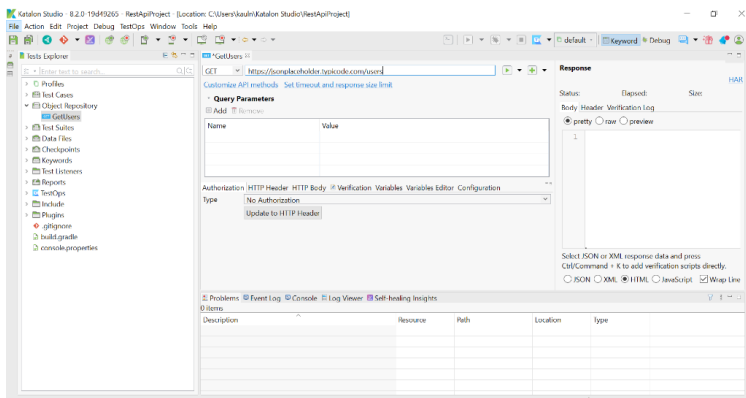
We select the type of Request, RESTful in our case. We can also provide the API URL (URL of the dummy REST API). The URL could be left blank at this step.

We click ‘OK’ to create the request.

The request is created as shown below:



5. We now edit the request to update the URL, http method and other API-related details.



Here, we have updated the URL to `https://jsonplaceholder.typicode.com/users`, which is the GET request that gets all the users’ information as follows:

```
[
{
  "id": 1,
```

```
“name”: “Leanne Graham”,
“username”: “Bret”,
“email”: “Sincere@april.biz”,
“address”: {
“street”: “Kulas Light”,
“suite”: “Apt. 556”,
“city”: “Gwenborough”,
“zipcode”: “92998-3874”,
“geo”: {
“lat”: “-37.3159”,
“lng”: “81.1496”
}
},
“phone”: “1-770-736-8031 x56442”,
“website”: “hildegard.org”,
“company”: {
“name”: “Romaguera-Crona”,
“catchPhrase”: “Multi-layered client-server neural-net”,
“bs”: “harness real-time e-markets”
}
},
{
“id”: 2,
“name”: “Ervin Howell”,
“username”: “Antonette”,
“email”: “Shanna@melissa.tv”,
“address”: {
“street”: “Victor Plains”,
“suite”: “Suite 879”,
“city”: “Wisokyburgh”,
“zipcode”: “90566-7771”,
```

```
“geo”: {
“lat”: “-43.9509”,
“lng”: “-34.4618”
}
},
“phone”: “010-692-6593 x09125”,
“website”: “anastasia.net”,
“company”: {
“name”: “Deckow-Crist”,
“catchPhrase”: “Proactive didactic contingency”,
“bs”: “synergize scalable supply-chains”
}
},
{
“id”: 3,
“name”: “Clementine Bauch”,
“username”: “Samantha”,
“email”: “Nathan@yesenia.net”,
“address”: {
“street”: “Douglas Extension”,
“suite”: “Suite 847”,
“city”: “McKenziehaven”,
“zipcode”: “59590-4157”,
“geo”: {
“lat”: “-68.6102”,
“lng”: “-47.0653”
}
},
“phone”: “1-463-123-4447”,
“website”: “ramiro.info”,
“company”: {
```

```
“name”: “Romaguera-Jacobson”,
“catchPhrase”: “Face to face bifurcated interface”,
“bs”: “e-enable strategic applications”
}
},
{
“id”: 4,
“name”: “Patricia Lebsack”,
“username”: “Karianne”,
“email”: “Julianne.OConner@kory.org”,
“address”: {
“street”: “Hoeger Mall”,
“suite”: “Apt. 692”,
“city”: “South Elvis”,
“zipcode”: “53919-4257”,
“geo”: {
“lat”: “29.4572”,
“lng”: “-164.2990”
}
},
“phone”: “493-170-9623 x156”,
“website”: “kale.biz”,
“company”: {
“name”: “Robel-Corkery”,
“catchPhrase”: “Multi-tiered zero tolerance productivity”,
“bs”: “transition cutting-edge web services”
}
},
{
“id”: 5,
“name”: “Chelsey Dietrich”,
```

```
“username”: “Kamren”,
“email”: “Lucio_Hettinger@annie.ca”,
“address”: {
“street”: “Skiles Walks”,
“suite”: “Suite 351”,
“city”: “Roscoeview”,
“zipcode”: “33263”,
“geo”: {
“lat”: “-31.8129”,
“lng”: “62.5342”
}
},
“phone”: “(254)954-1289”,
“website”: “demarco.info”,
“company”: {
“name”: “Keebler LLC”,
“catchPhrase”: “User-centric fault-tolerant solution”,
“bs”: “revolutionize end-to-end systems”
}
},
{
“id”: 6,
“name”: “Mrs. Dennis Schulist”,
“username”: “Leopoldo_Corkery”,
“email”: “Karley_Dach@jasper.info”,
“address”: {
“street”: “Norberto Crossing”,
“suite”: “Apt. 950”,
“city”: “South Christy”,
“zipcode”: “23505-1337”,
“geo”: {
```

```
“lat”: “-71.4197”,
“lng”: “71.7478”
}
},
“phone”: “1-477-935-8478 x6430”,
“website”: “ola.org”,
“company”: {
“name”: “Considine-Lockman”,
“catchPhrase”: “Synchronized bottom-line interface”,
“bs”: “e-enable innovative applications”
}
},
{
“id”: 7,
“name”: “Kurtis Weissnat”,
“username”: “Elwyn.Skiles”,
“email”: “Telly.Hoeger@billy.biz”,
“address”: {
“street”: “Rex Trail”,
“suite”: “Suite 280”,
“city”: “Howemouth”,
“zipcode”: “58804-1099”,
“geo”: {
“lat”: “24.8918”,
“lng”: “21.8984”
}
},
“phone”: “210.067.6132”,
“website”: “elvis.io”,
“company”: {
“name”: “Johns Group”,
```

```
“catchPhrase”: “Configurable multimedia task-force”,
“bs”: “generate enterprise e-tailers”
}
},
{
“id”: 8,
“name”: “Nicholas Runolfsdottir V”,
“username”: “Maxime_Nienow”,
“email”: “Sherwood@rosamond.me”,
“address”: {
“street”: “Ellsworth Summit”,
“suite”: “Suite 729”,
“city”: “Aliyaview”,
“zipcode”: “45169”,
“geo”: {
“lat”: “-14.3990”,
“lng”: “-120.7677”
}
},
“phone”: “586.493.6943 x140”,
“website”: “jacynthe.com”,
“company”: {
“name”: “Abernathy Group”,
“catchPhrase”: “Implemented secondary concept”,
“bs”: “e-enable extensible e-tailers”
}
},
{
“id”: 9,
“name”: “Glenna Reichert”,
“username”: “Delphine”,
```

```
“email”: “Chaim_McDermott@dana.io”,
“address”: {
“street”: “Dayna Park”,
“suite”: “Suite 449”,
“city”: “Bartholomebury”,
“zipcode”: “76495-3109”,
“geo”: {
“lat”: “24.6463”,
“lng”: “-168.8889”
}
},
“phone”: “(775)976-6794 x41206”,
“website”: “conrad.com”,
“company”: {
“name”: “Yost and Sons”,
“catchPhrase”: “Switchable contextually-based project”,
“bs”: “aggregate real-time technologies”
}
},
{
“id”: 10,
“name”: “Clementina DuBuque”,
“username”: “Moriah.Stanton”,
“email”: “Rey.Padberg@karina.biz”,
“address”: {
“street”: “Kattie Turnpike”,
“suite”: “Suite 198”,
“city”: “Lebsackbury”,
“zipcode”: “31428-2261”,
“geo”: {
“lat”: “-38.2386”,
```



```

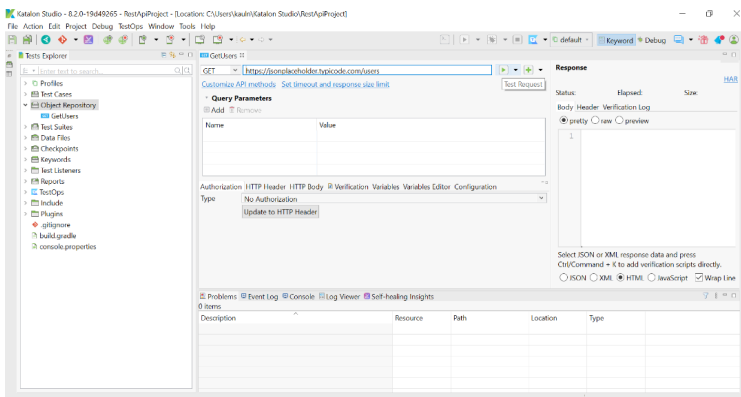
“lng”: “57.2232”
}
},
“phone”: “024-648-3804”,
“website”: “ambrose.net”,
“company”: {
“name”: “Hoeger LLC”,
“catchPhrase”: “Centralized empowering task-force”,
“bs”: “target end-to-end models”
}
}
]

```

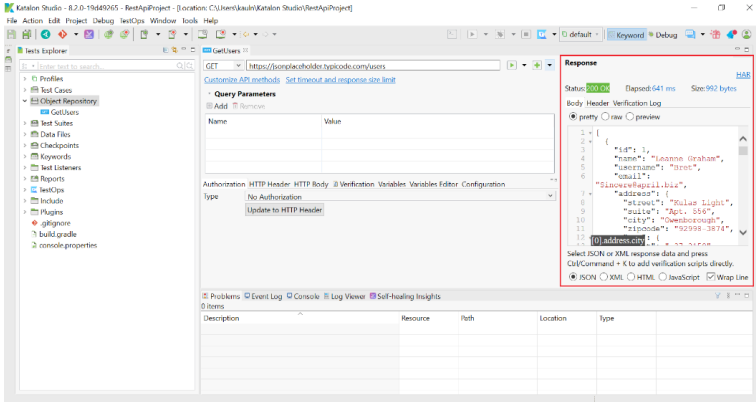
This dummy REST API returns a record of 10 users.

In addition to the URL, we can edit other information such as Authorization, HTTP Headers, Request Body, etc. Since this is a dummy API there is no authorization information entered. But, in real-life cases, this information should ideally be provided.

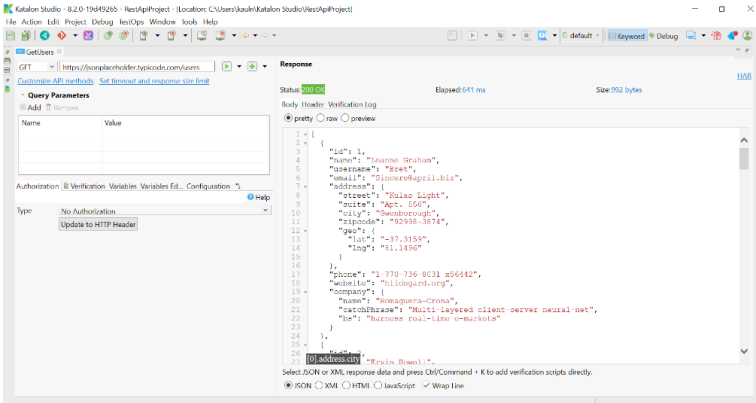
- To test this URL, we simply click on the play button located right next to URL.



- The response of the Request is shown in the ‘Response’ section on the right end.



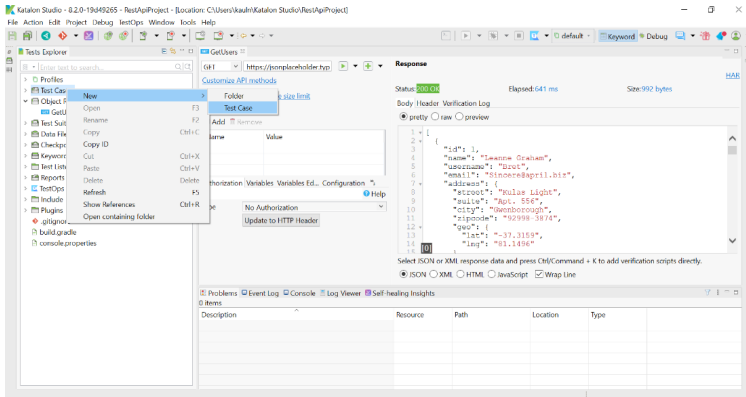
We can expand the Response window to see the response details as follows:



We can confirm that the status of the request is OK (return code 200), and that the response contains the data from the ‘users’ resource from JSONPlaceholder.

The response can be viewed in different formats such as XML, html, and JavaScript.

8. Our API request is set; the next step is to create a Test case for the API. To create a test case for this REST API, we go to Project Section, right-click on ‘Test Cases’ and select New → Test Case.



The following window requesting the basic details of the new test case is displayed:

New

Test Case

Create Test Case

Name

Description

Tag

OK Cancel

We provide a name for the test case.

New

Test Case

Create Test Case

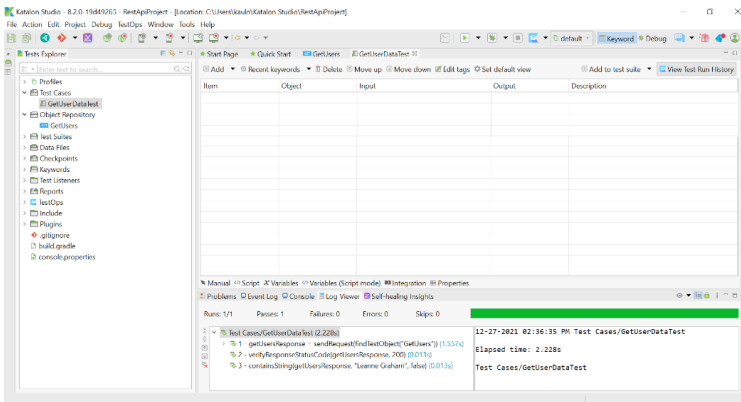
Name

Description

Tag

OK Cancel

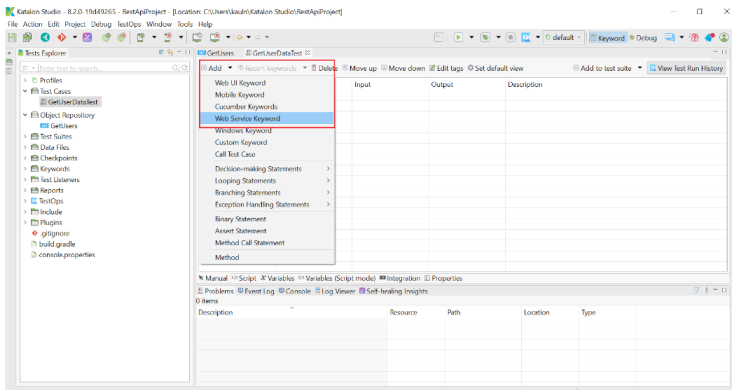
Once the name has been entered, click on ‘OK’ to create the test case. The test case is created as shown below:



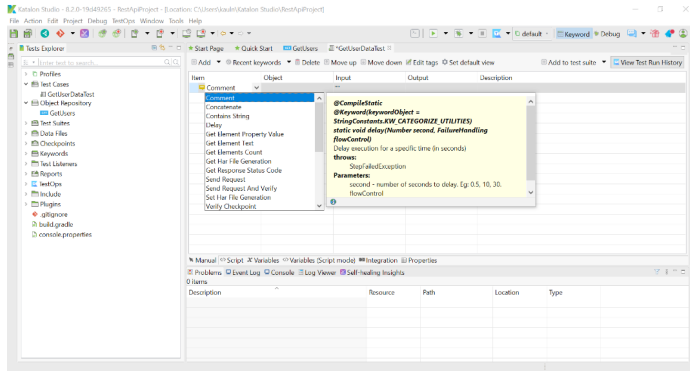
9. We now provide different parameters to test the API.

In this example, we provide three keywords/parameters that we use to test the API and its response. Keywords are like test parameters that can be used to program/configure the test case.

To add keywords, we click on the ‘Add’ button and click on ‘Web Service Keywords’ as follows:



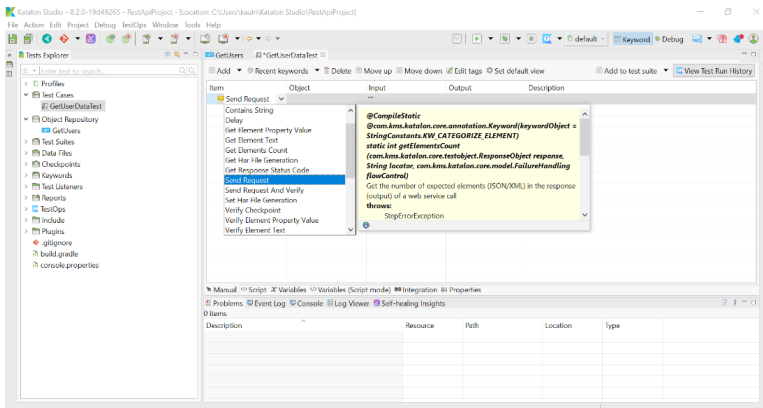
When we click on this, a new entry is added to our test case, and we can select the keyword of our choice from the drop-down list.



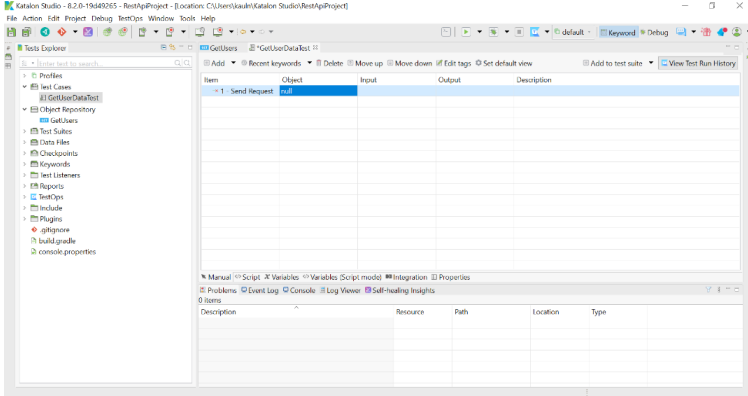
The three Web Service keywords that we are adding in this example are as follows:

- a. **Send Request:** This is used to call the API URL. This keyword is used to send an HTTP request to the web server ([WS] Send Request, 2022).

To add the ‘Send Request’ keyword, we select it from the available dropdown:

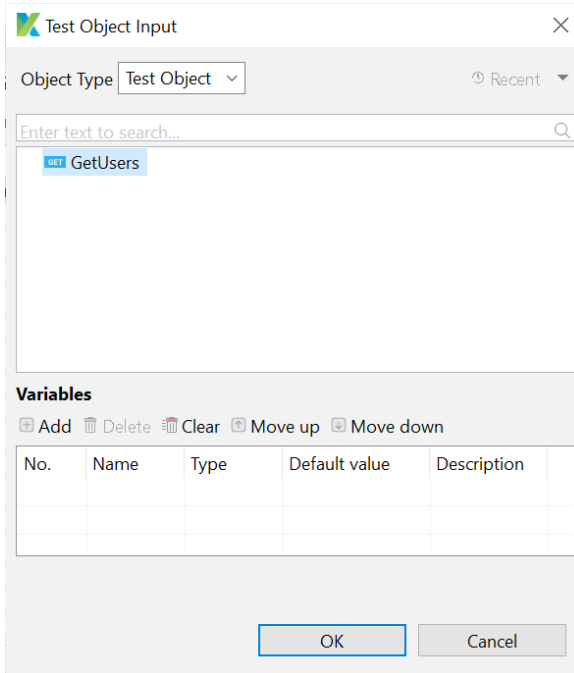


The following entry is added to the test case:

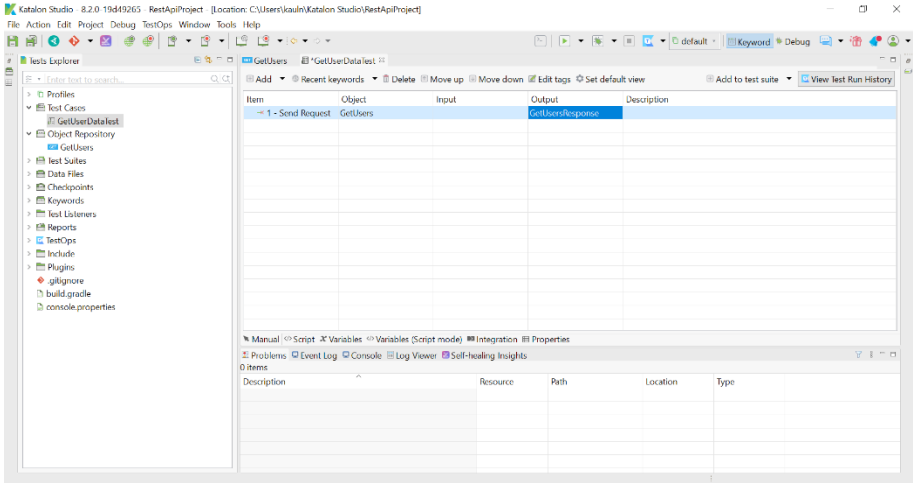


As we can see from the screenshot above, the Object is null here. We edit the object to select the correct API Request.

In this example, we call the getUsers web service request that we created previously as follows:



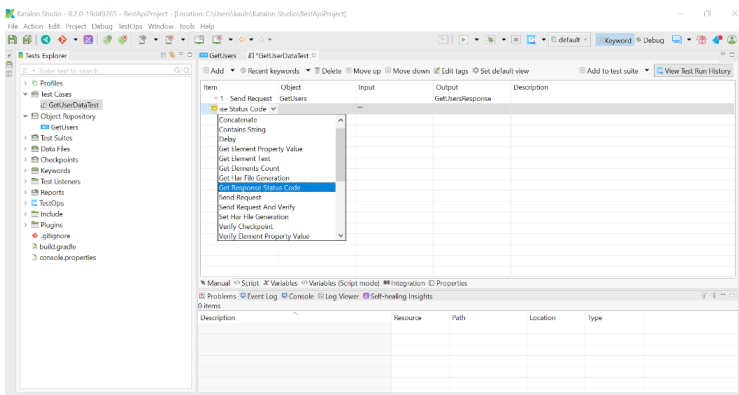
Additionally, we need to save the output of the request to a variable by setting the 'Output' field. This variable will be used in next two keywords of the test case. We edit the entry to add a variable name to the 'Output' field as shown below:



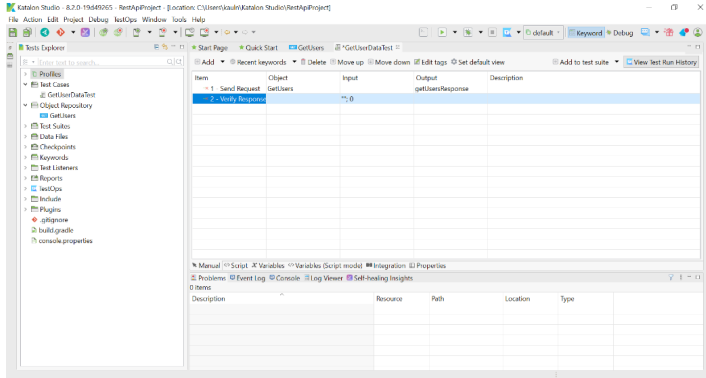
- b. Verify Response Status Code:** This keyword is used to verify response status code received from a call to a web service API ([WS] Verify Response Status Code, 2022).

To add this keyword, we click on the 'Add' button and select 'Web Service Keywords' as done in the previous step.

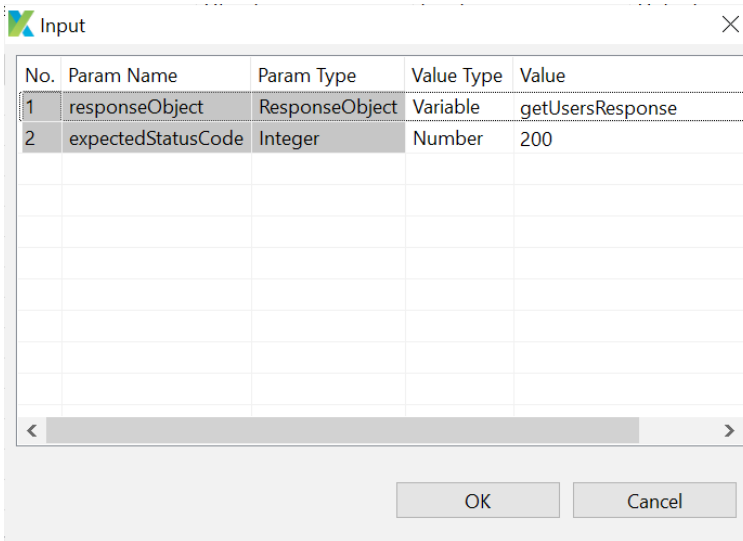
We select the option 'Verify Response Status Code' from the dropdown:



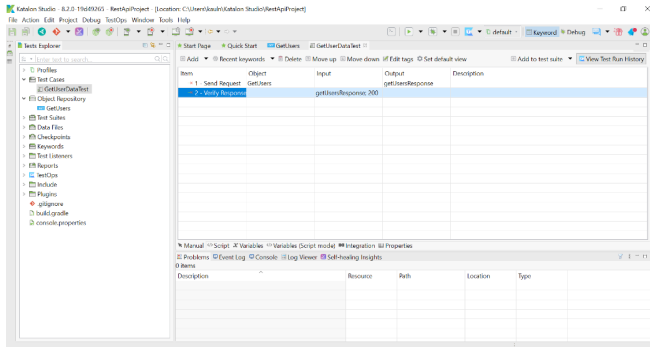
A second entry is added to the test case as follows:



We now edit the input column to choose the input parameter 'responseObject' which is the 'getUsersResponse' variable defined in the previous keyword/item and the parameter 'expectedStatusCode' which we set to 200 (OK).

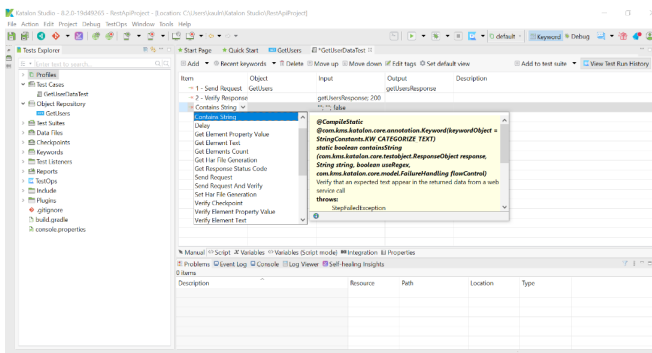


Once these values are set, we click on 'OK.' The changes are reflected as shown below:

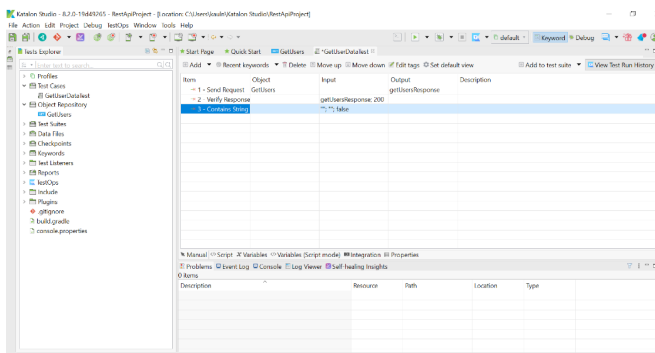


- c. **Contains String:** This keyword as its name implies, verifies that a particular text appears in the response received from a call to a web service API ([WS] Contains String, 2022).

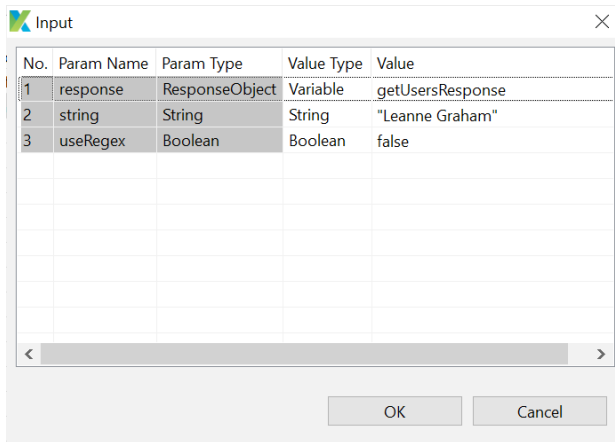
To add this keyword, we follow the same process as before: Add → Web Service Keywords and then selecting the option ‘Contains String’ from the dropdown list.



An entry is created as follows:

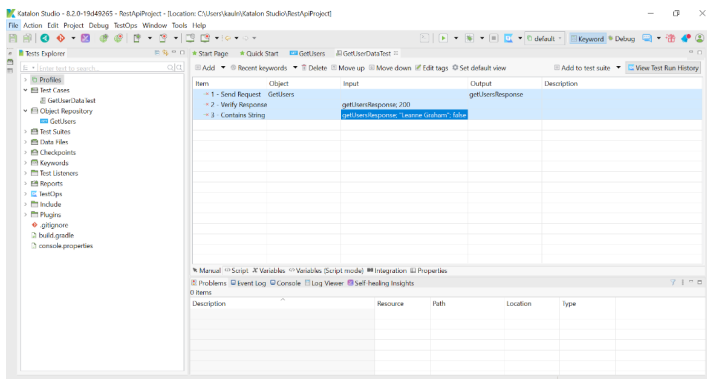


We edit the input column to add the string value that we wish to search for as follows:

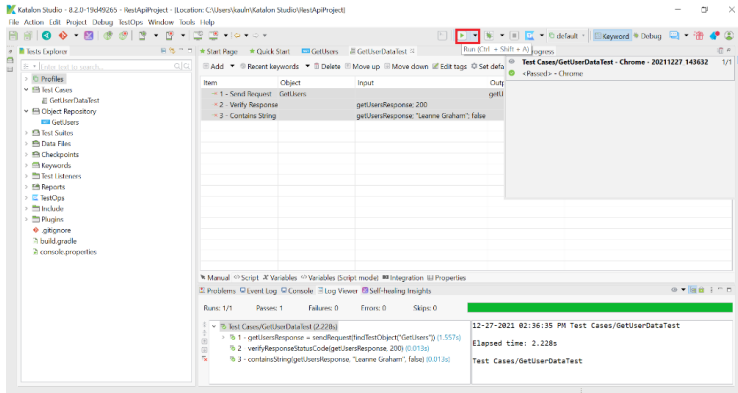


In the input parameters, first we set the ResponseObject parameter to 'getUsersResponse' which is the variable we created in step a. Then, we set the value of the string that will be searched as part of the 'Contains String' keyword. Here we set the value to be 'Leanne Graham' which is the first entry from API response 'getUsers.' Lastly, we set the 'useRegex' parameter which indicates whether the given text is a regular expression. We set this value to false and click on 'OK.'

Once done, we click on the save icon at the top left of the screen to save all the changes. The changes are reflected in the third entry as shown below:



- The last step is to execute the test case. To execute this test case, we simply click on the play button/icon at the top.



As observed from the image above, the test case is executed with success. Details of the test case including the different keywords executed, time elapsed, time taken by each keyword for execution, etc., can be found in the Log Viewer tab at the bottom of the screen.

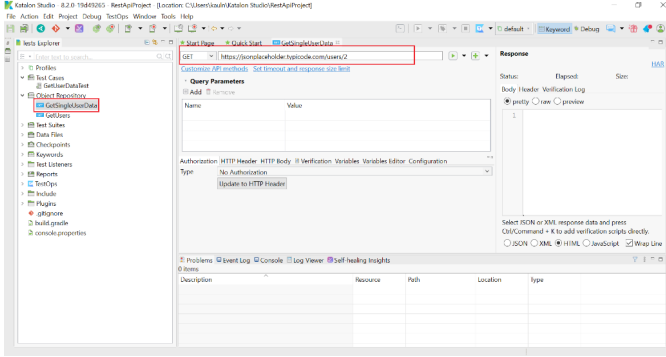
In the next example, we will create additional test cases followed by a test suite.

- **Example 2: Creating a Test Suite:** A test suite is a collection of test cases. This can be done by adding multiple test cases and then executing the test suite in a single go.

In this example, we will learn how to create and execute test suites. First, we will create another test case and then create a test suite that executes all the test cases in the test suite.

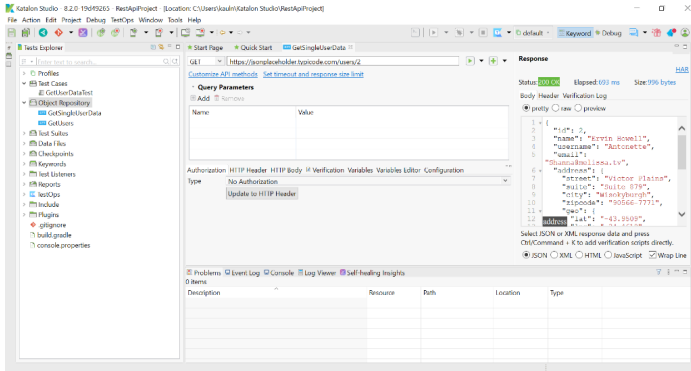
The steps to be followed are:

1. In the current project, we first create a new API request that get the details of a particular user by passing the user id in the REST API URL. Go to Object Repository → New → Web Service Request and create a new request.



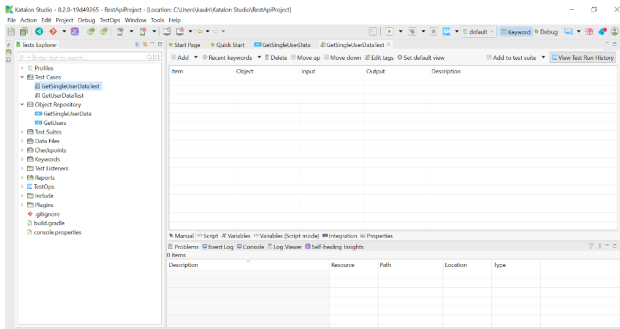
We provide the following URL to get data of the user with the id 2: <https://jsonplaceholder.typicode.com/users/2>.

- 2. To test this, we click on the play icon and execute the request.



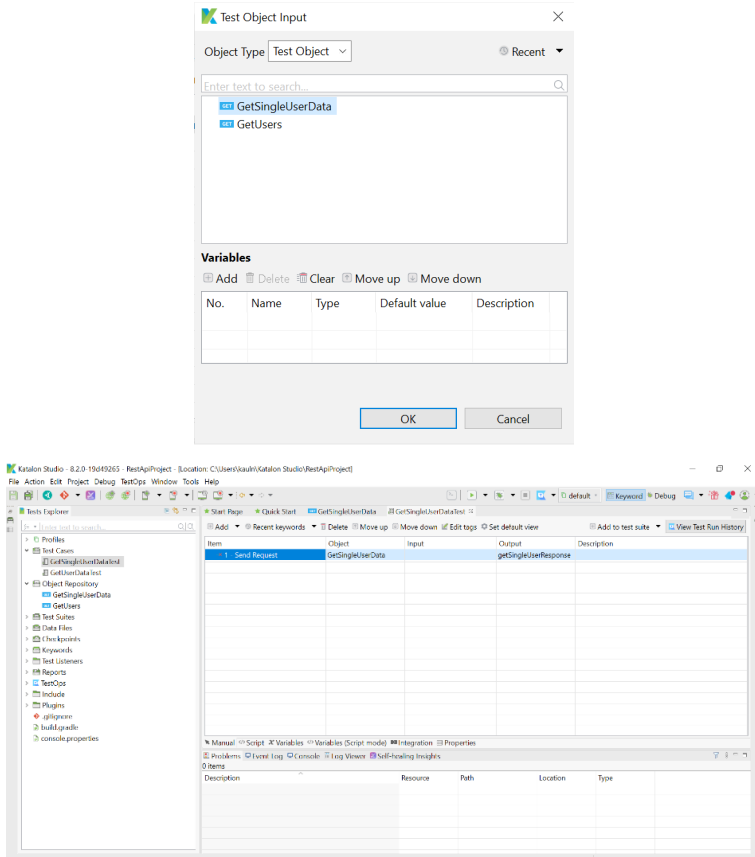
As we can see from the above screenshot, the response contains details of the user with the id 2.

- 3. We now proceed to create a test case for this API as follows by going to Test Cases → New → Test Case.

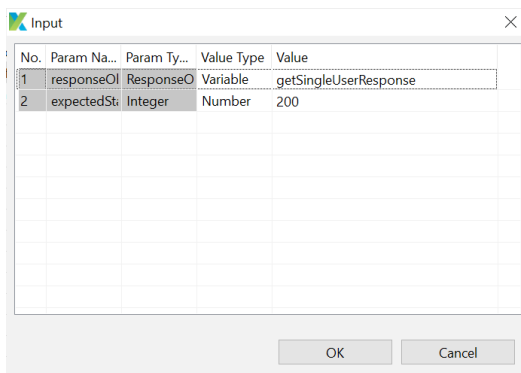


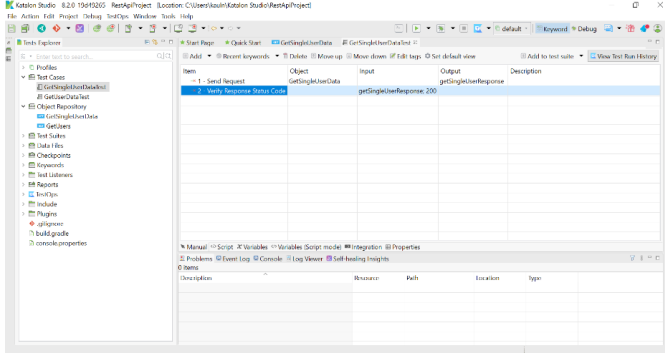
We proceed to add the following keywords to this test case to test the request: Send Request, Verify Response Status Code and Contains String.

a. Send Request:

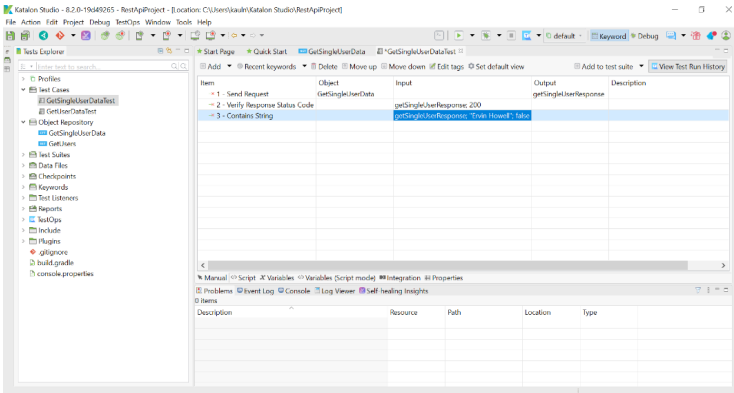
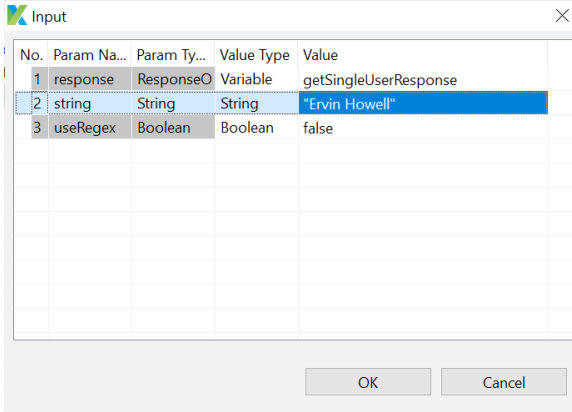


b. Verify Response Status Code:

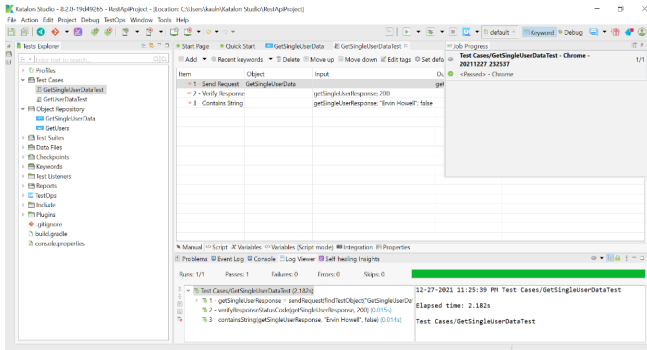




c. Contains String:

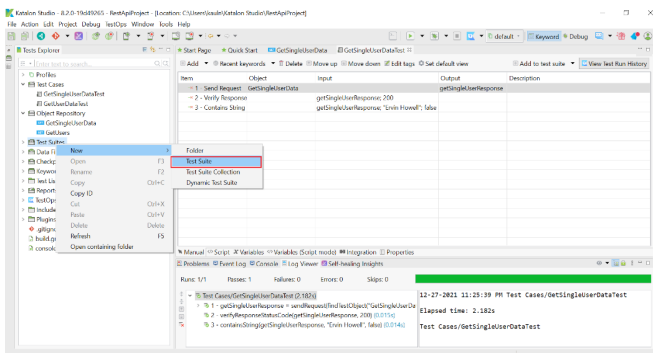


Now that the parameters of the test case have been added, we can execute the test case.

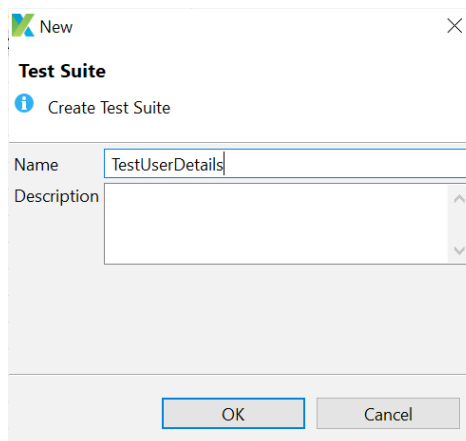


As we can see, the test case executes successfully.

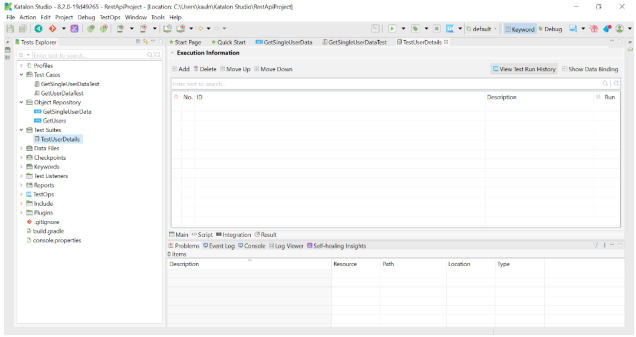
- Now that we have a couple of test cases, we will proceed to create a test suite in this step. To do this, we go to ‘Test Suites,’ right click on it and go to ‘New’ and click on ‘Test Suite’ as shown below:



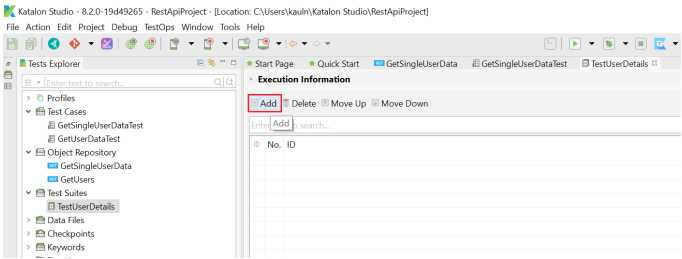
The following window pops up:



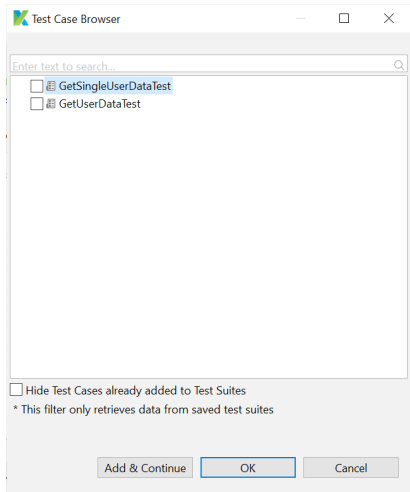
We provide the name of the suite and click on ‘OK.’ The test suite is created as shown below:



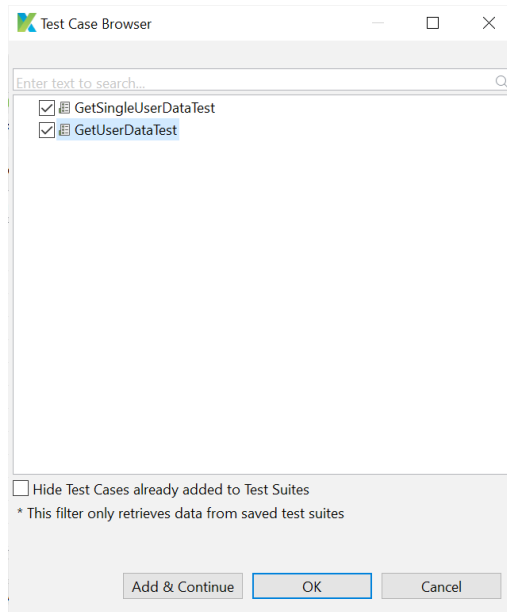
5. We now proceed to add the test cases to the test suite in this step. To add test cases to the suite, we simply click on the ‘Add’ button at the top:



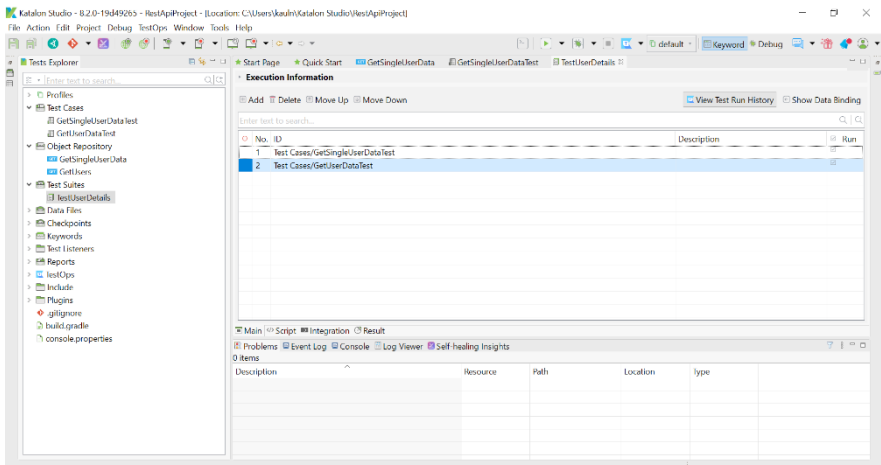
A window that prompts us to select test cases from the current project is displayed:



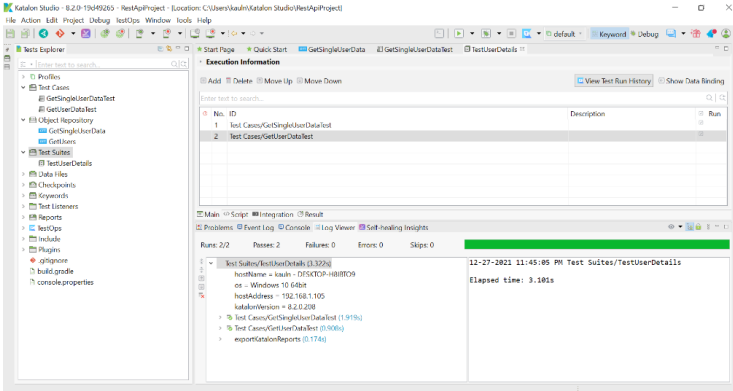
We select both the test cases and add them to the suite by clicking on 'OK.'



The test cases are added to test suite.



6. To execute the test suite, we click on the play icon at the top.



The test suite containing the two test cases is executed successfully.

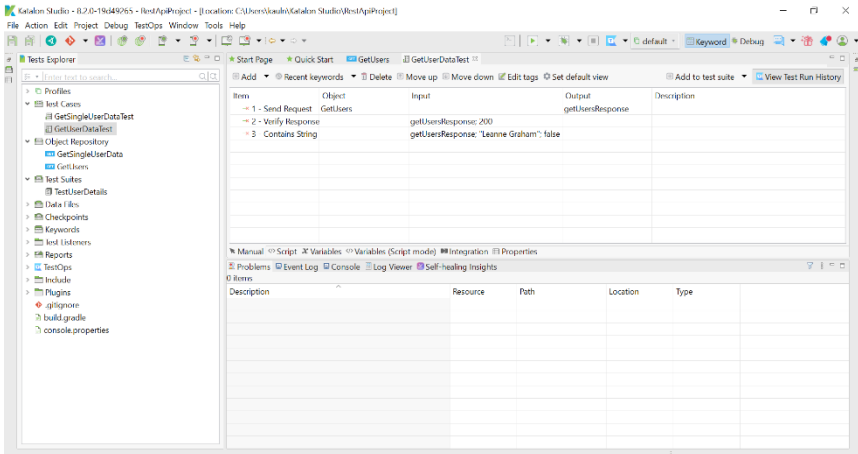
- **Example 3: Verifying the Response Count:** In this example, we will study the use of the ‘Verify Elements Count’ keyword.

This keyword helps verify the number of elements that were expected in the response of a web service request ([WS] Verify Elements Count, 2022).

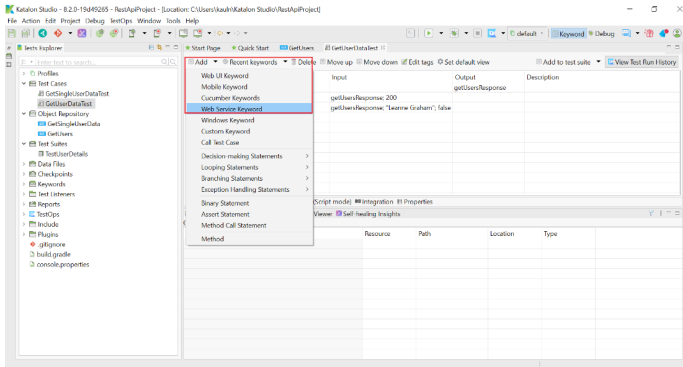
In this example, we will add this keyword to the `getUserDataTest` test case that we created in Example 1.

The steps are as follows:

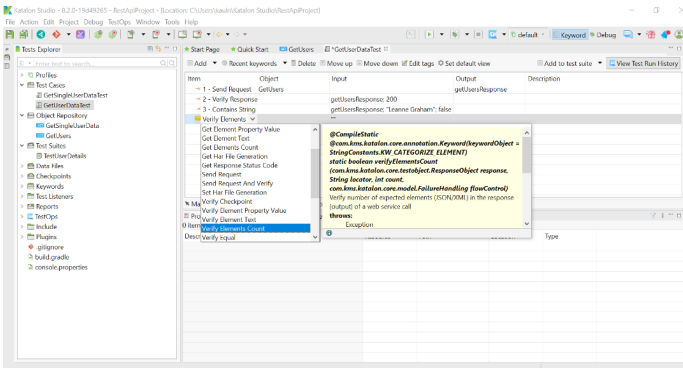
1. Open the ‘`getUserDataTest`’ test case in Katalon Studio.



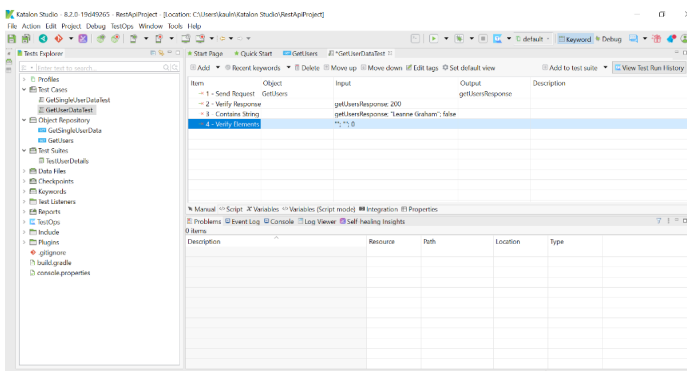
2. Click on ‘Add’ and select ‘New Web Service Keyword’ as shown in the image below:



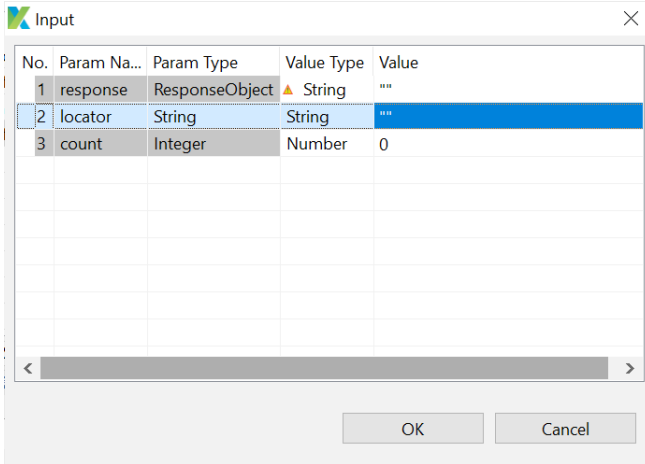
3. From the dropdown list, select the option 'Verify Elements Count'



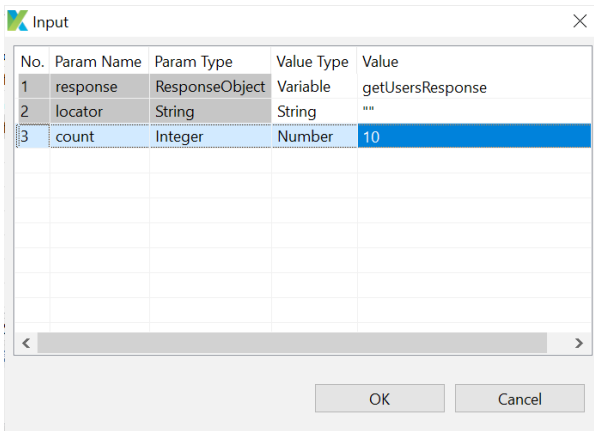
An entry is added to the test case as follows:



4. Double click on the input cell to open the Input dialog as shown below:

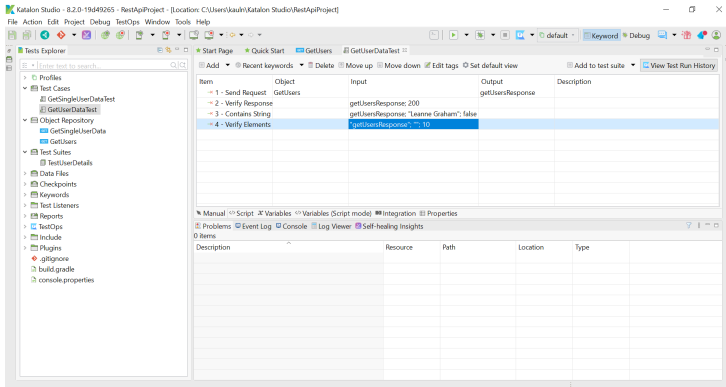


5. Edit this dialog to provide the response and count value as follows:

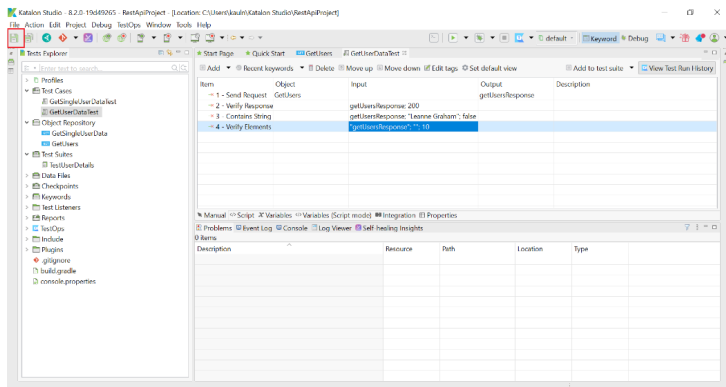


As you can see in the screenshot above, we have set the ResponseObject as 'getUsersResponse' which is the variable that captures the result of the request as defined in Example 1. As 'getUsersData' request returns 10 user records, we set the value for the 'count' parameter to 3. Once we enter this information, we click on 'OK.'

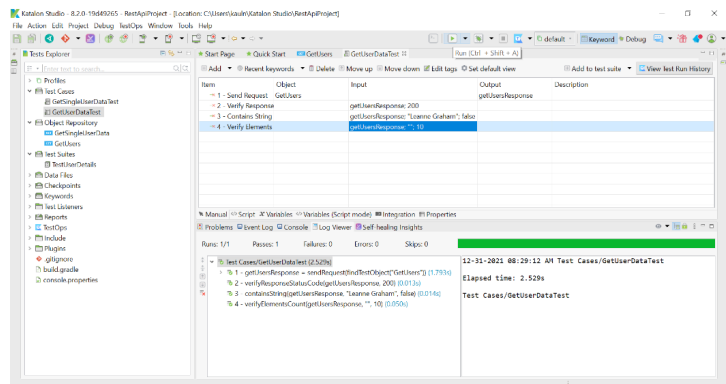
The changes are made to the entry as shown below:

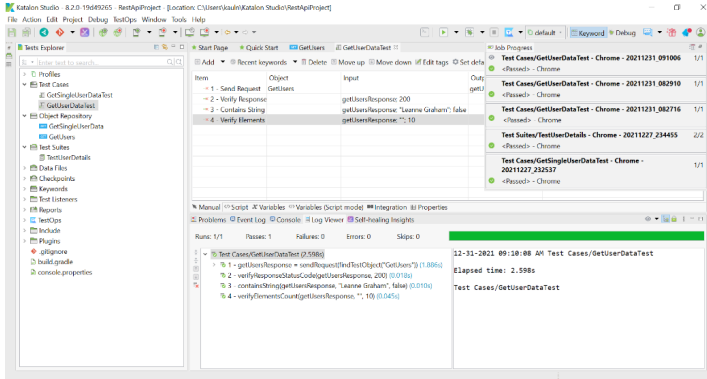


Click on the ‘Save’ button to save the changes to the test case.



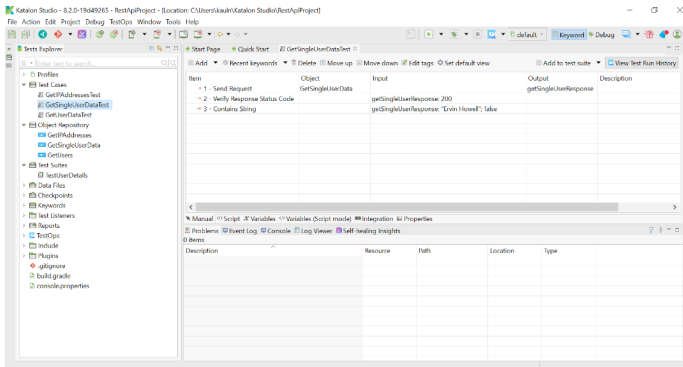
6. Test the newly updated test case by clicking on the play icon.



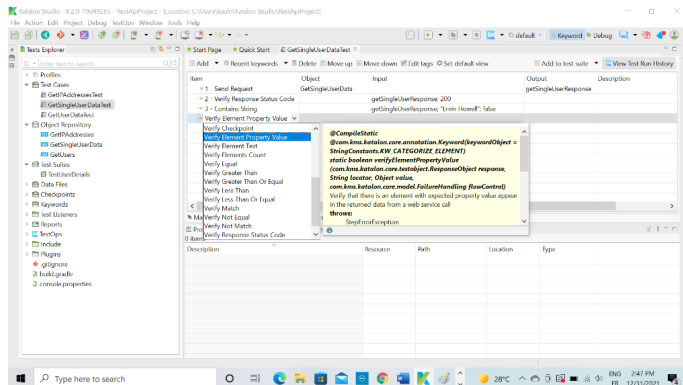


As we can see from the screenshot above the test case executes successfully.

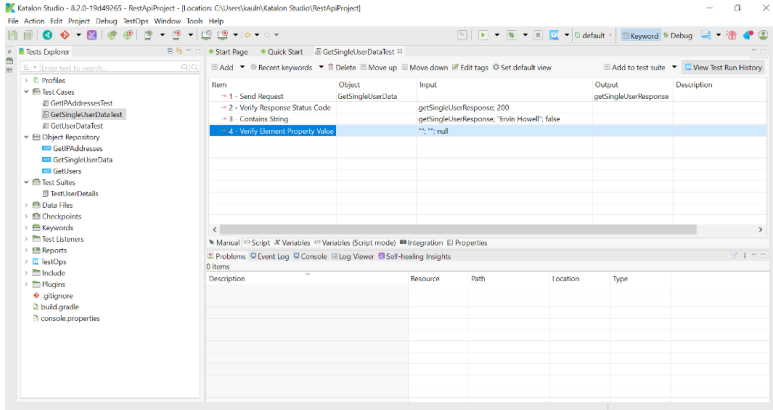
- Example 4: Verifying the Element Property Value:
 1. Open the 'getSingleUserData' test case in Katalon Studio.



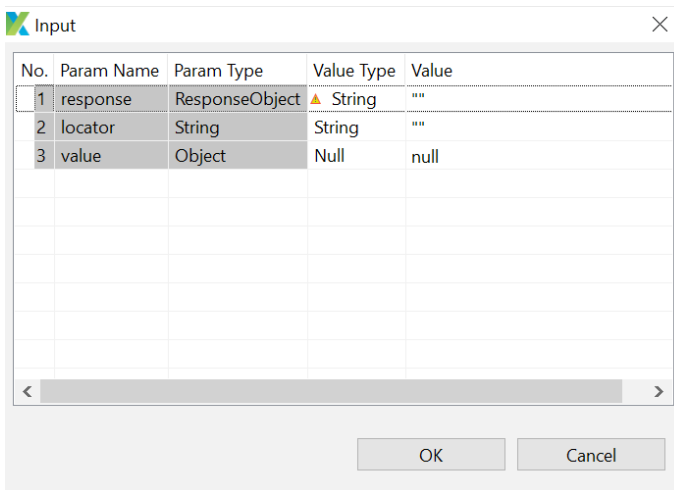
2. Click on 'Add' and select 'Web Service Keyword.' From the dropdown list, select 'Get Element Property Value.'



An entry is added to the test case keywords list as shown below:



- We click on the input cell to display the Input Window as follows:

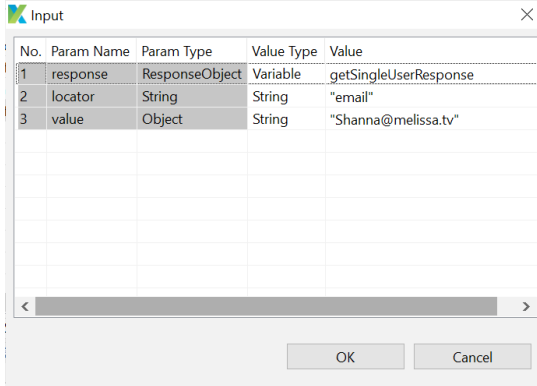


- We are prompted to enter the parameters response object and the locator field. Here, the response object is the variable 'getSingleUserResponse' that was created in Example 2 when we created the 'getSingleUserDataTest' test case. The locator parameter is the used to specify the location of the JSON property that we wish to locate in the returned data. Here, we provide the value 'email' as this is the JSON path (location of the email field in the response body).

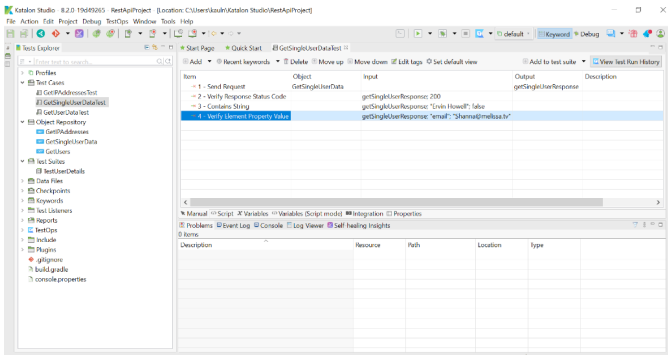
In the test case 'getSingleUserDataTest', we get the details of the user with the id 2. In this example, we will check if the element has the correct value for the property 'email', which should be 'Shanna@melissa.tv.' To do

this, we enter the value ‘Shanna@melissa.tv’ in the last parameter, which is the value parameter.

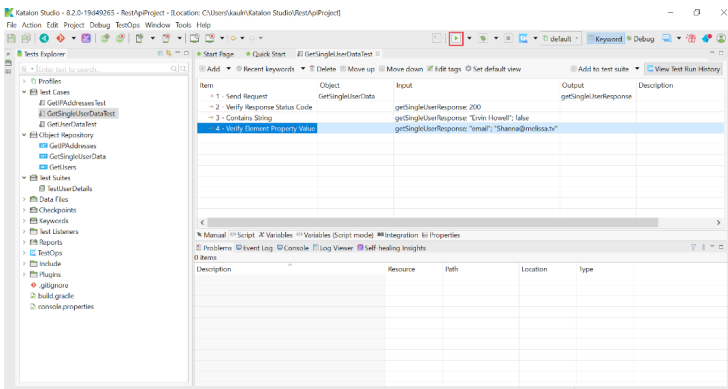
The Input has now been updated as follows:



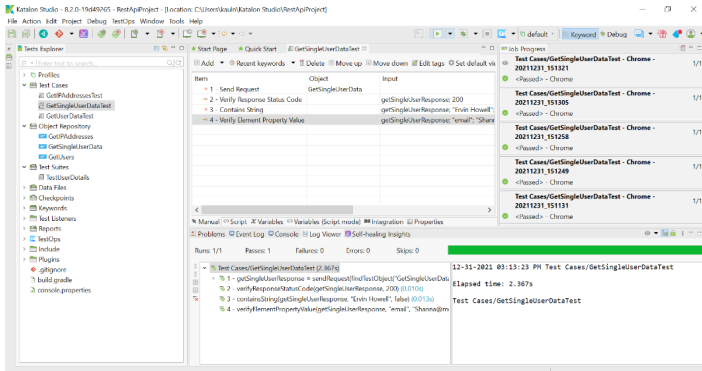
The entry in the test case is updated as shown below:



- The last step is the execution of the test case which can be launched by clicking on the play icon at the top.



The test case is executed correctly, as visible in the screenshot below:

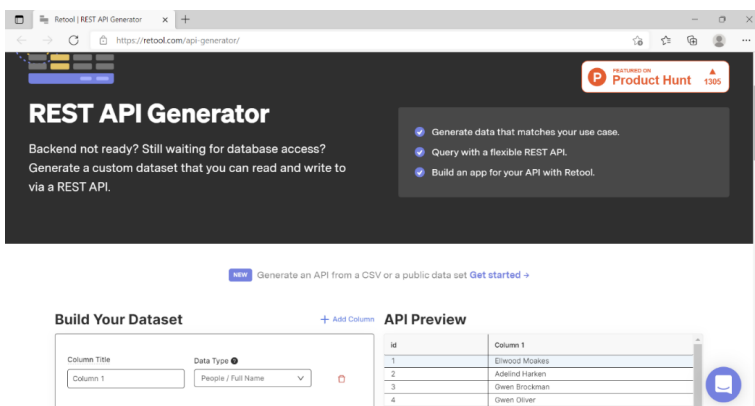


- Example 5: Verifying the Element Property Value with a User-Defined Web Service:** In this example, we will look at the keyword ‘Element Property Value’ keyword. This keyword helps check whether an element in the web service’s response object has the expected value for an element ([WS] Verify Element Property Value, 2022).

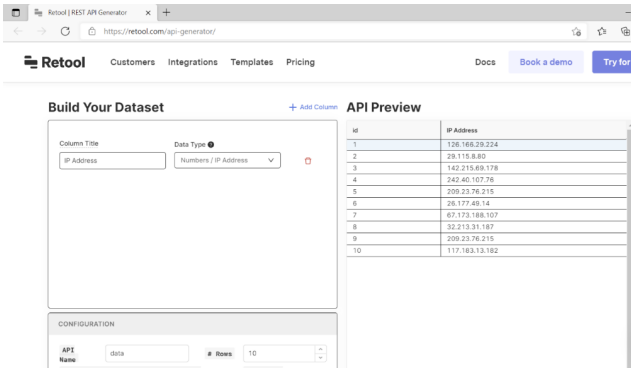
In this example, we will define a new web service request and add this keyword to test the property field in the response of this request.

The steps followed are:

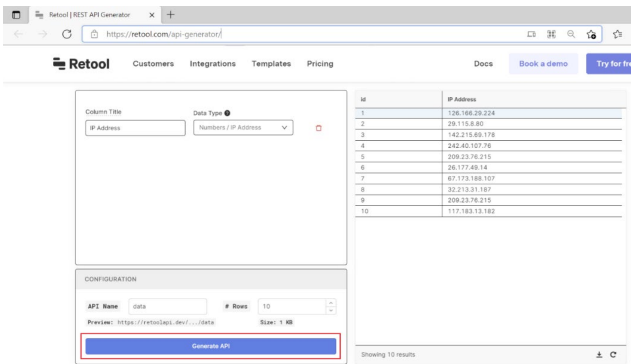
- First, we will use an online tool to generate a Rest API of our choosing. In this example, we will use the tool Retool-REST API Generator(<https://retool.com/api-generator/>) to create an API of our own. We navigate to the site.



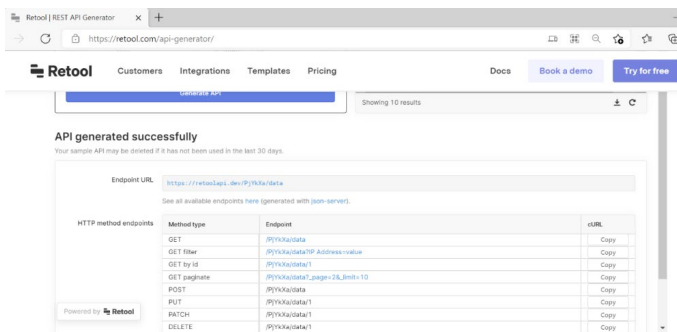
- We build a dataset of our choosing. For this example, we have created a list of IP Addresses as follows:



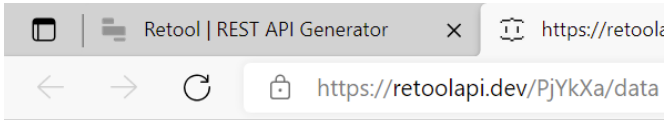
3. We click on the ‘Generate API’ button in the ‘Configuration’ section to generate the API.



The API is generated successfully. You can find the definition in lower half of the page as shown below:

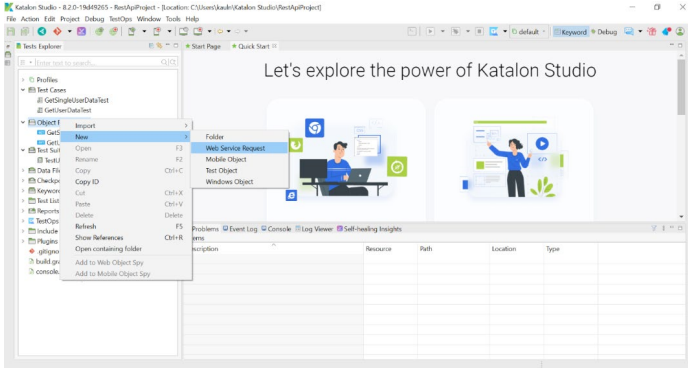


4. We click on the endpoint URL to get the data. It is shown below:

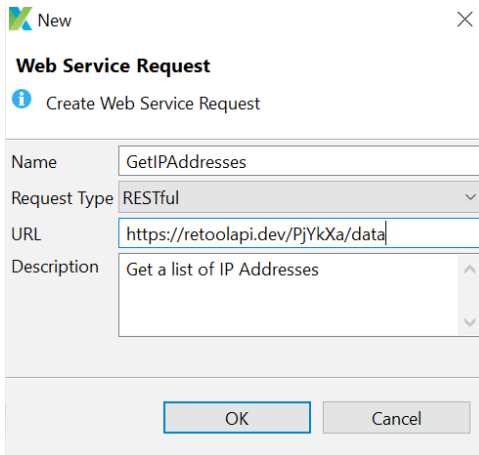


```
[
  {
    "id": 1,
    "IP Address": "126.166.29.224"
  },
  {
    "id": 2,
    "IP Address": "29.115.8.80"
  },
  {
    "id": 3,
    "IP Address": "142.215.69.178"
  },
  {
    "id": 4,
    "IP Address": "242.40.107.76"
  },
  {
    "id": 5,
    "IP Address": "209.23.76.215"
  },
  {
    "id": 6,
    "IP Address": "26.177.49.14"
  },
  {
    "id": 7,
    "IP Address": "67.173.188.107"
  },
  {
    "id": 8,
    "IP Address": "32.213.31.187"
  },
  {
    "id": 9,
    "IP Address": "209.23.76.215"
  },
  {
    "id": 10,
    "IP Address": "117.183.13.182"
  }
]
```

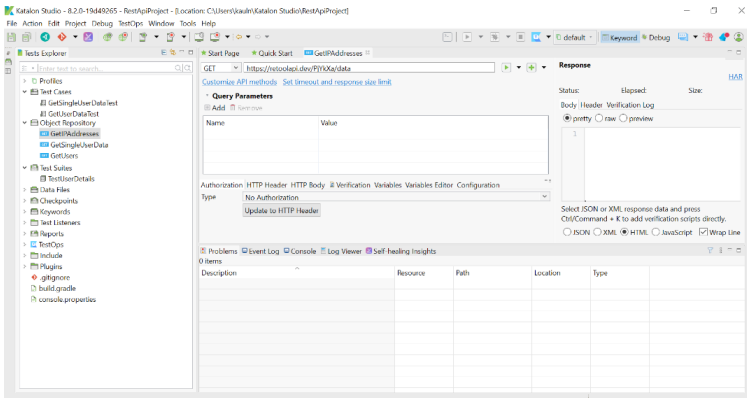
5. Now that the API has been created, we proceed to create a new Web Service Request in Katalon Studio. We right-click on 'Object Repository,' go to New → Web Service Request.



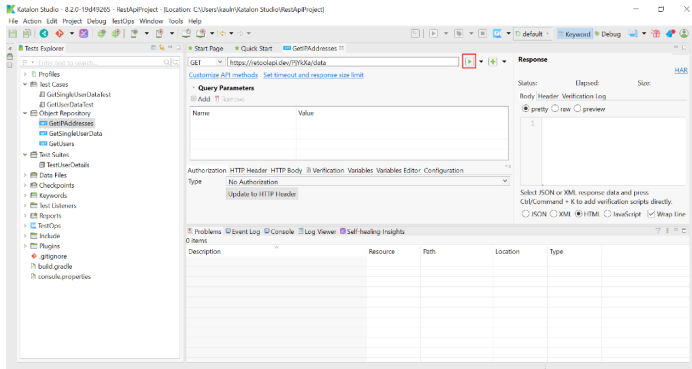
We then enter the name, description, and URL of this request as follows:



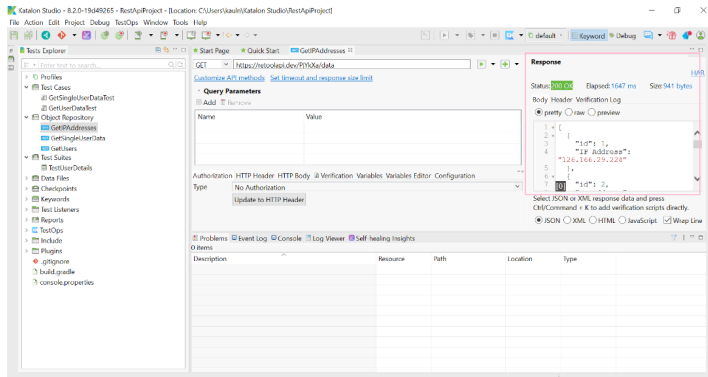
The Rest API request is created successfully as shown below:



6. In this step, we run the request to see if it returns the data by clicking on the play icon located at the end of the request bar.

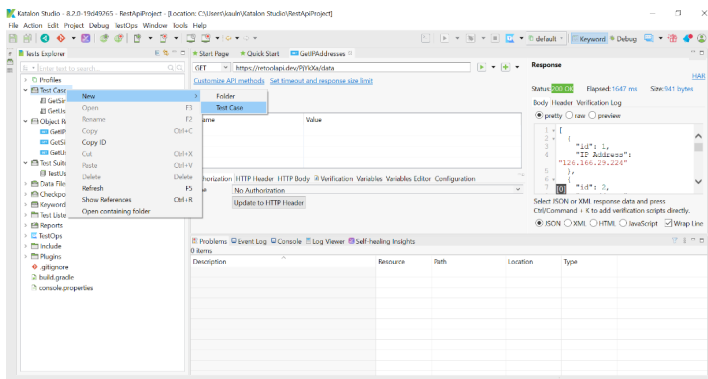


The results are displayed in the response tab as shown as follows:

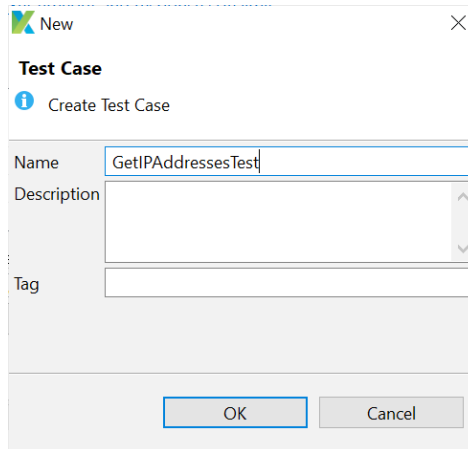


We can verify that a list of 10 IP addresses is present in the response.

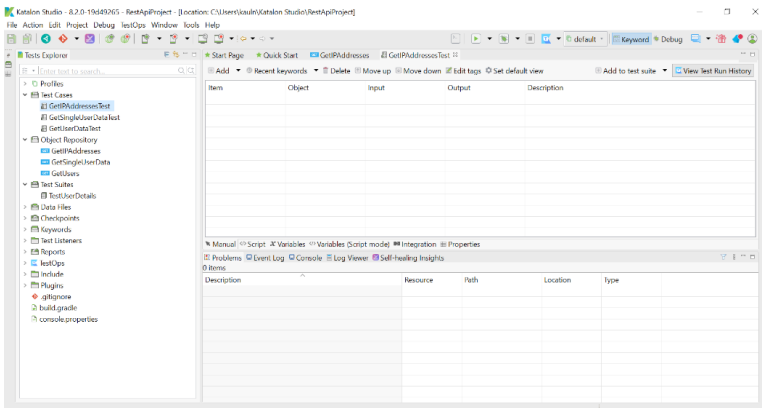
7. We now proceed to create a test case to test this web service. We go to 'Test Cases,' right-click 'Test Cases', then go to New → Test Case.



We provide the test case name and click on ‘OK’ to create the test case as follows:



The test case is added to the list of test cases as displayed below:

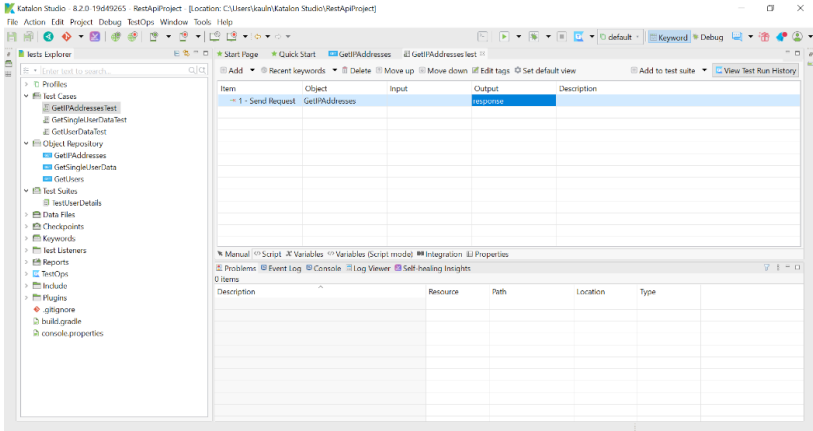


8. In this step, we proceed to adding keywords to the test case. Note that to use the ‘Verify Element Property Value’ variable ([WS] Verify Element Property Value, 2022), we first need to get the response of the API and store in a variable using the ‘Send Request’ keyword ([WS] Send Request, 2022). This variable will be used to access the results when defining the ‘Verify Element Property Value’ keyword.

We add the keywords as follows:

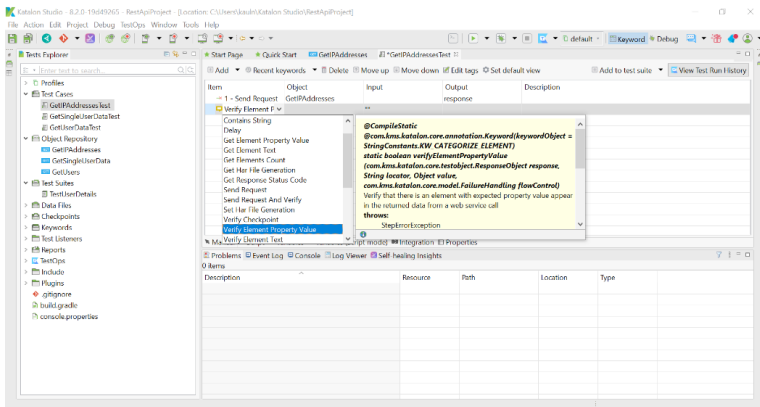
- a. Send Request:

We add a variable to the ‘Output’ column, which stores the response returned by request.

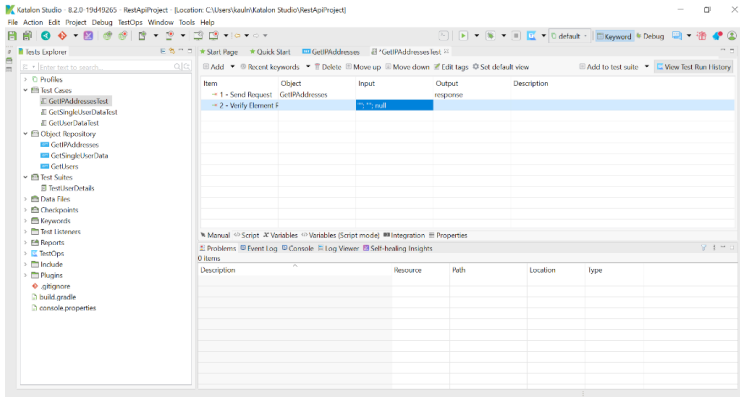


b. Verify Element Property Value: We proceed to add the keyword ‘Verify Element Property Value’ to the test case.

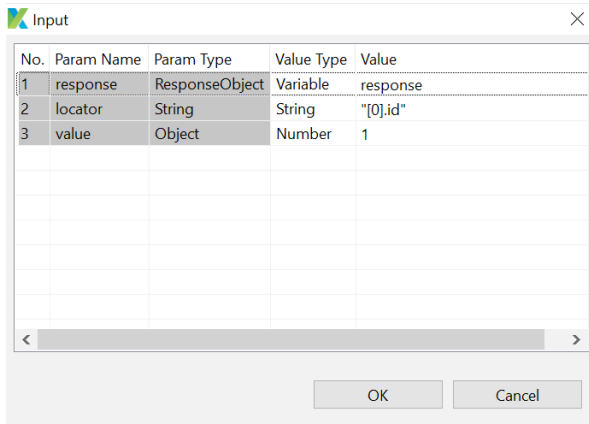
To do this, we add another web service keyword using the ‘Add’ button and select the keyword ‘Verify Element Property Value’ from the dropdown list.



A second entry is added to the test case.



We edit the input column to search for a particular value from the response:

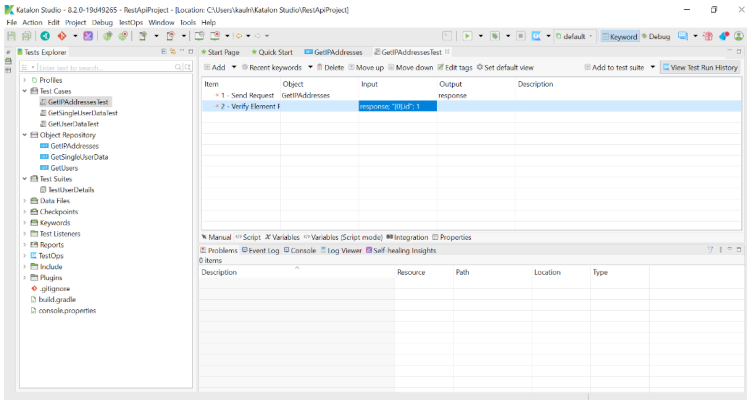


Here, the first parameter we enter is the response parameter. We set it to response which is the output of the API request.

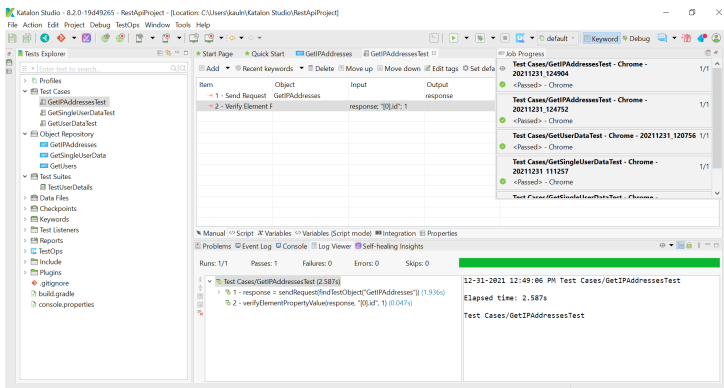
The next parameter we enter is the locator parameter which is the location of the element that we want to check. Here, based on the test data that we created, we want to check that the value of the first element's id property is 1, so, the value we enter for the locator parameter is [0].id. The value [0].id tells Katalon Studio to locate the first record from the data and find the property called id within this record.

The last parameter we enter is the expected value of the field/property specified in the locator. Here, we are checking the first record where the value of the 'id' field is 1. We click on 'OK' to update the input.

The input is updated, and we then save the test case by clicking on the 'Save' icon located at the top left section of the window.



9. We execute the test case by clicking on the play icon at the top.



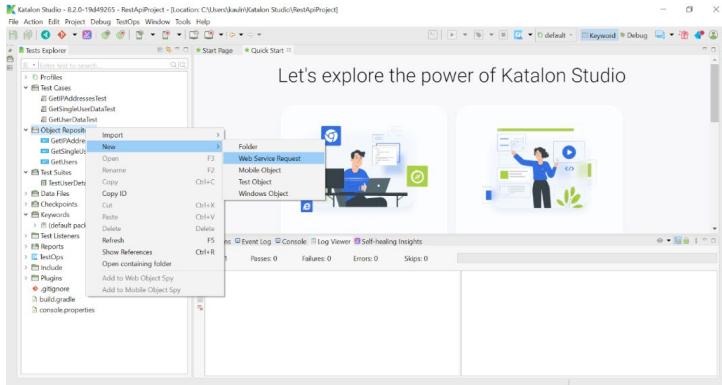
The test case passes successfully as shown above.

- Example 6: Testing a SOAP Web Service Request:** In the previous examples we studied different ways in which REST API requests can be tested. In this example, we will cover the creation of a SOAP web service request followed by creation of test cases for a SOAP request using Katalon Studio.

The steps are:

- First, we create the SOAP request. This is done in the same manner that we created a REST request.

To create a SOAP request, we go to Object Repository → New → Web Service Request.



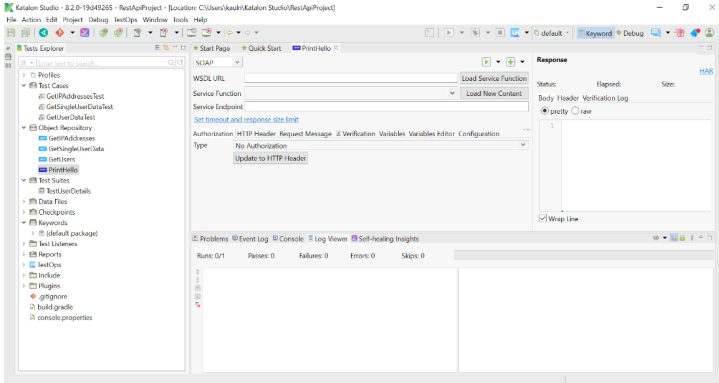
The following window pops up and prompts us to provide details of the SOAP request.

A dialog box titled "New" with a close button (X) in the top right corner. The main title is "Web Service Request". Below the title is an information icon and the text "Create Web Service Request". The dialog contains four input fields: "Name" with the value "PrintHello", "Request Type" with a dropdown menu showing "SOAP", "URL" with the value "URL", and "Description" which is an empty text area. At the bottom, there are two buttons: "OK" and "Cancel".

At this point, we provide just the name of the web service request. We need to select the Request Type which is 'SOAP.' The URL can be left blank. The description is optional too.

We click on 'OK' to create the request.

The request is created as shown below:



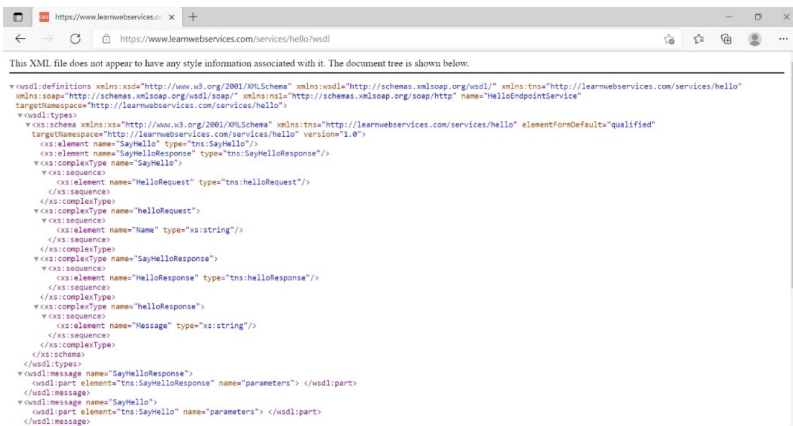
2. We proceed to use a freely available WSDL that prints ‘Hello <name>’ where name is a value entered in the request body.

In this example, we use the following site: Learn web services (<https://www.learnwebservices.com/>).

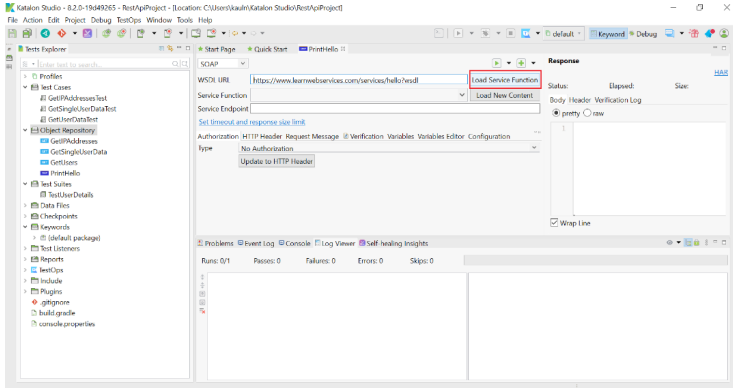
The WSDL provided is as follows:

<http://www.learnwebservices.com/services/hello?WSDL>

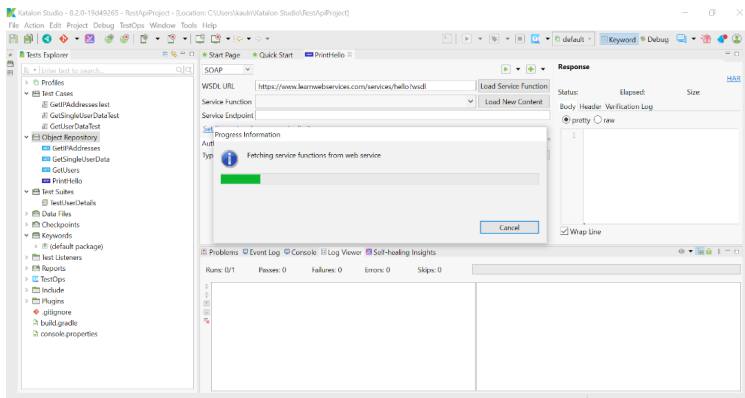
We access this URL in the browser to see the WSDL definition.



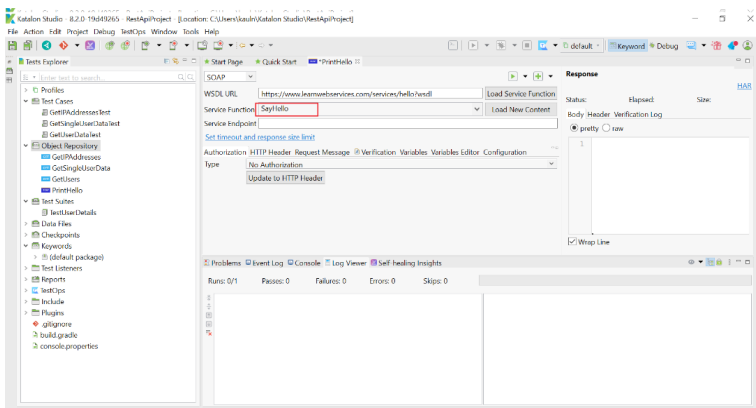
3. In Katalon Studio, in the SOAP request we just created, we provide the WSDL URL and click on the ‘Load Service Function’ to load the functions provided by the SOAP request.



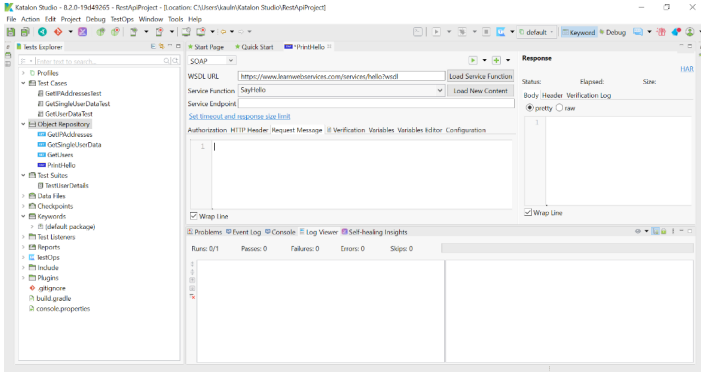
When we click on the button 'Load Service Function' the service functions are fetched from the web service.



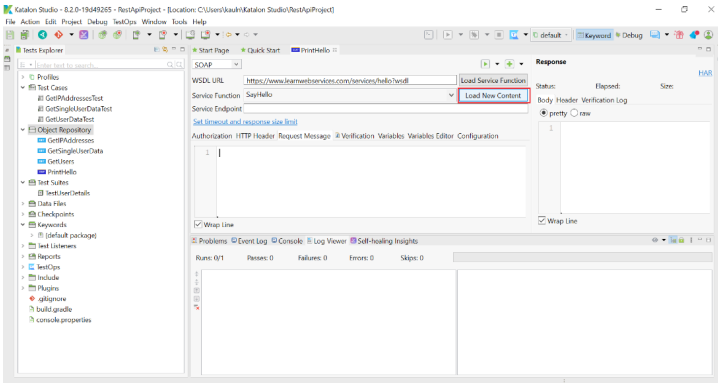
Once the functions have been fetched, we can see them in the 'Service Function' tab as shown below:



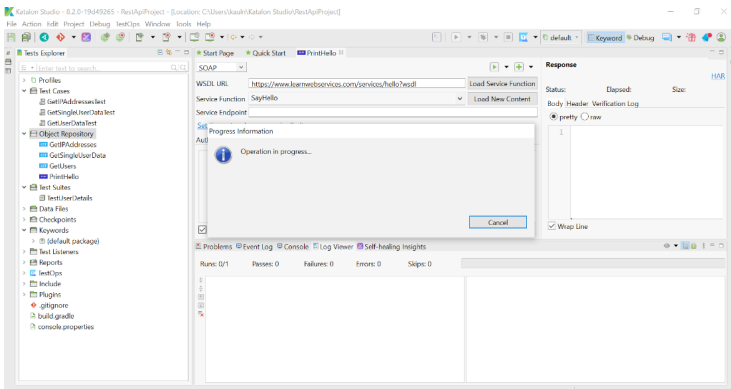
- 4. We now need to provide the body of the message to be able to execute the request. By default, it is empty as shown below:



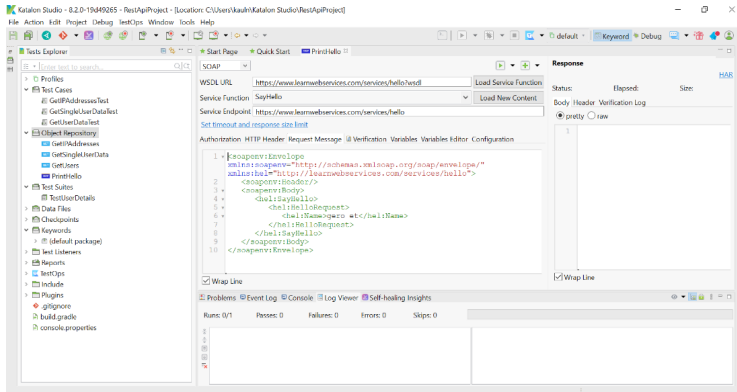
- 5. To load the request body, we click on ‘Load New Content.’



The Service function fetches the content of the WSDL request.



Once the operation is terminated, the Request message is populated with the request body as shown below:

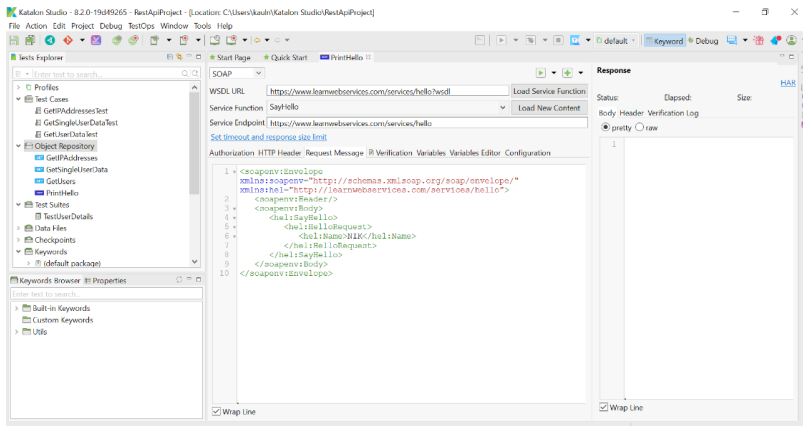


We have also provided the ‘Service Endpoint’ value that contains the available services list for the WSDL.

When we access the link <https://www.learnwebservices.com/services> we see the list of available services as displayed in the screenshot below:

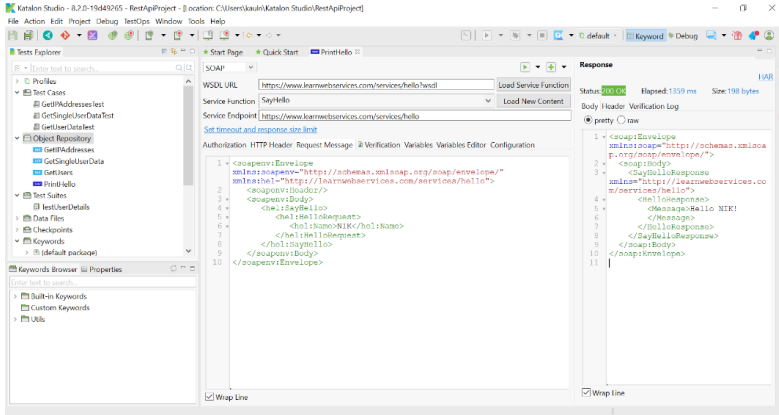
Available SOAP services:	
HelloEndpoint <ul style="list-style-type: none"> SayHello 	Endpoint address: https://www.learnwebservices.com/services/hello WSDL: http://learnwebservices.com/services/hello?HelloEndpointService Target namespace: http://learnwebservices.com/services/hello
TempConverterEndpoint <ul style="list-style-type: none"> CelsiusToFahrenheit FahrenheitToCelsius 	Endpoint address: https://www.learnwebservices.com/services/tempconverter WSDL: http://learnwebservices.com/services/tempconverter?TempConverterEndpointService Target namespace: http://learnwebservices.com/services/tempconverter

We edit the request body and change the name to ‘NIK’ as shown below:



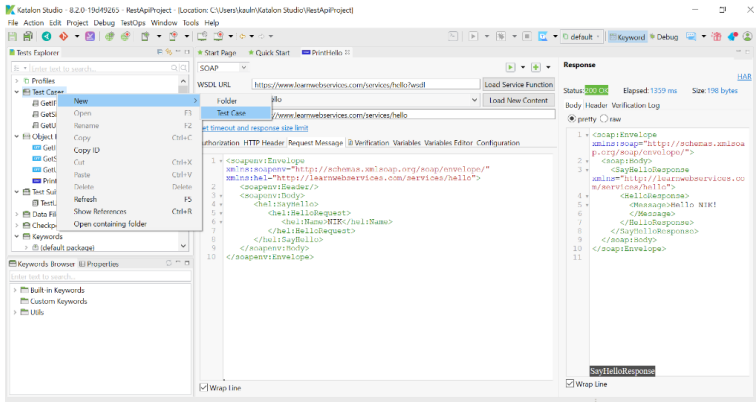
- We execute the SOAP request by clicking on the play icon at the top.

The results are available in the Response tab on the right. It is as follows:

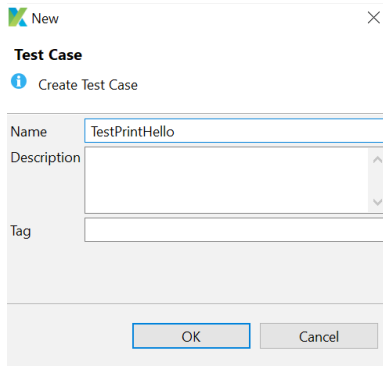


As visible from the screenshot above, we can see the response which contains the message 'Hello NIK!.'

- Now, we proceed to test this web service request. To create a test case, we go to Test Cases → New → Test Case.

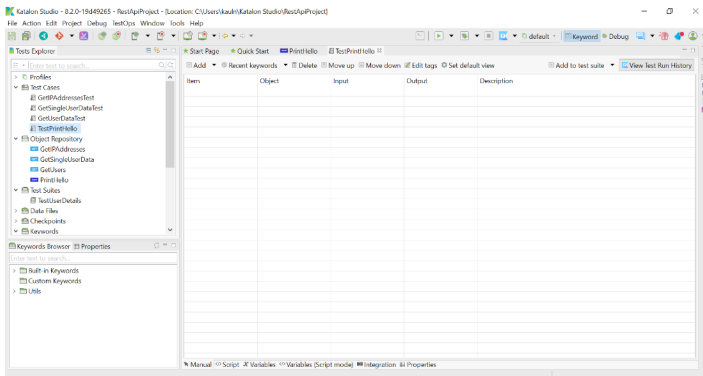


We are asked to provide the name, description, and tag. We provide the name and leave the other optional fields blank.

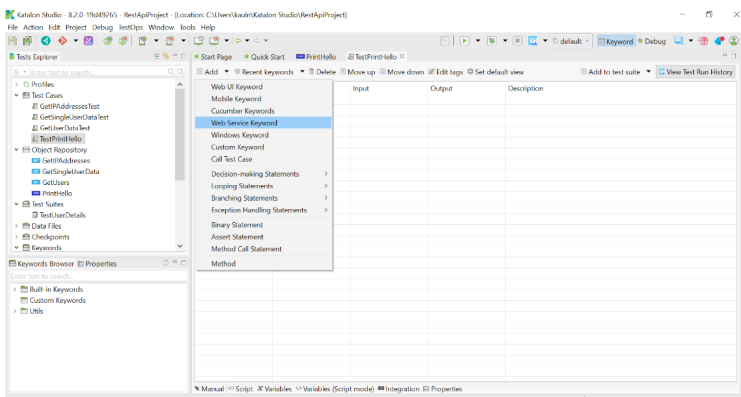


We click on 'OK' to proceed.

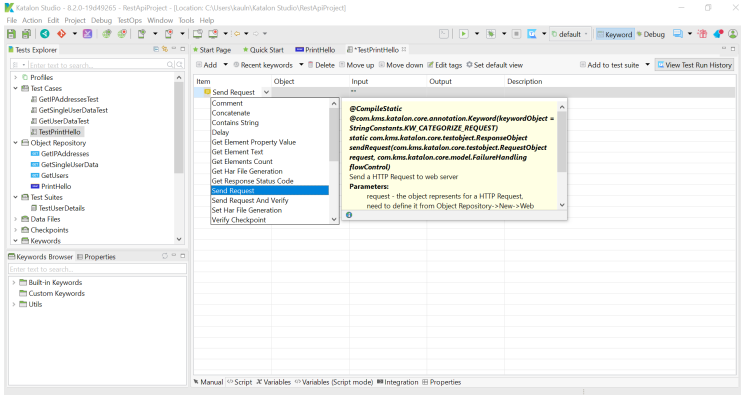
The test case is created as follows:



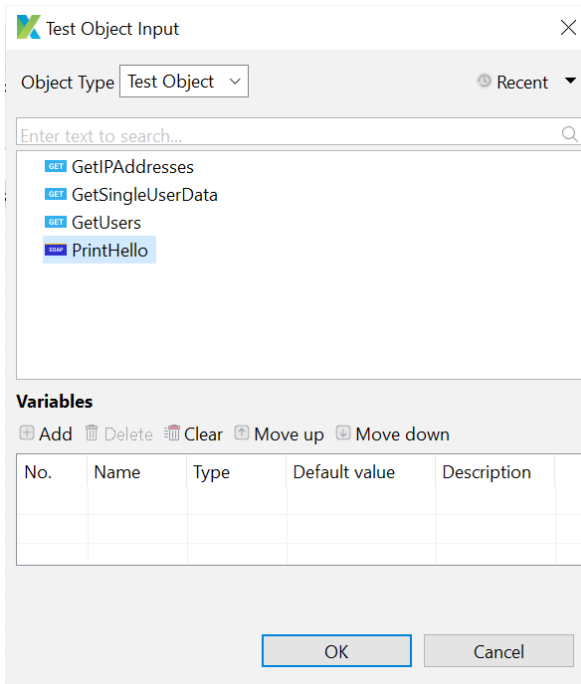
8. We proceed to add keywords to the test case as follows:



First, we need to add the 'Send request' keyword to call the web service request and store the response in a variable which will be used in the next keywords.

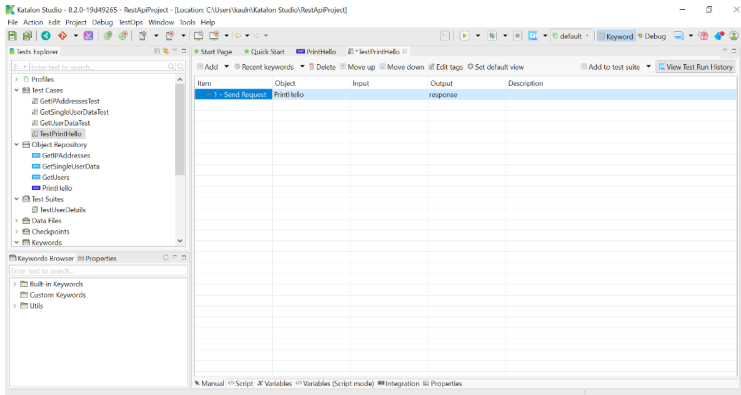


We select the 'PrintHello' object/web service request in the object Input column.

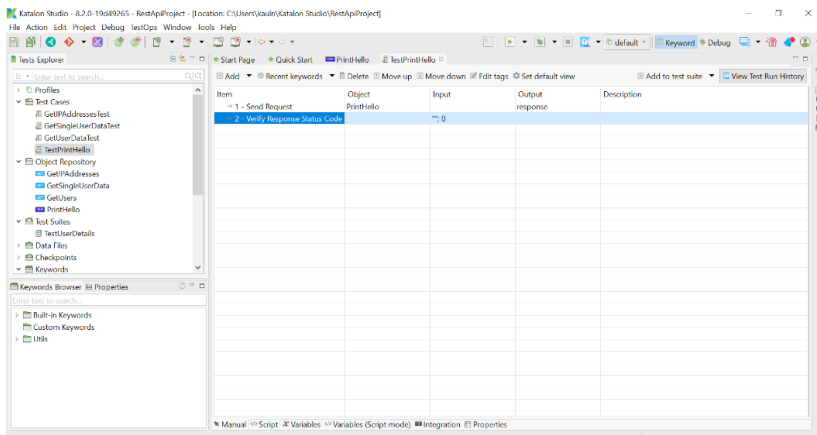


We click on 'OK' to select the PrintHello web service request.

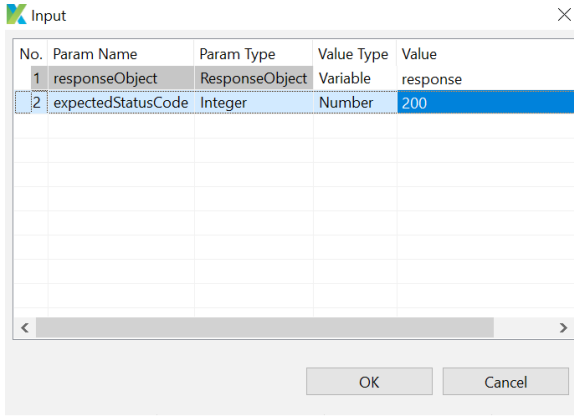
We also add a variable 'response' to the output field as shown in the screenshot below:



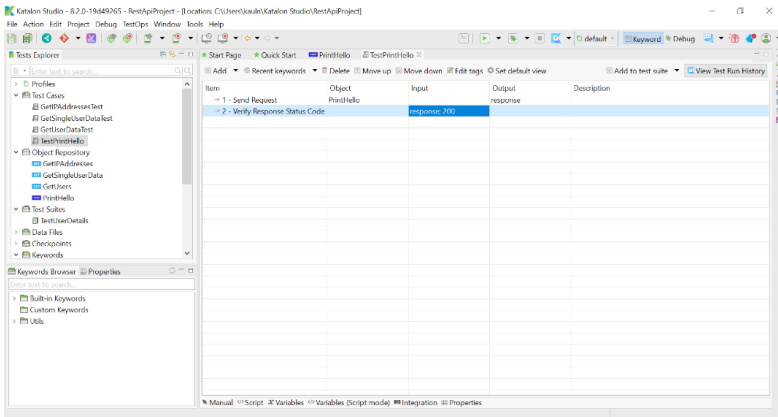
The next variable that we add is the ‘Verify Response Status Code’ to check whether the request returns a 200 (OK) request. The entry is created as follows:



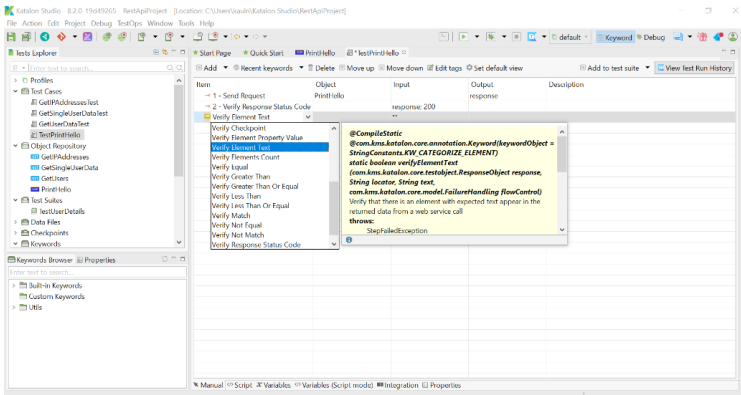
We edit the Input column and provide values for the parameters responseObject and expectedStatusCode. We select the ‘response’ variable that we created during creation of the ‘Send Request’ variable. For the expectedStatusCode we enter 200 as it is what is expected.



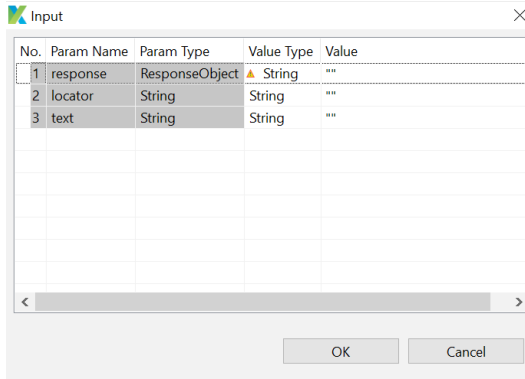
The entry is updated as shown below:



The next keyword that we add is the ‘Verify Element Text’ keyword that is used for checking the element’s text from the response returned by the web service.

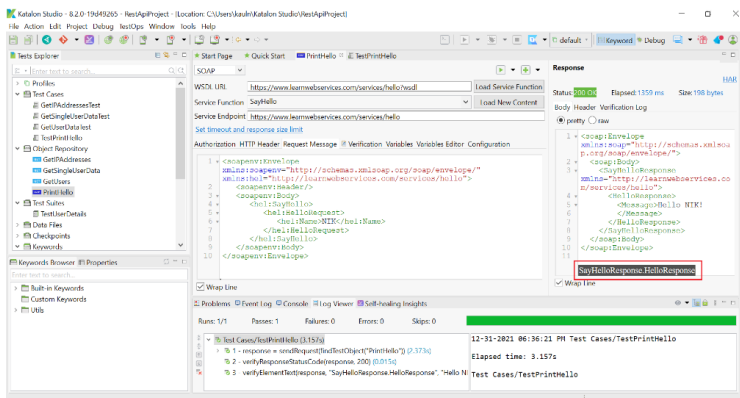


We now need to edit the input column.



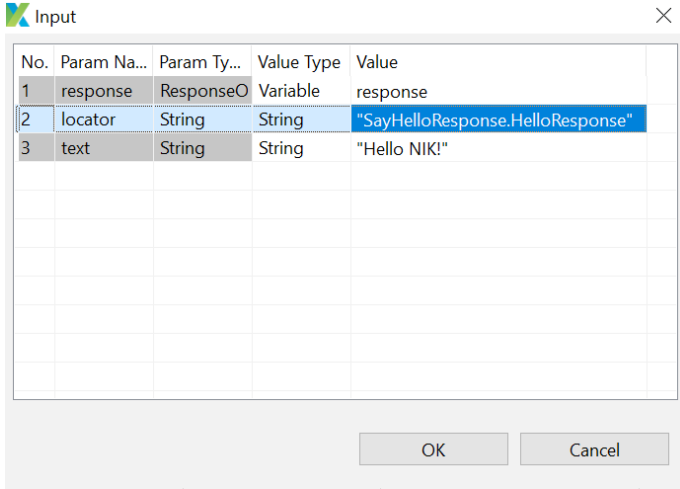
In the input, we provide the following values:

- ResponseObject: Response:** This value is the variable ‘response’ created in the first step.
- Locator:** This is the location of the element in the response of the web service request. This value for the field can be found in the soap request’s response as shown below:

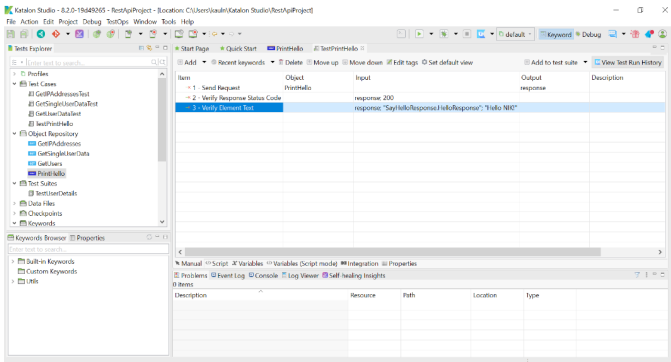


As visible from the screenshot above, the location of the field ‘message’ is SayHelloResponse.HelloResponse. We provide this value to the locator field.

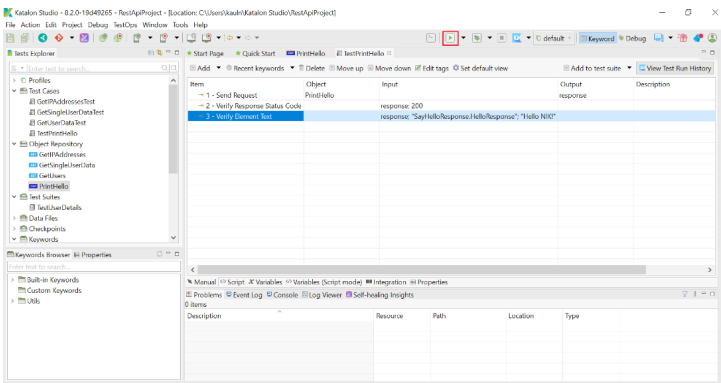
- Text:** This parameter is the value of the ‘message’ property which is ‘Hello NIK!’



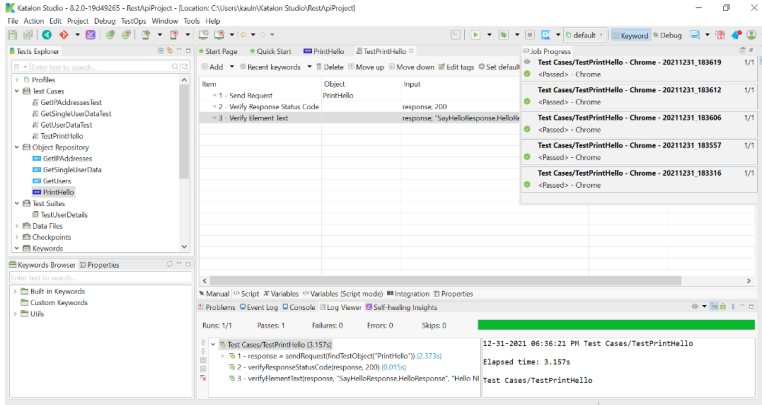
We click on 'OK' to add the changes which are reflected in the entry as shown below:



9. We run the test case by clicking on the play icon.



The test case is executed as shown below:

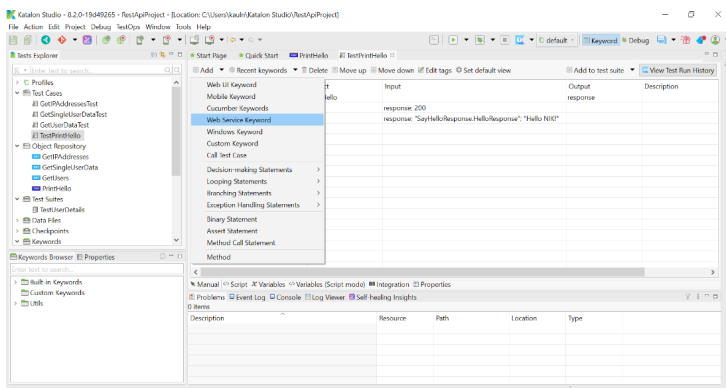


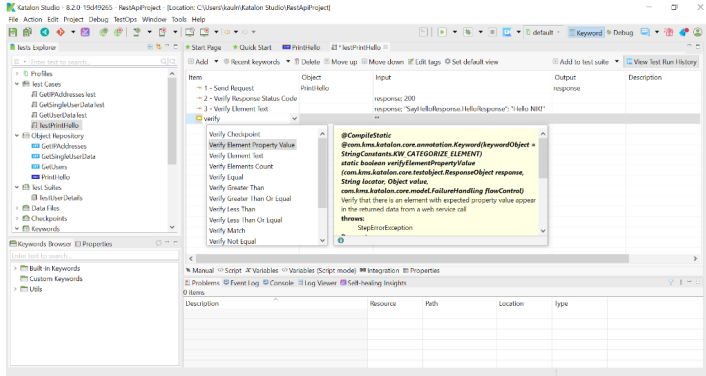
As seen from the results of the test case, the test case passed successfully.

10. In this test case, we can also use the ‘Verify Element Property Value’ to check the value of the message.

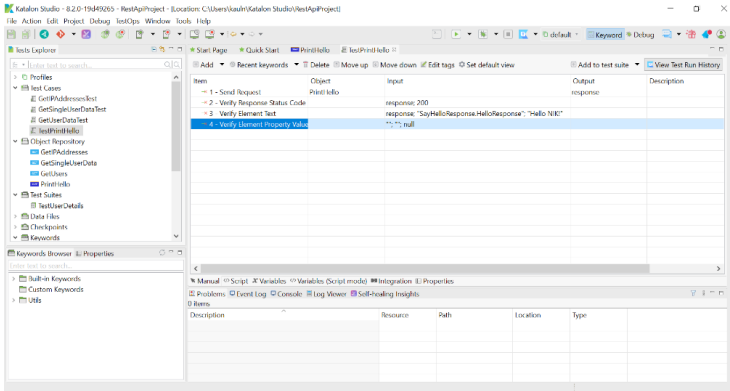
This is done as follows:

We click on ‘Add’ then select ‘Web Service Keyword.’ From the dropdown, we select the keyword ‘Verify Element Property Value.’

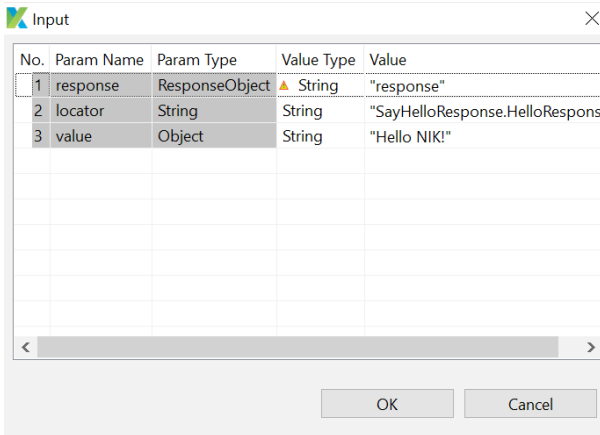




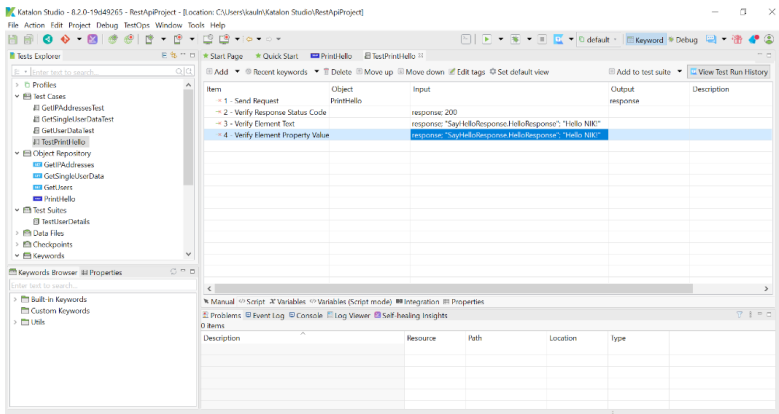
An entry is added to the test case as shown in the screenshot below:



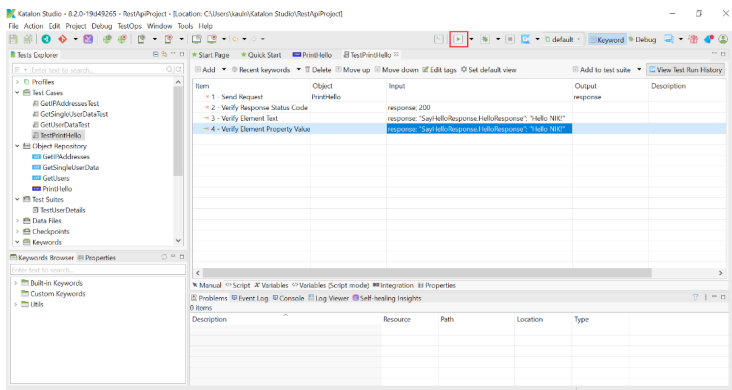
We edit the 'Input' column and provide the three parameters: request, locator, and value. These values are the same as done in Step 8.



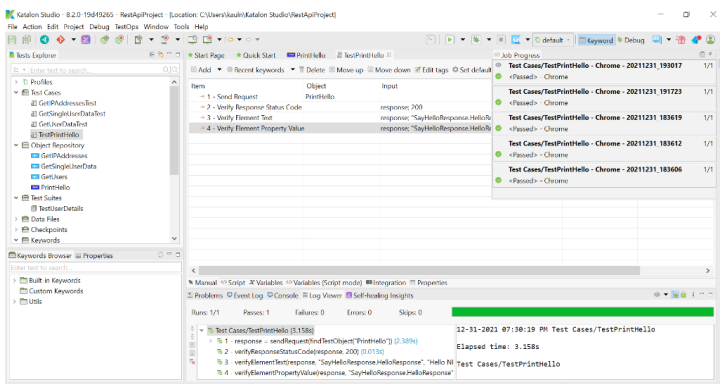
We click on 'OK' to register the changes to the input column. The changes are registered as shown below:



We click on the play icon to run the test case.



The test case executes successfully as follows:

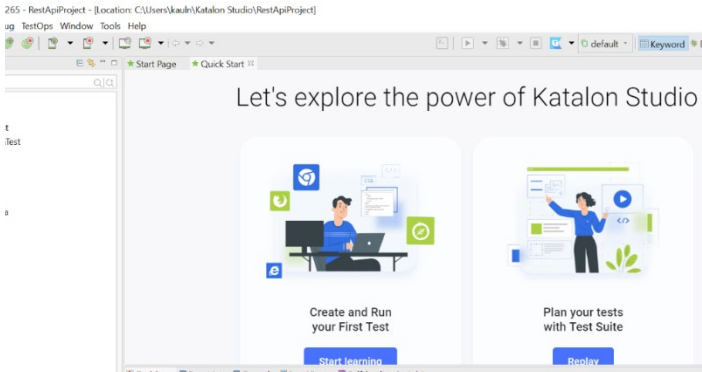


- **Example 7: Defining Custom Keywords:** Katalon Studio provide a wide range of built-in keywords that are available for testing different types of projects. But an interesting feature offered by Katalon Studio is that it allows users to define custom keywords. Customer keywords can be used to extend the functionality provided by Katalon studio (Introduction to Custom Keywords, 2022).

Once we define a new custom keyword, it can be used in test cases just like the other built-in keywords that we have seen until now.

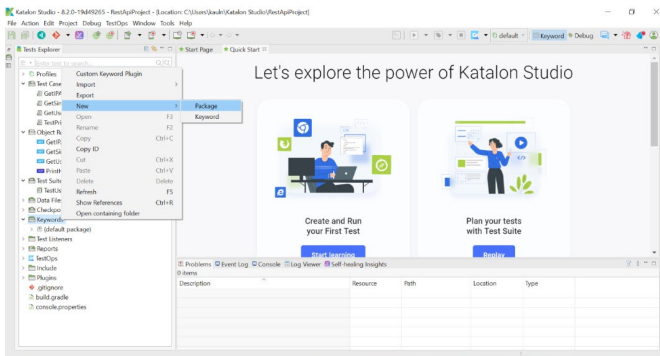
The steps to be followed are explained below:

1. First, we navigate to the ‘Keywords’ option in the Tests Explorer as shown below:

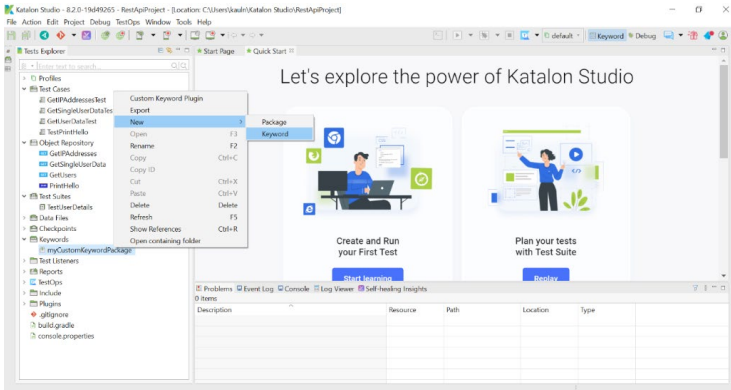


2. We create a new package by right-clicking on ‘Keywords’ and going to New → Package.

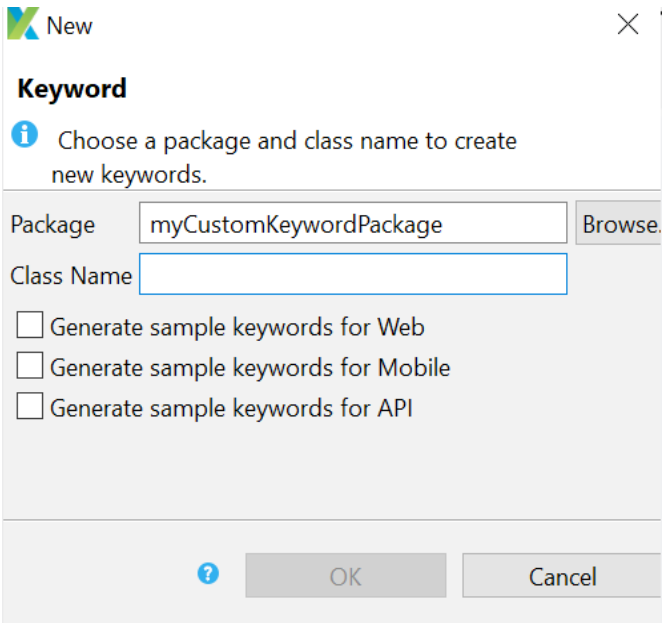
We can also create a new package by going to File → New → Package.



The New Package dialog appears:

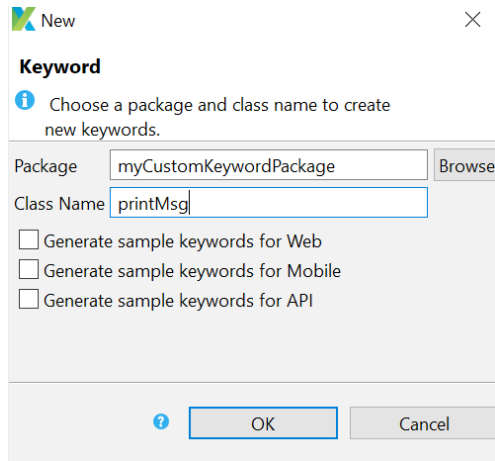


The create new Keyword dialog opens as follows:



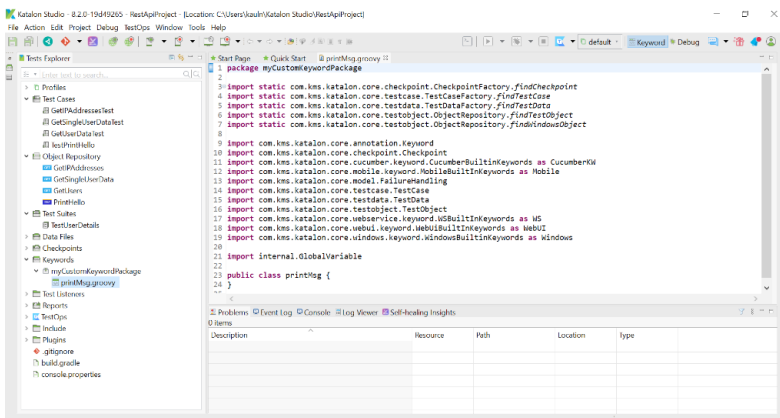
As seen from the screenshot above, the package name is already pre-filled as we had selected the package before entering the menu.

We provide the name of the keyword class as shown in the screenshot below. By default, the naming rules followed are like those of the Java language which means that the name of a class cannot start with a number, contain spaces, or have special characters in it. The naming convention for the Java language suggests creating a class name using a noun or a noun phrase and following the camel case rule where the first letter of each word is capitalized which helps better manage the project (Joy et al., 2000).



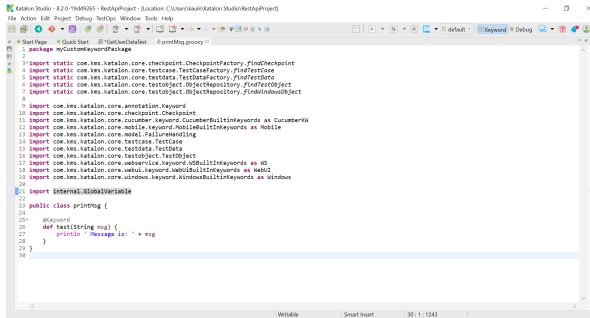
We can also create a new Keyword by going to File → New → Keyword from the main menu. The only difference when using this option is that the Package name will not be pre-selected in the Package field in the New Keyword dialog, one would have to browse and choose the package manually.

Once, we click on ‘OK’ the keyword class is added to the custom package as shown below:



We now proceed to create a new keyword by defining a method in this newly created keyword class. In this example, we define a keyword called test that simply prints a message ‘Message is:’ followed by the value of the argument/parameter provided. The keyword can be defined in either groovy or java which are both compatible in Katalon Studio.

The newly added keyword is shown below:



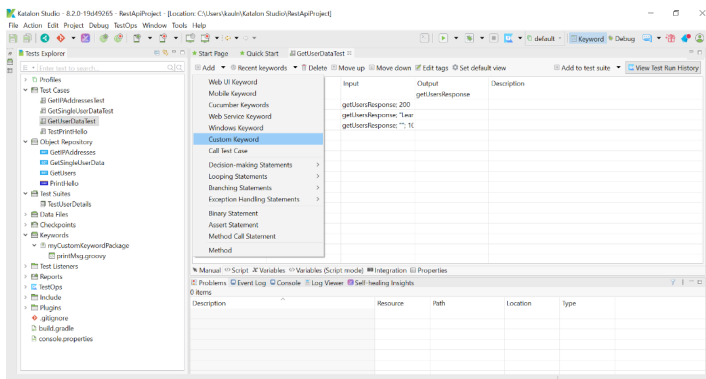
```

1 package myCustomKeywordPackage;
2
3 import static com.kalen.core.cheapest.CheapestFactory;
4 import static com.kalen.core.testdata.TestFactory;
5 import static com.kalen.core.testdata.TestFactory;
6 import static com.kalen.core.testdata.TestFactory;
7 import static com.kalen.core.testdata.TestFactory;
8
9 import com.kalen.core.annotation.Keyword;
10 import com.kalen.core.cheapest.Cheapest;
11 import com.kalen.core.customer.keyword.CustomerBuildKeywords as Customer;
12 import com.kalen.core.model.Facility;
13 import com.kalen.core.testdata.TestData;
14 import com.kalen.core.testdata.TestData;
15 import com.kalen.core.testdata.TestData;
16 import com.kalen.core.testdata.TestData;
17 import com.kalen.core.testdata.TestData;
18 import com.kalen.core.testdata.TestData;
19 import com.kalen.core.testdata.TestData;
20 import com.kalen.core.testdata.TestData;
21
22 import internal.GlobalVariable;
23
24 public class printing {
25     @Keyword
26     def test(String msg) {
27         println "Message is: " + msg;
28     }
29 }

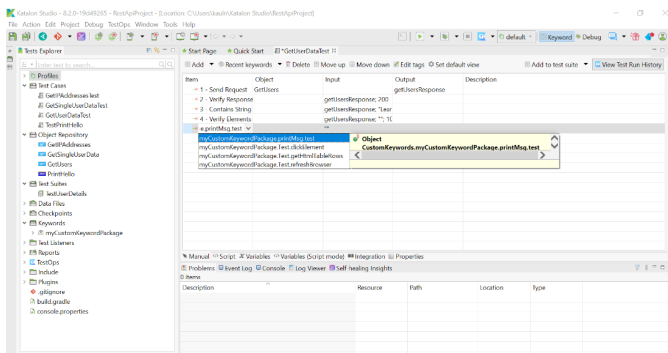
```

Once we add the keyword, we save the file.

4. We now proceed to implement this keyword in one of our test cases. We proceed to edit the 'GetUsersDataTest' test case. We go to 'Add' and select the option 'Custom Keyword' as shown below:

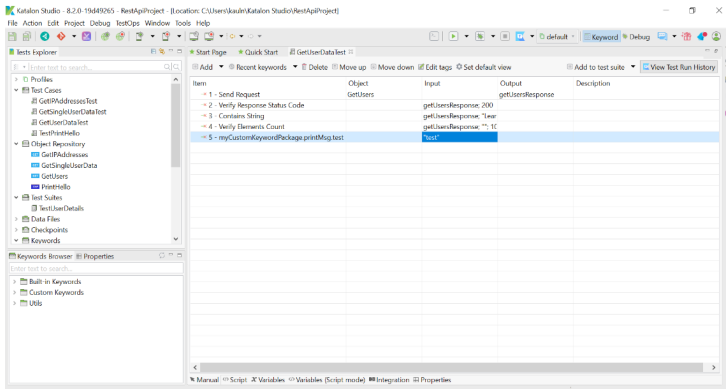


A dropdown list of all the available custom keywords is provided as shown below:

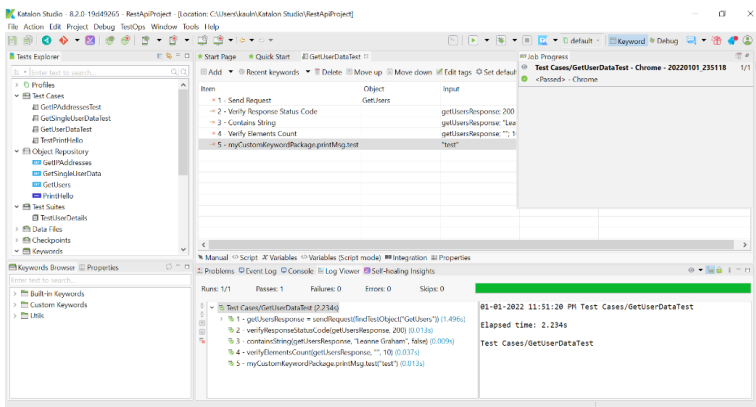


We select the keyword that we created which is ‘myCustomKeywordPackage.printMsg.test.’

We provide the value ‘test’ for the ARG parameters that we defined in the custom keyword. This is shown below:



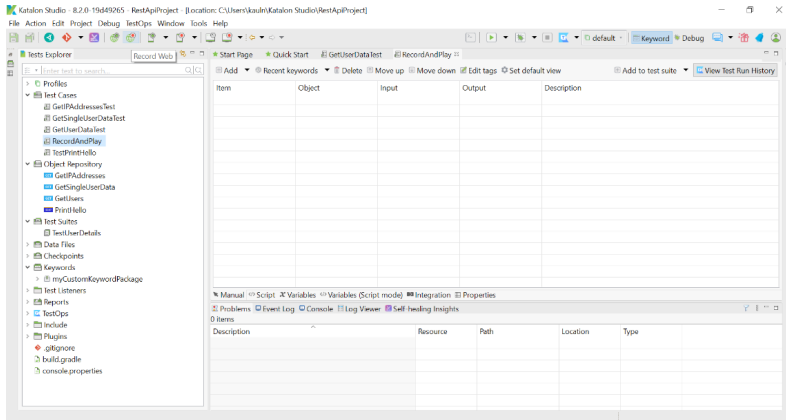
We run the test case by clicking on the play icon at the top. The test case executes with success. This is visible in the screenshot below:



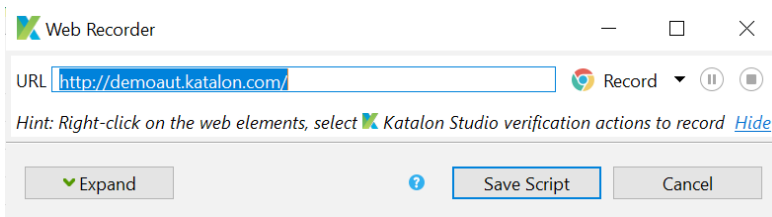
In this example, we added a simple test case to study how a custom keyword can be added. In real-life scenarios, meaningful keywords that help extend/enhance the quality of testing must be used.

Additionally, Katalon provides us with readily available sample custom keywords for Web, Mobile, and API projects. The documentation is available here: <https://docs.katalon.com/katalon-studio/docs/sample-custom-keywords.html>.

- **Example 8: Recording and Playback Using Katalon Studio:** An excellent feature that Katalon Studio has to offer is record

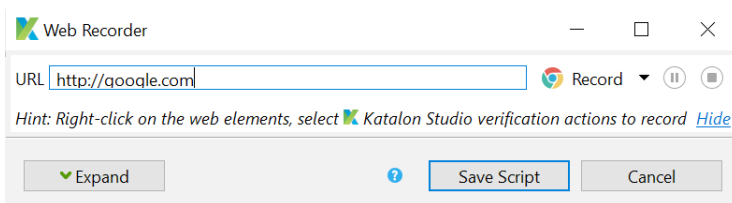


- The Record dialog appears and asks for information such as the URL to access and the browser that we wish to open such Chrome, Firefox, etc.

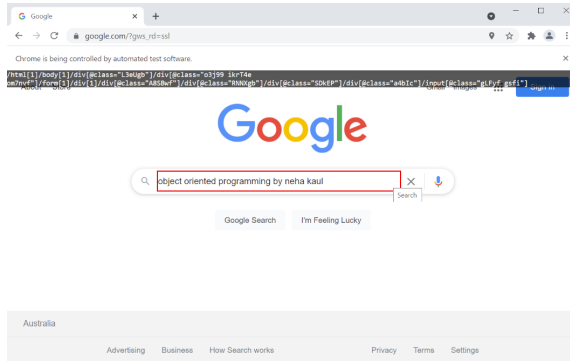


By default, the URL provided is `http://demoaut.katalon.com/`

In this example, we update the URL to `google.com` and keep the browser selection at Chrome which is the default value.

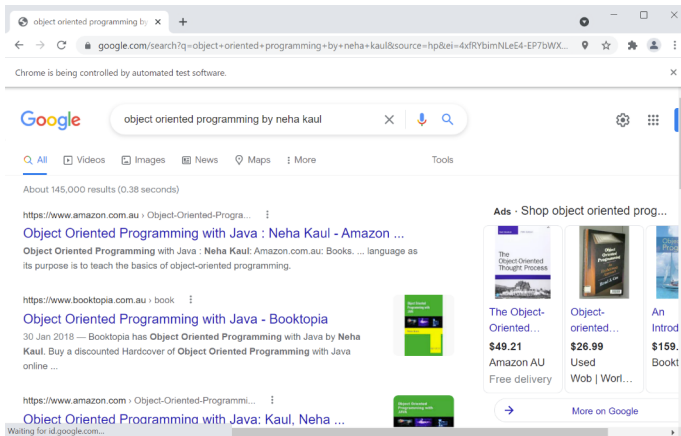


- We click on 'Record' to start recording actions taken in the browser. When we click on the 'Record' button, a Chrome Browser with the URL `http://google.com` opens as shown below:

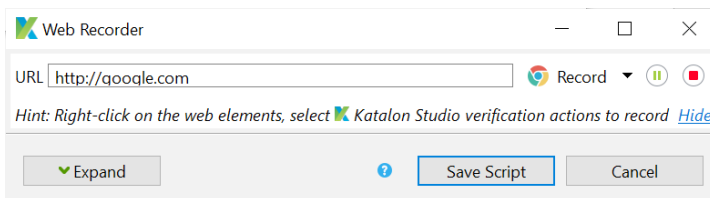


We enter the text ‘object-oriented programming by Neha Kaul’ in the google search bar and click on enter.

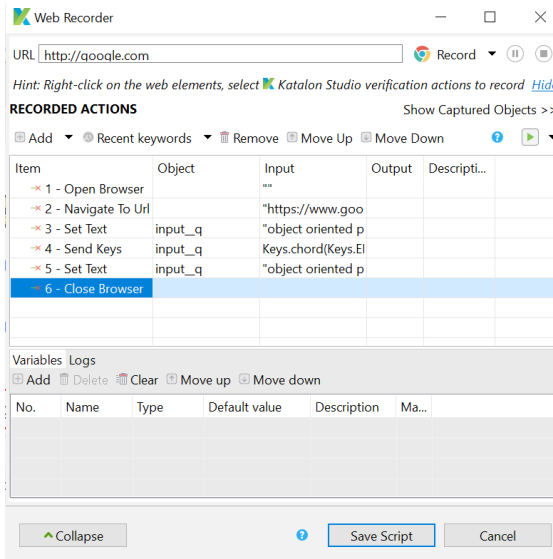
Google displays the results as follows:



- 5. We close the browser opened by automation and click on the Stop button in the Web Record Dialog:

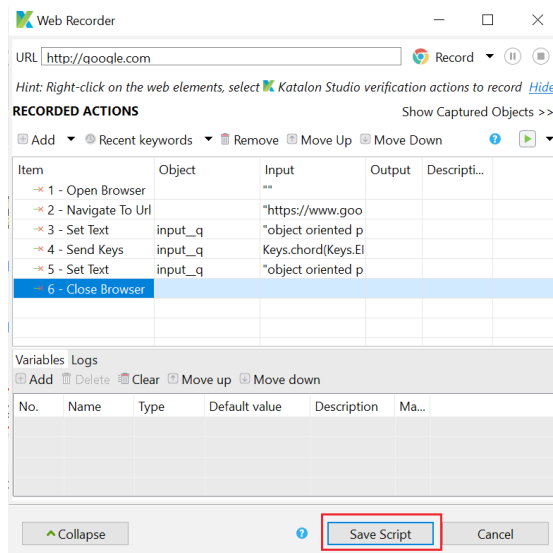


Once we stop recording, the Web Recorder dialog is updated with actions that we performed in the browser. This is shown below:



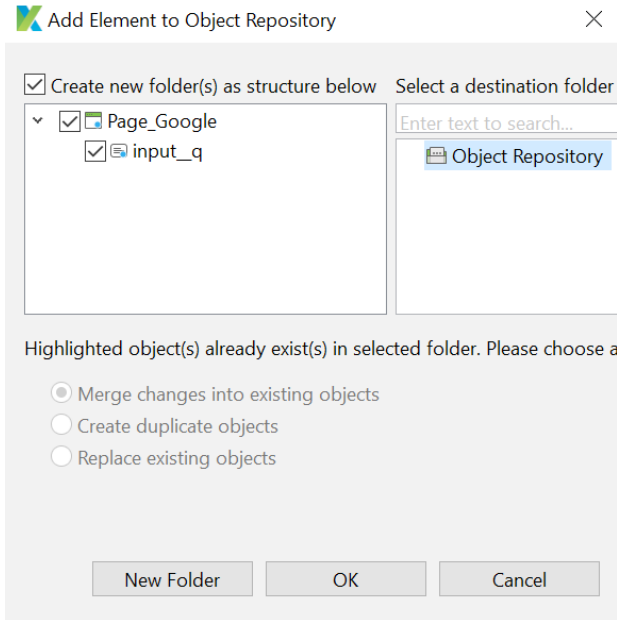
6. If we are happy with the test case, we can save the script. If not, you can record again.

To save the script generated, we click on the ‘Save Script.’

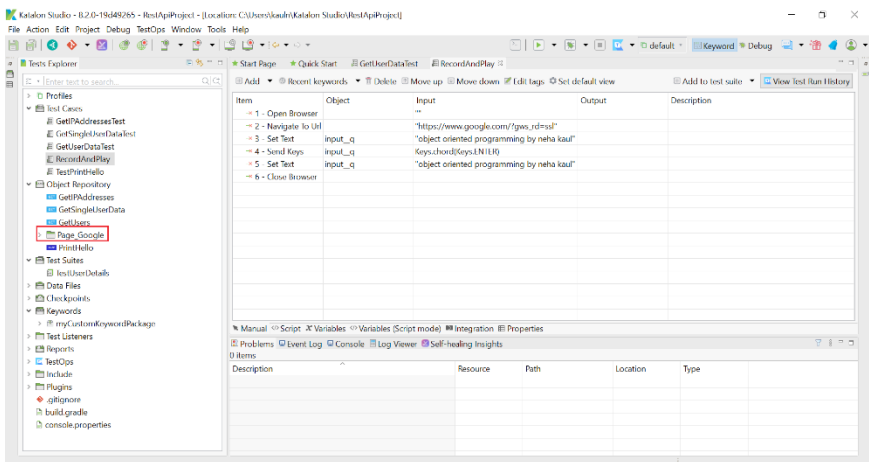


When we save the script, we are prompted to also save the element that we browsed (google page in this case) to the object repository.

To do this, click on ‘OK’ in ‘Add Element to Object Repository’ dialog as follows:

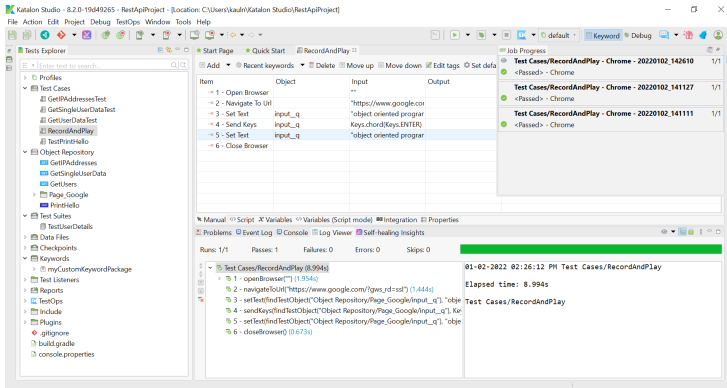


Once we save the element, it is added to the Object Repository as shown below:



We can also see that the test cases have an entry pertaining to each action that we took while recording.

7. To run the test case, we click on the play button at the top.



The test case is executed successfully.

CONTENTS

4.1. Important Watir Commands and Terminology	112
4.2. Watir Installation	114
4.3. Examples	127

This chapter describes the testing tool Watir. In this chapter, we learn what this tool is, how to install it and how to use it for automation testing.

Watir is an open-source library meant for test automation which develop in the Ruby language. The full form of Watir is Web Application Testing in Ruby and Watir is pronounced as “Water.” This tool helps testers develop automated tests for testing web applications. The automation tool is independent of the language in which the application under test is written. Watir supports the following web browsers: Firefox, Chrome, Internet Explorer, Edge, and Safari (Watir Project, 2009).

This tool is developed in Ruby and is available as a RubyGems install. This automation tool is created with the intent of interacting with a web browser in the same way as humans. Watir mimics user actions such filling online forms, clicking on different links, opening tabs, etc. (Watir Project, 2009).

The latest version of Watir is 7.1.0. Since it is developed in Ruby, it requires a user of this tool to possess elementary knowledge of Ruby. To run Ruby code, you need to install a Ruby interpreter (Watir Project, 2009).

4.1. IMPORTANT WATIR COMMANDS AND TERMINOLOGY

In this section, we cover some Watir basic commands that will help us start with this tool (“Module: Watir,” 2022).

- Loading the Watir Library:

`require 'watir'`

- Opening a Browser:

`browser = Watir::Browser.new:firefox`

`browser = Watir::Browser.new:chrome`

`browser = Watir::Browser.new:ie`

- Opening the Default Browser:

`browser = Watir::Browser.new`

- Opening a URL

`browser.goto "http://mysite.com"`

- Refreshing the Browser

`browser.refresh`

- Closing the Browser

`browser.quit`

`browser.close`

- Check if Button is Enabled

`browser.button(:id => "button").enabled?`

- Get the Button Text

`browser.button(:id => "button").text`

- Button Click

`browser.button(:id => "button").click`

- Take a Screenshot

`Browser.screenshot.save "filename.png"`

- Maximize Browser Window

`browser.window.maximize`

- Set Text Field Value

`browser.text_field(:id => "textField").set "test"`

- Get Text Field Value

`browser.text_field(:id => "textField").value`

- Clear Text Field Value

`browser.text_field(:id => "textField").clear`

- Select Checkbox

`browser.checkbox(:id => "checkbox1").set`

`browser.checkbox(:id => "checkbox1").set(true)`

- Unselect Checkbox

`browser.checkbox(:id => "checkbox1").clear`

`browser.checkbox(:id => "checkbox1").set(false)`

- Check if Checkbox is Selected

`browser.checkbox(:id => "checkbox1").set?`

- Select from Dropdown List

`dropdownList = browser.select(id: 'names')`

`dropdownList.select(text: 'Tom')`

`dropdownList.select "Tom"`

- Clear Dropdown List Selection

`dropdownList.clear`

- Select Radio Button

`browser.radio(:id => "radioButton").set`

- Check if Radio Button is Selected

`browser.radio(:id => "radioButton").selected`

- Click on a Hyperlink Text

`browser.link(:text => "Test").click`

`browser.link(:text => /Test/).click`

- Wait until a Condition Becomes True

`browser.link(:text => /Edit/).wait_until(&:present?)`

`browser.text_field(name: "test").wait_until(&:present?).click`

`browser.text_field(name: "test").wait_until(timeout: 30, &:present?)`

- Sleep for Set Duration

`Sleep 10`

- Check if the Browser Includes a Certain Text (this will print true/false)

`browser.text.include? 'Apple'`

Use puts to print the output:

`puts browser.text.include? 'Apple'`

4.2. WATIR INSTALLATION

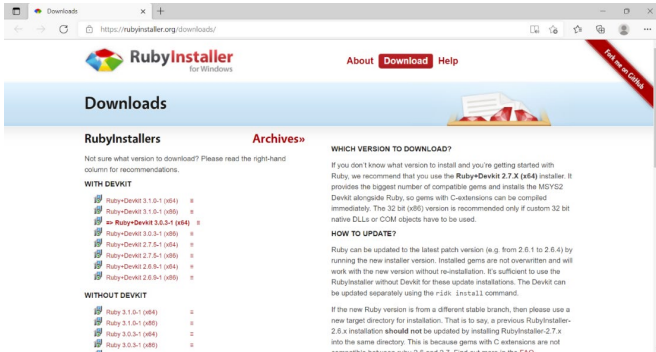
In this section, we look at the installation process for Watir.

The installation process for Watir is straightforward. The online documentation provided on their official website is accurate, helpful, and up to date.

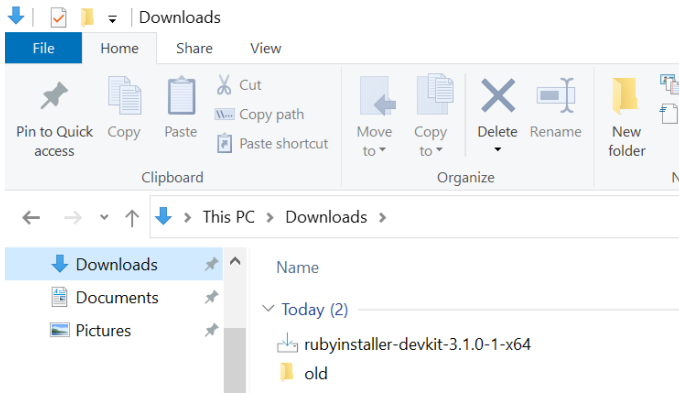
In this book, we will be installing Watir on a windows machine.

The steps followed are:

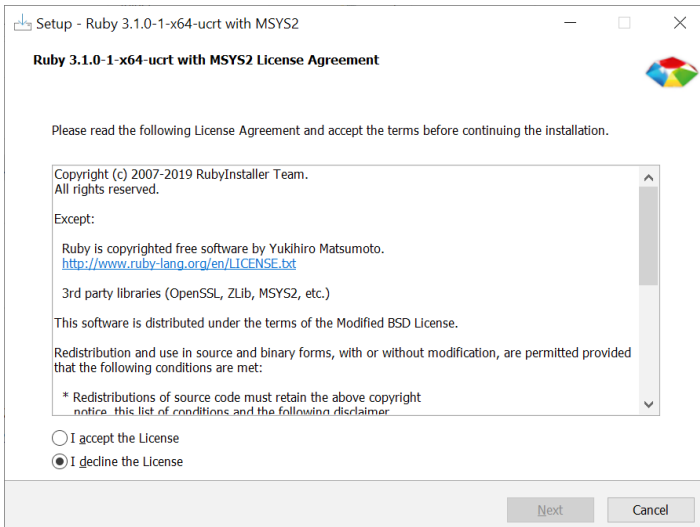
1. The first step is to install a Ruby Interpreter as it is required to run code written in the Ruby language. To install the Ruby Interpreter for windows, we go to the following site <https://rubyinstaller.org/downloads/> and download the latest ruby interpreter for windows.

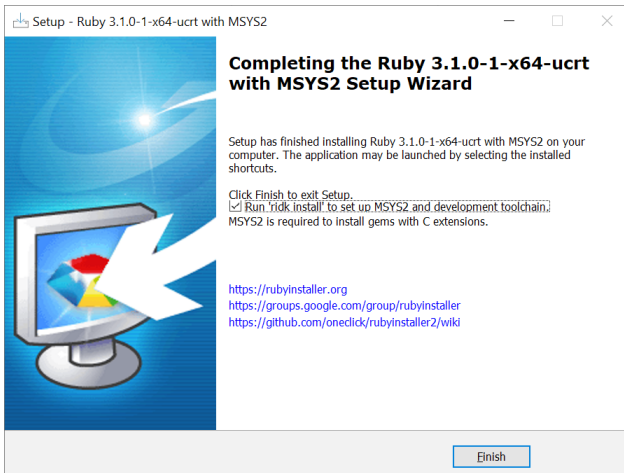
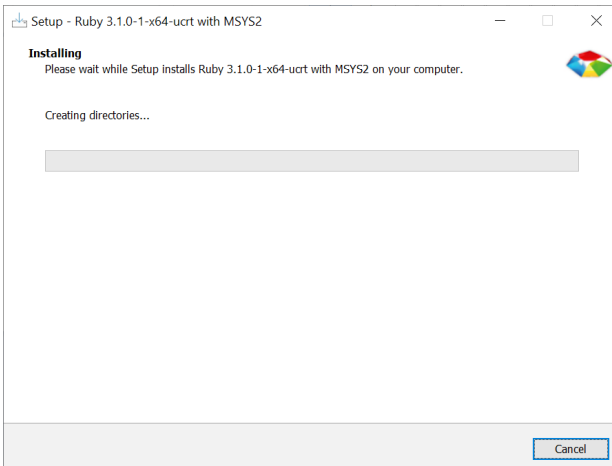
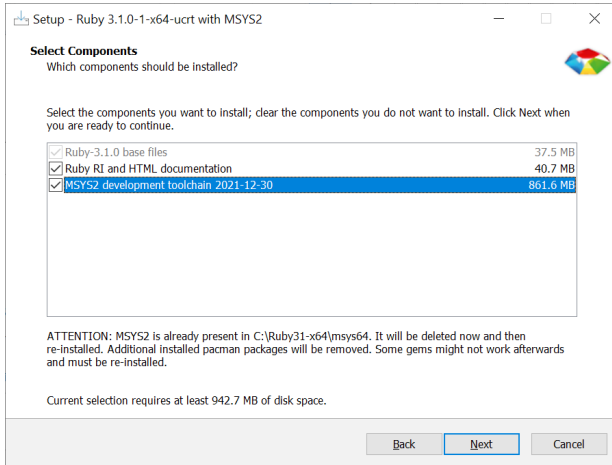


We download the 64-bit 3.1.0-1 version of the installer.



2. We run the ruby installer and follow the setup as follows:





- Once the setup is complete, a command prompt window opens and asks us to install additional components that are needed as part of the installation. The window that opens is as follows:

```

C:\Windows\system32\cmd.exe
Ruby Installer
for Windows

1 - MSYS2 base installation
2 - MSYS2 system update (optional)
3 - MSYS2 and MINGW development toolchain

Which components shall be installed? If unsure press ENTER [1,3] 1,3

```

We choose option 1 and 3 and click on the ‘Enter’ button on our keyboard.

```

Select C:\Windows\system32\cmd.exe
warning: gawk-5.1.0-2 is up to date -- skipping
warning: grep-3.0-3 is up to date -- skipping
warning: libtool-2.4.6-12 is up to date -- skipping
warning: m4-1.4.19-2 is up to date -- skipping
warning: make-4.3-1 is up to date -- skipping
warning: patch-2.7.6-1 is up to date -- skipping
warning: sed-4.8-2 is up to date -- skipping
warning: texinfo-6.8-1 is up to date -- skipping
warning: texinfo-tex-6.8-1 is up to date -- skipping
warning: wget-1.21.2-1 is up to date -- skipping
warning: mingw-w64-ucrt-x86_64-binutils-2.37-4 is up to date -- skipping
warning: mingw-w64-ucrt-x86_64-crt-git-9.0.0.6357.eac8c38c1-3 is up to date -- skipping
warning: mingw-w64-ucrt-x86_64-gcc-11.2.0-5 is up to date -- skipping
warning: mingw-w64-ucrt-x86_64-gcc-libs-11.2.0-5 is up to date -- skipping
warning: mingw-w64-ucrt-x86_64-headers-git-9.0.0.6357.eac8c38c1-1 is up to date -- skipping
warning: mingw-w64-ucrt-x86_64-libmangle-git-9.0.0.6357.eac8c38c1-1 is up to date -- skipping
warning: mingw-w64-ucrt-x86_64-libwinpthread-git-9.0.0.6357.eac8c38c1-1 is up to date -- skipping
warning: mingw-w64-ucrt-x86_64-make-4.3-1 is up to date -- skipping
warning: mingw-w64-ucrt-x86_64-tools-git-9.0.0.6357.eac8c38c1-1 is up to date -- skipping
warning: mingw-w64-ucrt-x86_64-winpthreads-git-9.0.0.6357.eac8c38c1-1 is up to date -- skipping
warning: pkgconf-1.8.0-1 is up to date -- skipping
warning: mingw-w64-ucrt-x86_64-pkgconf-1.8.0-2 is up to date -- skipping
there is nothing to do
Install MSYS2 and MINGW development toolchain succeeded

1 - MSYS2 base installation
2 - MSYS2 system update (optional)
3 - MSYS2 and MINGW development toolchain

Which components shall be installed? If unsure press ENTER []

```

The installation is successful, and we are again prompted to install it. But, as visible in the screenshot above, there are no options within the square brackets which means that the installation of all the compulsory components is complete. We click on enter and the window terminates.

- The next step is to install water using ruby gems.

To install watir, we first open a new command prompt window. To install watir we simply need to run the following command:

```
gem install watir
```

Let us first try to understand the command that we will execute.

A gem is like a package that you can download and install. A gem is

nothing but a collection of Ruby code that we put into a “collection” that we can name later (Pargal, 2020). The Gems software permits us to download, install, and make use of ruby packages. Each software package that is downloaded is called a “gem” and each “gem” contains a packaged Ruby application or a Ruby library.

The gem install command allows us to specify the gems that we want to use and the versions of the gems that we wish to use (Pargal, 2020).

In simpler terms, the ‘gem install’ command extracts the gem and places it into a directory on your machine. Here, we are installing the watir package to our system.

This is shown below:



Watir is installed successfully as shown in the image below:

```
Command Prompt
C:\Users\kauln>gem install watir
Successfully installed watir-7.1.0
Parsing documentation for watir-7.1.0
Done installing documentation for watir after 3 seconds
1 gem installed

C:\Users\kauln>
```

5. We then proceed to install the necessary drivers to run watir test cases and suites.

We begin by installing the selenium web driver in this step. The command to install it is: `sudo gem install selenium-webdriver --no-ri --no-rdoc` his is shown below:

```
Command Prompt - gem install the selenium-web driver -no-ri -no-rdoc
C:\Users\kauln>gem install watir
Successfully installed watir-7.1.0
Parsing documentation for watir-7.1.0
Done installing documentation for watir after 3 seconds
1 gem installed

C:\Users\kauln>gem install the selenium-web driver -no-ri -no-rdoc
Successfully installed the-0.0.3
Parsing documentation for the-0.0.3
Done installing documentation for the after 0 seconds
```

On running the command, selenium web driver is installed.

```
Command Prompt
C:\Users\kauln>gem install watir
Successfully installed watir-7.1.0
Parsing documentation for watir-7.1.0
Done installing documentation for watir after 3 seconds
1 gem installed

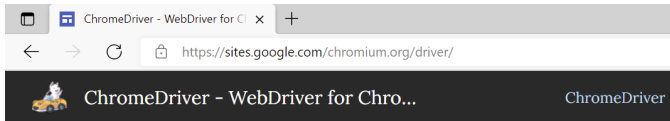
C:\Users\kauln>gem install the selenium-web driver -no-ri -no-rdoc
Successfully installed the-0.0.3
Parsing documentation for the-0.0.3
Done installing documentation for the after 0 seconds
ERROR: Could not find a valid gem 'selenium-web' (= 0) in any repository
ERROR: Possible alternatives: bbc-selenium-webdriver, rainux-selenium-webdriver, selenium-webdriver, selenium-webdriver
element-decorators, selenium-webdriver-element-extend_click_again, selenium-webdriver-full-screenshot, selenium-webdriv
er-rails-support-via-monkeypatch, selenium-webdriver-rails-support-via-monkeypatch.gemspec, selenium-webdriver-viewers,
superbot-selenium-webdriver
Successfully installed driver-0.0.4
Parsing documentation for driver-0.0.4
Done installing documentation for driver after 0 seconds
2 gems installed

C:\Users\kauln>
```

6. The next webdriver that we install is the browser driver. We install the web driver for google chrome called the chromedriver. This

is done by going to the ChromeDriver website and downloading the chromedriver for the chrome browser installed on the test machine.

In this case, since we are using the version 97 of chrome web browser, we download the corresponding chromedriver for windows.



You can view the current implementation status of the WebDriver standard [here](#).

All versions available in [Downloads](#)

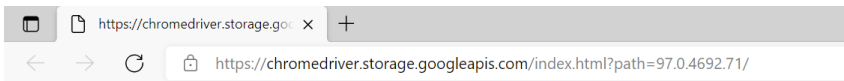
- Latest stable release: [ChromeDriver 97.0.4692.71](#)
- Previous stable release: [ChromeDriver 96.0.4664.45](#)

ChromeDriver Documentation

- [Getting started with ChromeDriver on Desktop](#) (Windows, Mac, Linux)
 - [ChromeDriver with Android](#)
 - [ChromeDriver with ChromeOS](#)
- [ChromeOptions](#), the capabilities of ChromeDriver
- [Mobile emulation](#)
- [Security Considerations](#), with recommendations on keeping ChromeDriver safe
- [Chrome Extension installation](#)
- [Verbose logging](#) and [performance data logging](#)

Troubleshooting

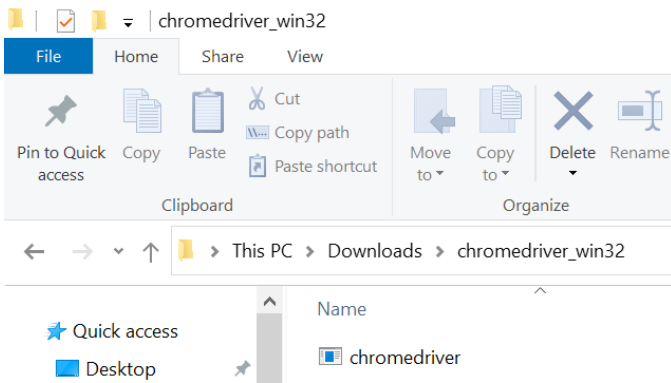
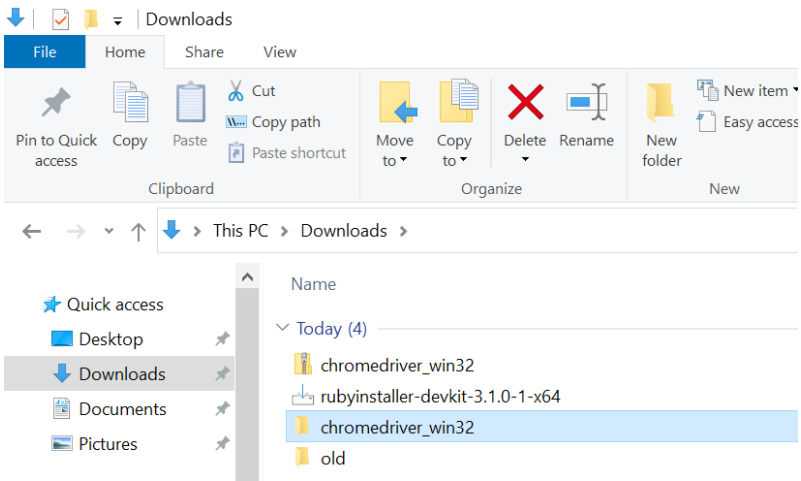
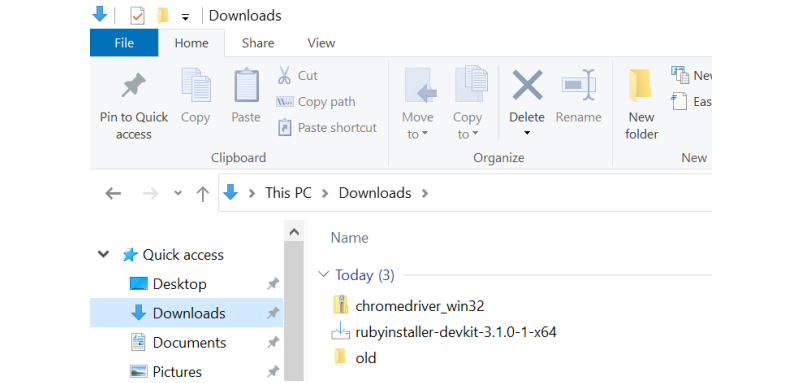
- ① [Chrome crashes immediately or doesn't start](#)
- [ChromeDriver crashes](#)

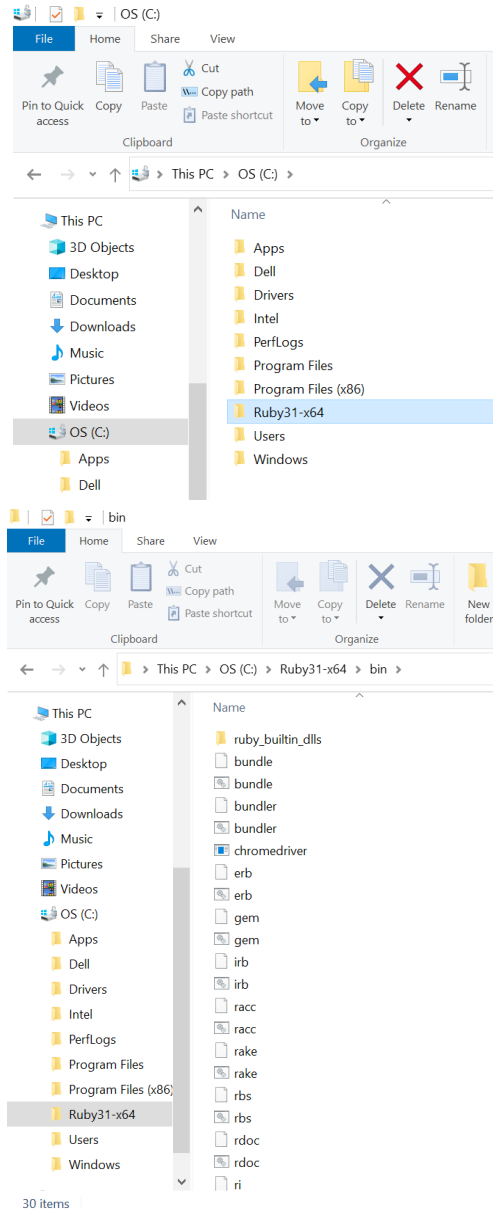


Index of /97.0.4692.71/

	Name	Last modified	Size	ETag
	Parent Directory		-	
	chromedriver_linux64.zip	2022-01-05 05:45:10	9.52MB	156efec320a9760140fcbeac740513d2
	chromedriver_mac64.zip	2022-01-05 05:45:13	7.89MB	952c27f9ca42748db82b15f8c0c59d3c
	chromedriver_mac64_m1.zip	2022-01-05 05:45:15	7.48MB	bb0e354876e64b8725b620d482247dff
	chromedriver_win32.zip	2022-01-05 05:45:17	5.89MB	58ac3bf76466773680a5fe04b69ad1d3
	notes.txt	2022-01-05 05:45:22	0.00MB	0fe69c56feb42175dd29cf69e4f38e9d

Once we have downloaded the chromedriver, we proceed to extract it and paste a copy of the chrome driver in the bin folder of our Ruby installation.





As seen from the screenshot above, we have successfully added the chrome driver to the following location: `C:\Ruby31-x64\bin`.

- To test the successful addition of the chromedriver we run the following command in an irb session: `browser = Watir::Browser.new`.

We do not need to provide the browser name as the default browser for Watir is chrome.

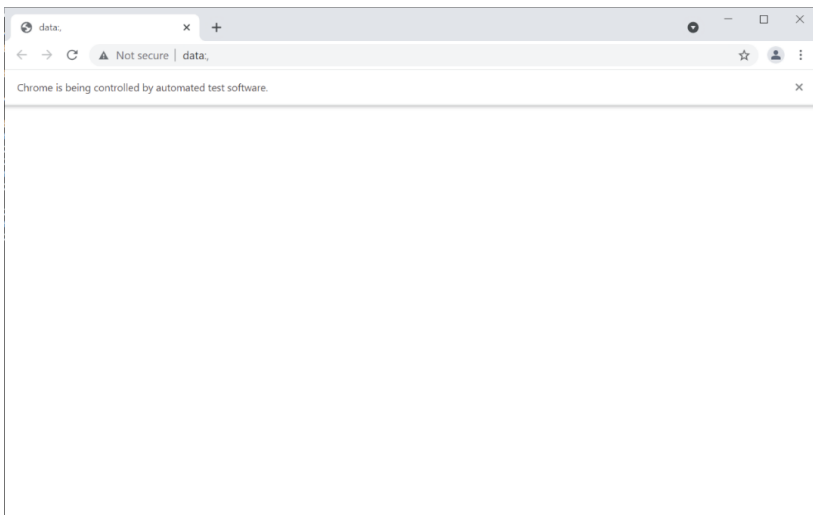
```
Command Prompt - irb

C:\Users\kauIn>irb
irb(main):001:0> require 'watir'
=> true
irb(main):002:0> browser = Watir::Browser.new
```

```
Command Prompt - irb

C:\Users\kauIn>irb
irb(main):001:0> require 'watir'
=> true
irb(main):002:0> browser = Watir::Browser.new

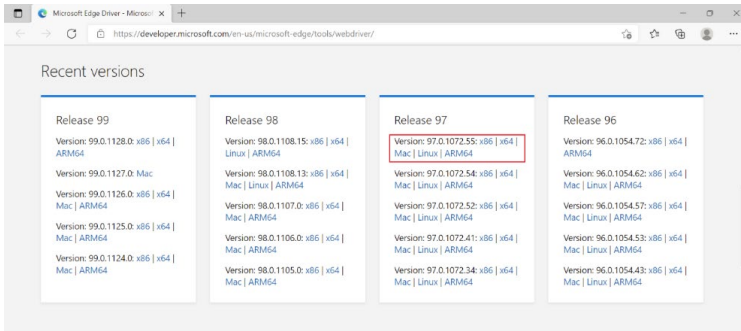
DevTools listening on ws://127.0.0.1:52157/devtools/browser/78c51f58-f944-4b9f-ac9d-5ce4562d36a8
=> #Watir::Browser@x-4e4b358a8 url="data:" title=""
irb(main):003:0> [22944:25992:0110/175317.821:ERROR:chrome_browser_main_extra_parts_metrics.cc(227)] START: ReportBluetoothAvailability(). If you don't see the END: message
this is crbug.com/1216328.
[22944:25992:0110/175317.821:ERROR:chrome_browser_main_extra_parts_metrics.cc(238)] END: ReportBluetoothAvailability()
[22944:10048:0110/175317.822:ERROR:device_event_log_impl.cc(214)] [17:53:17.822] USB: usb_device_handle_win.cc:1050 Failed to read descriptor from node connection: A device
attached to the system is not functioning. (0x1f)
[22944:25992:0110/175317.822:ERROR:chrome_browser_main_extra_parts_metrics.cc(235)] START: GetDefaultBrowser(). If you don't see the END: message, this is crbug.com/1216328
[22944:10048:0110/175317.822:ERROR:device_event_log_impl.cc(214)] [17:53:17.822] USB: usb_device_handle_win.cc:1050 Failed to read descriptor from node connection: A device
attached to the system is not functioning. (0x1f)
[22944:25992:0110/175317.826:ERROR:chrome_browser_main_extra_parts_metrics.cc(239)] END: GetDefaultBrowser()
```



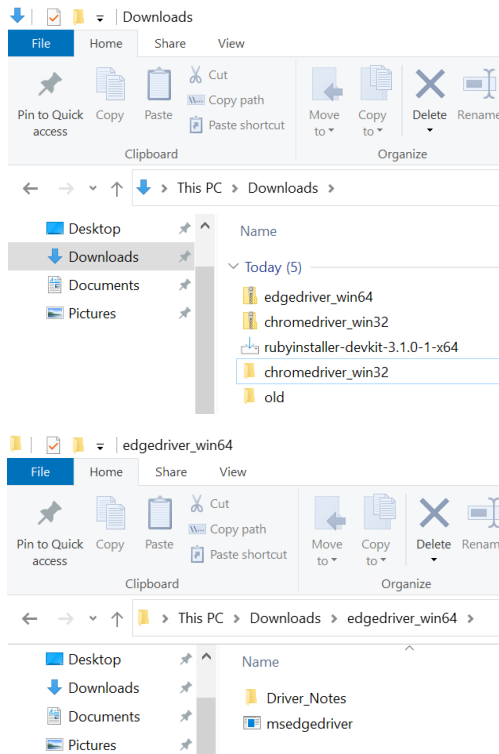
As we can see, a chrome web page is opened confirming that the driver has been installed correctly.

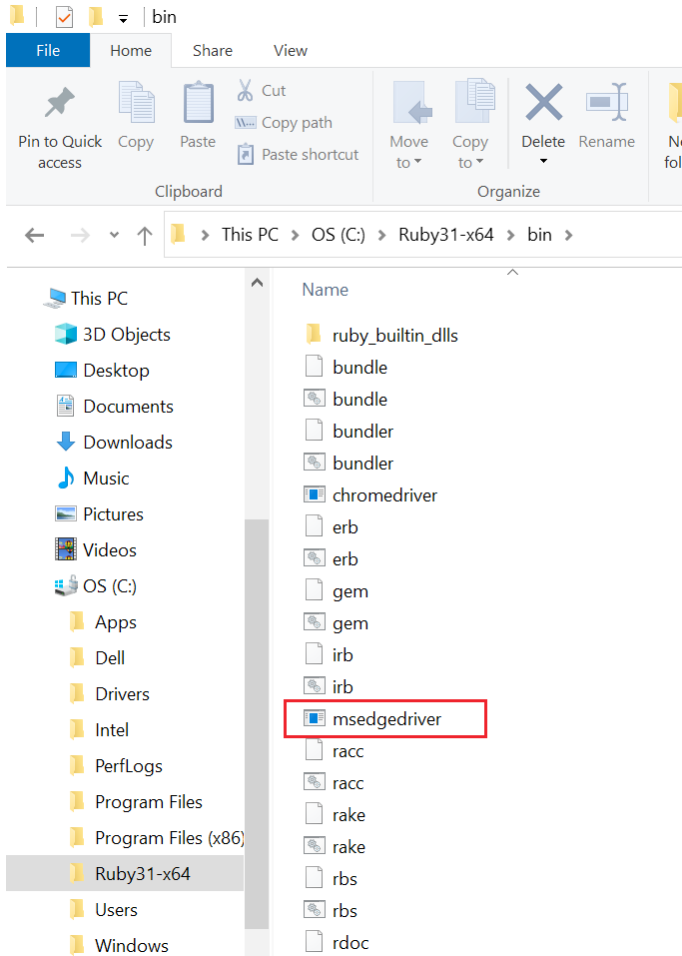
8. We proceed to install the driver for Microsoft Edge in this step.

To install the webdriver for Edge, first, we check the version of Microsoft installed on the system and then proceed to download the web driver for that version.



We extract the webdriver and paste it in the bin folder of our ruby installation.





To test whether we are able to open a Microsoft Edge browser using Watir, we open an irb session. irb or IRB stands for Interactive Ruby Shell which is a REPL (read-eval print loop) for programming in the object-oriented scripting language Ruby (Cross Browser Automation Testing Using Watir, 2019).

We run the following command in an open irb session:

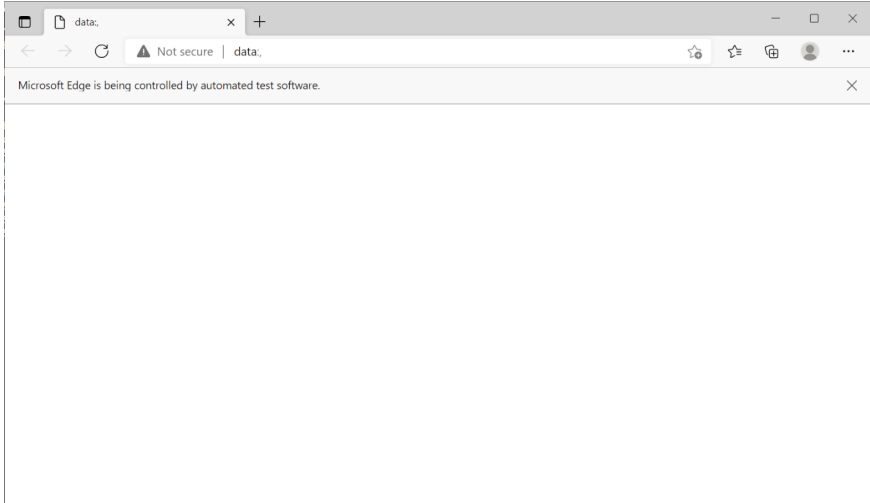
```
Browser = Watir::Browser.new:edge
```

In this command, we specify the browser that we wish to open because watir will open a Google Chrome browser by default.

```
Command Prompt - irb
C:\Users\kauin>irb
irb(main):001:0> require 'watir'
=> true
irb(main):002:0> browser = Watir::Browser.new:edge
```

```
Command Prompt - irb
C:\Users\kauin>irb
irb(main):001:0> require 'watir'
=> true
irb(main):002:0> browser = Watir::Browser.new:edge

DevTools listening on ws://127.0.0.1:53335/devtools/browser/a5000eaf-4482-46cb-81e0-d00f88739132
=> #Watir::Browser=0x268172e url: 'data:', title: ''
irb(main):003:0> [25528:2656:0110/180742.783:ERROR:chrome_browser_main_extra_parts_metrics.cc(251)] START: GetDefaultBrowser(). If you don't see the END: message, this is c
pbug.com/1216328.
[25528:25408:0110/180742.789:ERROR:device_event_log_impl.cc(214)] [18:07:42.789] USB: usb_device_handle_win.cc:1050 Failed to read descriptor from node connection: A device
attached to the system is not functioning. (0x1f)
[25528:25408:0110/180742.790:ERROR:device_event_log_impl.cc(214)] [18:07:42.790] USB: usb_device_handle_win.cc:1050 Failed to read descriptor from node connection: A device
attached to the system is not functioning. (0x1f)
[25528:2656:0110/180742.794:ERROR:chrome_browser_main_extra_parts_metrics.cc(255)] END: GetDefaultBrowser()
[25528:25408:0110/180742.797:ERROR:profile_manager.cc(1076)] Cannot create profile at path C:\Users\kauin\AppData\Local\Microsoft\Edge\User Data\Default
[25528:25408:0110/180742.797:ERROR:profile_manager.cc(2067)] Cannot create profile at path C:\Users\kauin\AppData\Local\Microsoft\Edge\User Data\Default
```



As we can see from the screenshot above, a Microsoft edge browser tab is opened which indicates that our web driver has been picked up.

This concludes the basic installation of Watir. In the next section, we look at different examples that demonstrate the creation of test cases and automation using watir.

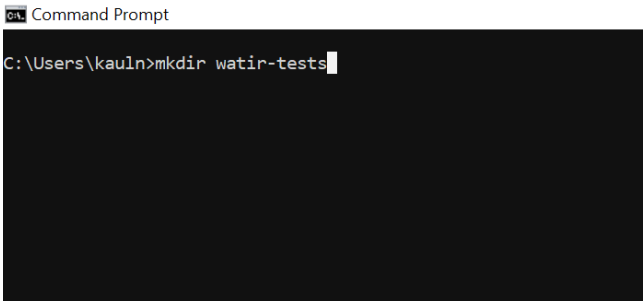
4.3. EXAMPLES

In this section, we look at writing test cases and implementing test suites using watir and ruby gems.

- **Example 1: Creating a Simple Test Case to Open the Default Browser:** In this example, we will write a simple test case and execute it using watir.

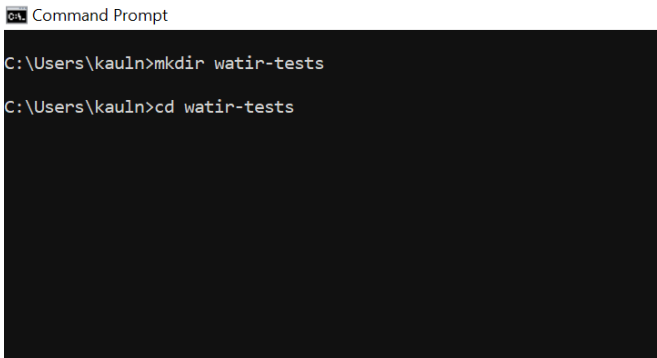
The steps followed are:

1. We open a command prompt and create a folder named watir-tests as shown below:



```
Command Prompt
C:\Users\kau\>mkdir watir-tests
```

2. We open the folder and create a new ruby file named test.rb in this folder.



```
Command Prompt
C:\Users\kau\>mkdir watir-tests
C:\Users\kau\>cd watir-tests
```

```
Select Command Prompt
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. All rights reserved.
C:\Users\kauIn>cd watir-tests
C:\Users\kauIn\watir-tests>type nul > test1.rb
```

```
Select Command Prompt
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. All rights reserved.
C:\Users\kauIn>cd watir-tests
C:\Users\kauIn\watir-tests>type nul > test1.rb
C:\Users\kauIn\watir-tests>dir
Volume in drive C is OS
Volume Serial Number is D458-377D

Directory of C:\Users\kauIn\watir-tests

01/12/2022 09:17 PM <DIR>      .
01/12/2022 09:17 PM <DIR>      ..
01/11/2022 10:14 PM             105 test1.rb
01/13/2022 12:30 AM              0 test1.rb
                3 File(s)          262 bytes
                2 Dir(s) 363,180,797,952 bytes free

C:\Users\kauIn\watir-tests>
```

3. We check the contents of the folder to check that the file is created.

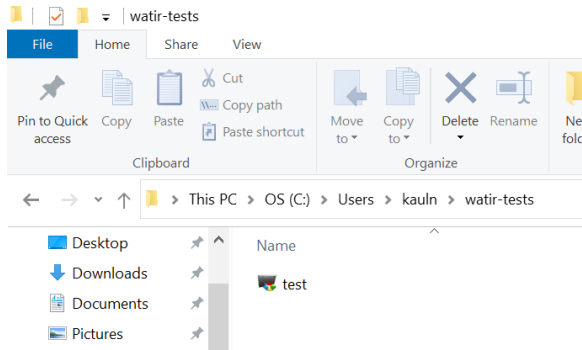
```
Command Prompt
C:\Users\kauIn>mkdir watir-tests
C:\Users\kauIn>cd watir-tests
C:\Users\kauIn\watir-tests>type nul > test.rb
C:\Users\kauIn\watir-tests>dir
Volume in drive C is OS
Volume Serial Number is D458-377D

Directory of C:\Users\kauIn\watir-tests

01/11/2022 10:11 PM <DIR>      .
01/11/2022 10:11 PM <DIR>      ..
01/11/2022 10:11 PM             0 test.rb
                1 File(s)           0 bytes
                2 Dir(s) 364,526,575,616 bytes free

C:\Users\kauIn\watir-tests>
```

4. We proceed to the folder where the file 'test.rb' is created and proceed to edit it.



We edit the file as follows:

```

File Edit Format View Help
require 'watir'
browser = Watir::Browser.new:edge
browser.goto "https://www.google.com/"
browser.close

```

Let us look at the commands that we have entered.

The first line of our code says → `require 'watir'`

This statement is like a declaration and here we communicate the fact that we will be needing the Watir library to run this code. This line of code ensures that the Watir library gets loaded.

The second statement is → `browser = Watir::Browser.new:edge`

In this command we specify the browser that we are using.

The next command is → `browser.goto "https://google.com"`

Here, we specify the URL that we wish to open in the Edge browser.

The last command is → `browser.close`

- Now, we proceed to executing the test case from the command line by using the load command as shown in the screenshot below:

```
Select Command Prompt - irb

C:\Users\kauln>mkdir watir-tests
C:\Users\kauln>cd watir-tests
C:\Users\kauln\watir-tests>type nul > test.rb
C:\Users\kauln\watir-tests>dir
Volume in drive C is OS
Volume Serial Number is D458-377D

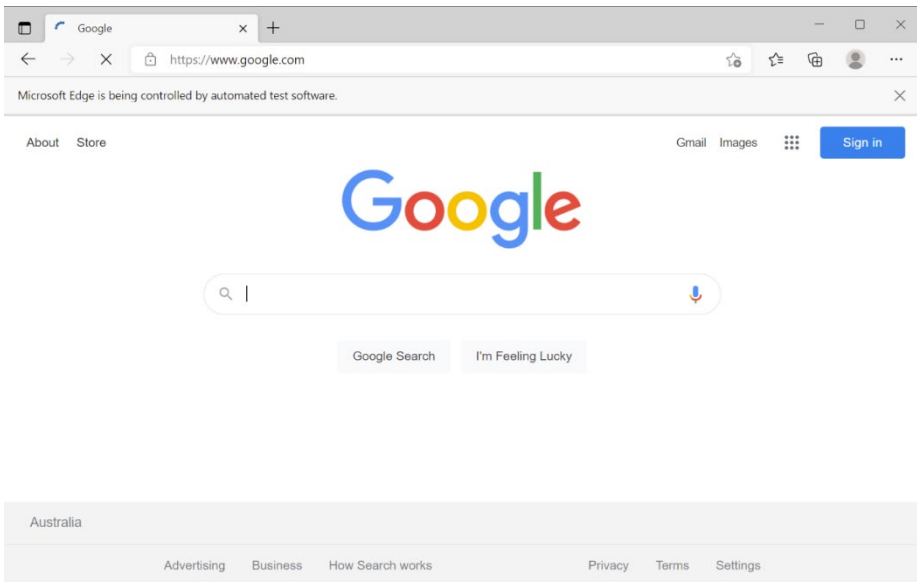
Directory of C:\Users\kauln\watir-tests

01/11/2022 10:11 PM <DIR>      .
01/11/2022 10:11 PM <DIR>      ..
01/11/2022 10:11 PM           0 test.rb
                1 File(s)        0 bytes
                2 Dir(s)   364,526,575,616 bytes free

C:\Users\kauln\watir-tests>irb
irb(main):001:0> load './test.rb'

DevTools listening on ws://127.0.0.1:56843/devtools/browser/1f3038c8-8472-4cbd-bb41-50a93da7e17d
=> true
irb(main):002:0>
```

The test case is executed and the URL google.com is opened in the Microsoft Edge browser as shown below:



- **Example 2: Opening the Default Browser Using Watir:** In this example, we will write a simple test case where we open the google search page in the default web browser Google chrome.

The steps we follow are:

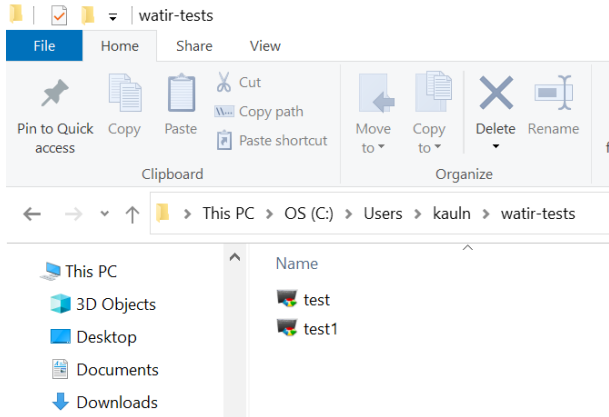
1. First, we create new file in the folder watir-tests named 'test1.rb.'



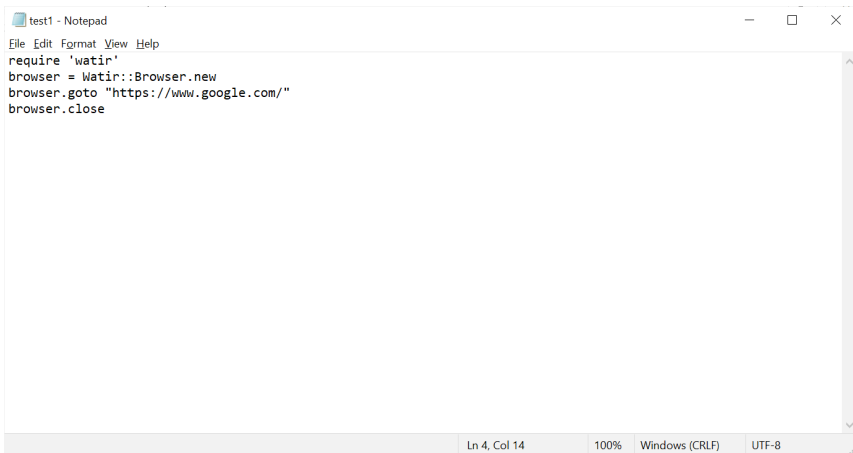
```
Command Prompt
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kauln>cd watir-tests

C:\Users\kauln\watir-tests>nuul > test1.rb
```



2. We edit this file as follows:



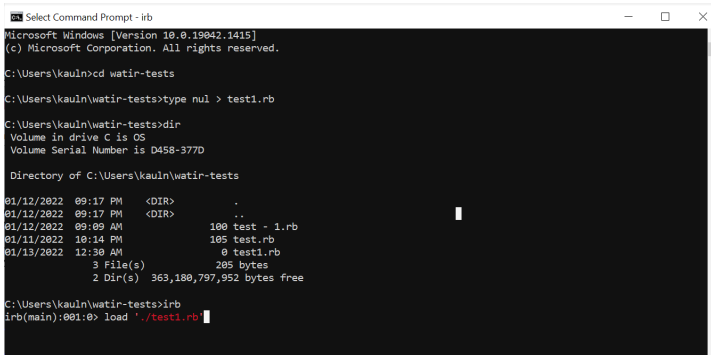
```
test1 - Notepad
File Edit Format View Help
require 'watir'
browser = Watir::Browser.new
browser.goto "https://www.google.com/"
browser.close
```

The screenshot shows a Notepad window titled 'test1 - Notepad'. The text content is as follows: `require 'watir'`, `browser = Watir::Browser.new`, `browser.goto "https://www.google.com/"`, and `browser.close`. The status bar at the bottom indicates 'Ln 4, Col 14', '100%', 'Windows (CRLF)', and 'UTF-8'.

The test1.rb file content is very similar to the file test.rb created in the previous example except the browser command. In this example, we simply change the second statement where we set the browser.

In this example, we do not specify the browser so that the default browser is opened.

3. We save the file and proceed to run this file from the command line.



```

Select Command Prompt - irb
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kauln>cd watir-tests
C:\Users\kauln\watir-tests>type nul > test1.rb
C:\Users\kauln\watir-tests>dir
Volume in drive C is OS
Volume Serial Number is D458-377D

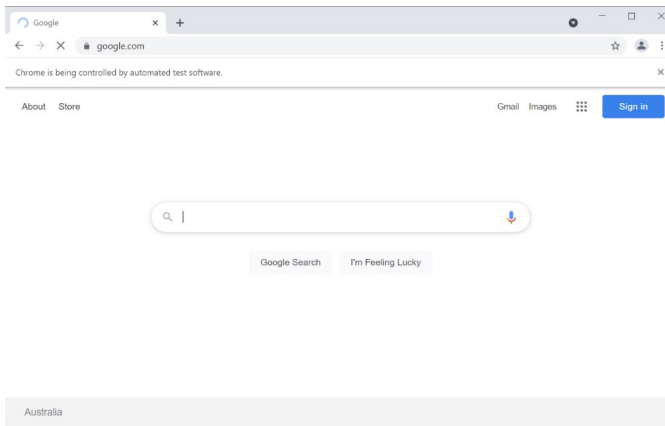
Directory of C:\Users\kauln\watir-tests

01/12/2022 09:17 PM <DIR>          .
01/12/2022 09:17 PM <DIR>          ..
01/12/2022 09:09 AM             100 test - 1.rb
01/11/2022 10:14 PM             105 test.rb
01/13/2022 12:30 AM              3 File(s)          205 bytes
                2 Dir(s)  363,180,797,952 bytes free

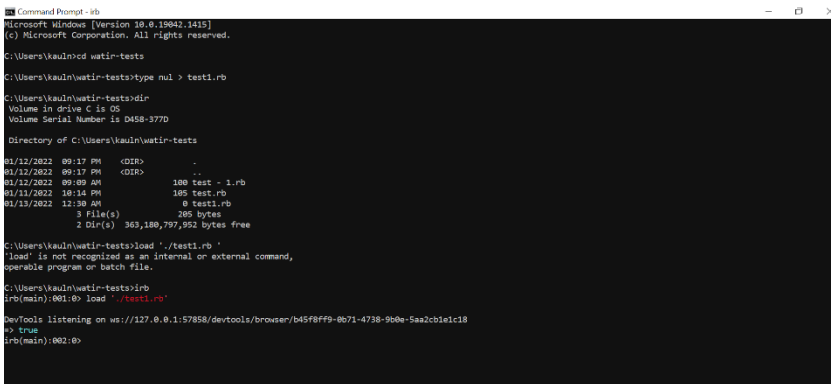
C:\Users\kauln\watir-tests>irb
irb(main):001:0> load './test1.rb'

```

4. The test is executed and the URL google.com is opened in the default browser which is Google Chrome.



The test is completed successfully as shown below:



```

Command Prompt - sb
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kauln>cd watir-tests
C:\Users\kauln\watir-tests>type nul > test1.rb
C:\Users\kauln\watir-tests>dir
Volume in drive C is OS
Volume Serial Number is D458-377D

Directory of C:\Users\kauln\watir-tests

01/12/2022 09:17 PM <DIR>          .
01/12/2022 09:17 PM <DIR>          ..
01/12/2022 09:09 AM             100 test - 1.rb
01/11/2022 10:14 PM             105 test.rb
01/13/2022 12:30 AM              3 File(s)          205 bytes
                2 Dir(s)  363,180,797,952 bytes free

C:\Users\kauln\watir-tests>load './test1.rb'
'load' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\kauln\watir-tests>irb
irb(main):001:0> load './test1.rb'

NewTool listening on ws://127.0.0.1:57858/devtools/browser/b45f8ff9-0671-4738-900e-5aa2c61e1c18
-> true
irb(main):002:0>

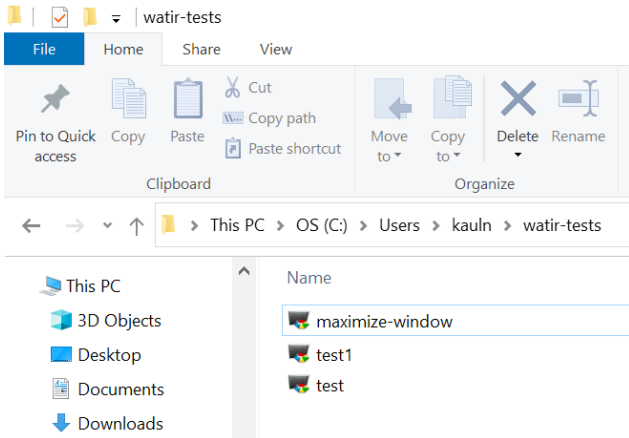
```

We can also run the ruby test case directly by double-clicking on the file which is an executable. We choose to run it from the command line to see if any errors are thrown. The command line helps us debug any issues that might arise. This example is a very simple example of a test case but in the case of complex test cases, it is always easier to run from the command line.

- **Example 3: Using the Maximize Command:** In this example, we look at the maximize command to maximize the browser when opened.

The steps to follow are:

1. First, we create a new ruby file in the same location as before and name it 'maximize-window.rb.'



2. We edit this file to maximize the browser as follows:

```

maximize-window - Notepad
File Edit Format View Help
require 'watir'
browser = Watir::Browser.new
browser.goto "https://www.google.com/"
browser.window.maximize
browser.close

```

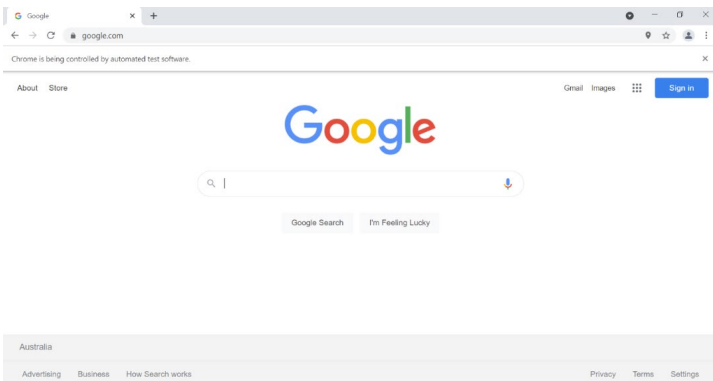
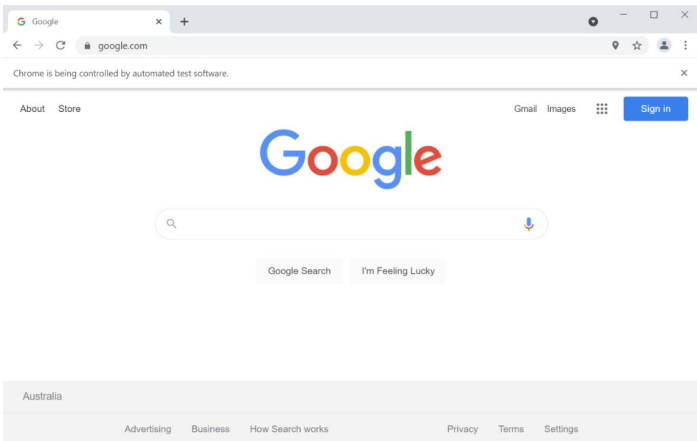
As visible in the screenshot above, we make use of `browser.window.maximize` command to maximize the browser window.

3. We execute this ruby file from the command prompt.

Command Prompt - irb

```
C:\Users\kauln>cd watir-tests  
C:\Users\kauln\watir-tests>irb  
irb(main):001:0> load './maximize-window.rb'
```

- 4. The automation executes the ruby commands and opens the URL configured in the file using Google chrome. This window is then maximized.



The test executes with success.

Command Prompt - irb

```
C:\Users\kauIn>cd watir-tests
C:\Users\kauIn\watir-tests>irb
irb(main):001:0> load './maximize-window.rb'

DevTools listening on ws://127.0.0.1:63868/devtools/browser/6b9e2acc-4ca5-401b-8b06-9e81bff75bc5
=> true
irb(main):002:0> |
```

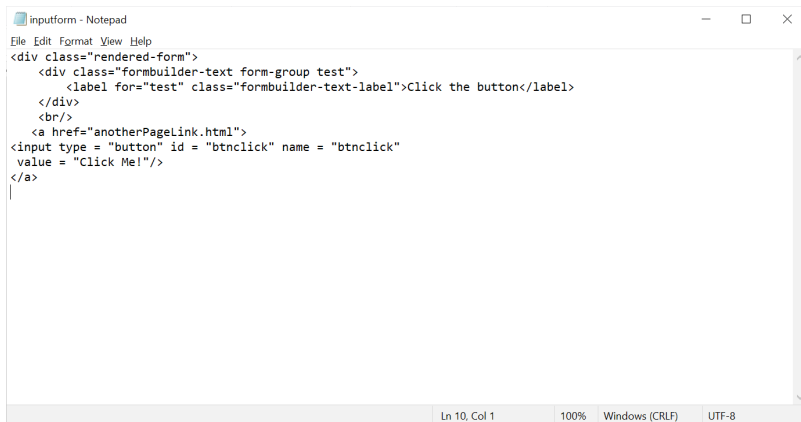
- **Example 4: Searching for a Text in a Simple html form Using Watir:** In this example, we use watir to test a simple html form.

The steps are as follows:

1. First, we create a simple html form in the same directory as our tests.

In this page, we add a button that redirects to another html page when clicked.

The main page is as follows:

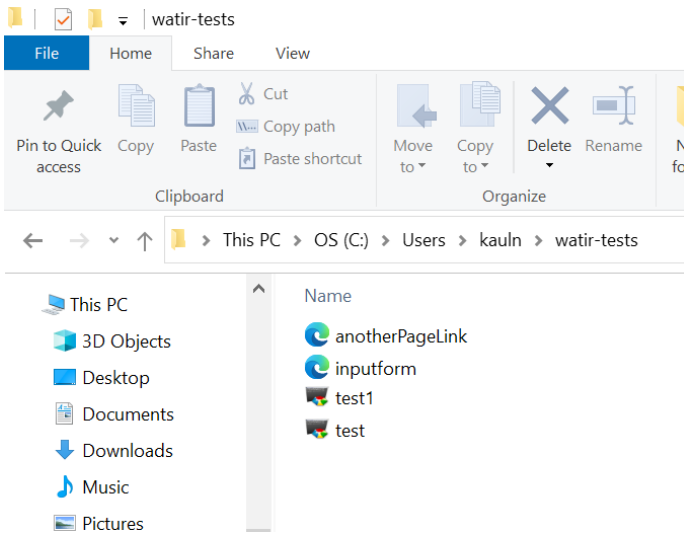


```
inputform - Notepad
File Edit Format View Help
<div class="rendered-form">
  <div class="formbuilder-text form-group test">
    <label for="test" class="formbuilder-text-label">Click the button</label>
  </div>
  <br/>
  <a href="anotherPageLink.html">
<input type = "button" id = "btnclick" name = "btnclick"
value = "Click Me!"/>
</a>
```

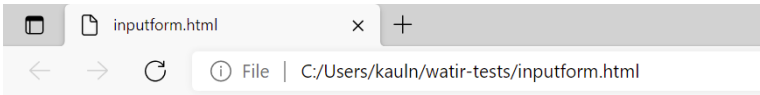
As seen from the content of the file above, we have a simple html page with a label and button which is redirected to the page ‘anotherPageLink.html’ when clicked.

The contents of the file ‘anotherPageLink.html’ are as follows:

These files can be viewed in directory as follows:



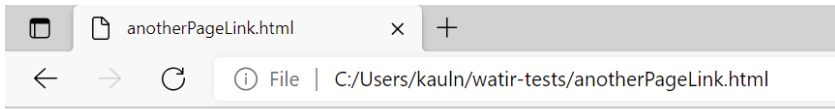
We double-click on the file ‘inputform’ to open it in the Microsoft edge browser.



Click the button

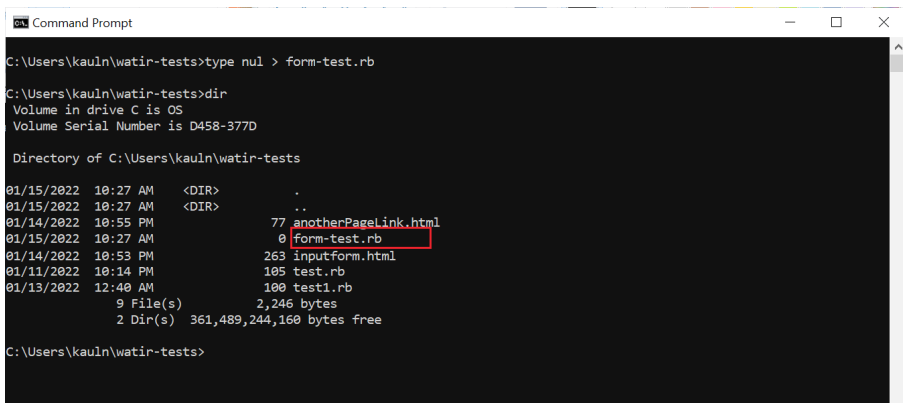
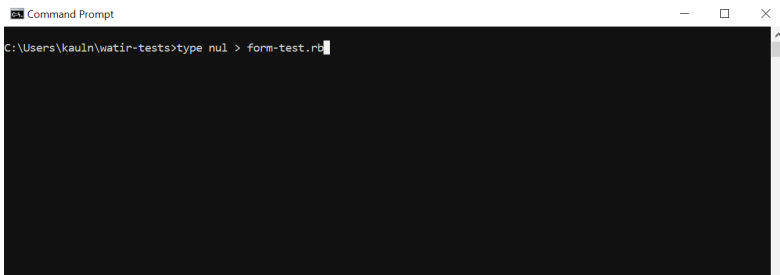


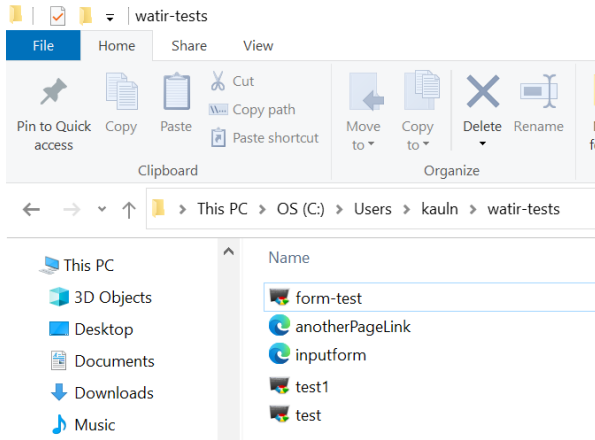
On click on the ‘Click Me!’ button, the redirected page is opened as shown below:



Hello!!!!

2. We proceed to create a new ruby file in the same directory.





3. We proceed to edit the file ‘form-test’ as follows:



The test case we created is very simple. Let us go through it step by step.

The first step is a common step that is needed for all test cases to be executed using ‘Waitr.’

The second statement is where we open the browser and save it in a variable which we will be used in the further steps. Here, we choose to open the default browser which is chrome.

The third statement is where we instruct that the html form be opened.

The fourth statement is where we provide the instruction to click on the button and provide its id to watir. We make use of the ‘wait_until’ command which waits until the given conditions become true (Watir Project – Waiting, 2009). The condition we give is that we wait until the button is present in the browser form before clicking it.

4. We run the file from the command line as follows:

```

Command Prompt - irb
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kauln>cd watir-tests
C:\Users\kauln\watir-tests>irb
irb(main):001:0> load './form-test.rb'

```

```

Command Prompt - irb
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kauln>cd watir-tests
C:\Users\kauln\watir-tests>irb
irb(main):001:0> load './form-test.rb'

DevTools listening on ws://127.0.0.1:59786/devtools/browser/818f50fc-6a58-42be-8224-11d8b8e39b08
irb(main):002:0> true
irb(main):002:0> [31892:29352:0115/115554.477:ERROR:chrome_browser_main_extra_parts_metrics.cc(227)] START: ReportBluetoothAvailability(). If you don't see the END: message, this is crbug.com/1216328.
[31892:29352:0115/115554.477:ERROR:chrome_browser_main_extra_parts_metrics.cc(230)] END: ReportBluetoothAvailability()
[31892:33344:0115/115554.477:ERROR:device_event_log_impl.cc(214)] [11:55:54.477] USB: usb_device_handle_win.cc:1050 Failed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[31892:29352:0115/115554.477:ERROR:chrome_browser_main_extra_parts_metrics.cc(235)] START: GetDefaultBrowser(). If you don't see the END: message, this is crbug.com/1216328.
[31892:33344:0115/115554.477:ERROR:device_event_log_impl.cc(214)] [11:55:54.477] USB: usb_device_handle_win.cc:1050 Failed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[31892:29352:0115/115554.483:ERROR:chrome_browser_main_extra_parts_metrics.cc(239)] END: GetDefaultBrowser()

```

The browser is opened, and the link is clicked.



- Example 5: Using Watir to Test an html for an Take a Screenshot:** In this example, we build upon the previous example where we test an html file containing a button which when clicked

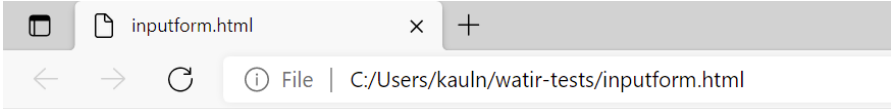
redirects to another html page. Here, we will take screenshots during the execution of the test case and save them.

The watir command to take a screenshot and save it is:

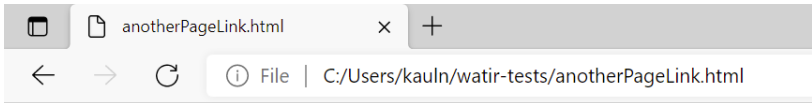
`browser.screenshot.save`

Let us see how this this is implemented. The steps we follow are:

1. We first open the html page manually.

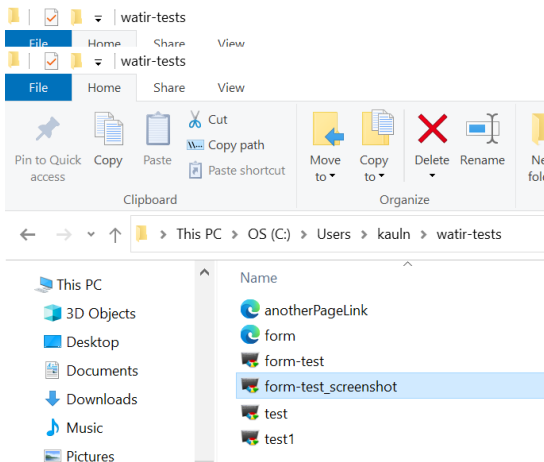


Click the button

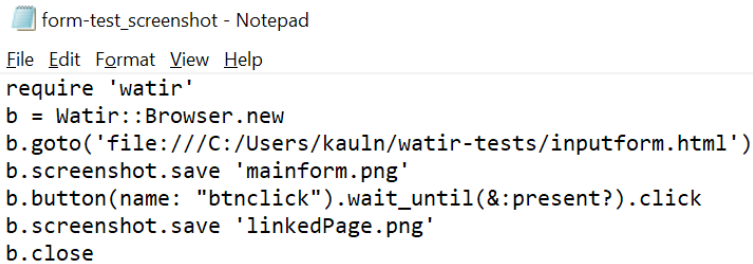


Hello!!!!

2. We create a new test file named 'form-test_screenshot.rb.' We create this file in the same location as the input form.



3. We edit the file to add commands where we take a screenshot.



```
form-test_screenshot - Notepad
File Edit Format View Help
require 'watir'
b = Watir::Browser.new
b.goto('file:///C:/Users/kauln/watir-tests/inputform.html')
b.screenshot.save 'mainform.png'
b.button(name: "btnclick").wait_until(&:present?).click
b.screenshot.save 'linkedPage.png'
b.close
```

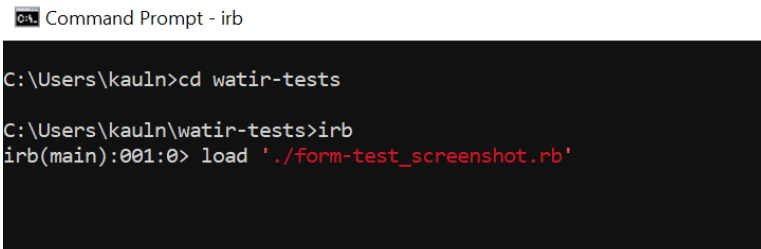
In this test case, we open the html page, take a screenshot, then click on the 'Click me' button and take a screenshot again before closing the browser.

We use the following command:

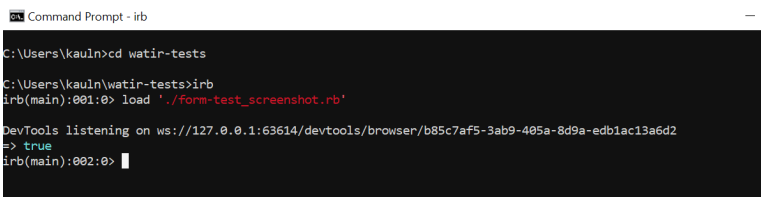
```
b.screenshot.save 'filename.png'
```

The above command takes a screenshot and saves it in the current repository with the file name provided.

4. We run the file using the command prompt.

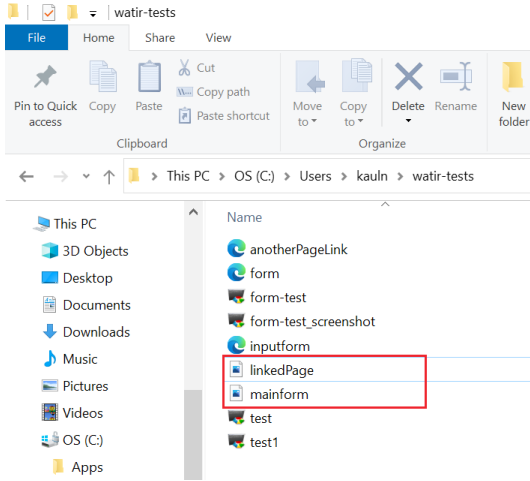


```
Command Prompt - irb
C:\Users\kauIn>cd watir-tests
C:\Users\kauIn\watir-tests>irb
irb(main):001:0> load './form-test_screenshot.rb'
```



```
Command Prompt - irb
C:\Users\kauIn>cd watir-tests
C:\Users\kauIn\watir-tests>irb
irb(main):001:0> load './form-test_screenshot.rb'
DevTools listening on ws://127.0.0.1:63614/devtools/browser/b85c7af5-3ab9-405a-8d9a-edb1ac13a6d2
=> true
irb(main):002:0> █
```

The screenshots are created in the same location as the test cases:



Click the button

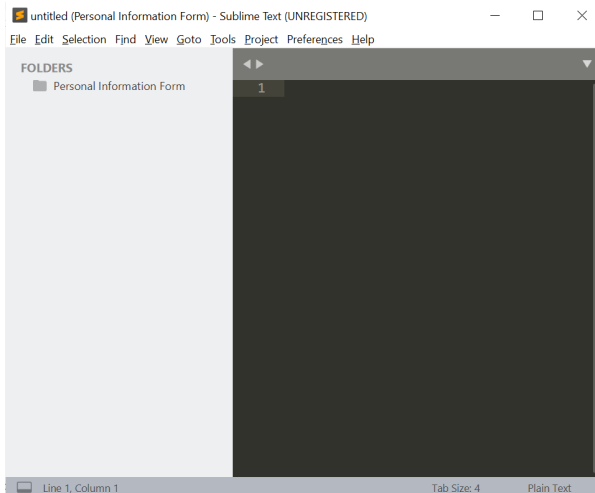


Hello!!!!

- **Example 6: Using Watir to Test an html Form to Test Various Web Elements:** In this example, we use watir to test a sample html form that accepts input and performs basic CRUD operations on records such as addition, editing, and deleting.
 1. First, we create a simple html project containing a form, a CSS stylesheet and a JavaScript controlling the CRUD actions.

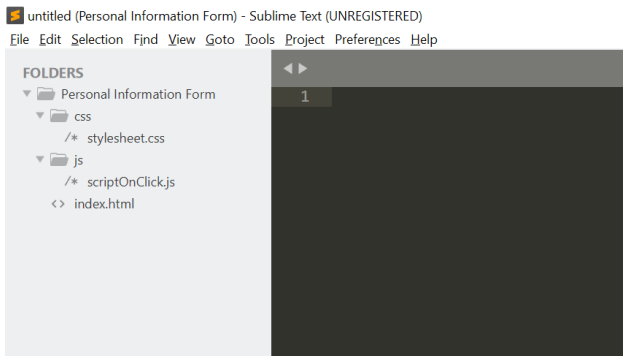
We make use of an HTML editor called Sublime Text (2000) to edit the files in our html project.

We open Sublime Text and create a new folder named ‘Personal Information Form’ as follows:



We proceed to create a project structure with an index.html file, a CSS folder containing a CSS file and a JS folder containing a JavaScript file.

This is shown below:



We look at each of the files in detail:

a. Index.html:

```

ion Form\index.html (Personal Information Form) - Sublime Text (UNREGISTERED)
Project Preferences Help
index.html x
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Personal User Information</title>
8   <link rel="stylesheet" href="css/styleSheet.css">
9 </head>
10 <body>
11 <CENTER><h2>User Information</h2></CENTER>
12 <table>
13 <tr>
14 <td>
15   <form onsubmit="event.preventDefault();onFormSubmit();" autocomplete="off">
16     <h3>User Information Form</h3>
17     <div>
18       <label>Full Name</label>
19       <label class="validation-error hide" id="fullNamevalidate">Full name should not
20       be empty!</label>
21       <input type="text" name="fullName" id="fullName" placeholder="Enter Full Name">
22     </div>
23     <div>
24       <label>Building Number</label>
25       <label class="validation-error hide" id="bldgNumvalidate">Building number
26       should not be empty!</label>
27       <input type="text" name="bldgNumber" id="bldgNumber" placeholder="Enter
28       building number">
29     </div>
30     <div>
31       <label>Street Address</label>
32       <label class="validation-error hide" id="streetAddressvalidate">Street Address
33       should not be empty!</label>
34       <input type="text" name="streetAddress" id="streetAddress" placeholder="Enter
35       Street Address">
36     </div>
37     <div>
38       <label>Age</label>
39       <label class="validation-error hide" id="agevalidate">Age should not be empty!
40       </label>
41       <input type="number" name="age" id="age" placeholder="Enter User Age">
42     </div>
43     <div class="form-action-buttons">
44       <input type="submit" name="submit" value="Submit">
45     </div>
46   </form>
47 </td>
48 <td>
49   <table class="list" id="userlist" style="background-color: #C2976D;">
50     <thead>
51       <tr>
52         <th>Full Name</th>
53         <th>Building Number</th>
54         <th>Street Address</th>
55         <th>Age</th>
56         <th>Action</th>
57       </tr>
58     </thead>
59     <tbody>
60     </tbody>
61   </table>
62 </td>
63 </tr>
64 </table>
65 <script src="js/scriptOnClick.js"></script>
66 </body>
67 </html>

```

As seen from the form above, we have created a simple form with four fields in a table separated by div tabs. A submit button is provided at the end to submit the data. A second table is created to display the data. At the end, we have a script element specifying the script to be used for the actions.

b. scriptOnClick.js:

tion Form)/js/onclick.js (Personal Information Form) - Sublime text (UNREGISTERED)

Project Preferences Help

```

1  var selectedRow = null
2
3  function onFormSubmit() {
4      if (validate()) {
5          var userData = readUserData();
6          if (selectedRow == null)
7              {
8                  insertNewUserRecord(userData);
9              }
10         else
11             {
12                 updateUserRecord(userData);
13             }
14         resetPersonalInformationForm();
15     }
16 }
17
18 function resetPersonalInformationForm() {
19     document.getElementById("fullName").value = "";
20     document.getElementById("bldgNumber").value = "";
21     document.getElementById("streetAddress").value = "";
22     document.getElementById("age").value = "";
23     selectedRow = null;
24 }
25
26 function readUserData() {
27     var userData = {};
28     userData["fullName"] = document.getElementById("fullName").value;
29     userData["bldgNumber"] = document.getElementById("bldgNumber").value;
30     userData["streetAddress"] = document.getElementById("streetAddress").value;
31     userData["age"] = document.getElementById("age").value;
32     return userData;
33 }
34
35 function insertNewUserRecord(data) {
36     var table = document.getElementById("userlist").getElementsByTagName("tbody")[0];
37     var newRow = table.insertRow(table.length);
38     cell1 = newRow.insertCell(0);
39     cell1.innerHTML = data.fullName;
40     cell2 = newRow.insertCell(1);
41     cell2.innerHTML = data.bldgNumber;
42     cell3 = newRow.insertCell(2);
43     cell3.innerHTML = data.streetAddress;
44     cell4 = newRow.insertCell(3);
45     cell4.innerHTML = data.age;
46     cell5 = newRow.insertCell(4);
47     cell5.innerHTML = "<a onClick='onEditUser(this)'>Edit/<a>
48     <a onClick='onDeleteUser(this)'>Delete/<a>;
49 }
50
51 function onEditUser(td) {
52     selectedRow = td.parentElement.parentElement;
53     document.getElementById("fullName").value = selectedRow.cells[0].innerHTML;
54     document.getElementById("bldgNumber").value = selectedRow.cells[1].innerHTML;
55     document.getElementById("streetAddress").value = selectedRow.cells[2].innerHTML;
56     document.getElementById("age").value = selectedRow.cells[3].innerHTML;
57 }
58
59
60 function updateUserRecord(userData) {
61     selectedRow.cells[0].innerHTML = userData.fullName;
62     selectedRow.cells[1].innerHTML = userData.bldgNumber;
63     selectedRow.cells[2].innerHTML = userData.streetAddress;
64     selectedRow.cells[3].innerHTML = userData.age;
65 }
66
67 // Check User validation
68 function validate() {
69     isValid = true;
70     if (document.getElementById("fullName").value == "") {
71         isValid = false;
72         document.getElementById("fullNamevalidate").classList.remove("hide");
73     }
74     else {
75         isValid = true;
76         if (document.getElementById("fullNamevalidate").classList.contains("hide"))
77             {
78                 document.getElementById("fullNamevalidate").classList.add("hide");
79             }
80     }
81     if (document.getElementById("bldgNumber").value == "") {
82         isValid = false;
83         document.getElementById("bldgNumbervalidate").classList.remove("hide");
84     }
85     else {
86         isValid = true;
87         if (document.getElementById("bldgNumbervalidate").classList.contains("hide"))
88             {
89                 document.getElementById("bldgNumbervalidate").classList.add("hide");
90             }
91     }
92     if (document.getElementById("streetAddress").value == "") {
93         isValid = false;
94         document.getElementById("streetAddressvalidate").classList.remove("hide");
95     }
96     else {
97         isValid = true;
98         if (document.getElementById("streetAddressvalidate").classList.contains("hide"))
99             {
100                 document.getElementById("streetAddressvalidate").classList.add("hide");
101             }
102     }
103     if (document.getElementById("age").value == "") {
104         isValid = false;
105         document.getElementById("agevalidate").classList.remove("hide");
106     }
107     else {
108         isValid = true;
109         if (document.getElementById("agevalidate").classList.contains("hide"))
110             {
111                 document.getElementById("agevalidate").classList.add("hide");
112             }
113     }
114     return isValid;
115 }
116
117 function onDeleteUser(td) {
118     if (confirm("Are you sure to delete this user record?")) {
119         row = td.parentElement.parentElement;
120         document.getElementById("userlist").deleteRow(row.rowIndex);
121         resetPersonalInformationForm();
122     }
123 }

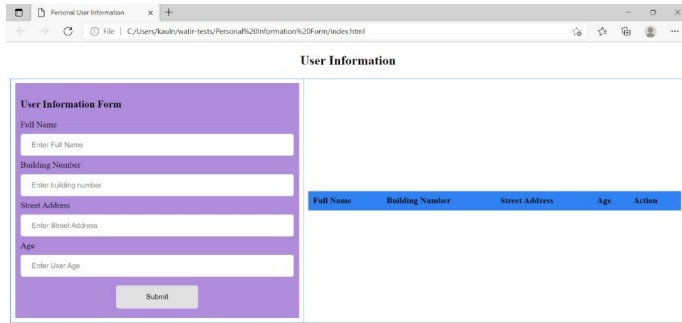
```

This file consists of several functions that perform CRUD operations such as insert, update, and delete. In addition to these methods, this file contains a validation method that validates whether each field is filled and throws an error if its empty.

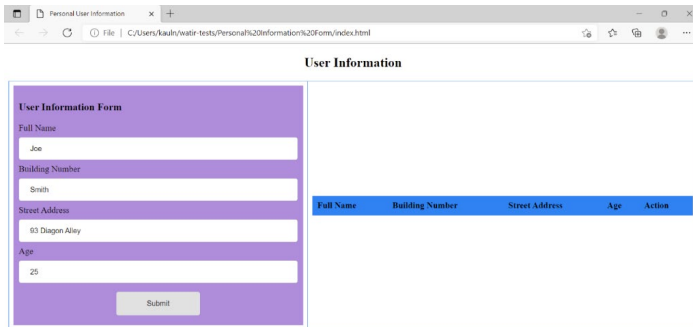
- c. **Stylesheet.css:** This file is a CSS file that defines the style for different elements of the html form.

```
ation Form)\css\stylesheet.css (Personal Information Form) - Sublime Text (UNREGISTERED)
Project Preferences Help
stylesheet.css x
1 body > table{
2   width: 100%;
3 }
4
5 table{
6   border-collapse: collapse;
7   text-align: center;
8 }
9 table.list{
10  width:100%;
11  text-align: center;
12 }
13 table.list td{
14  text-align: center;
15  background-color: #ccc;
16 }
17 td, th {
18  border: 1px solid #0076ee;
19  text-align: left;
20  padding: 8px;
21 }
22 tr:nth-child(even),table.list thead>tr {
23  background-color: #0076ee;
24 }
25 input[type=text], input[type=number] {
26  width: 100%;
27  padding: 12px 20px;
28  margin: 8px 0;
29  display: inline-block;
30  border: 1px solid #ccc;
31  border-radius: 4px;
32  box-sizing: border-box;
33 }
34 input[type=submit]{
35  width: 30%;
36  background-color: #ddd;
37  color: #000;
38  padding: 14px 20px;
39  margin: 8px 0;
40  border: none;
41  border-radius: 4px;
42  cursor: pointer;
43 }
44 form{
45  background-color: #ab81d6;
46  padding: 10px;
47 }
48 }
49 form div.form-action-buttons{
50  text-align: center;
51 }
52 }
53 a{
54  cursor: pointer;
55  text-decoration: underline;
56  color: #0000ee;
57  margin-right: 4px;
58 }
59 label.validation-error{
60  color: #990000;
61  margin-left: 5px;
62 }
63 .hide{
64  display:none;
65 }
```

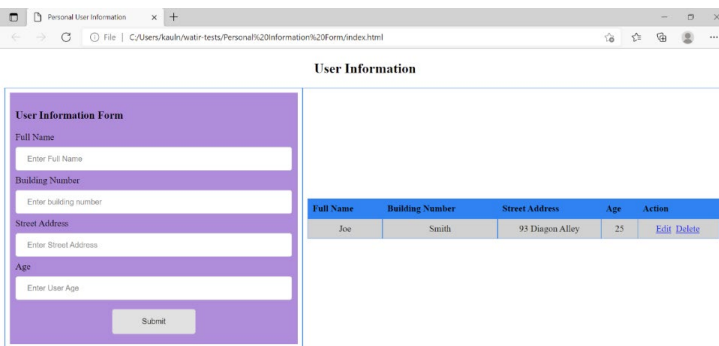
2. We proceed to open the html file in the browser and fill out the form to check if it works.



We fill in some dummy data.

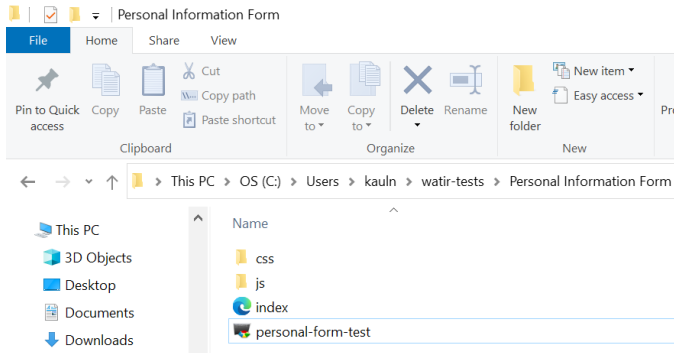


We click on Submit to process the data and display it in the adjoining table.



The data is displayed in the table as shown above.

3. We proceed to create a test case that does the same actions that we performed.



4. We edit this file to perform the actions of creating records automatically. The contents of this test case are as follows:

```

personal-form-test - Notepad
File Edit Format View Help
require 'watir'
browser = Watir::Browser.new :chrome
browser.goto('file:///C:/Users/kauln/watir-tests/Personal%20Information%20Form/index.html')
fullName = browser.text_field(name: 'fullName')
fullName.exists?
fullName.set 'Harry Potter'
bidgNumber = browser.text_field(name: 'bidgNumber')
bidgNumber.exists?
bidgNumber.set '4'
address = browser.text_field(name: 'streetAddress')
address.exists?
address.set 'Privet Drive'
age = browser.text_field(name: 'age')
age.exists?
age.set '13'
Watir::Wait.until(timeout: 60) { browser.text.include? 'Privet Drive' }
Ln 16, Col 72 100% Windows (CRLF) UTF-8

```

5. We run this file from the command line as follows:

```

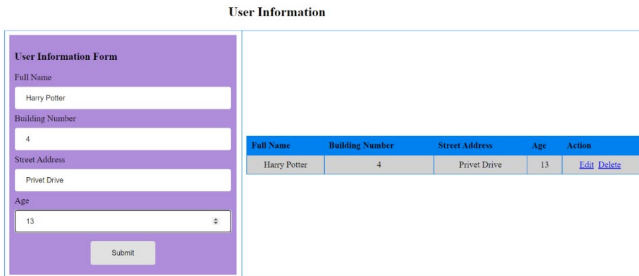
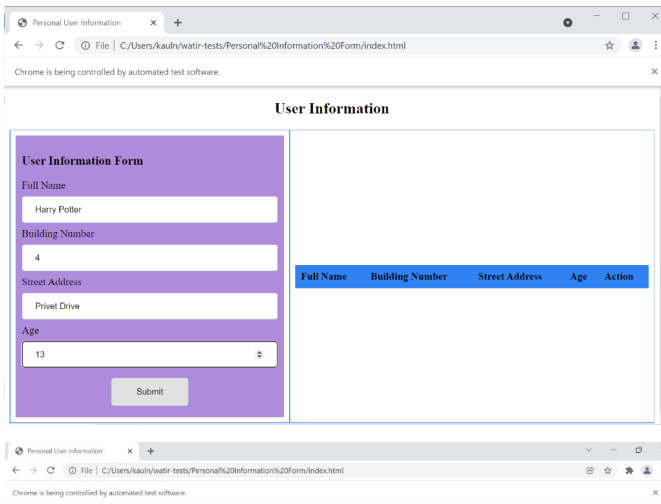
Command Prompt - irb
C:\Users\kauln>cd watir-tests
C:\Users\kauln\watir-tests>cd "Personal Information Form"
C:\Users\kauln\watir-tests\Personal Information Form>irb
irb(main):001:0> load './personal-form-test.rb'

```

```

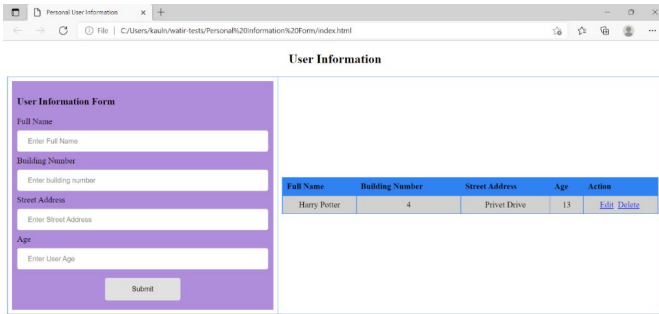
Select C:\Ruby31-x64\bin\ruby.exe
DevTools listening on ws://127.0.0.1:61737/devtools/browser/582aea4-c982-4784-bf1f-8c8de87cf998
[19896:21624:0115/192641.658:ERROR:chrome_browser_main_extra_parts_metrics.cc(227)] START: ReportBluetoothAvailability()
. If you don't see the END: message, this is crbug.com/1216328.
[19896:20836:0115/192641.658:ERROR:device_event_log_impl.cc(214)] [19:26:41.658] USB: usb_device_handle_win.cc:1050 Failed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[19896:21624:0115/192641.658:ERROR:chrome_browser_main_extra_parts_metrics.cc(238)] END: ReportBluetoothAvailability()
[19896:21624:0115/192641.660:ERROR:chrome_browser_main_extra_parts_metrics.cc(235)] START: GetDefaultBrowser(). If you don't see the END: message, this is crbug.com/1216328.
[19896:20836:0115/192641.660:ERROR:device_event_log_impl.cc(214)] [19:26:41.660] USB: usb_device_handle_win.cc:1050 Failed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[19896:21624:0115/192641.668:ERROR:chrome_browser_main_extra_parts_metrics.cc(239)] END: GetDefaultBrowser()
    
```

- The test is executed successfully, and the record is created as shown below:

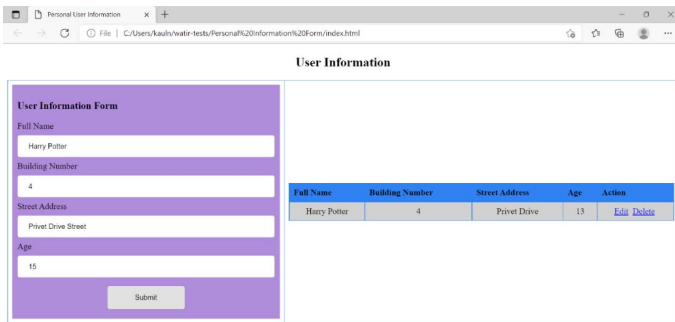


Example 7: Testing the Edit Functions in an html Form Using Watir: In this example, we will test the edit functionality of the html form that we created in the previous example.

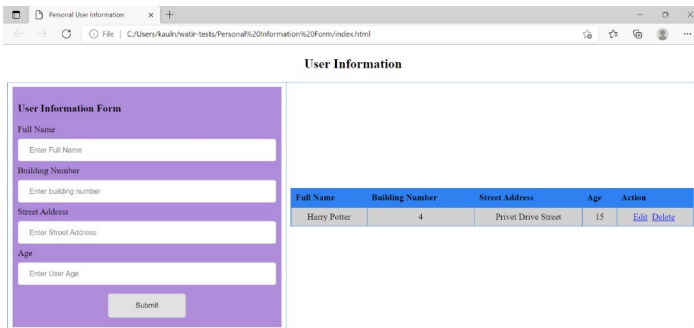
1. We begin by first testing the editing functionality. We enter data into the form and submit it. The data is loaded and then we click on the edit link to edit the data.



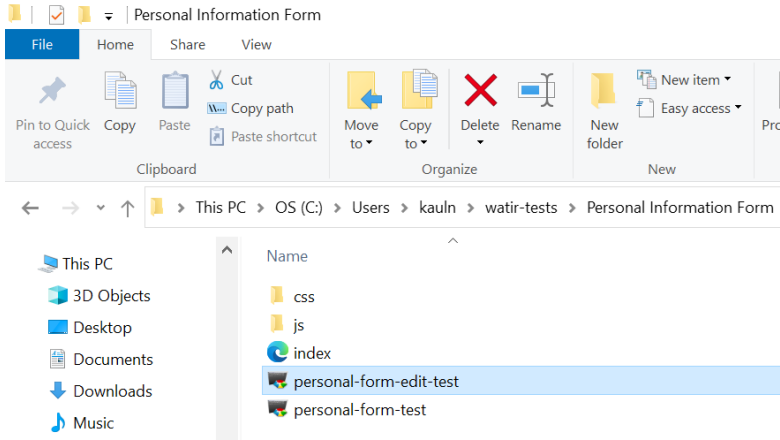
We edit the Street address and age of the same user record.



We click on submit to register the changes.



2. We proceed to create a test for this scenario. In the same folder as the index.html file, we create a new ruby file named personal-form-edit-test.rb.



We edit this file and create a watir test case that performs the same actions for deleting a user as we did manually.

```

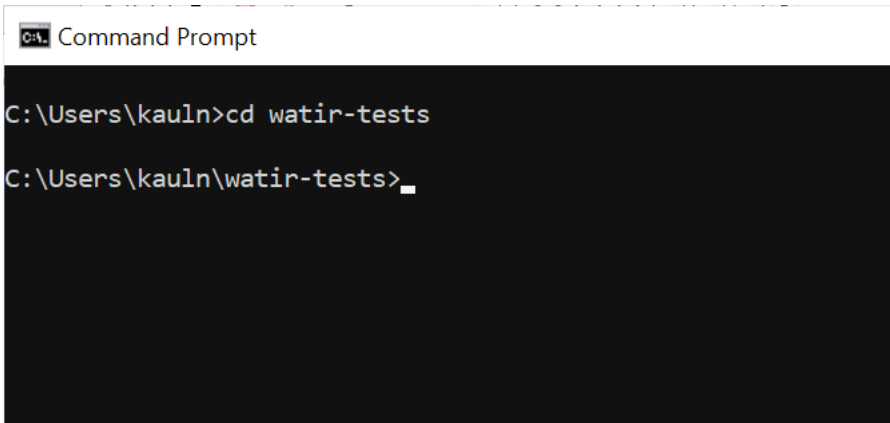
personal-form-edit-test - Notepad
File Edit Format View Help
require 'watir'
browser = Watir::Browser.new :chrome
browser.goto('file:///C:/Users/kauln/watir-tests/Personal%20Information%20Form/index.html')
fullName = browser.text_field(name: 'fullName')
fullName.exists?
fullName.set 'Harry Potter'
bldgNumber = browser.text_field(name: 'bldgNumber')
bldgNumber.exists?
bldgNumber.set '4'
address = browser.text_field(name: 'streetAddress')
address.exists?
address.set 'Privet Drive'
age = browser.text_field(name: 'age')
age.exists?
age.set '13'
browser.button(name: "submit").wait_until(&:present?).click
#display the links on the page
puts browser.links.collect(&:text)
#click the 'Edit link'
browser.link(:text => /Edit/).wait_until(&:present?)
browser.link(:text => /Edit/).click
address = browser.text_field(name: 'streetAddress')
address.exists?
address.set 'Privet Drive Street'
age = browser.text_field(name: 'age')
age.exists?
age.set '11'
sleep 30
browser.button(name: "submit").wait_until(&:present?).click
sleep 30

```

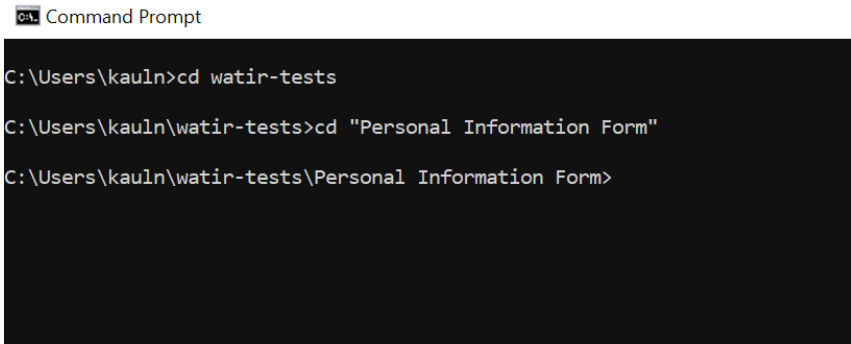
In this test case, we add to the previously created test case ‘personal-test.rb.’ So, this new test consists of opening the form, adding a record, and submitting it. We now add the steps to edit a record that is displayed in the table. First, we click on the edit link using the ‘browser.link’ functionality. After clicking the link, we proceed to edit some information in the user. As we can see from the screenshot of the code above, we edit the fields Street Address and age. We set the Street address to ‘Privet Drive Street’ and age to 11.

3. The next step is to execute the test case.

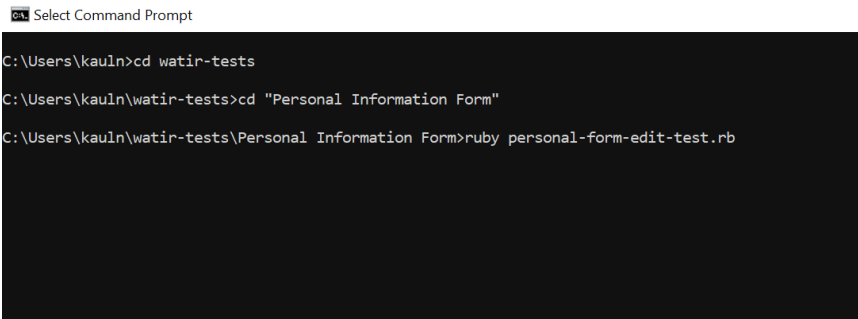
We run the test case from the command prompt as follows:



```
CA. Command Prompt
C:\Users\kauIn>cd watir-tests
C:\Users\kauIn\watir-tests>
```



```
CA. Command Prompt
C:\Users\kauIn>cd watir-tests
C:\Users\kauIn\watir-tests>cd "Personal Information Form"
C:\Users\kauIn\watir-tests\Personal Information Form>
```



```
CA. Select Command Prompt
C:\Users\kauIn>cd watir-tests
C:\Users\kauIn\watir-tests>cd "Personal Information Form"
C:\Users\kauIn\watir-tests\Personal Information Form>ruby personal-form-edit-test.rb
```

Note that here we use the command ‘ruby <filename.rb>’ to execute the test case. This is an easier way of running ruby files.

The test case is executed as shown below:


```

Command Prompt - ruby personal-form-edit-test.rb
C:\Users\kauln>cd watir-tests
C:\Users\kauln\watir-tests>cd "Personal Information Form"
C:\Users\kauln\watir-tests\Personal Information Form>ruby personal-form-edit-test.rb
DevTools listening on ws://127.0.0.1:50692/devtools/browser/03fe8bc2-a3e8-4ff9-9f09-373b11cc7a0e
Edit
Delete
[29620:26072:0116/004712.562:ERROR:device_event_log_impl.cc(214)] [00:47:12.562] USB: usb_device_handle_win.cc:1050 Failed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[29620:984:0116/004712.562:ERROR:chrome_browser_main_extra_parts_metrics.cc(227)] START: ReportBluetoothAvailability(). If you don't see the END: message, this is crbug.com/1216328.
[29620:26072:0116/004712.562:ERROR:device_event_log_impl.cc(214)] [00:47:12.562] USB: usb_device_handle_win.cc:1050 Failed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[29620:984:0116/004712.563:ERROR:chrome_browser_main_extra_parts_metrics.cc(230)] END: ReportBluetoothAvailability()
[29620:984:0116/004712.564:ERROR:chrome_browser_main_extra_parts_metrics.cc(235)] START: GetDefaultBrowser(). If you don't see the END: message, this is crbug.com/1216328.
[29620:984:0116/004712.572:ERROR:chrome_browser_main_extra_parts_metrics.cc(239)] END: GetDefaultBrowser()

```

4. The automation opens the page in the Chrome browser by default and the changes are made.

First, the data is entered, and the table is populated with the entered data. This is shown below:

The screenshot shows a web browser window with the address bar displaying the URL: `C:/Users/kauln/watir-tests/Personal%20Information%20Form/index.html`. The page title is "Personal User Information". A notification bar at the top states "Chrome is being controlled by automated test software." The main content area is titled "User Information" and contains a form and a table.

User Information Form

Full Name

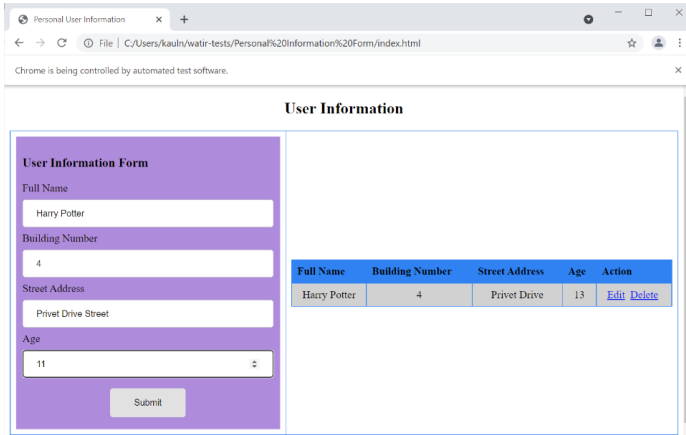
Building Number

Street Address

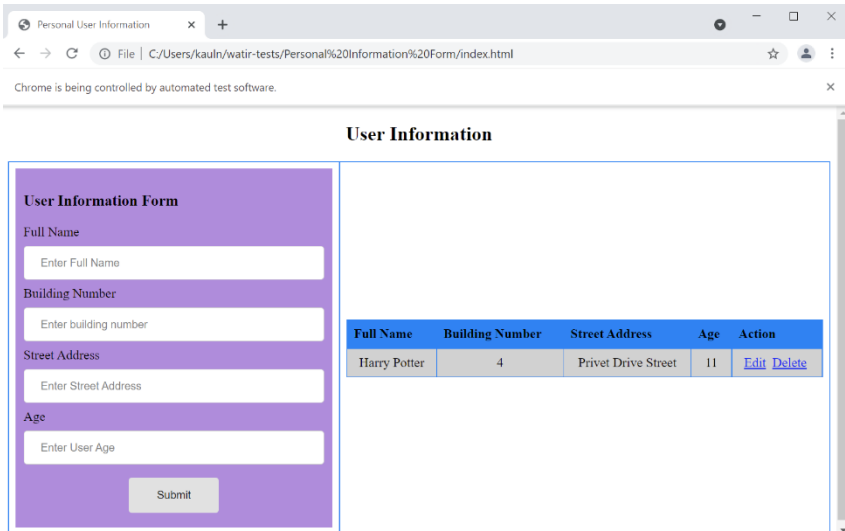
Age

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	13	Edit Delete

Then, the 'Edit' link is clicked and the information pertaining to that record is loaded in the form. The new changes are made in the form as shown below:



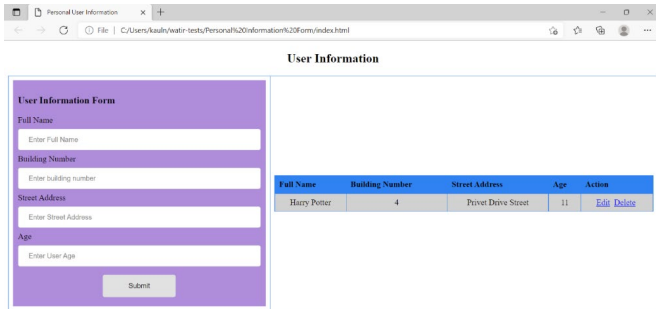
The 'Submit' button is clicked to register the changes as seen below:



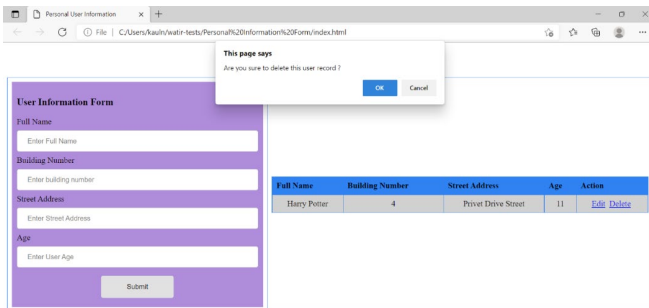
- Example 8: Deleting a User Record from an html Page:** In this example, we build up on the previous example and proceed to delete a user record after the editing is done. Here, we would be accessing the form, populating the form, creating a record, editing it, and then deleting the record.

The steps are:

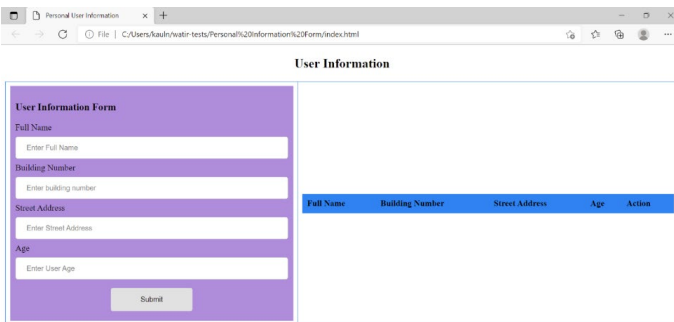
1. First, we open the html page to test the delete link. We create a record and then click on the 'Delete' link.



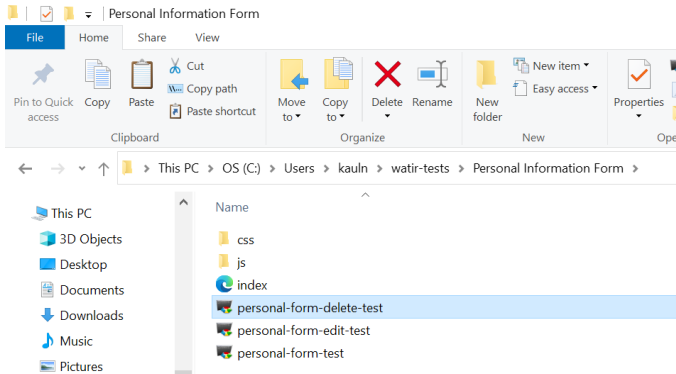
A popup dialog asks us to confirm if we want to proceed with the deletion. We click on 'OK.'



The record is deleted from the table is shown below:



- We then create a new test file in the same location as the html form and name it 'personal-form-delete-test.rb.'



We edit this file in notepad and add the commands that would click on the 'Delete' link. In this example, like the previous one, we build up on the test case that we wrote previously. Thus, our new test case for testing the deletion link consists of opening the page, entering a record, editing it, saving it followed by the delete action.

This is shown below:

personal-form-delete-test - Notepad

```
File Edit Format View Help
require 'watir'
browser = Watir::Browser.new :chrome
browser.goto('file:///C:/Users/kauln/watir-tests/Personal%20Information%20Form/index.html')
fullName = browser.text_field(name: 'fullName')
fullName.exists?
fullName.set 'Harry Potter'
bldgNumber = browser.text_field(name: 'bldgNumber')
bldgNumber.exists?
bldgNumber.set '4'
address = browser.text_field(name: 'streetAddress')
address.exists?
address.set 'Privet Drive'
age = browser.text_field(name: 'age')
age.exists?
age.set '13'
browser.button(name: "submit").wait_until(&.present?).click
#display the links on the page
puts browser.links.collect(&:text)
#click the 'Edit link'
sleep 30
browser.link(:text => /Edit/).wait_until(&.present?)
browser.link(:text => /Edit/).click
address = browser.text_field(name: 'streetAddress')
address.exists?
address.set 'Privet Drive Street'
age = browser.text_field(name: 'age')
age.exists?
age.set '11'
sleep 30
browser.button(name: "submit").wait_until(&.present?).click
#click the 'Delete' link
browser.link(:text => /Delete/).wait_until(&.present?)
browser.link(:text => /Delete/).click
#click OK on alert confirmation
browser.alert.exists?
browser.alert.text
browser.alert.ok
sleep 30
```

We make use of the `browser.link` command to check if the link exists and then click on it. Then we confirm that we wish to delete the record. We have added sleep commands throughout the test case to be able to slow down the test and take screenshots.

3. We run the test from the command line as shown below:

```

Command Prompt

C:\Users\kauln>cd watir-tests
C:\Users\kauln\watir-tests>cd "Personal Information Form"
C:\Users\kauln\watir-tests\Personal Information Form>ruby personal-form-delete-test.rb

```

The test is executed as shown below:

```

Command Prompt - ruby personal-form-delete-test.rb

C:\Users\kauln>cd watir-tests
C:\Users\kauln\watir-tests>cd "Personal Information Form"
C:\Users\kauln\watir-tests\Personal Information Form>ruby personal-form-delete-test.rb

DevTools listening on ws://127.0.0.1:53424/devtools/browser/bbd2ca73-2298-430f-a376-deadf3ff2b31
Edit
Delete
[41596:36484:0116/093234.272:ERROR:chrome_browser_main_extra_parts_metrics.cc(227)] START: ReportBluetoothAvailability()
. If you don't see the END: message, this is crbug.com/1216328.
[41596:36484:0116/093234.272:ERROR:chrome_browser_main_extra_parts_metrics.cc(230)] END: ReportBluetoothAvailability()
[41596:36484:0116/093234.274:ERROR:chrome_browser_main_extra_parts_metrics.cc(235)] START: GetDefaultBrowser(). If you d
on't see the END: message, this is crbug.com/1216328.
[41596:36484:0116/093234.279:ERROR:chrome_browser_main_extra_parts_metrics.cc(239)] END: GetDefaultBrowser()
[41596:28432:0116/093234.279:ERROR:device_event_log_impl.cc(214)] [09:32:34.280] USB: usb_device_handle_win.cc:1050 Fail
ed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[41596:28432:0116/093234.280:ERROR:device_event_log_impl.cc(214)] [09:32:34.281] USB: usb_device_handle_win.cc:1050 Fail
ed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)

```

4. The automation launches the chrome browser and actions entered in the test case are performed.

The record is first entered and then edited.

The screenshot shows a web browser window titled 'Personal User Information' with the URL 'C:\Users\kauln\watir-tests\Personal%20Information%20Form\index.html'. The page content is as follows:

User Information

User Information Form

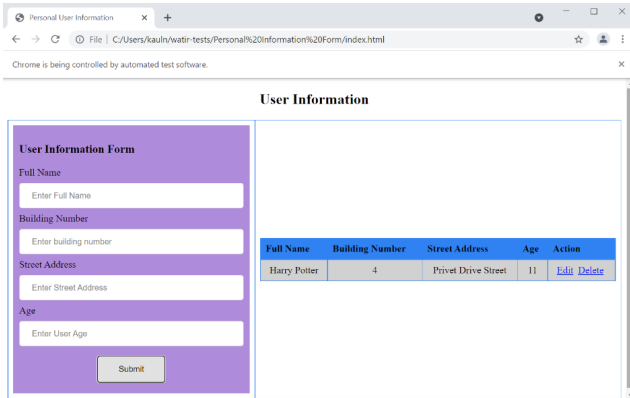
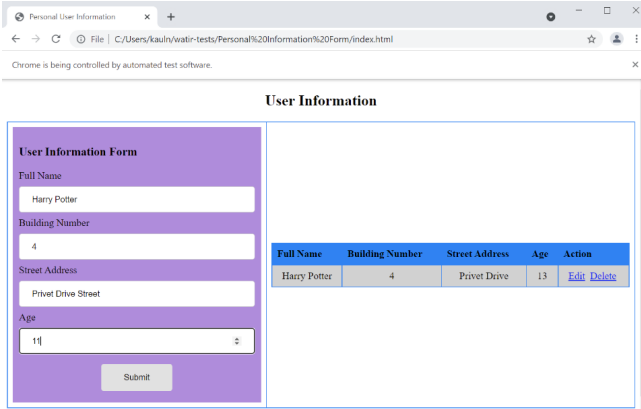
Full Name

Building Number

Street Address

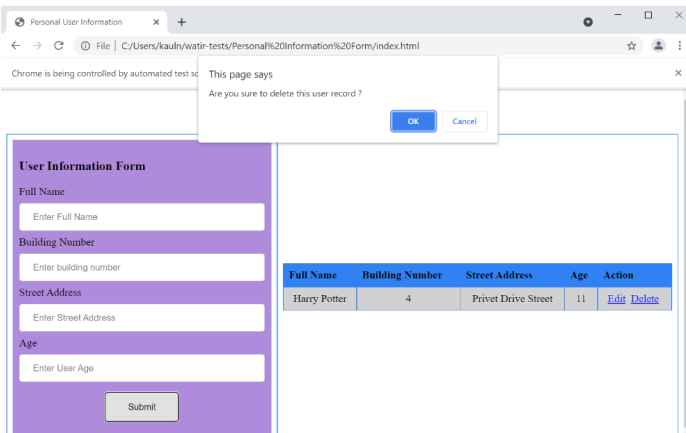
Age

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	13	Edit Delete

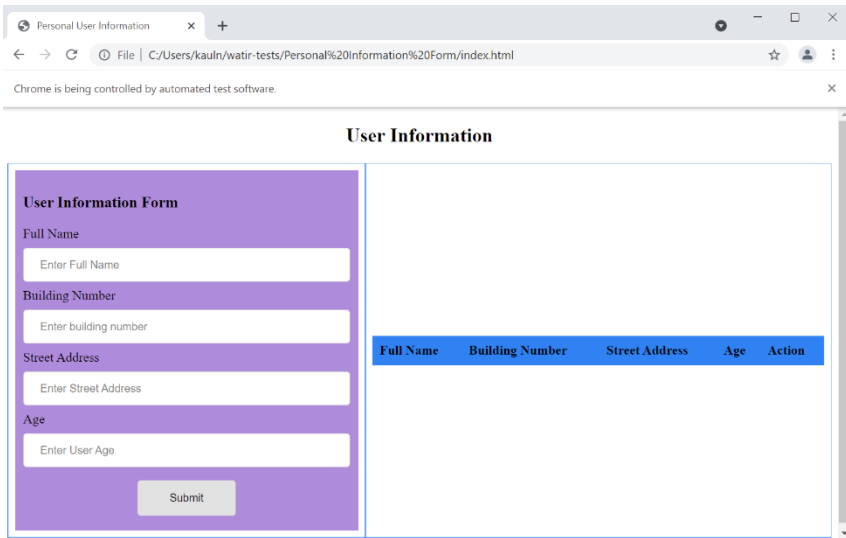


The edited record is saved, as shown in the screenshot above.

The delete link is clicked which opens the confirmation popup as shown below:



The ‘OK’ button on the confirmation alert dialog is clicked and the record is deleted successfully as shown below:

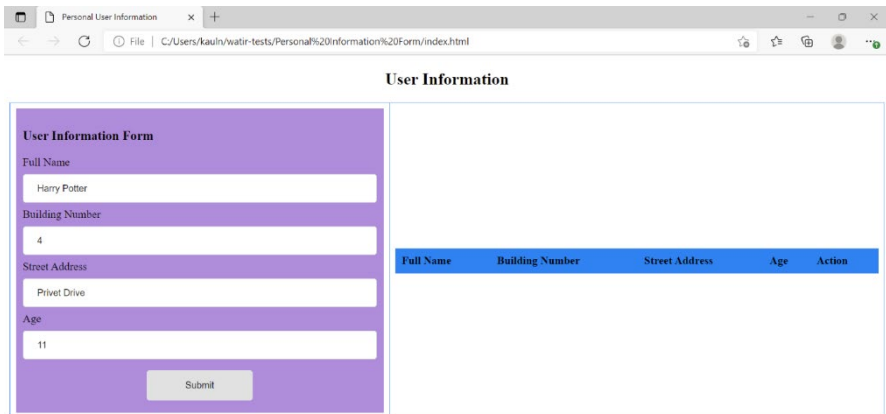


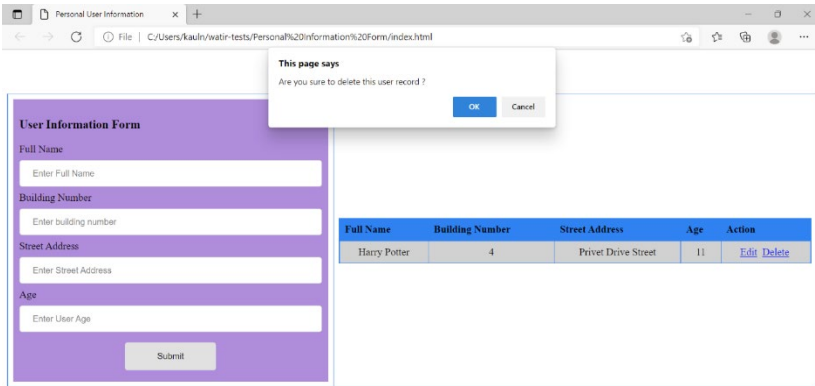
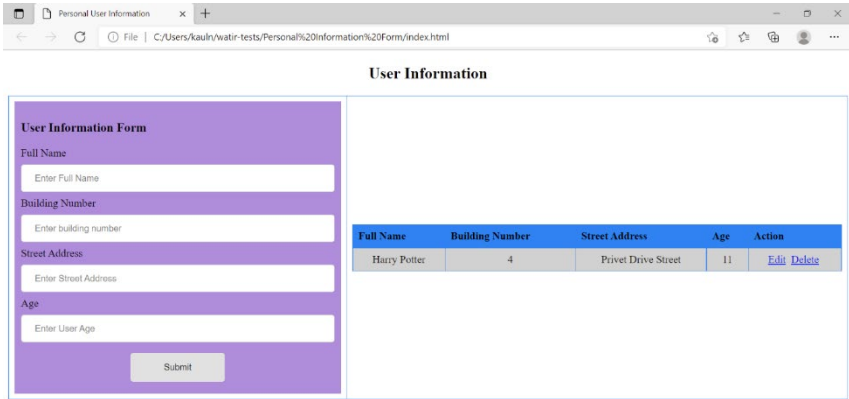
- **Example 9: Canceling a Deletion Request on an html Page Using Watir:** In this example, we will proceed to delete a user record, but cancel the operation.

We proceed to complete all the operations from the previous example and when we get the confirmation alert, we cancel the operation.

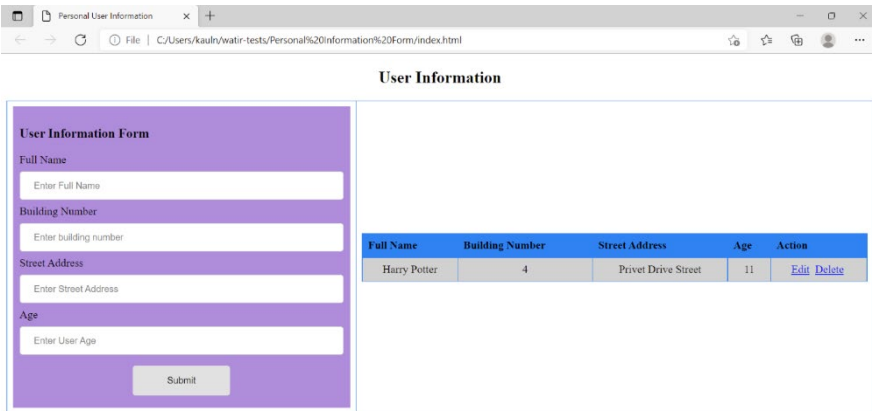
The steps are as follows:

1. We create a new record and click on the ‘Delete’ link.



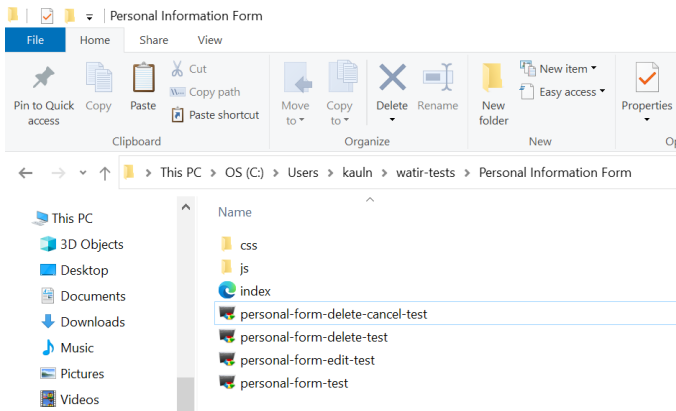


We click on 'Cancel' to cancel the delete operation.



2. We create a new ruby test file named personal-form-delete-cancel-test.rb.

The test case is very similar to the previous example but here we click on cancel instead of ok when the alert confirmation message pops up.



3. The contents of the test case are as follows:

```
personal-form-delete-test - Notepad
File Edit Format View Help
require 'watir'
browser = Watir::Browser.new :chrome
browser.goto('file:///C:/Users/kauln/watir-tests/Personal%20Information%20Form/index.html')
fullName = browser.text_field(name: 'fullName')
fullName.exists?
fullName.set 'Harry Potter'
bldgNumber = browser.text_field(name: 'bldgNumber')
bldgNumber.exists?
bldgNumber.set '4'
address = browser.text_field(name: 'streetAddress')
address.exists?
address.set 'Privet Drive'
age = browser.text_field(name: 'age')
age.exists?
age.set '13'
browser.button(name: "submit").wait_until(&:present?).click
#display the links on the page
puts browser.links.collect(&:text)
#click the 'Edit link'
sleep 30
browser.link(:text => /Edit/).wait_until(&:present?)
browser.link(:text => /Edit/).click
address = browser.text_field(name: 'streetAddress')
address.exists?
address.set 'Privet Drive Street'
age = browser.text_field(name: 'age')
age.exists?
age.set '11'
sleep 30
browser.button(name: "submit").wait_until(&:present?).click
#click the 'Delete' link
browser.link(:text => /Delete/).wait_until(&:present?)
browser.link(:text => /Delete/).click
#click OK on alert confirmation
browser.alert.exists?
browser.alert.text
browser.alert.close
sleep 30
```

4. To run the test, we execute the ruby file from the command prompt. This is shown below:

```

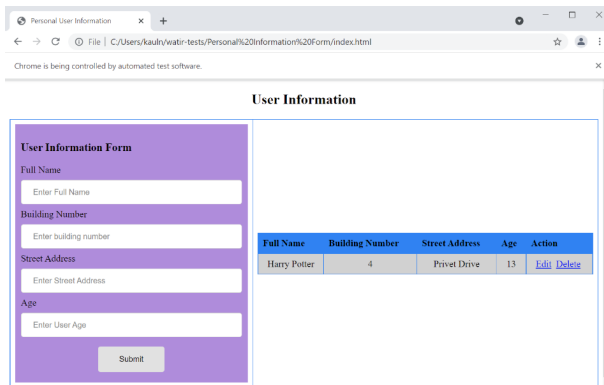
Command Prompt
C:\Users\kauin>cd watir-tests
C:\Users\kauin\watir-tests>cd "Personal Information Form"
C:\Users\kauin\watir-tests\Personal Information Form>ruby personal-form-delete-cancel-test.rb
    
```

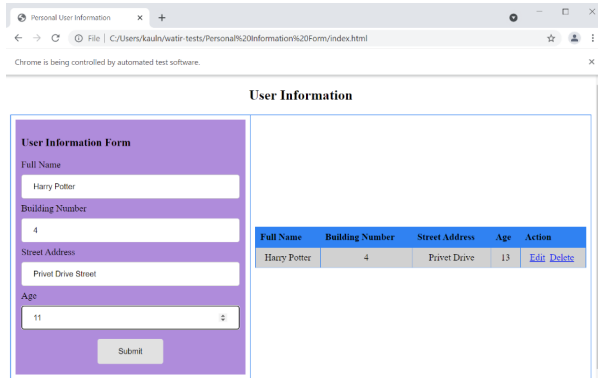
```

Select Command Prompt - ruby personal-form-delete-cancel-test.rb
C:\Users\kauin>cd watir-tests
C:\Users\kauin\watir-tests>cd "Personal Information Form"
C:\Users\kauin\watir-tests\Personal Information Form>ruby personal-form-delete-cancel-test.rb
DevTools listening on ws://127.0.0.1:59428/devtools/browser/9d2e8c16-7db6-43f6-bd65-5ee467bdb18e
Edit
Delete
[16204:4864:0118/092832.627:ERROR:chrome_browser_main_extra_parts_metrics.cc(227)] START: ReportBluetoothAvailability().
If you don't see the END: message, this is crbug.com/1216328.
[16204:4864:0118/092832.628:ERROR:chrome_browser_main_extra_parts_metrics.cc(230)] END: ReportBluetoothAvailability()
[16204:4864:0118/092832.629:ERROR:chrome_browser_main_extra_parts_metrics.cc(235)] START: GetDefaultBrowser(). If you do
n't see the END: message, this is crbug.com/1216328.
[16204:39328:0118/092832.630:ERROR:device_event_log_impl.cc(214)] [09:28:32.630] USB: usb_device_handle_win.cc:1050 Fail
ed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[16204:39328:0118/092832.631:ERROR:device_event_log_impl.cc(214)] [09:28:32.631] USB: usb_device_handle_win.cc:1050 Fail
ed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[16204:4864:0118/092832.635:ERROR:chrome_browser_main_extra_parts_metrics.cc(239)] END: GetDefaultBrowser()
    
```

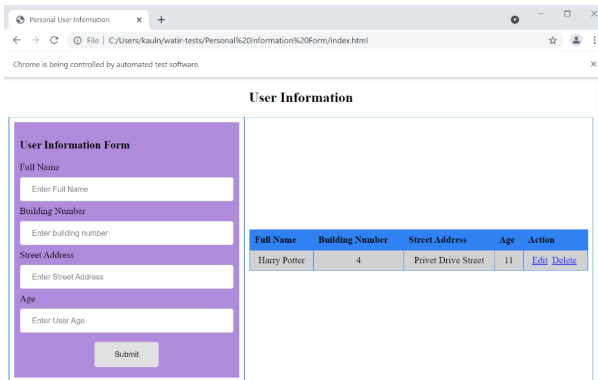
5. The file is executed, and the automation executes the commands from the test file one by one in the default browser.

A record is created as shown below:

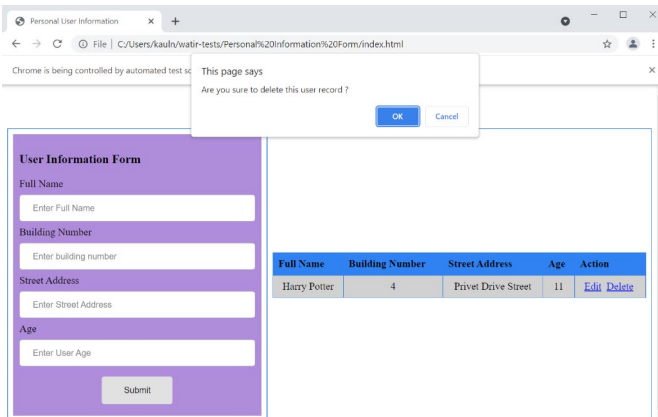




The delete link for this record is clicked and the alert dialog pops up.



The automation clicks on 'Cancel' and the entry remains.



Personal User Information x +

File | C:\Users\kauln\waair-tests\Personal%20Information%20Form\index.html

Chrome is being controlled by automated test software.

User Information

User Information Form

Full Name
Enter Full Name

Building Number
Enter building number

Street Address
Enter Street Address

Age
Enter User Age

Submit

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive Street	11	Edit Delete

- Example 10: Validating the Fields:** In this example, we will test the simple validation implemented on all the 4 fields on the personal information form.
 - First, we test the validation on the form manually. This form has a validation implemented on all the input fields that shows error message when you try to submit a field that is empty.

Personal User Information x +

File | C:\Users\kauln\waair-tests\Personal%20Information%20Form\index.html

User Information

User Information Form

Full Name
Enter Full Name

Building Number
Enter building number

Street Address
Enter Street Address

Age
Enter User Age

Submit

Full Name	Building Number	Street Address	Age	Action
-----------	-----------------	----------------	-----	--------

Personal User Information x +

File | C:\Users\kauln\waair-tests\Personal%20Information%20Form\index.html

User Information

User Information Form

Full Name Full name should not be empty!
Enter Full Name

Building Number Building number should not be empty!
Enter building number

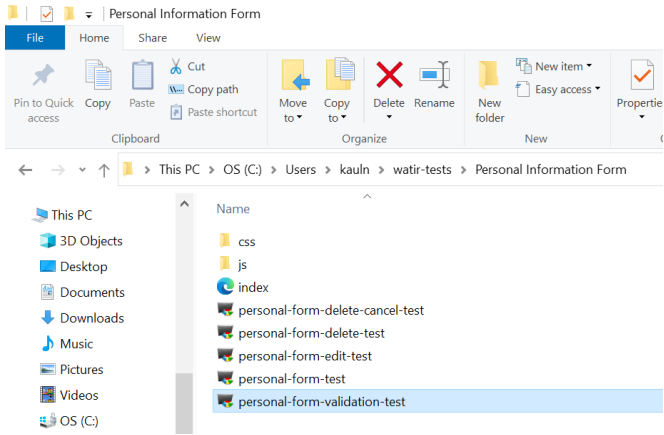
Street Address Street Address should not be empty!
Enter Street Address

Age Age should not be empty!
Enter User Age

Submit

Full Name	Building Number	Street Address	Age	Action
-----------	-----------------	----------------	-----	--------

2. We create a new test file named personal-form-validation-test.rb.



The contents of this file are as follows:

```

personal-form-validation-test - Notepad
File Edit Format View Help
require 'watir'
browser = Watir::Browser.new :chrome
browser.goto('file:///C:/Users/kauln/watir-tests/Personal%20Information%20Form/index.html')
browser.button(name: "submit").wait_until(&:present?).click
sleep 30
fullName = browser.text_field(name: 'fullName')
fullName.exists?
fullName.set 'Harry Potter'
bldgNumber = browser.text_field(name: 'bldgNumber')
bldgNumber.exists?
bldgNumber.set '4'
address = browser.text_field(name: 'streetAddress')
address.exists?
address.set 'Privet Drive'
sleep 30
browser.button(name: "submit").wait_until(&:present?).click
sleep 30
age = browser.text_field(name: 'age')
age.exists?
age.set '13'
sleep 30
browser.button(name: "submit").wait_until(&:present?).click
sleep 30
Ln 24, Col 1      100%  Windows (CRLF)  UTF-8

```

3. We execute the test from the command prompt as shown below:

```

Command Prompt
C:\Users\kauln>cd watir-tests
C:\Users\kauln\watir-tests>cd "Personal Information Form"
C:\Users\kauln\watir-tests\Personal Information Form>ruby personal-form-validation-test.rb_

```

```

Command Prompt - ruby_personal-form-validation-test.rb
C:\Users\kau\>cd watir-tests
C:\Users\kau\watir-tests>cd "Personal Information Form"
C:\Users\kau\watir-tests\Personal Information Form>ruby personal-form-validation-test.rb
DevTools listening on ws://127.0.0.1:54038/devtools/browser/21d2a42b-1716-446f-a216-0639c5918f3f
[38388:39120:0118/101249.297:ERROR:chrome_browser_main_extra_parts_metrics.cc(227)] START: ReportBluetoothAvailability()
. If you don't see the END: message, this is crbug.com/1216328.
[38388:39120:0118/101249.297:ERROR:chrome_browser_main_extra_parts_metrics.cc(238)] END: ReportBluetoothAvailability()
[38388:39120:0118/101249.298:ERROR:chrome_browser_main_extra_parts_metrics.cc(235)] START: GetDefaultBrowser(). If you d
on't see the END: message, this is crbug.com/1216328.
[38388:17248:0118/101249.299:ERROR:device_event_log_impl.cc(214)] [10:12:49.299] USB: usb_device_handle_win.cc:1050 Fail
ed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[38388:17248:0118/101249.301:ERROR:device_event_log_impl.cc(214)] [10:12:49.301] USB: usb_device_handle_win.cc:1050 Fail
ed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[38388:39120:0118/101249.305:ERROR:chrome_browser_main_extra_parts_metrics.cc(239)] END: GetDefaultBrowser()

```

- The automation mimics our actions and click on the ‘Submit’ button without entering any data first. The validation errors appear as expected.

Personal User Information

Chrome is being controlled by automated test software.

User Information

User Information Form

Full Name *Full name should not be empty!*

Building Number *Building number should not be empty!*

Street Address *Street Address should not be empty!*

Age *Age should not be empty!*

Full Name	Building Number	Street Address	Age	Action

The data is then entered by the automation except for the ‘Age’ field.

Personal User Information

Chrome is being controlled by automated test software.

User Information

User Information Form

Full Name *Full name should not be empty!*

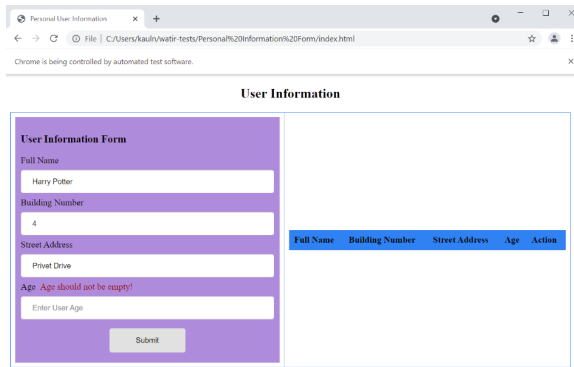
Building Number *Building number should not be empty!*

Street Address *Street Address should not be empty!*

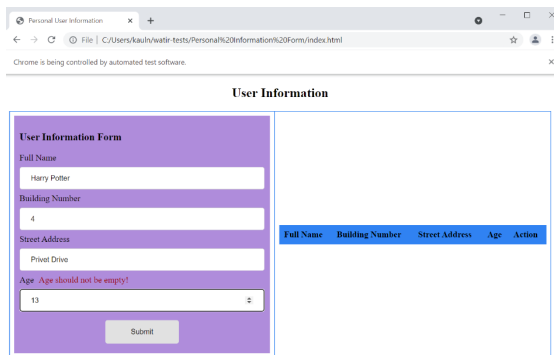
Age *Age should not be empty!*

Full Name	Building Number	Street Address	Age	Action

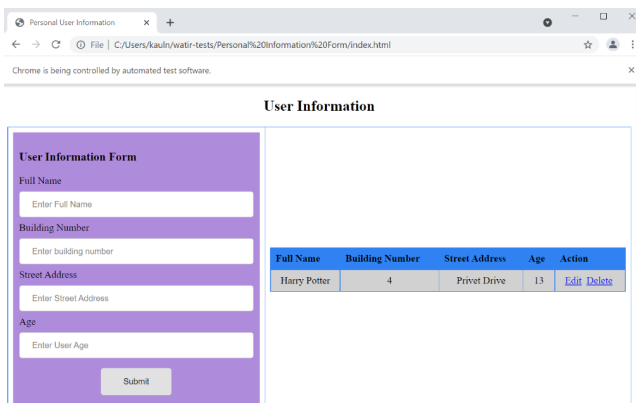
As clear from the screenshot below, all fields except the ‘Age’ field are filled in the form.



On submitting this incomplete form, we get the validation error for the ‘age’ field as follows:



Finally, a value is entered for this field and the record is created.



- **Example 11: Manipulating Multiple Records:** In this example, we will be testing the personal information form by populating it with multiple records and performing crud actions on the records.

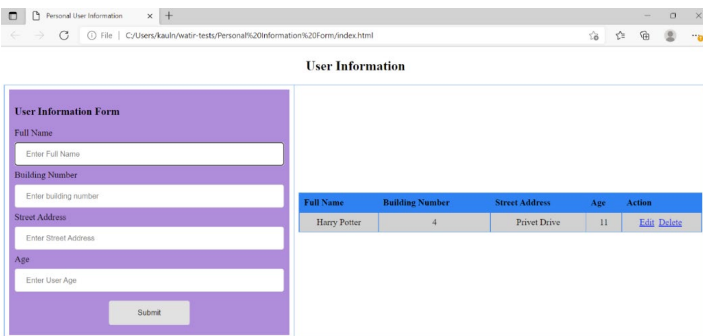
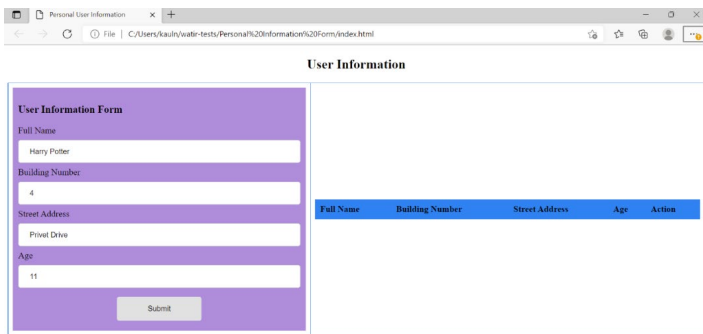
The sequence of events that we will follow in this example are as follows:

- Creating three records;
- Editing a record;
- Testing the validation of a field;
- Deleting a record;
- Canceling a deletion.

The steps are as follows:

1. First, we complete the actions manually in the browser.

Three records are created first.



User Information

User Information Form

Full Name
Ronald Weasley

Building Number
1

Street Address
Diagon Alley

Age
11

Submit

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	11	Edit Delete

User Information

User Information Form

Full Name
Enter Full Name

Building Number
Enter building number

Street Address
Enter Street Address

Age
Enter User Age

Submit

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	11	Edit Delete
Ronald Weasley	1	Diagon Alley	11	Edit Delete

User Information

User Information Form

Full Name
Hermione Granger

Building Number
12

Street Address
Grimmauld Place

Age
11

Submit

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	11	Edit Delete
Ronald Weasley	1	Diagon Alley	11	Edit Delete

User Information

User Information Form

Full Name
Enter Full Name

Building Number
Enter building number

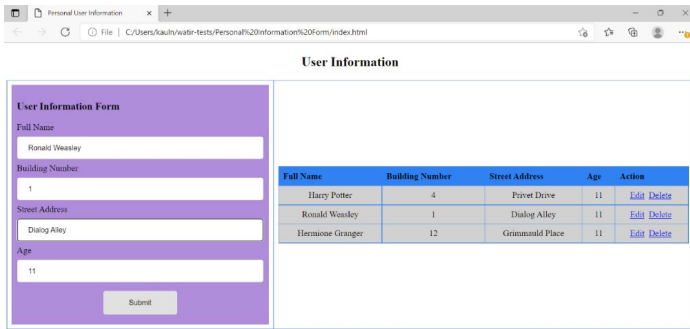
Street Address
Enter Street Address

Age
Enter User Age

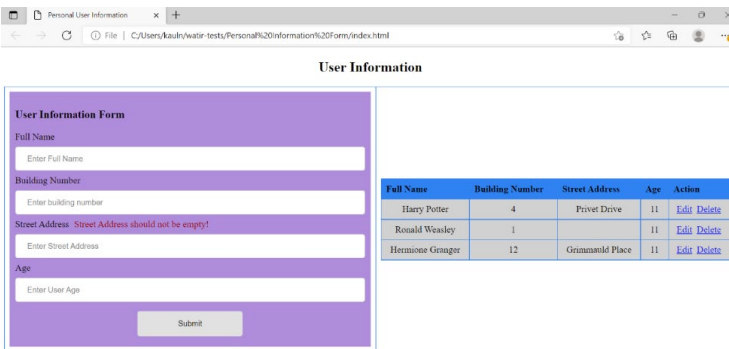
Submit

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	11	Edit Delete
Ronald Weasley	1	Diagon Alley	11	Edit Delete
Hermione Granger	12	Grimmauld Place	11	Edit Delete

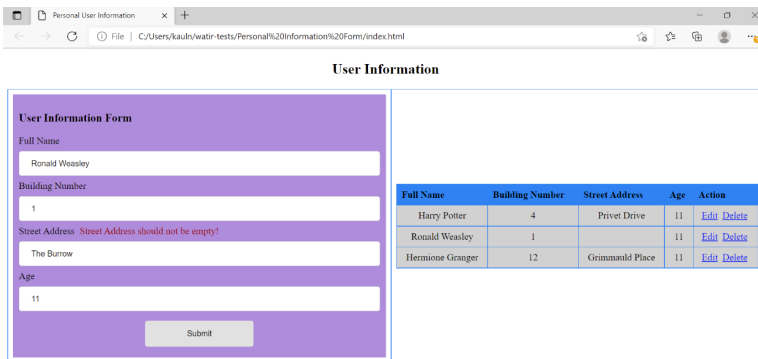
Then, we edit the 2nd record by clicking on the edit link located at index 1. For watir, the indexes begin at 0 (Watir Project, 2017).

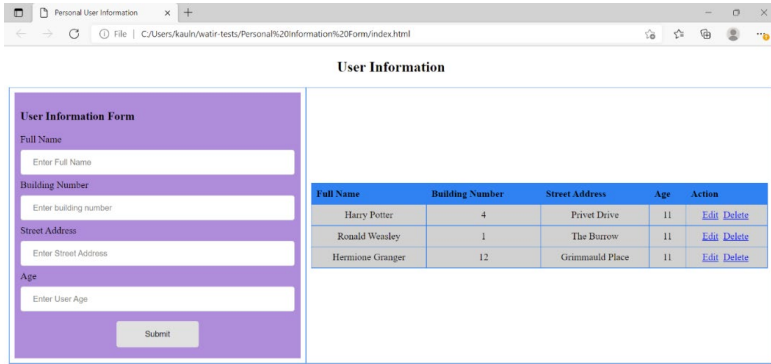


While editing, the ‘address’ field is left empty to test if the validation error is thrown. The validation error is thrown as seen in the image below:

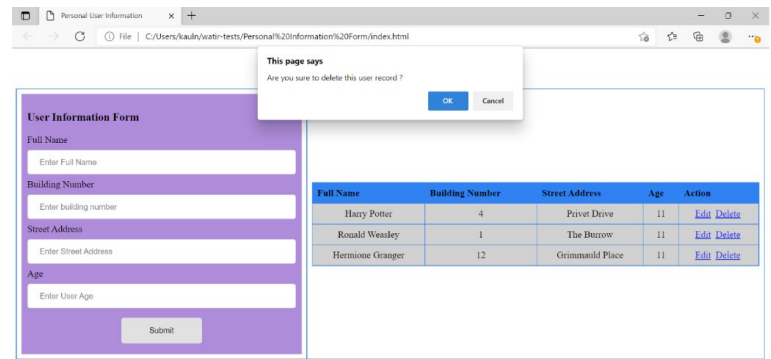
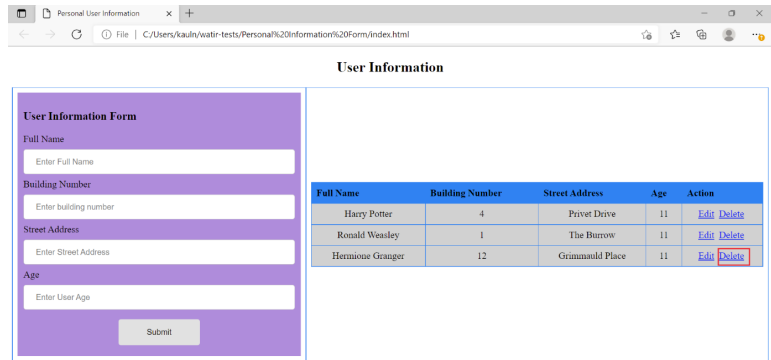


The address is then filled, and the record is updated.

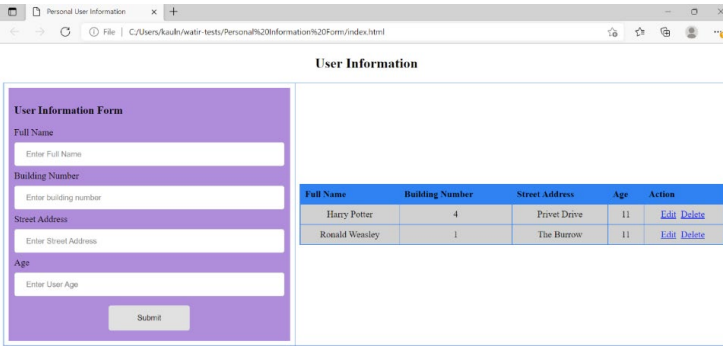




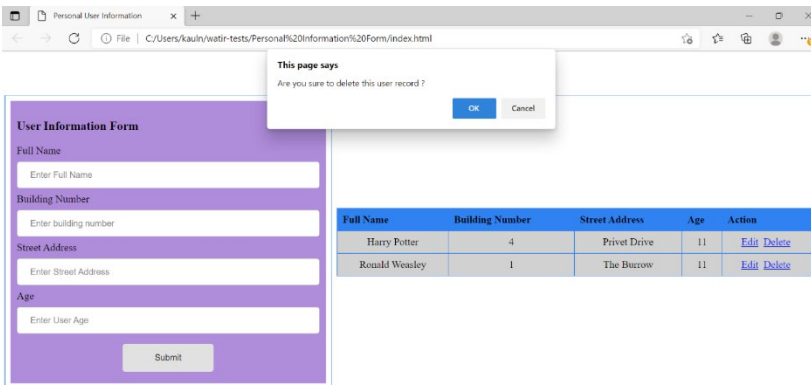
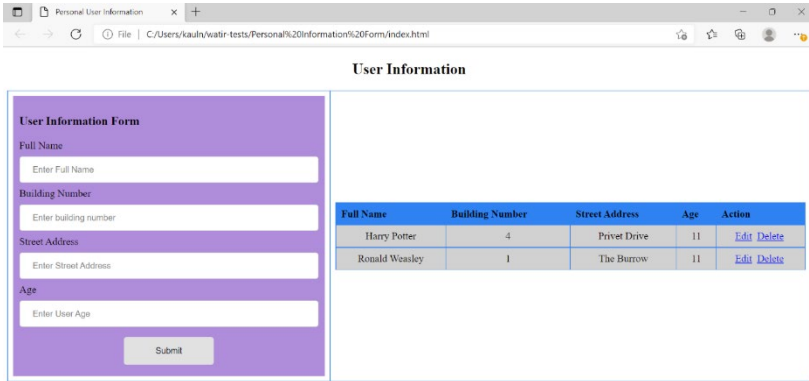
We then proceed to deleting the last record.



The record is deleted as shown below:



The last thing we do is to click on the delete link for the 2nd record and then cancel the deletion request.



The screenshot shows a web browser window with the title 'Personal User Information'. The address bar shows the URL: `C:/Users/kauln/watir-tests/Personal%20Information%20Form/index.html`. The page content is titled 'User Information' and is divided into two main sections.

The left section is a 'User Information Form' with a purple background. It contains the following fields:

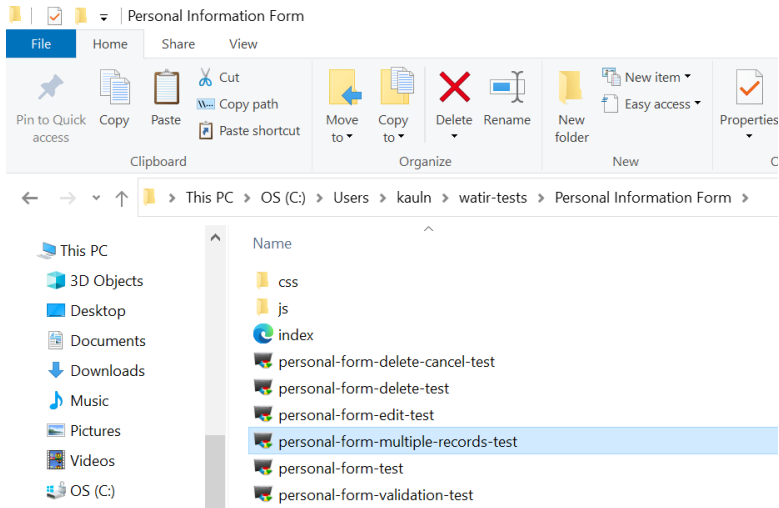
- Full Name:** A text input field with the placeholder text 'Enter Full Name'.
- Building Number:** A text input field with the placeholder text 'Enter building number'.
- Street Address:** A text input field with the placeholder text 'Enter Street Address'.
- Age:** A text input field with the placeholder text 'Enter User Age'.

Below these fields is a 'Submit' button.

The right section contains a table with the following data:

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	11	Edit Delete
Ronald Weasley	1	The Burrow	11	Edit Delete

- The next step is to create a test file. We create the test file in the same location as before.



We edit this file and write the commands that we need to complete all the actions we did manually.

The contents of this file are as follows:

sonal-form-multiple-records-test - Notepad

```

dit Format View Help
re 'watir'
er = Watir::Browser.new :chrome
er.goto('file:///C:/Users/kauln/watir-tests/Personal%20Information%20Form/index.html')
ame = browser.text_field(name: 'fullName')
ame.exists?
ame.set 'Harry Potter'
mber = browser.text_field(name: 'bldgNumber')
mber.exists?
mber.set '4'
ss = browser.text_field(name: 'streetAddress')
ss.exists?
ss.set 'Privet Drive'
browser.text_field(name: 'age')
ists?
at '11'
er.button(name: "submit").wait_until(&:present?).click
ame = browser.text_field(name: 'fullName')
ame.exists?
ame.set 'Ronald Weasley'
mber = browser.text_field(name: 'bldgNumber')
mber.exists?
mber.set '1'
ss = browser.text_field(name: 'streetAddress')
ss.exists?
ss.set 'Diagon Alley'
browser.text_field(name: 'age')
ists?
at '11'
er.button(name: "submit").wait_until(&:present?).click
ame = browser.text_field(name: 'fullName')
ame.exists?
ame.set 'Hermione Granger'
mber = browser.text_field(name: 'bldgNumber')
mber.exists?
mber.set '12'
ss = browser.text_field(name: 'streetAddress')
ss.exists?
ss.set 'Grimmauld Place'
browser.text_field(name: 'age')
ists?
at '11'
er.button(name: "submit").wait_until(&:present?).click
lay the links on the page
rowser.links.collect(&:text)
< the 'Edit' link and edit the 2nd record with index 1
er.link(:text => /Edit/).wait_until(&:present?)
er.link(:text => /Edit/, :index => 1).click
ss = browser.text_field(name: 'streetAddress')
ss.exists?
the validation by entering empty address while editing
ss.set ''
er.button(name: "submit").wait_until(&:present?).click
the address again and submit
er.link(:text => /Edit/).wait_until(&:present?)
er.link(:text => /Edit/, :index => 1).click
ss = browser.text_field(name: 'streetAddress')
ss.exists?
ss.set 'The Burrow'
er.button(name: "submit").wait_until(&:present?).click
te the 3rd record at index 2
er.link(:text => /Delete/).wait_until(&:present?)
er.link(:text => /Delete/, :index => 2).click
er.button(name: "submit").wait_until(&:present?).click

```

3. We execute the ruby file from the command prompt as follows:

```

Command Prompt
C:\Users\kauIn>cd watir-tests
C:\Users\kauIn\watir-tests>cd "Personal Information Form"
C:\Users\kauIn\watir-tests\Personal Information Form>ruby personal-form-multiple-records-test.rb

```

```

Command Prompt - ruby personal-form-multiple-records-test.rb
C:\Users\kauIn\watir-tests\Personal Information Form>ruby personal-form-multiple-records-test.rb
DevTools listening on ws://127.0.0.1:63100/devtools/browser/dc1cb2ca-0056-4500-a292-ab64486e2bd7
Edit
Delete
Edit
Delete
Edit
Delete
[29484:38580:0118/222401.686:ERROR:chrome_browser_main_extra_parts_metrics.cc(227)] START: ReportBluetoothAvailability()
. If you don't see the END: message, this is crbug.com/1216328.
[29484:38580:0118/222401.687:ERROR:chrome_browser_main_extra_parts_metrics.cc(230)] END: ReportBluetoothAvailability()
[29484:38580:0118/222401.687:ERROR:chrome_browser_main_extra_parts_metrics.cc(235)] START: GetDefaultBrowser(). If you d
on't see the END: message, this is crbug.com/1216328.
[29484:41112:0118/222401.687:ERROR:device_event_log_impl.cc(214)] [22:24:01.688] USB: usb_device_handle_win.cc:1050 Fail
ed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[29484:41112:0118/222401.688:ERROR:device_event_log_impl.cc(214)] [22:24:01.689] USB: usb_device_handle_win.cc:1050 Fail
ed to read descriptor from node connection: A device attached to the system is not functioning. (0x1F)
[29484:38580:0118/222401.691:ERROR:chrome_browser_main_extra_parts_metrics.cc(239)] END: GetDefaultBrowser()

```

4. The automation executes the actions described in the ruby file. First, the records are created.

The screenshot shows a web browser window with the URL `C:\Users\kauIn\watir-tests\Personal%20Information%20Form\index.html`. The page title is "User Information". On the left, there is a "User Information Form" with the following fields:

- Full Name:
- Building Number:
- Street Address:
- Age:

Below the form is a "Submit" button. On the right, there is a table with the following structure:

Full Name	Building Number	Street Address	Age	Action

Personal User Information x +
 C:/Users/kauln/eaatn-tests/Personal%20Information%20Form/index.html
 Chrome is being controlled by automated test software.

User Information

User Information Form

Full Name

Building Number

Street Address

Age

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	11	Edit Delete

Personal User Information x +
 C:/Users/kauln/eaatn-tests/Personal%20Information%20Form/index.html
 Chrome is being controlled by automated test software.

User Information

User Information Form

Full Name

Building Number

Street Address

Age

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	11	Edit Delete

Personal User Information x +
 C:/Users/kauln/eaatn-tests/Personal%20Information%20Form/index.html
 Chrome is being controlled by automated test software.

User Information

User Information Form

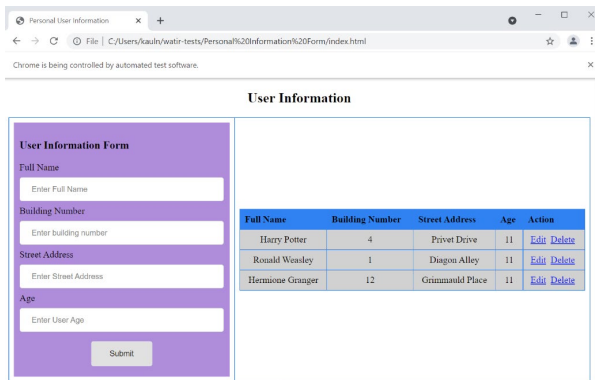
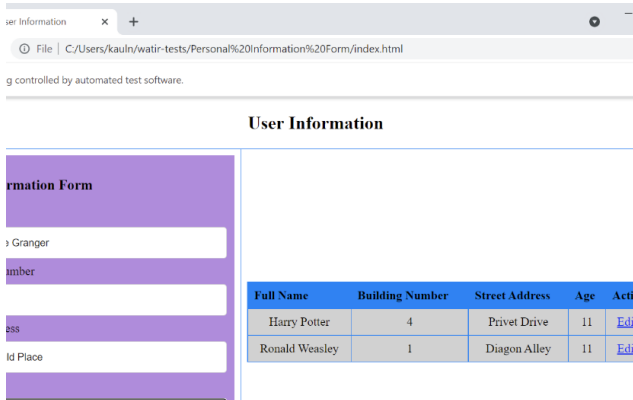
Full Name

Building Number

Street Address

Age

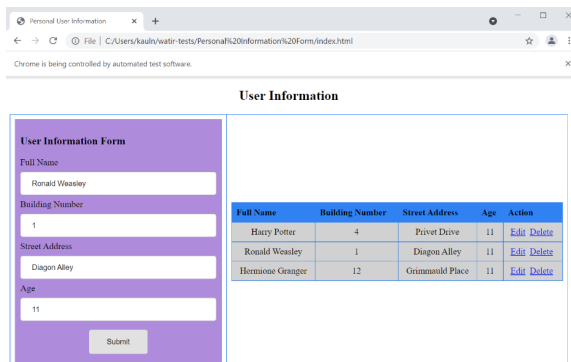
Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	11	Edit Delete
Ronald Weasley	1	Diagon Alley	11	Edit Delete



Then, the second record is edited, and the validation on the ‘address’ field is tested. We edit the 2nd record by providing the index in the ‘browser.link’ command as follows:

```
browser.link(:text => /Edit/, :index => 1).click
```

In watir, the index begins at 0 (Watir Project, 2017).



Personal User Information

Chrome is being controlled by automated test software.

User Information

User Information Form

Full Name
Ronald Weasley

Building Number
1

Street Address
Enter Street Address

Age
11

Submit

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	11	Edit Delete
Ronald Weasley	1	Diagon Alley	11	Edit Delete
Hermione Granger	12	Grimmauld Place	11	Edit Delete

Personal User Information

Chrome is being controlled by automated test software.

User Information

User Information Form

Full Name
Enter Full Name

Building Number
Enter building number

Street Address **Street Address should not be empty!**
Enter Street Address

Age
Enter User Age

Submit

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	11	Edit Delete
Ronald Weasley	1		11	Edit Delete
Hermione Granger	12	Grimmauld Place	11	Edit Delete

Once the validation for the 'address' field is tested, data is entered into the address field and the record is updated. This is shown below:

Personal User Information

Chrome is being controlled by automated test software.

User Information

User Information Form

Full Name
Ronald Weasley

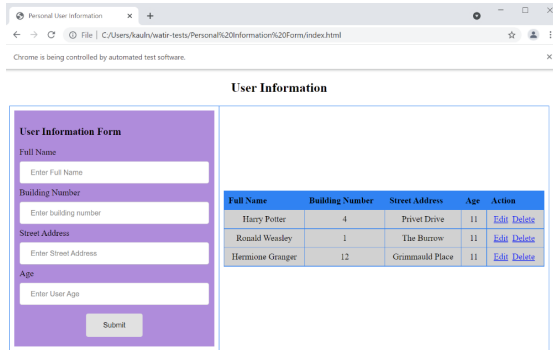
Building Number
1

Street Address **Street Address should not be empty!**
The Burrow

Age
11

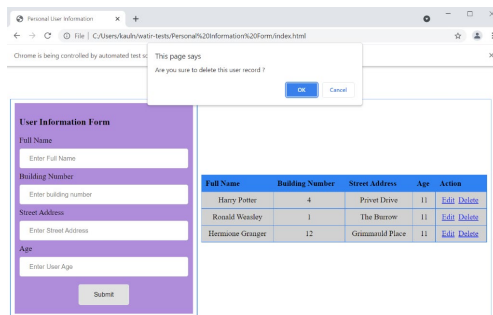
Submit

Full Name	Building Number	Street Address	Age	Action
Harry Potter	4	Privet Drive	11	Edit Delete
Ronald Weasley	1		11	Edit Delete
Hermione Granger	12	Grimmauld Place	11	Edit Delete

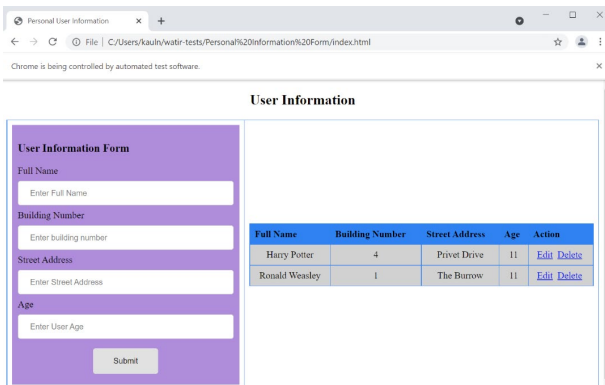


The automation proceeds to deleting the last record. This is done by providing the index of the delete link in the 'browser.link' command as follows:

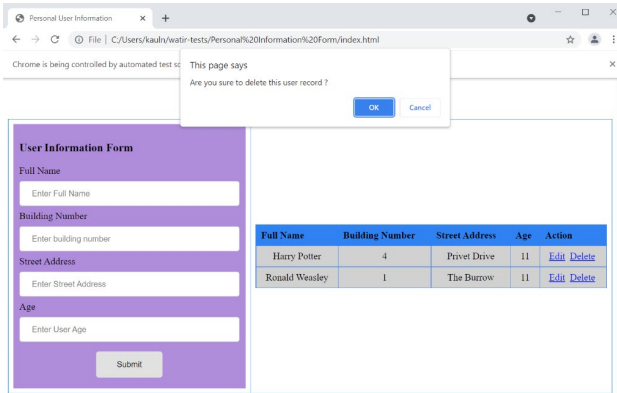
```
browser.link(:text => /Delete/, :index => 2).click
```



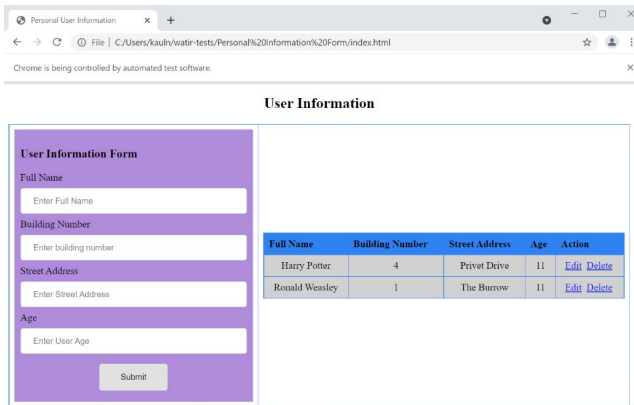
The record is deleted as shown:



Next, the automation tries to delete the 2nd record from the remaining records. The alert confirmation dialog opens, and the automation cancels this request.



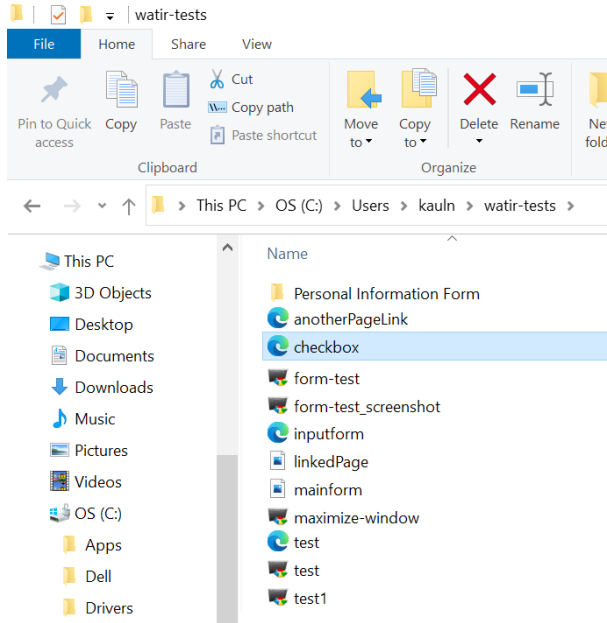
The delete operation is canceled as shown below:



- Example 12: Testing Checkboxes on a UI Page:** In this example, we test a simple html page containing checkboxes using watir. We will also be using an IDE to edit and run our watir test case.

The steps are as follows:

- First, we create a simple html page containing a few checkboxes. We create a new file named checkbox.html in the 'watir-tests' folder.



2. We edit this file using an html file editor to add the required html and JavaScript code needed to display checkboxes and display the selected values.

The contents of this file are as follows:

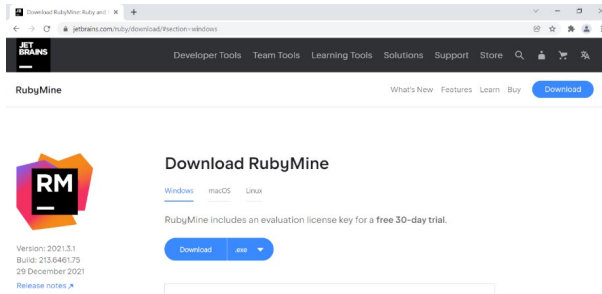
```

C:\Users\kauln\watir-test\checkbox.html - Sublime Text [UNREGISTERED]
File Edit Selection Find View Goto Tools Project Preferences Help

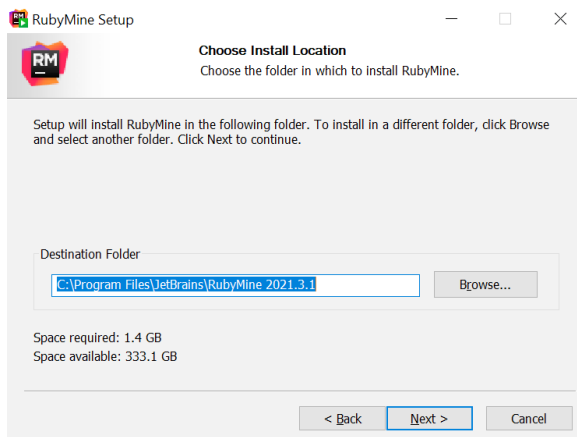
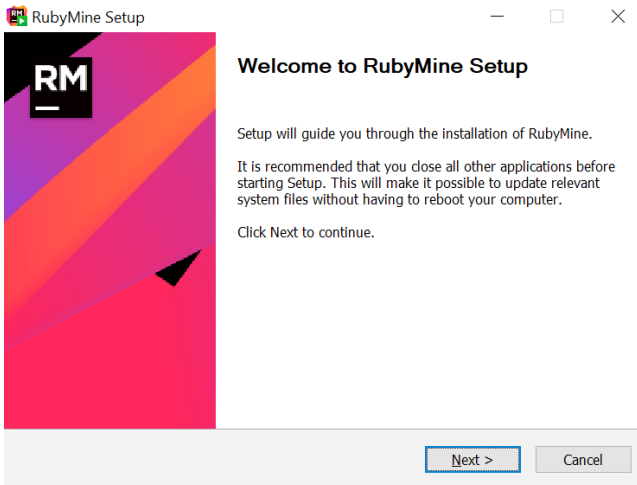
checkbox.html
1 <html>
2 <body>
3 <h3> Select your favourite fruits </h3>
4 <div>
5 <td> Orange: <input type="checkbox" id="Orange" class="pl" value="Orange" />
6 <td> Apple: <input type="checkbox" id="Apple" class="pl" value="Apple" />
7 </td> <td>
8 <td> Kiwi: <input type="checkbox" id="Kiwi" class="pl" value="Kiwi" />
9 <td> Watermelon: <input type="checkbox" id="Watermelon" class="pl" value="Watermelon" />
10 </td> <td>
11 <td> Banana: <input type="checkbox" id="Banana" class="pl" value="Banana" />
12 <button name="submit" onclick="getCheckboxValue()">Submit</button> <br>
13 <div style="color:blue" id="result"></div>
14
15 <script>
16 function getCheckboxValue() {
17   var orange = document.getElementById("Orange");
18   var apple = document.getElementById("Apple");
19   var kiwi = document.getElementById("kiwi");
20   var watermelon = document.getElementById("Watermelon");
21   var banana = document.getElementById("Banana");
22
23   var result = "";
24   if (orange.checked == true){
25     var porange = document.getElementById("Orange").value;
26     result = result + porange + ",";
27   }
28   if (apple.checked == true){
29     var papple = document.getElementById("Apple").value;
30     result = result + papple + ",";
31   }
32   if (kiwi.checked == true){
33     document.write(result);
34     var pkiwi = document.getElementById("kiwi").value;
35     result = result + pkiwi + ",";
36   }
37   if (watermelon.checked == true){
38     var pwatermelon = document.getElementById("Watermelon").value;
39     result = result + pwatermelon + ",";
40   }
41   if (banana.checked == true){
42     var pbanana = document.getElementById("Banana").value;
43     result = result + pbanana + ",";
44   }
45
46   return document.getElementById("result").innerHTML = "You have selected the following fruit/fruits:" + result;
47 }
48 </script>
49 </body>
50 </html>

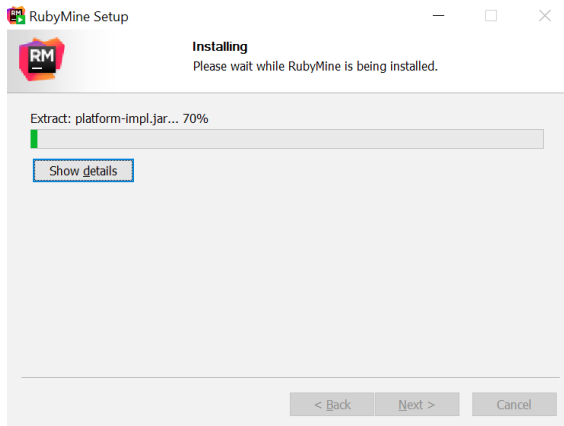
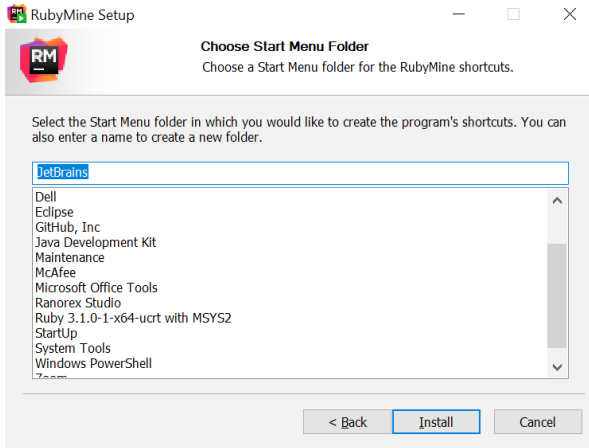
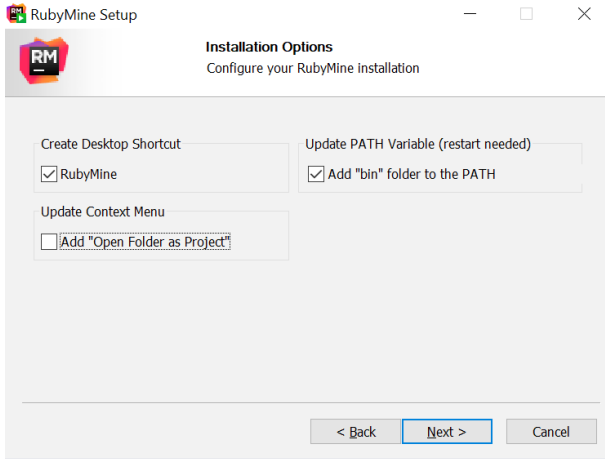
```

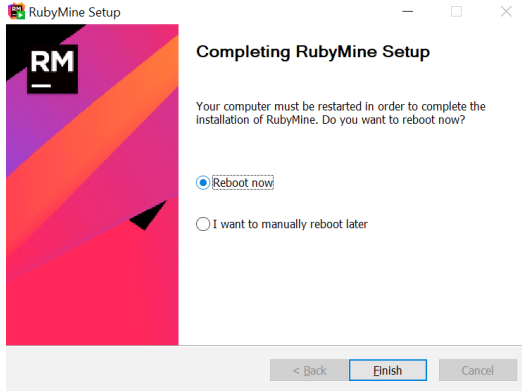
- We will now proceed to creating a ruby file and write our test case using watir. Before that, we install the RubyMine IDE from the JetBrains website: <https://www.jetbrains.com/ruby/>



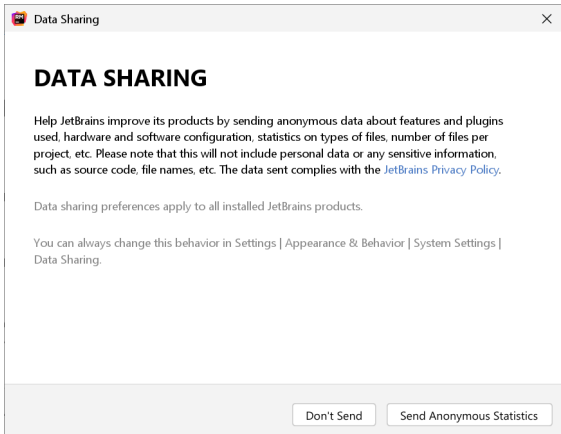
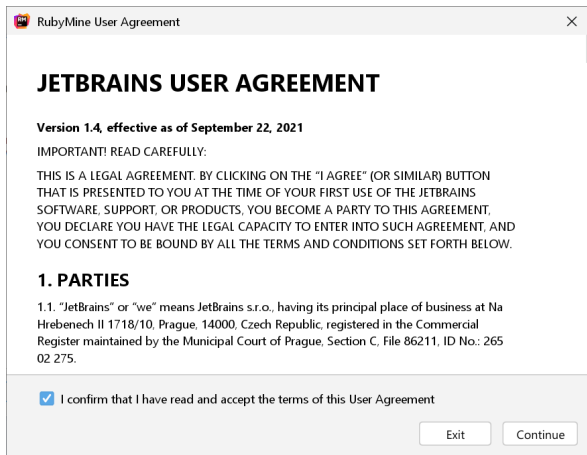
We download the exe for windows and proceed with the setup.



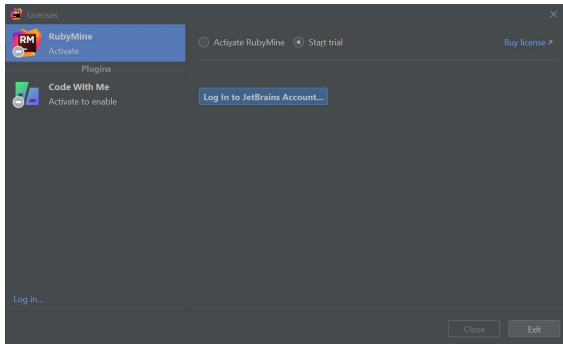




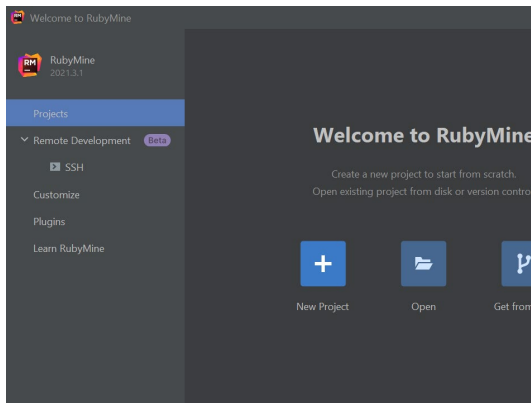
Once we reboot the system, we open the IDE. We accept the user agreement and proceed.



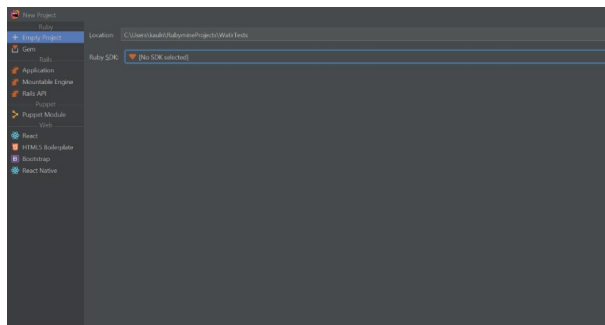
When the IDE opens, the first thing you are asked to do is to activate your account. Here, we start a free 30-day trial. RubyMine IDE is a commercially licensed IDE (RubyMine: The Ruby on Rails IDE by JetBrains, 2022).

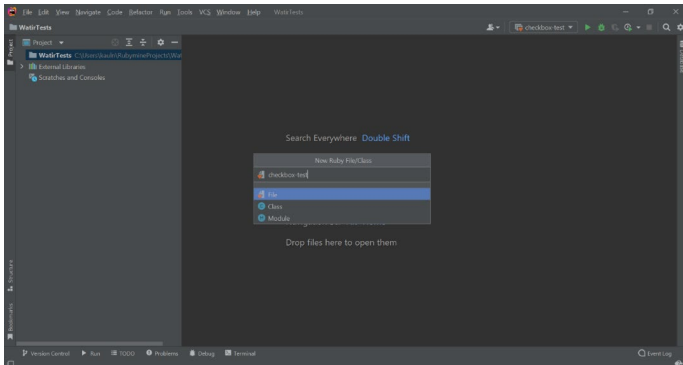
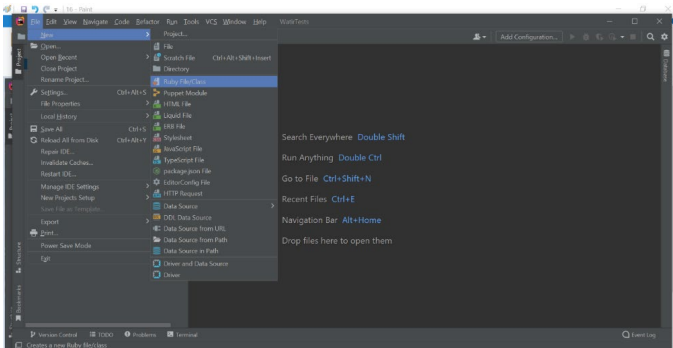
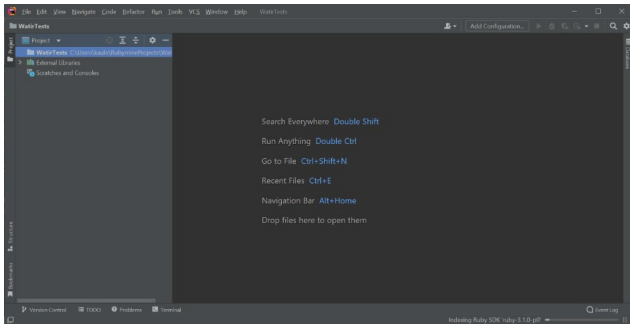
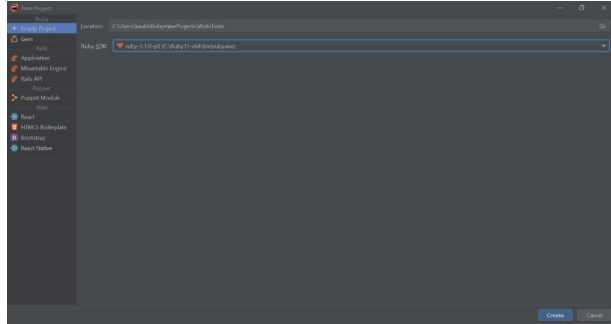


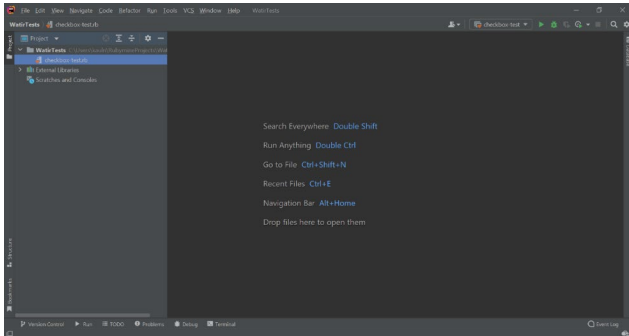
Once we setup an account, we can access the Welcome page.



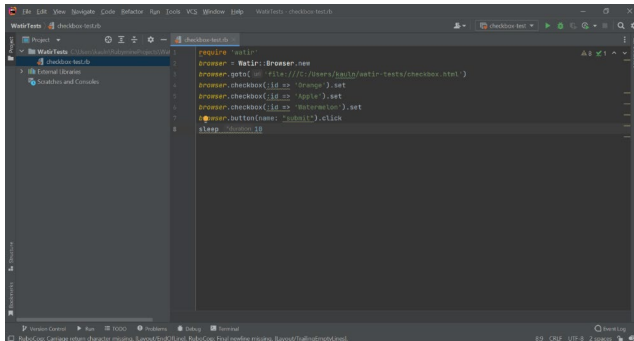
4. We proceed to create a new project as follows:





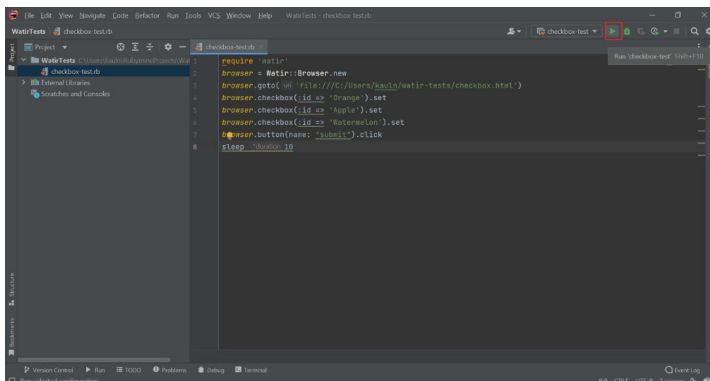


5. We edit this file to add a test case that tests the html form we created in Step-1.

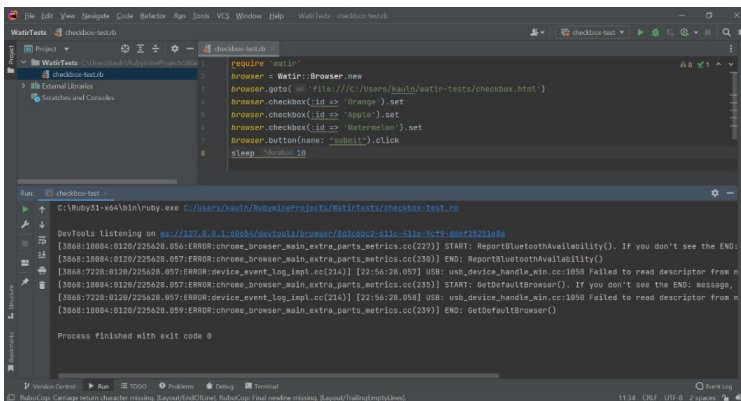
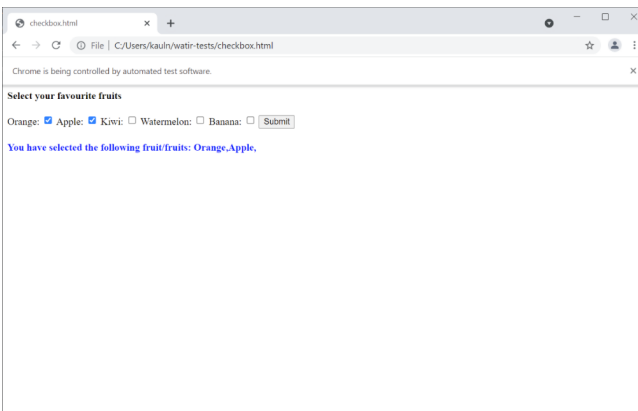
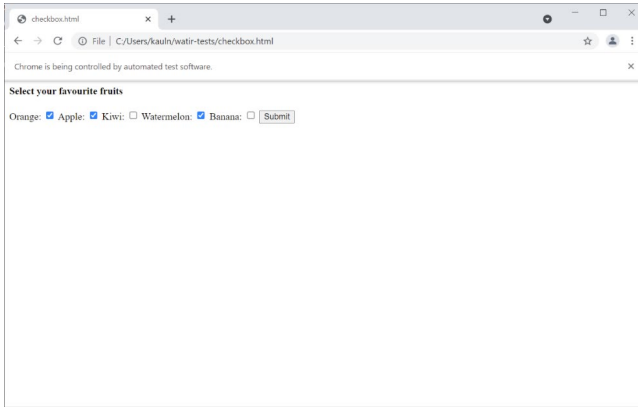


In this test case, we open the html page in the browser and then make use of the checkbox command to select the checkboxes using the ids for the checkbox values. Here, we select three options: orange, apple, and watermelon.

6. We now run the test case by clicking on the run button at the top.



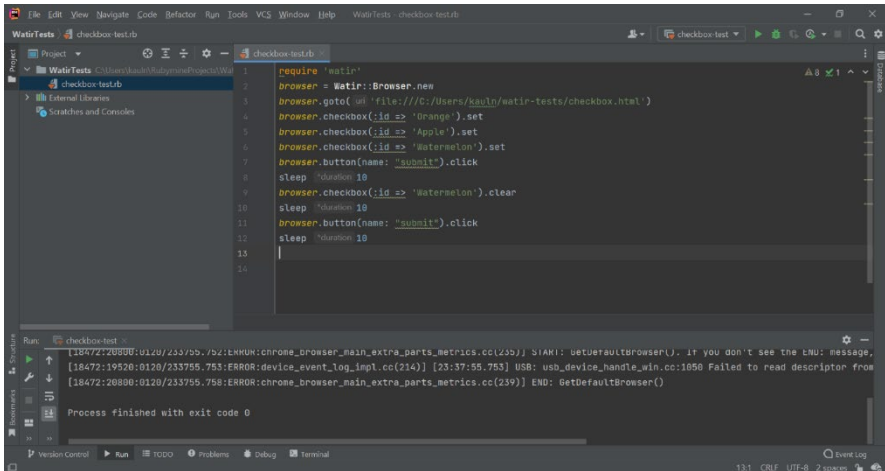
The automation launches the browser and selects the checkboxes. Then the submit button is clicked.



The test case is executed with success as seen in the screenshot above.

- We can also unselect the values that we selected. This can be done by using the watir command `clear`. The command is: `browser.checkbox(:id => 'id').clear`

We apply this change in the `checkbox-test.rb` file as follows:



```

1  [require 'watir'
2  browser = Watir::Browser.new
3  browser.goto(url: 'file:///C:/Users/kauln/watir-tests/checkbox.html')
4  browser.checkbox(:id => 'Orange').set
5  browser.checkbox(:id => 'Apple').set
6  browser.checkbox(:id => 'Watermelon').set
7  browser.button(name: 'submit').click
8  sleep :duration => 10
9  browser.checkbox(:id => 'Watermelon').clear
10 sleep :duration => 10
11 browser.button(name: 'submit').click
12 sleep :duration => 10
13
14
15

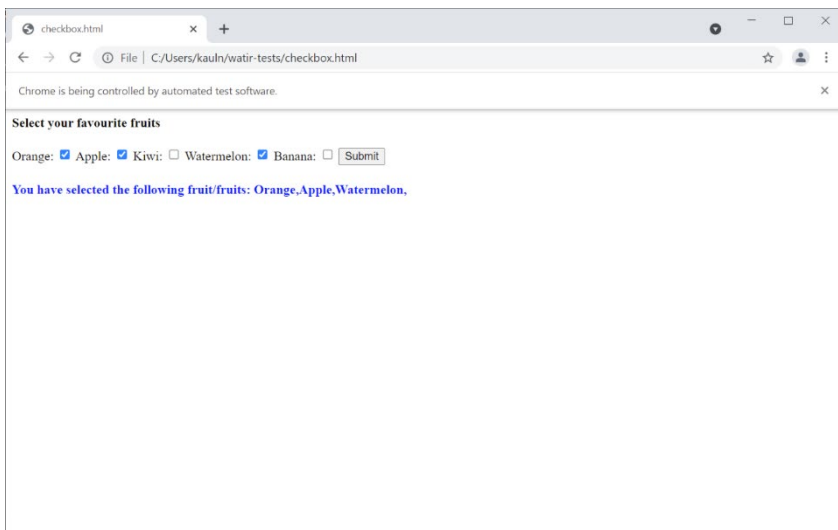
```

```

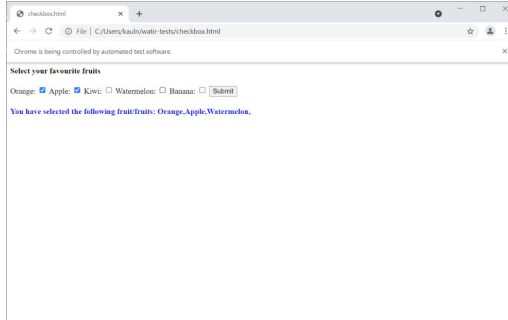
Run: checkbox-test
[18472:28890:0120/233755.752:EXHUN:chrome_browser_main_extra_parts_metrics.cc(235)] START: GetDefaultBrowser(), if you don't see the ENU message,
[18472:19526:0120/233755.753:ERROR:device_event_log_impl.cc(214)] [23:37:55.753] USB: usb_device_handle_win.cc:1850 Failed to read descriptor from
[18472:28890:0120/233755.758:ERROR:chrome_browser_main_extra_parts_metrics.cc(239)] END: GetDefaultBrowser()
Process finished with exit code 0

```

- We run the test case and observe its behavior.

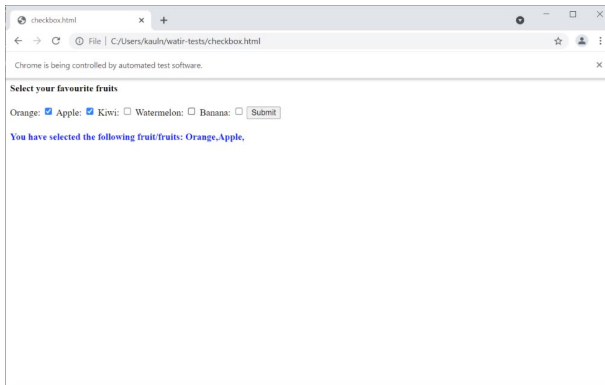


First, the three fruits Orange, Apple, and Watermelon are selected and the submit button is pressed. This gives us the text saying that these fruits have been selected.

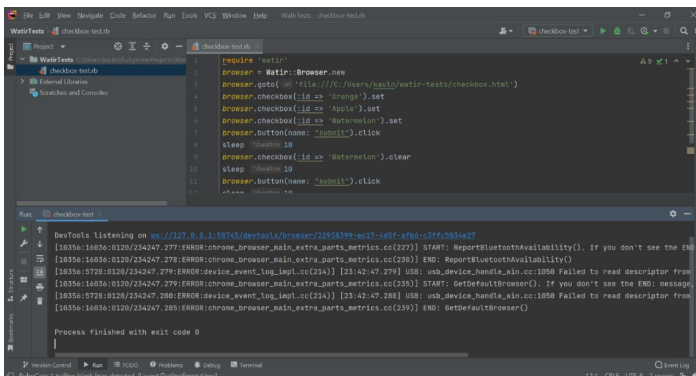


As clear from the screenshot above, the checkbox for ‘Watermelon’ has been unselected.

Now, the submit button is pressed and the text changes to reflect the checkboxes that are shown below:



The test case is successful, which is also reflected in the console output as follows:



RANOREX STUDIO

CONTENTS

5.1. Setup and Installation	192
5.2. Ranorex Studio Basics.....	199
5.3. Examples	201

Ranorex Studio is a powerful tool designed to automate tests for web applications, mobile applications, and standalone applications. This tool provides excellent GUI Testing features for web, mobile, and desktop applications. It is a tool that is straightforward to use and people with a non-coding background can easily create automated tests. It is a versatile tool that supports parallel testing, cross-browser testing and allows for remote testing too. A variety of coding languages are supported by this tool such as .Net, HTML, Java (Ranorex, 2017).

This tool allows users to build test cases and automate them quickly. Some additional features of this tool are: regression testing, keyword-driven testing, data-driven testing, and cross-browser testing (Ranorex, 2017).

Before we begin working with the tool, we need to download and install the Ranorex Studio. Ranorex provides a trial version available for download which is easily accessible on their website.

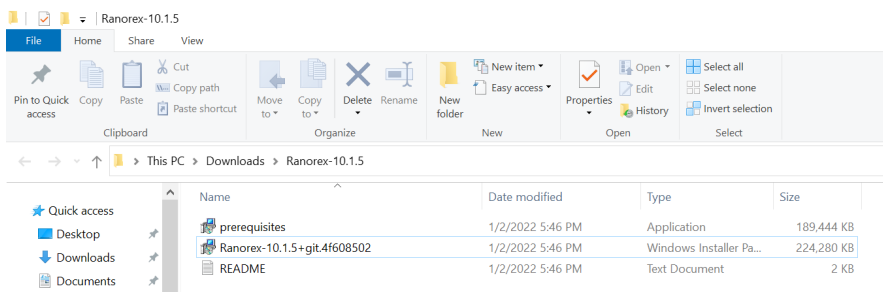
5.1. SETUP AND INSTALLATION

To download the tool, you need to register on their official website with a business email address. On successful registration, a link is sent to your email so you could download and install the trial version. If you do not have a valid business email, then you can directly contact the sales team and they would help you with downloading a trial version (Ranorex, 2017).

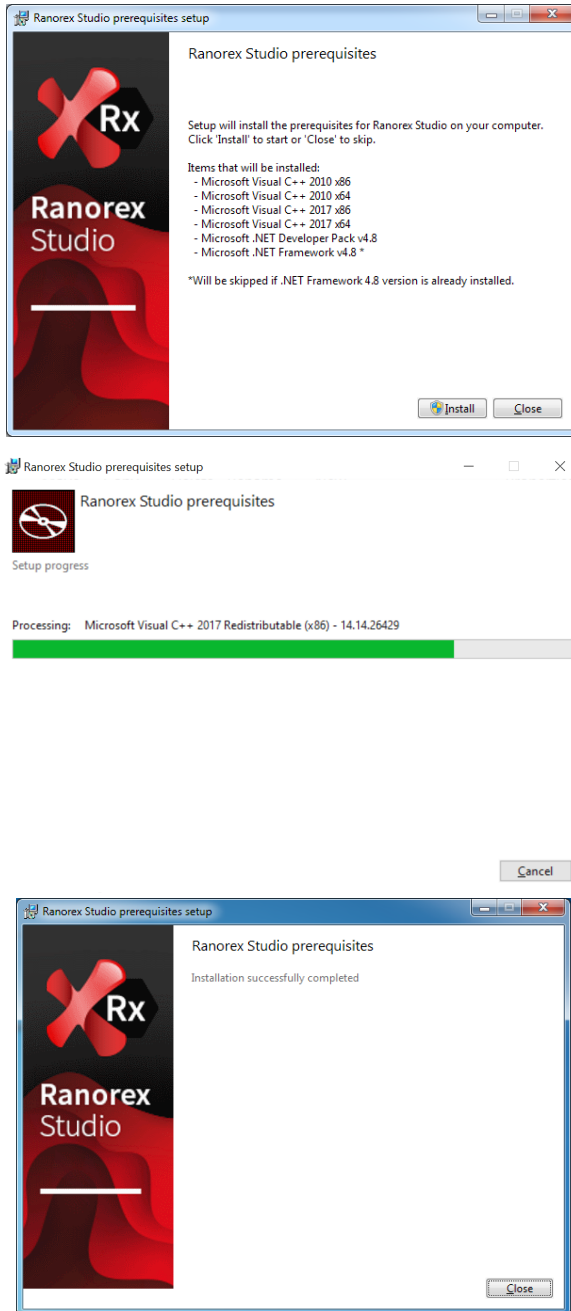
On successful registration on the official Ranorex website, you are sent a link to download the trial version.

Click on the link provided and download the trial version of Ranorex.

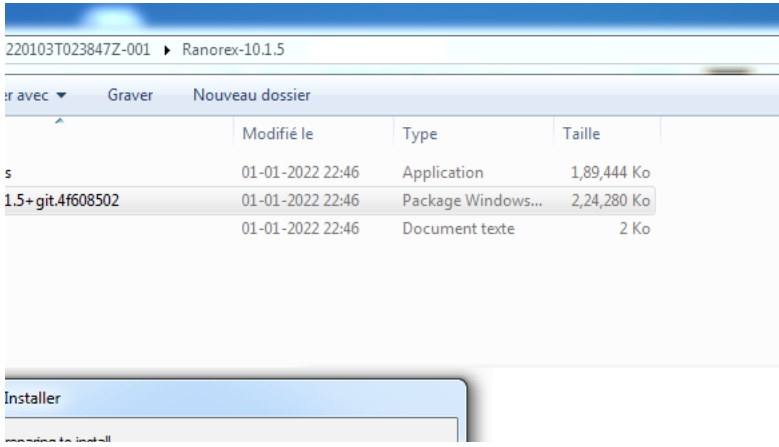
A zip file is downloaded on your local machine and the contents of the zip provided are as follows:



Follow the instructions provided in the README file. The first step is to install the prerequisites.



Once the prerequisites are installed successfully, we proceed to installing Ranorex by clicking on the MSI installer package provided.

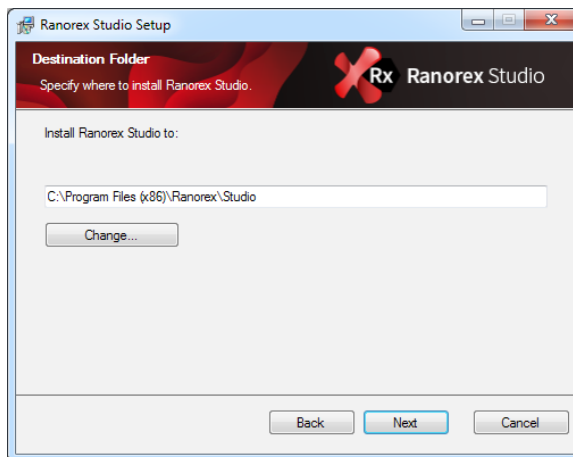


We click on ‘Next.’



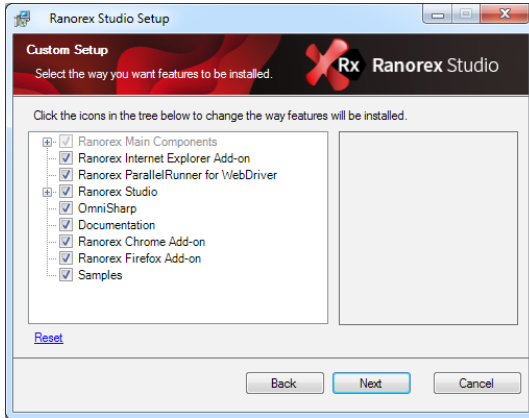


We accept the End User License Agreement and click on 'Next.' We are asked to provide the location where Ranorex studio would be installed. You can use the location selected by default or navigate to a location of your choice. Here, we install Ranorex in the default location which is in Program Files.

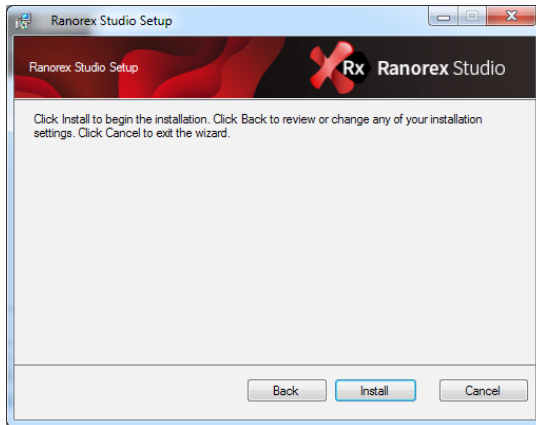


We then choose the features that we wish to install. Here, we select all features. But you can choose the features you wish to install based on your testing requirements.

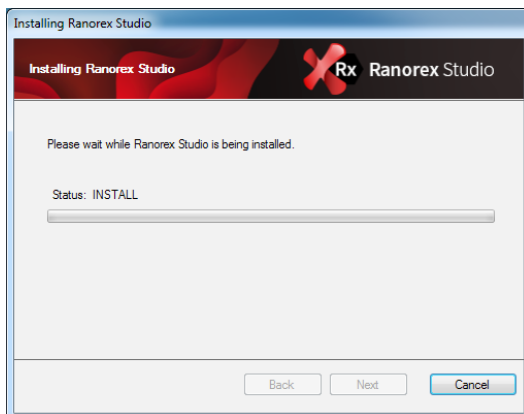
For example, if the application that you are testing is never run-on Firefox, then you can choose to omit the Firefox Add-on from the setup.



To start the installation, we click in ‘Install.’



The installation starts and the status is updated.



The installation takes a few minutes. On completion of the installation process, the following window appears:

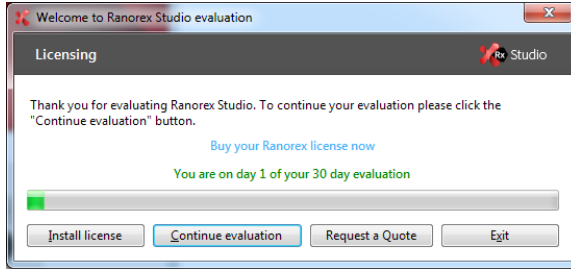


The 'Launch Ranorex Studio when setup exits' is checked by default. We click on 'Finish' to open Ranorex Studio.

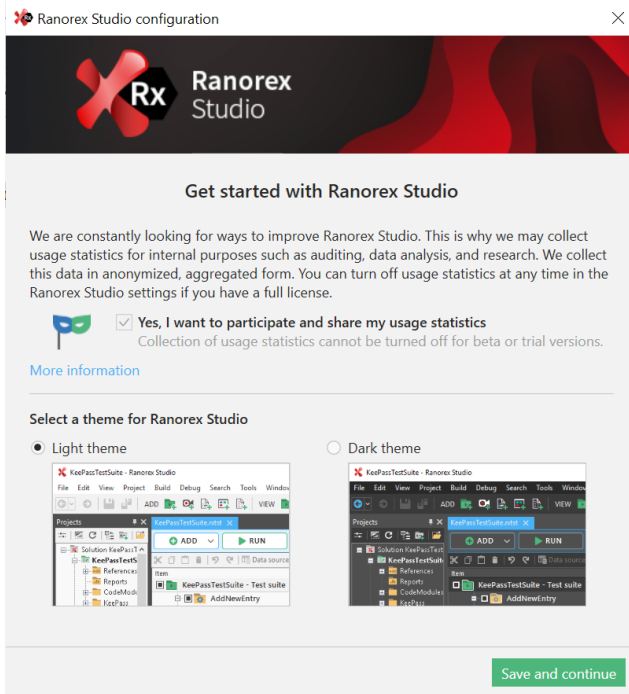
We are prompted to the licensing window that asks us to provide a license or start a free 30-day trial.



Here, we choose the 30-day free trial option.



We click on ‘Continue evaluation’ which then opens the Ranorex Studio configuration window and asks us to choose a theme.



We select a theme after which we can begin creating a project. This is done in Section 5.3.

Note that to be able to use Ranorex Studio after its trial period of 30 days, you must obtain a license. Before we move forward with the creation of the first test case, let us look at a few basic details of this tool in the next section.

5.2. RANOREX STUDIO BASICS

In this section, we learn different components that form a part of Ranorex Studio.

5.2.1. Views

Ranorex Studio provides three important views (Ranorex, 2017).

1. **Project View:** This is a view that is common for most IDE's and it permits you to view all the files in the current project. The project view is composed of the project, different libraries that are used by the project and the application, the repository file, modules, and all the files that are needed to execute the project. The project view also permits working on multiple projects in a single solution.
2. **Module View:** This displays the test steps and permits users to execute single or multiple steps. A user can view the individual test modules here. Different modules can be combined to create a series of automated suites.

The module browser is made up of two types of folders. They are:

- i. **Groups:** These are nothing but a collection of different items. All the module groups are listed here.
- ii. **Modules:** This section lists all the modules in the project. This section lists all the recordings and the project's code files. Additionally, any variables that have been defined in the recordings are found here as well.

The primary use of the module browser is to view the modules, drag-drop the modules and automation groups and reuse them.

3. **File View:** The file view is shown when you double-click on a file name in an open project from the Project View or the Module View. If you double click on any file name, it is opened in the file view much like any editor. For example, if you want to view the recordings in more detail, you can double click on the recording file, and it is displayed in File view. With the help of this view users can display all the available files such as the This view displays all the available files in the project such as the Recordings, the repository, action tables, code modules, reports, etc.

5.2.2. Components

The main components of the Ranorex Studio Tool are (Ranorex, 2017):

- **Ranorex Recorder:** It is a useful device that helps users record the activities as test steps and providing interesting playback and editing options. Different experiments can be done by exploring the recordings, editing them, retesting, and fine-tuning the recordings. The activities that are recorded can be adjusted physically in the activity table, which is an interesting feature provided by this tool. Ranorex is interesting in its design as the activities recorded and the UI components related to the activities performed are stored separately (different modules), which makes it easier to perform changes and update the test cases.

Another useful feature of this tool is that users can record the actions performed by the mouse in addition to allowing recording of the console. This feature is very useful in the case of UI testing. Once these movements are recorded, they are available for editing/updating in the Recorder activity table, where you can perform a host of operations on the recordings. You are allowed to adjust the test cases and your recordings to suit all your testing needs. Although the Recorder is a device and can be used independently, it can be easily coordinated with other components such the spy, the models, etc.

- **Ranorex Spy:** It is a versatile test automation tool that is beneficial for UI testing as it consists of an object recognition tool. This tool is the Ranorex Spy. The Tanorex Spy is a tool that is used for object recognition which helps locate and identify the UI elements in the screen/website that is being tested (Peischel et al., 2011).

The Ranorex Spy is like a scanner that locates and scans the UI elements of an application. This helps in understanding the UI components of the application that help in writing better test cases to test the application. It permits analysis of the application from the UI viewpoint and allows the ability to discern different elements of the UI with incredible ease (Peischel et al., 2011). This scanning tool provides us with UI data mapping relative to the XPath. We shall cover this in the upcoming sections. This tool helps in creating tests/recordings that test the modules of the UI correctly. The Ranorex Spy can see the details of the UI components of the application that is under test, and it provides a hierarchical representation of the UI elements which is extremely helpful when creating test cases.

- **Ranorex Repository:** It consists of all the UI elements that are being analyzed. The Ranorex repository provides a view in which the UI elements are mapped logically. Any UI element that is being tested is found in this repository. The repository is a component of Ranorex Studio which is usually used in coordination with Ranorex Spy and Ranorex Recorder. The repository is like a storage unit that is used to create and store mappings of the UI components that help in UI testing by providing insights into the components of the application under test. The repository portrays the UI components used in the application under test in a logical manner set by its UI mapping. The repository stores and displays the UI elements in a tree-like structure. Each item in this structure has a ‘RanoreXPath’ that is used to identify the component/element and helps in creating test cases that interact with the UI components of the application that is being tested. A repository is created automatically for every test project created in Ranorex Studio. The repository file has the extension ‘.rxrep.’
- **Ranorex Test Suite Runner:** As its name suggests, is a program that executes the test suites created in Ranorex Studio (Peischel et al., 2011). The runner is opened automatically on double-clicking a test suite file in the application. The test suite is a standalone application which means that if you click on the file without opening Ranorex studio, it will simply execute the test suite in the Runner.

The Ranorex Test Suite Runner can be used to execute specific test cases, entire test suites or just specific modules/folders. To execute entire test suites, run certain test cases and smart folders, or just run a specific module (Ranorex, 2017). Each time a test suite is run, a test report is created. Tests can be integrated with other applications such as Jenkins, DevOps tools, etc., and run from there as well.

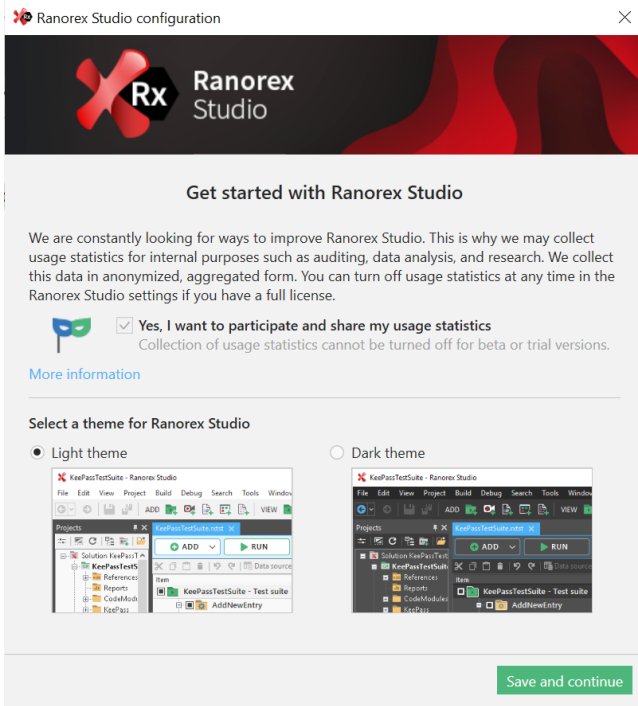
Now that we have looked at the components of Ranorex Studio we shall see practical examples demonstrating how to use Ranorex Studio to create test cases and test suites.

5.3. EXAMPLES

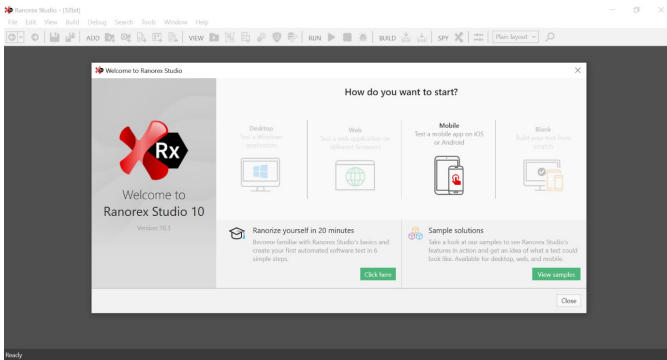
- **Example 1: Creating Your First Test Case in Ranorex Studio:** In this example, we will create a test project in Ranorex Studio. As this would be the first time that we open Ranorex Studio

and we do not have any projects created, the Ranorex Studio Configuration page is displayed.

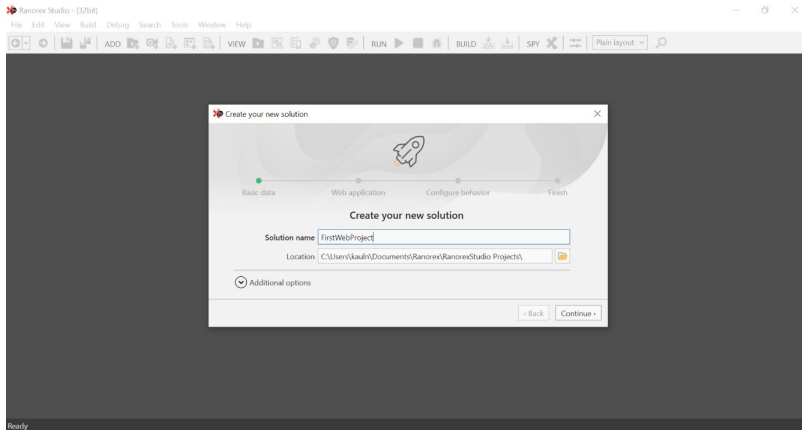
We choose the theme and proceed.



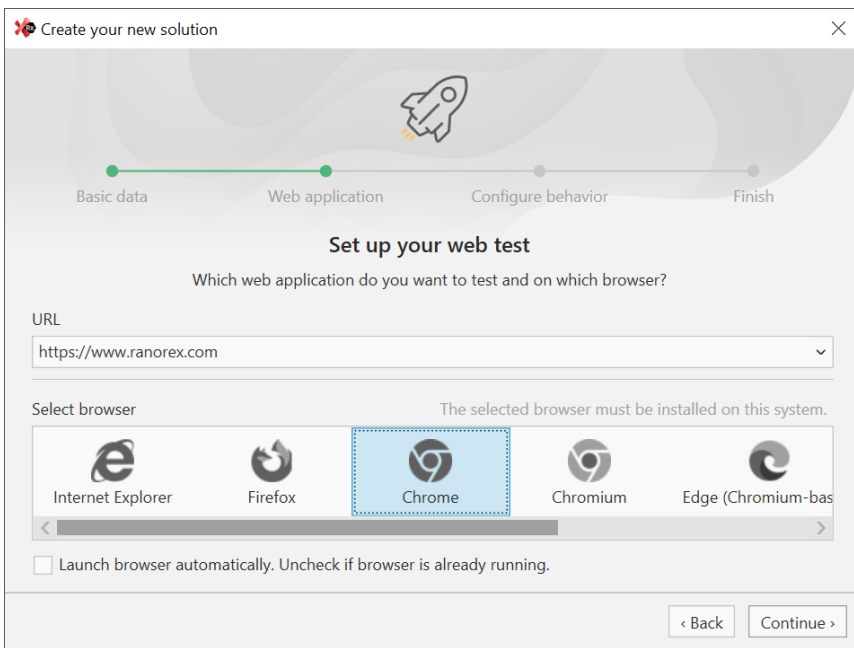
Once we select a theme, the RocketStart solution wizard appears and prompts us to choose the type of test project that we want to create.



Here, we create a web project. We provide a name for the project and choose the location for the project and click on ‘Continue.’

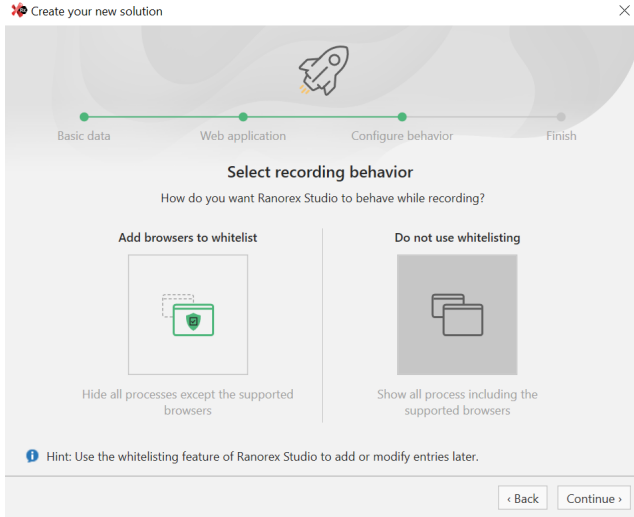


The wizard then requests us to provide the URL that we wish to test and the browser that we are going to use.

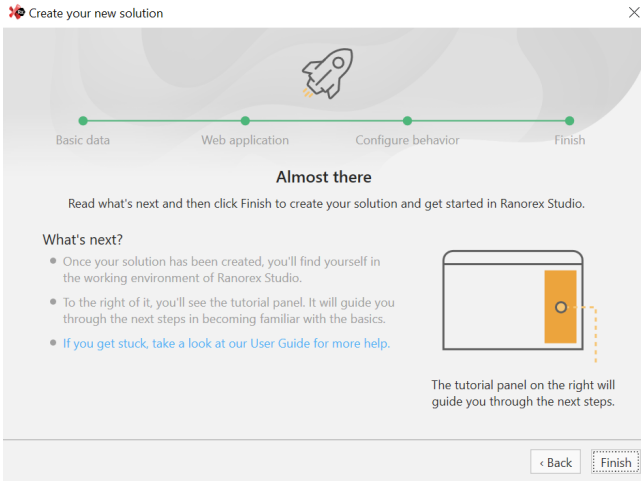


In this example, we use the URL provided by default and select 'Chrome' as the browser and click on 'Continue.'

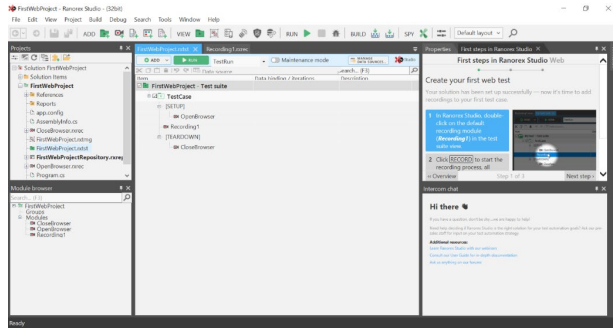
We are then prompted to choose if we wish to use whitelisting. In this case, we do not use whitelisting.



We click on 'Continue.'



We click on 'Finish' to create our project, which opens in Ranorex Studio as shown below:

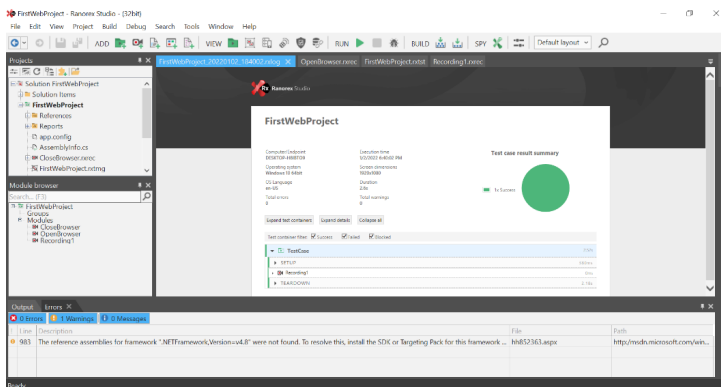


As we can see clearly in the screenshot above, in the Projects Tab, we have the solution items such as the recording file, the repository file, the reports folder, the configuration files, etc.

When we provide the URL while creating the project, a test case is created by default. A recording file containing basic actions such as opening the browser, accessing the URL, and closing the browser was created by default. On the right-hand side of the window, we have the tutorial page that guides us in creating our very first project. In the right-hand bottom corner, we also have the Intercom Chat window where we can ask any queries that we might have regarding the tool.

To run the test case that was created, we click on the play icon located on the toolbar. When we play this test case, it opens the Chrome Browser, opens the URL <http://www.ranorex.com> and then closes the browser.

After the execution of the test case, we receive the test case result and summary as shown below:

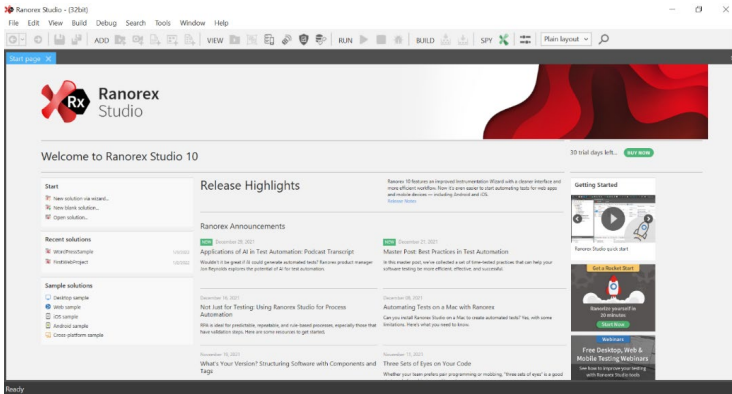


The test case summary provides details of the test case, such as the status of the test case (Success/Failed/Blocked), the steps of the test case, and details of the machine/system that ran the test case.

This example showed us the creation of a web project in Ranorex. In the next example, we will look at one of the sample projects provided by Ranorex and try to understand how it is configured.

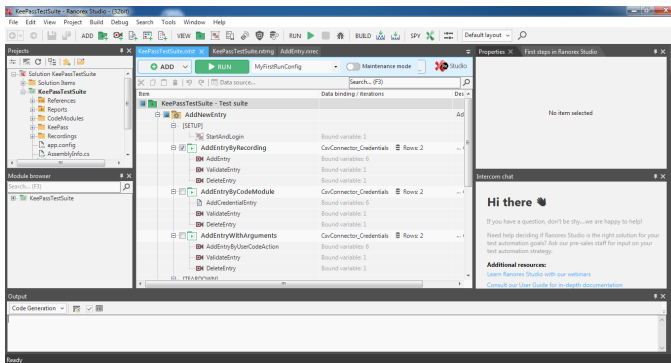
Ø **Example 2: Understanding a Sample Solution:** In this example, we shall look at the sample solution provided by Ranorex and try to look at how it is configured.

When we relaunch Ranorex, the start page is displayed. From the start page, we navigate to the ‘Sample Solutions’ tab and click on ‘Desktop sample.’



We open the Desktop example in Ranorex Studio. This is a sample that implements a KeePass Test suite where the keepass functionalities such as adding an entry, deleting an entry, adding a credential, deleting a credential, adding a group, deleting a group, etc., are tested.

The Desktop example is launched in Ranorex Studio as shown below:



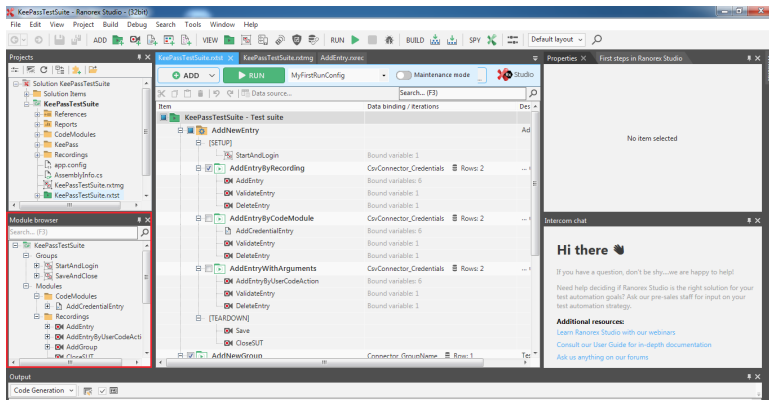
In the project view, we can see that it contains the files of the Solution – KeePass Test suite.

The module browser contains two folders: Groups and Modules.

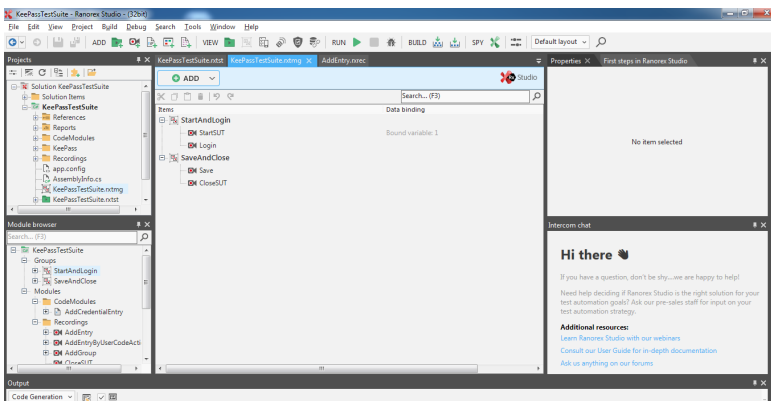
- The Groups folder consists of two groups: StartAndLogin and SaveAndClose.
- The Modules folder consists of additional two folders: CodeModules and Recordings.

The CodeModules consists of a.cs file which is a code file and consists of code written for the test suite.

The Recordings folder is composed of all the recordings that form a part of the test suite. A total of 12 recordings are present in this folder. Each recording is part of a test case that is outlined in the KeepassTestSuite.rxtst file which we will cover shortly. A file with the extension.rxtst is a Ranorex Test suite file. (rxtst is equivalent to ‘Ranorex test suite file’).

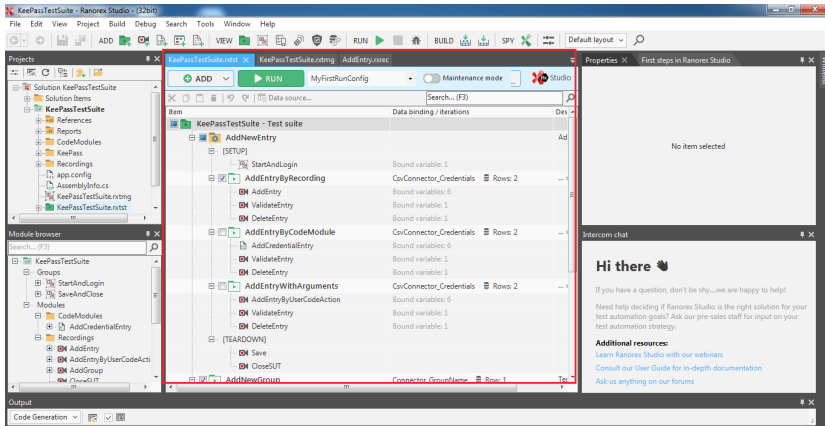


In Groups, if we double-click on ‘StartAndLogin,’ the groups are displayed as shown below:



As we can see clearly from the screenshot above, the items in the group are display in file view. We can see that the Groups folder has two items, and each item is comprised of two recordings. A recording file is indicated by the small camera icon before the name of the file.

To better understand how the recordings, work, we need to see the KeePassTestSuite.rxtst file as shown below:



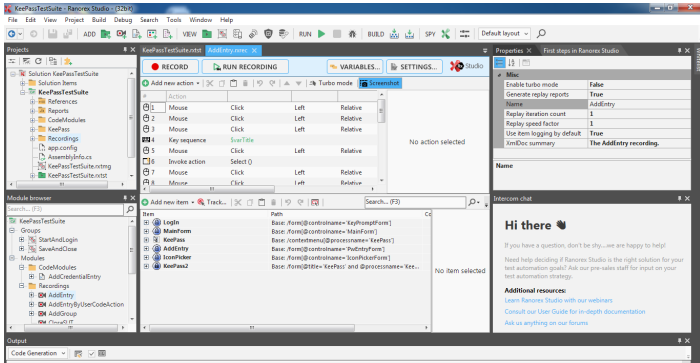
As we can see from the screenshot above, each recording is part of test case folder that is defined in this file.

In this sample solution, in the first folder/module of the test suite, each test case is adding an entry to Keepass using different ways. The first test case ‘AddEntryByRecording’ focuses on adding an entry via recording only. The second test case ‘AddEntryByCodeModule’ adds an entry using the code module file ‘AddCredentialEntry.’ The third test case tries to add entry by providing arguments in the test case. The last part of this module is the teardown where the Keepass data is saved and closed.

Each of these test cases in the first module, have three common types of recordings: add entry, validate entry, and delete entry.

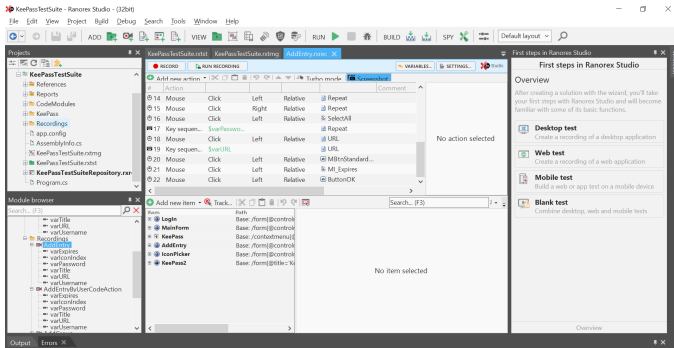
Let us take a brief look at each of these recordings in brief.

The AddEntry recording is as follows:

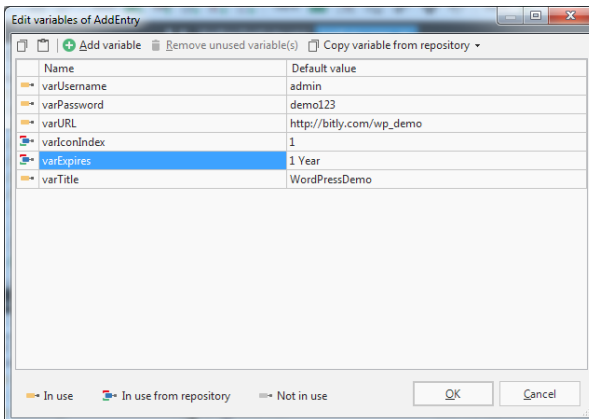


This recording consists of several mouse clicks, entering of keys, clicking on 'OK' buttons, etc.

If we expand the AddEntry recording in the module browser, we can see that it is made up some variables that are entered at runtime:

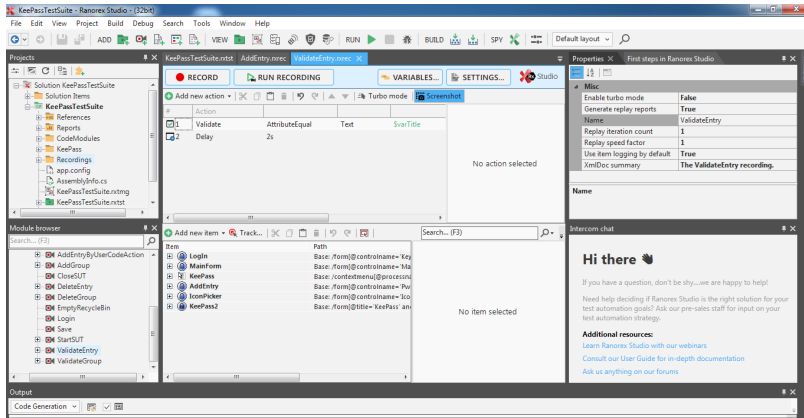


If we double-click on any one of the variables, the variables window pops up as follows:



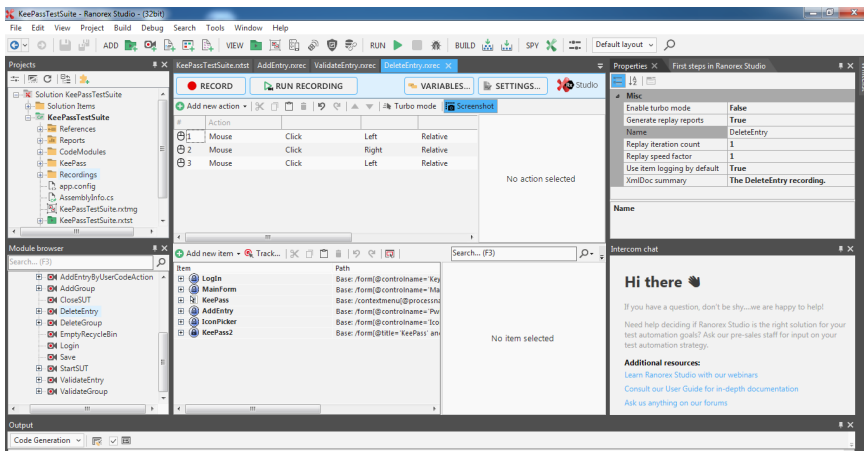
The values displayed in the window above are the values that are entered during the execution of the test.

The validateEntry recording is as follows:



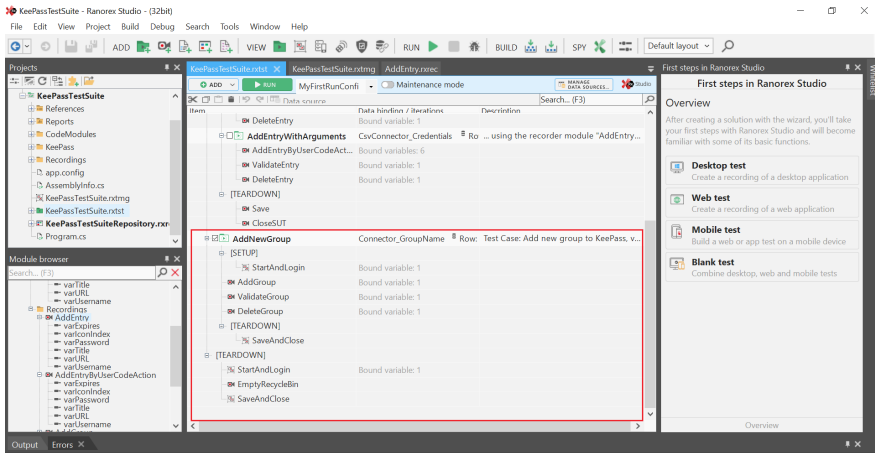
This recording is smaller as compared to the AddEntry recording and is made up of 2 actions. One action is the validate action which validates whether the entered password and re-entered password match.

The deleteEntry recording is as follows:



This recording consists of recording of different mouse clicks relative to the opened window that close the window.

The second module of the test suite consists of the 'AddNewGroup' test case where a new group is added, validated, and delete from Keepass.



The Teardown section consists of logging into KeePass, emptying the recycle, saving the changes and closing it.

Now that we have covered a few of the basics in the sample test suite, we proceed to run the test suite by click on the green play icon located on the toolbar.

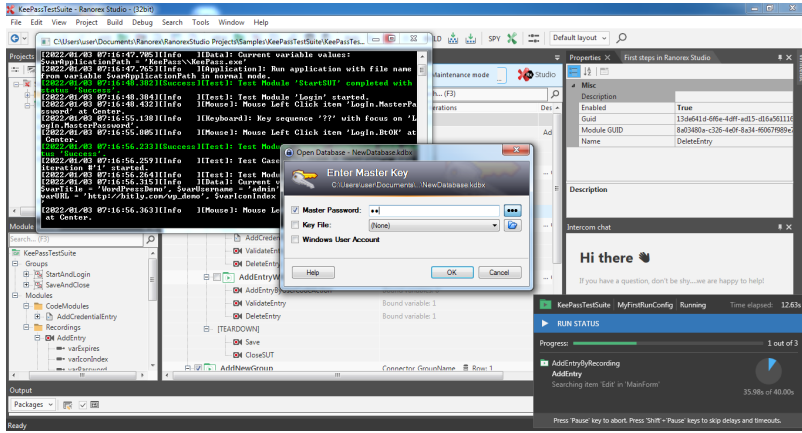
When we click on the play button, the test suite is launched. A command window is launched that initializes the test suite and logs the events.

```

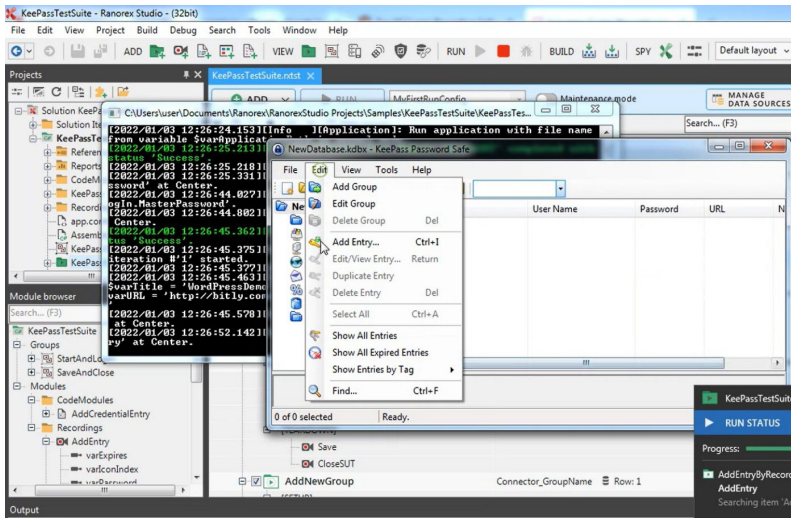
C:\Users\User\Documents\Ranorex\RanorexStudio\Projects\Samples\KeepPassTestSuite\KeepPassTes...
[2022/01/03 07:15:44.645][Debug][Logger]: Console logger starting.
[2022/01/03 07:15:45.475][Info][Test]: Test Suite 'KeepPassTestSuite' started.
[2022/01/03 07:15:45.505][Info][Test]: Smart Folder 'AddNewEntry' started.
[2022/01/03 07:15:45.638][Info][Test]: Test Module 'StartSUT' started.
[2022/01/03 07:15:45.719][Info][Data]: Current variable values:
$varApplicationPath = 'KeepPass\KeepPass.exe'
[2022/01/03 07:15:45.783][Info][Application]: Run application with file name
from variable $varApplicationPath in normal mode.
[2022/01/03 07:15:46.625][Success][Test]: Test Module 'StartSUT' completed with
status 'Success'.
[2022/01/03 07:15:46.628][Info][Test]: Test Module 'Login' started.
[2022/01/03 07:15:46.698][Info][Mouse]: Mouse Left Click item 'Login.MasterPa
ssword' at Center.
[2022/01/03 07:15:56.800][Info][Keyboard]: Key sequence '??' with focus on 'L
ogin.MasterPassword'.
[2022/01/03 07:15:57.468][Info][Mouse]: Mouse Left Click item 'Login.BtOK' at
Center.
[2022/01/03 07:15:57.898][Success][Test]: Test Module 'Login' completed with sta
tus 'Success'.
[2022/01/03 07:15:57.930][Info][Test]: Test Case 'AddEntryByRecording': data
iteration #'1' started.
[2022/01/03 07:15:57.933][Info][Test]: Test Module 'AddEntry' started.
[2022/01/03 07:15:58.002][Info][Data]: Current variable values:
$varTitle = 'WordPressDemo', $varUsername = 'admin', $varPassword = 'demo123', $

```

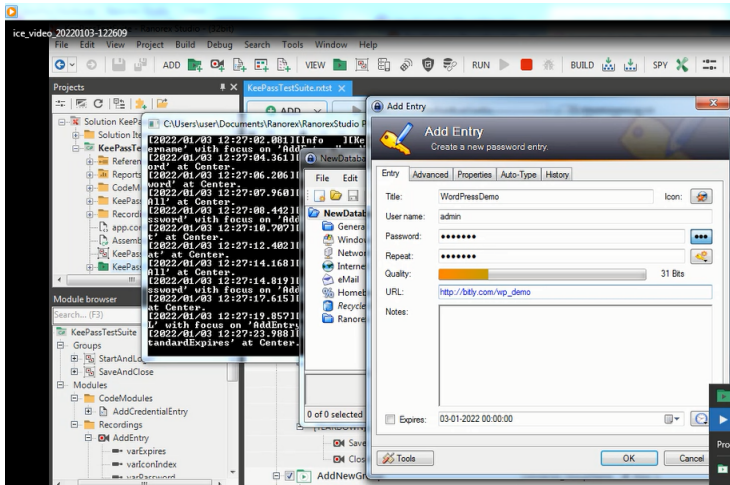
The login page is shown:



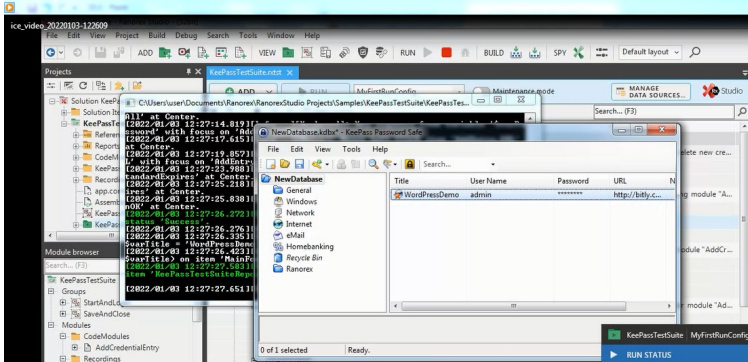
The automation proceeds to logging in and begins creating a new entry as follows:



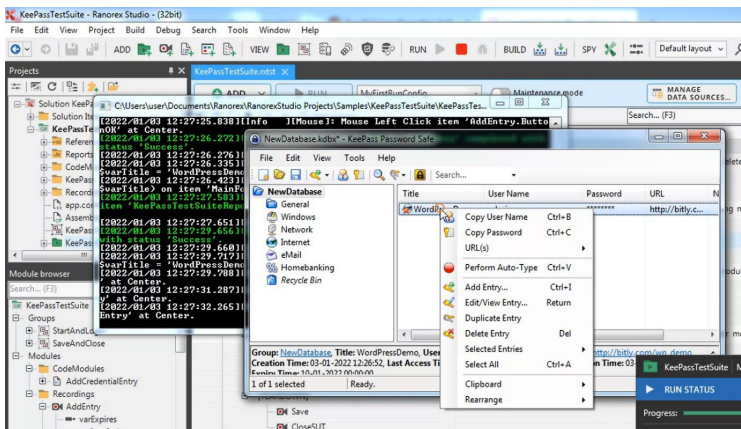
First, a new entry is created using the variables declared in the 'AddEntry' recording.

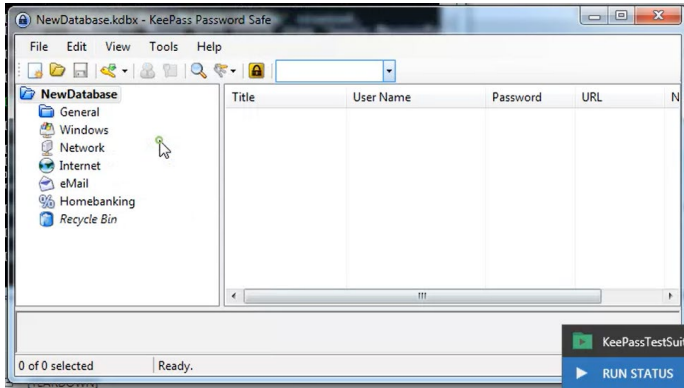


The entry is added as shown below:

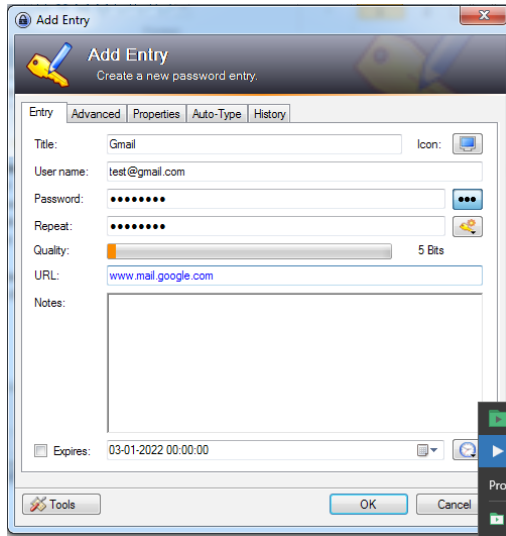
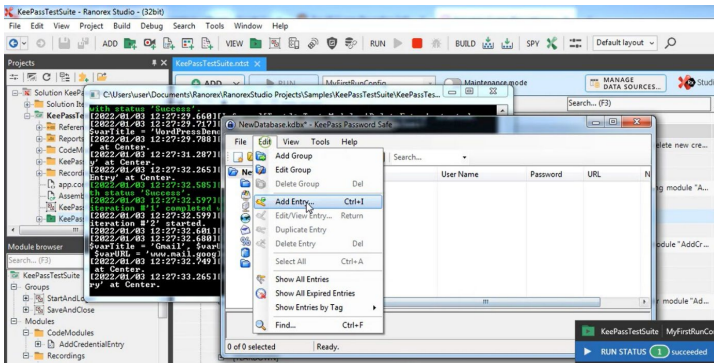


Once the entry is created, the automation deletes the entry and save the database.

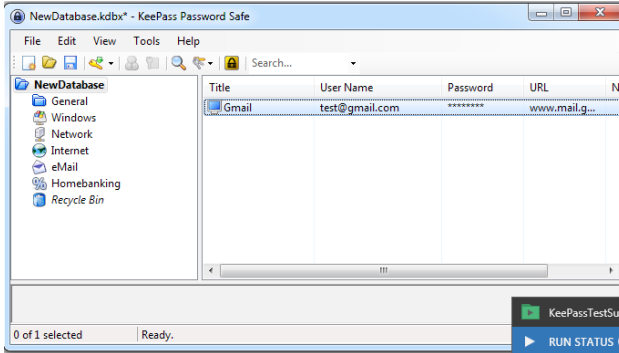




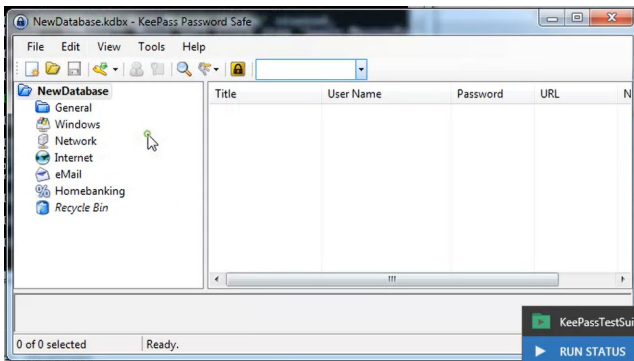
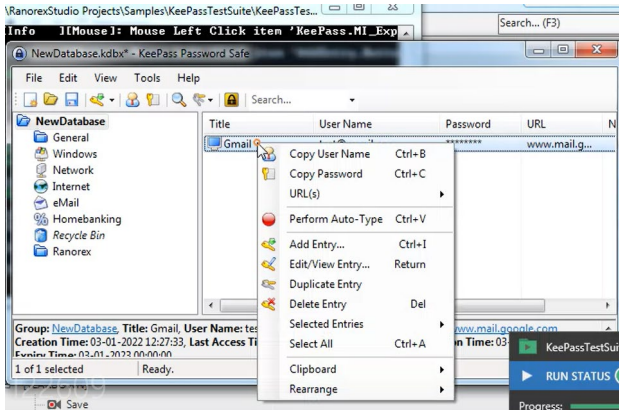
Next, the automation logs in again and adds another entry using the credentials file.



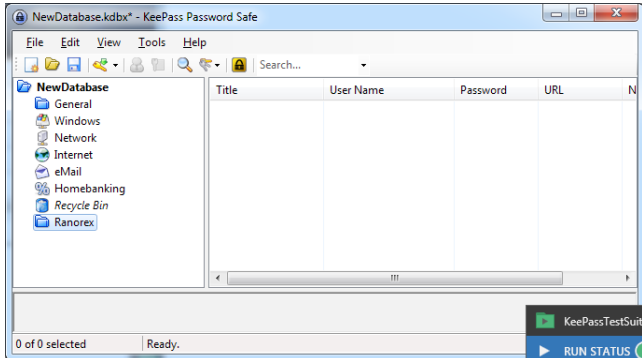
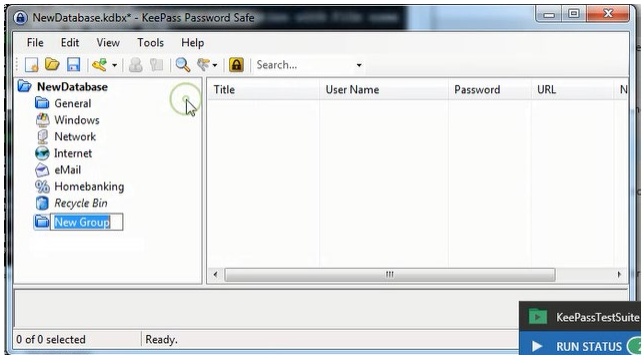
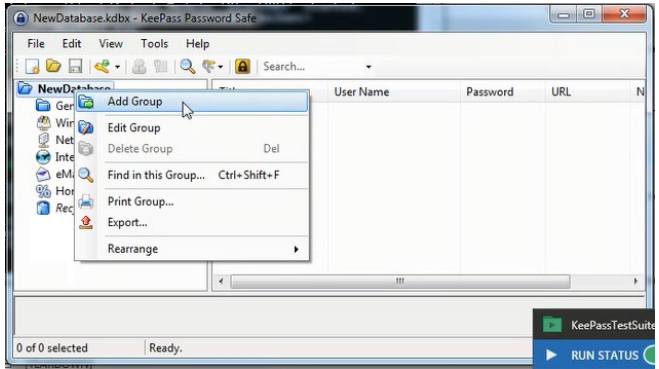
The credentials are added as shown below:



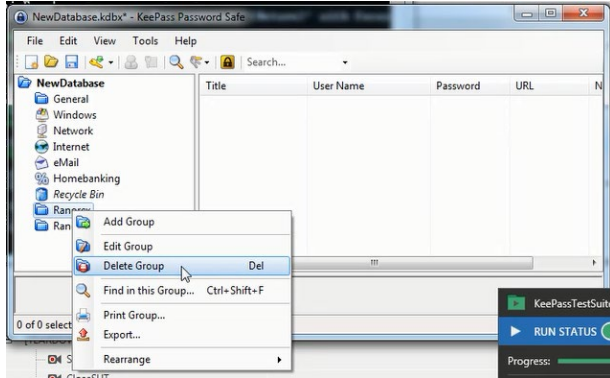
Once the credentials are added and validated, the entry is deleted.



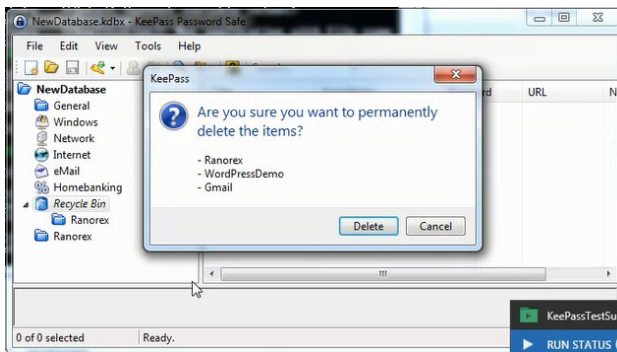
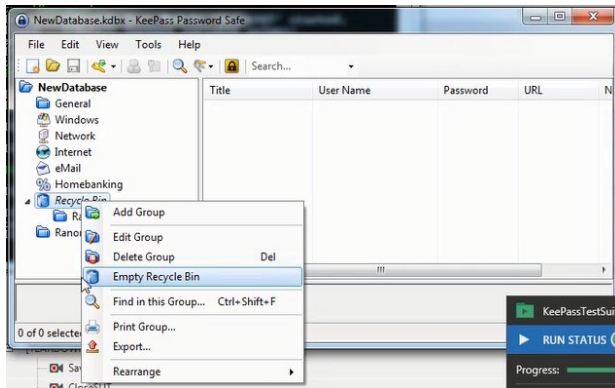
After the entry is deleted, the next part of the test suite is executed where a new group is added after which the recycle bin is emptied.



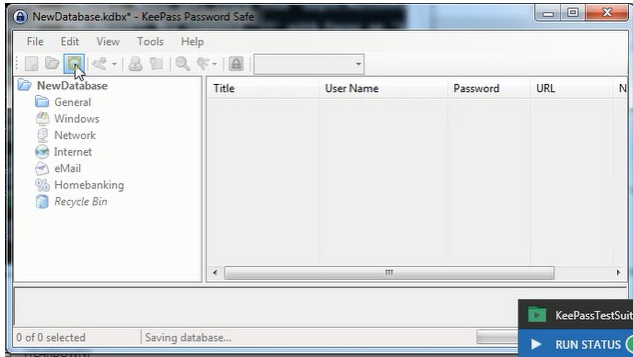
Once the group is added, it is deleted.



Next, the recycle bin is emptied.

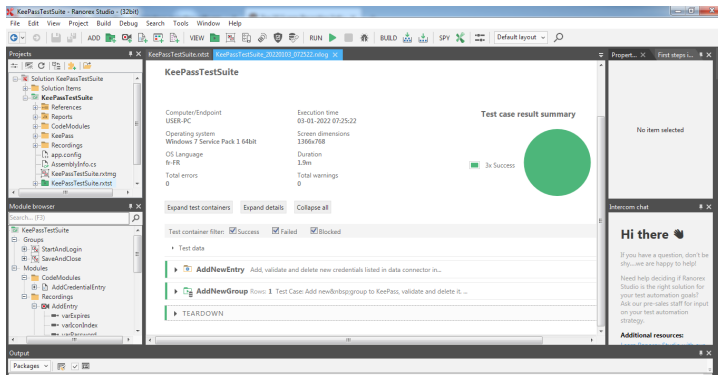


The items created are deleted and the keepass database is saved.



This step concludes the test suite.

Once the test is completed, the test suite result summary is displayed as shown below:

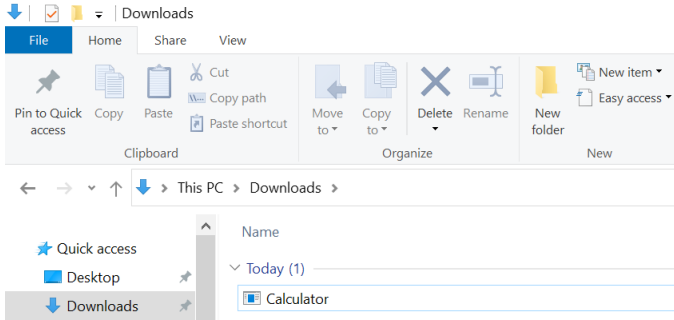


The summary details the test suite and the test case that were executed as part of the test suite. Additional details such as the execution time, machine details, etc., are provided as well.

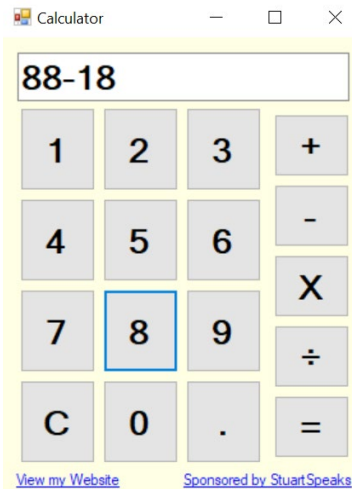
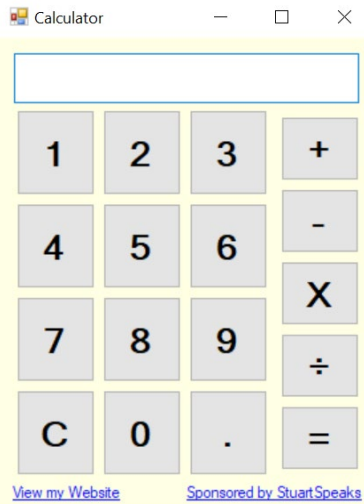
- Example 3: Create a Test Suite for a Desktop Calculator Application:** In this example, we download a sample calculator application and create a test suite that records and tests its functionality.

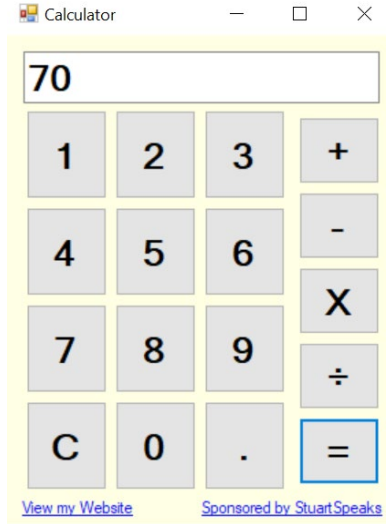
The steps followed are:

- We download a sample calculator application from the following website: <https://basic-calculator.en.softonic.com/>
- Once the calculator application is downloaded, we navigate to the downloads folder and launch it.



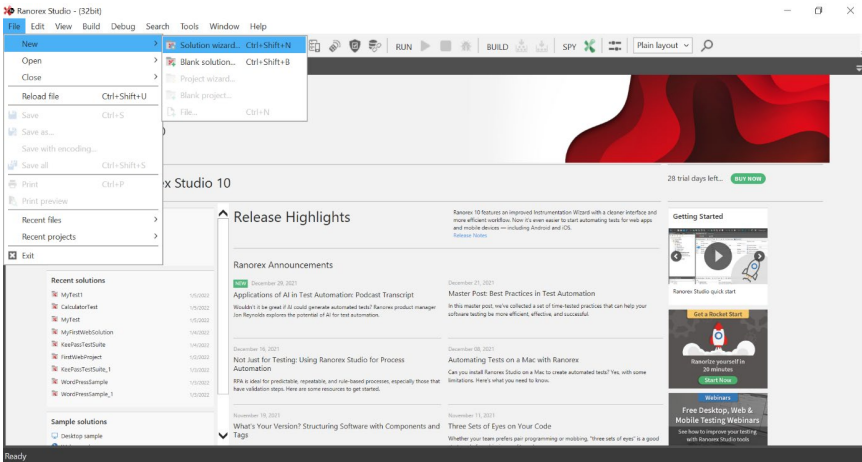
The application opens and we perform a few actions:



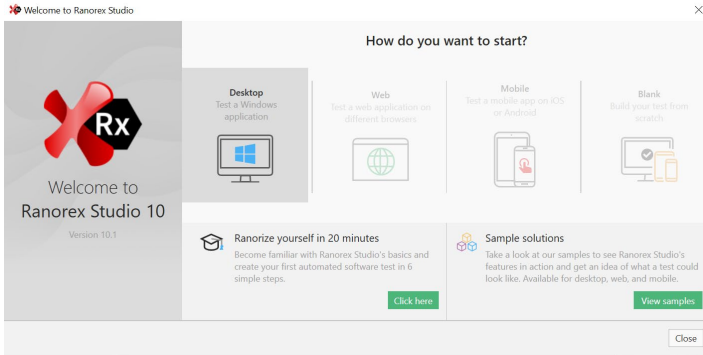


3. Once we have checked that the application works fine, we close it and proceed to create a new solution in Ranorex Studio. We first open Ranorex Studio.

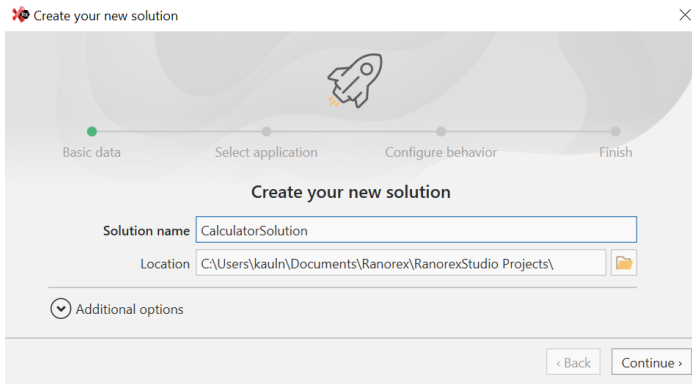
We navigate to File → New → Solution Wizard to create a new solution.



4. We are prompted to choose the type of application that we would like to test. We select 'Desktop' from the available options.



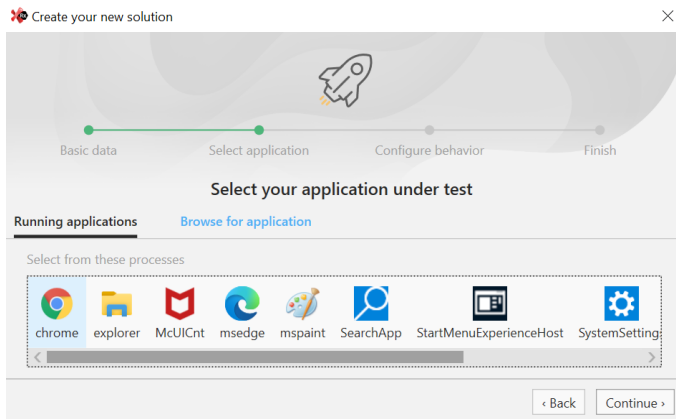
5. We provide the solution name and location as follows:



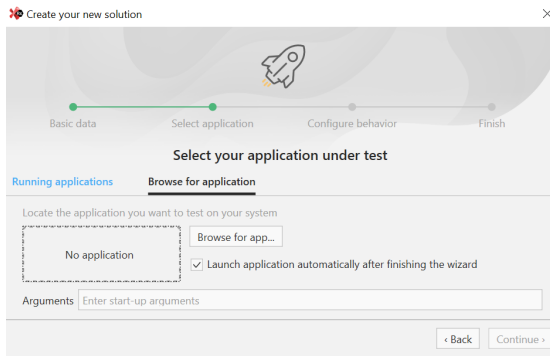
We click on 'Continue' to proceed.

6. We are then prompted to choose the application that we wish to test.

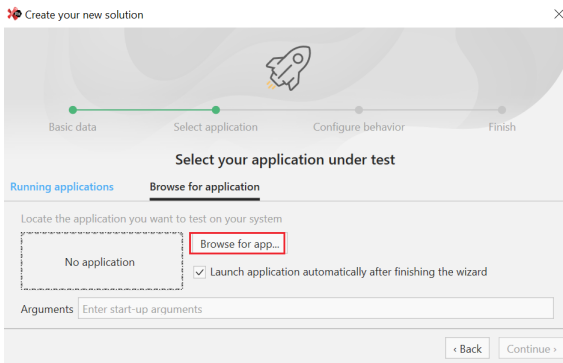
The following window appears:



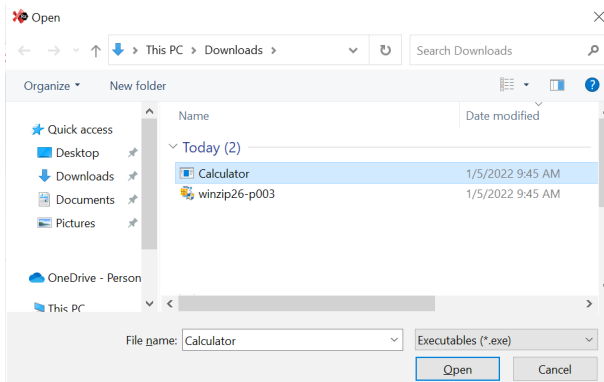
The above window asks us to choose the application that we would like to test. We go to ‘Browse for application’ tab.



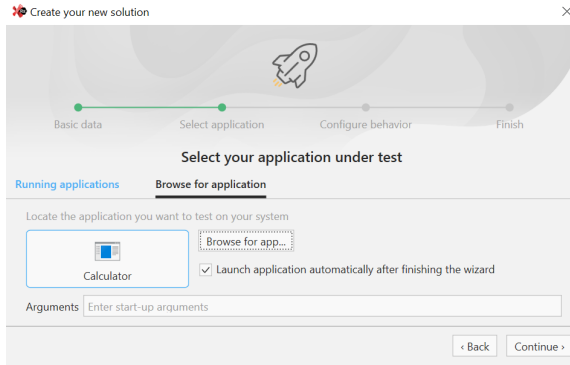
We click on the ‘Browse for app.’ button:



We browse to the ‘Downloads’ folder where we downloaded the calculator application and select it.



It is added successfully to the solution as shown below:

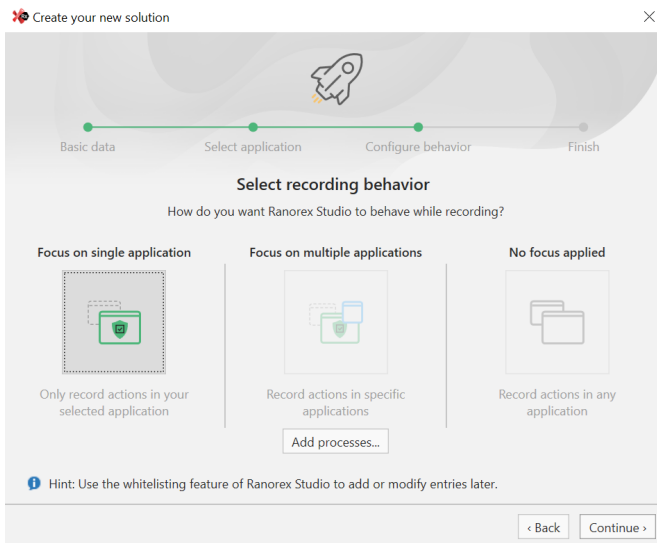


Note that we keep the ‘Launch application automatically after finishing the wizard’ option selected because we want the calculator app to be launched when the project is opened.

We click on ‘Continue’ to proceed with the project creation.

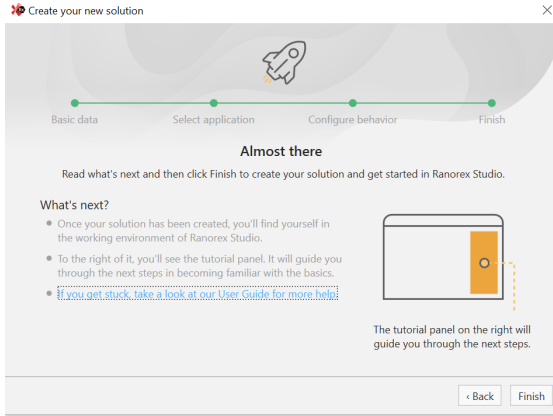
7. We are asked to configure the behavior of Ranorex while recording for a test case.

We select the first option which is ‘Focus on single application.’ We select this option as this will record only the actions performed by the ‘Calculator’ application and ignore any other applications that are running simultaneously on the machine.



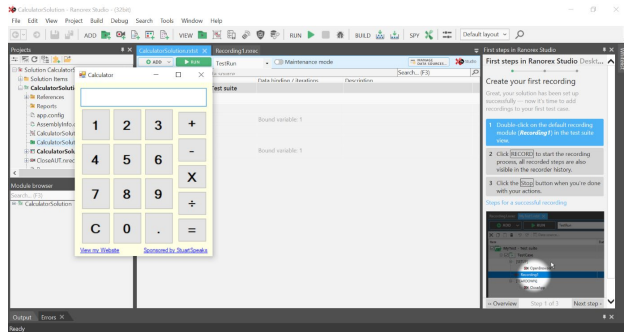
We click on 'Continue' to advance with the creation of the solution.

8. We click on 'Finish' to end the wizard and create the solution.

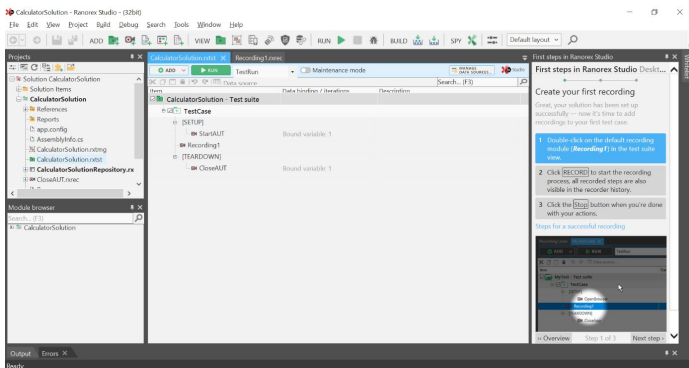


We click on finish to complete the wizard and create the new solution.

The solution is created and when its opened and the calculator application is launched with the solution as shown below:

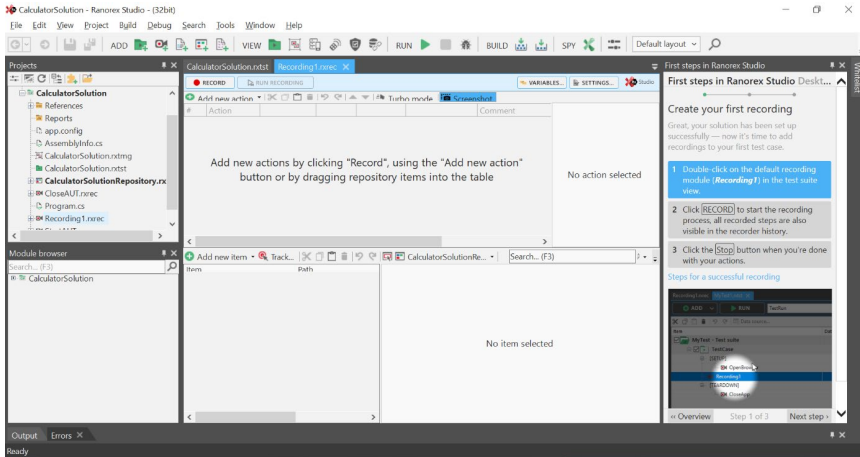


9. We see the CalculatorSolution test suite as follows:

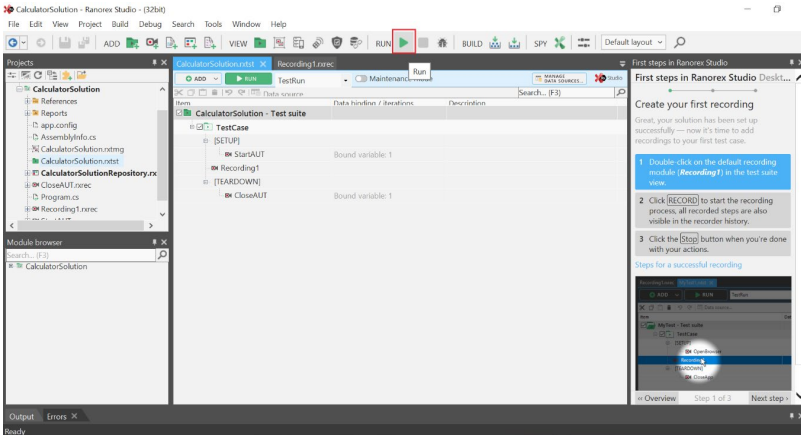


By default, the test suite is created with a test case that has a setup, an empty recording, and a teardown section.

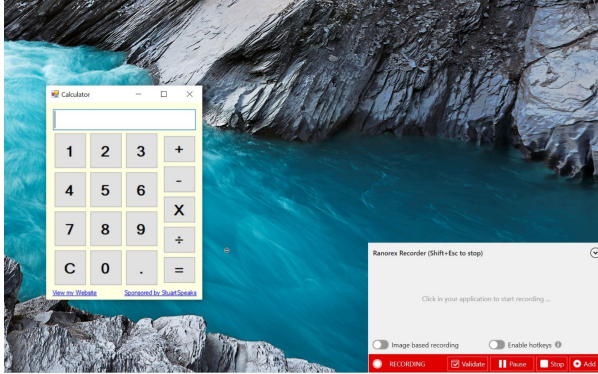
We click on 'Recording1' and navigate to the recording1.rxrec file and can begin recording for the test suite.



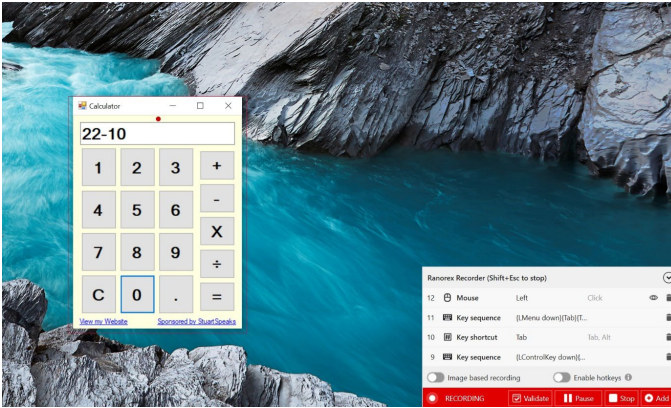
10. To begin recording for the test case, we click on the record button.



The record window is displayed in the corner of the screen along with the calculator application in the center as shown below:



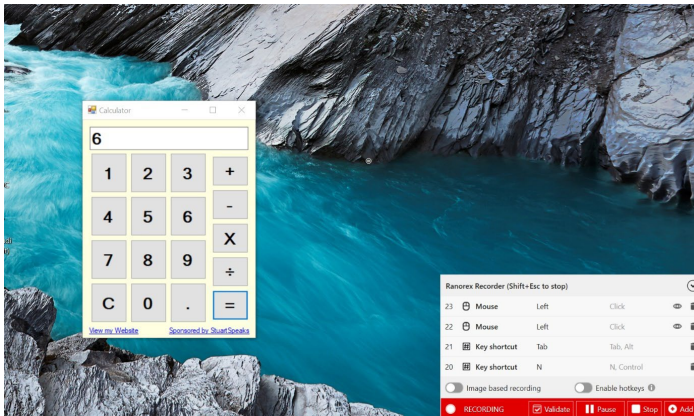
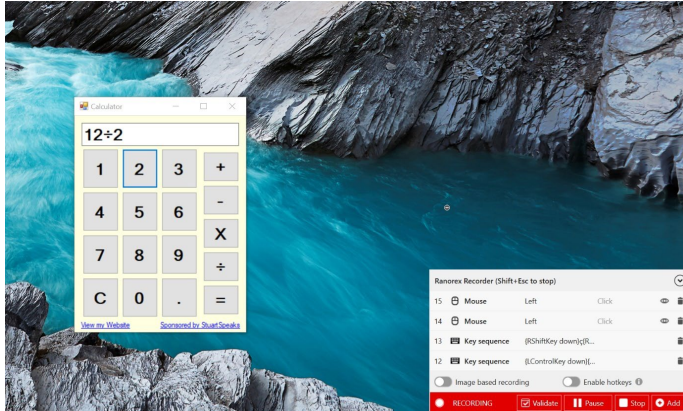
We begin recording by performing simple operations as follows:



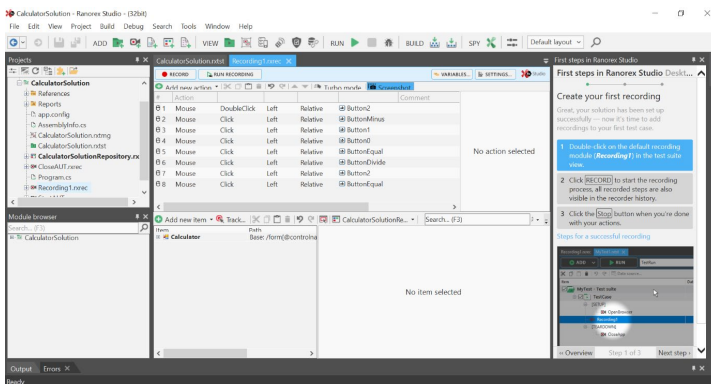
As we can see, each keystroke is recorded in the Ranorex Recorder at the left corner of the screen.



We keep on recording a few actions.



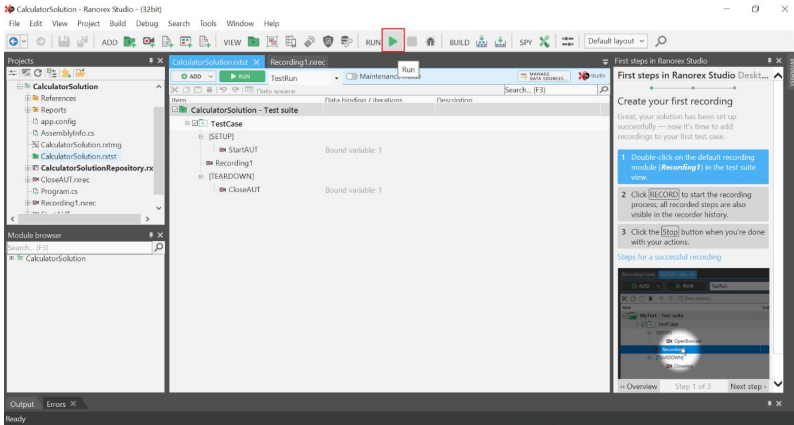
Once we are done with our recording, we can see that the file 'Recording 1. Rxrec' is now updated with the actions that we recorded.



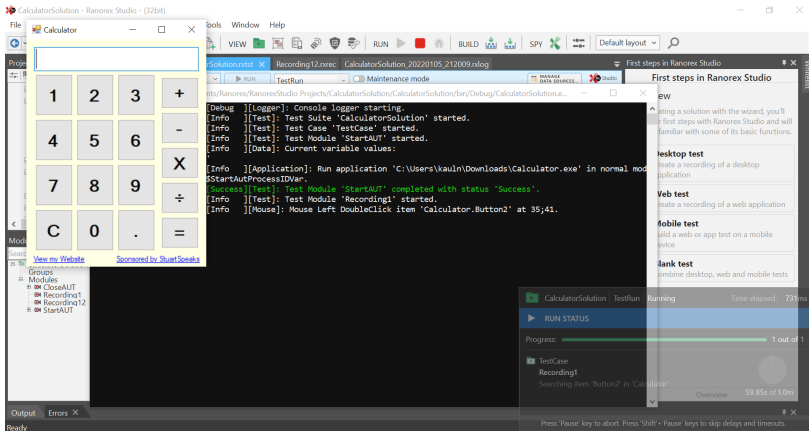
As we can see from the screenshot above, each movement has been recorded.

- 11. The last step would be to execute the test suite containing the updated recording.

We click on the green run button at the top to execute the test suite.

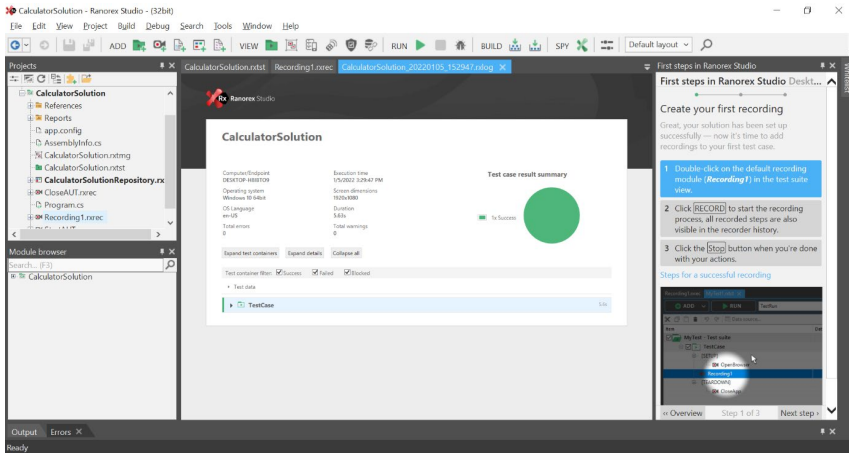


The test suite launches and goes on to repeat the actions that we performed earlier.



After the test case is executed, a test summary report is generated.

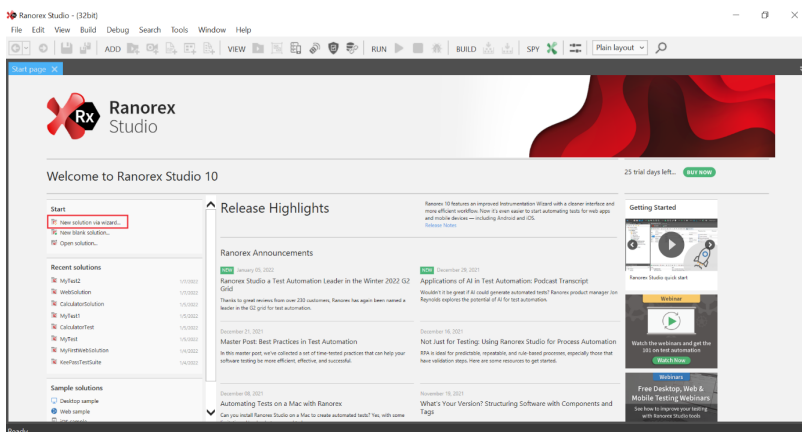
The test suite executes successfully as visible in the Test case result summary shown below:



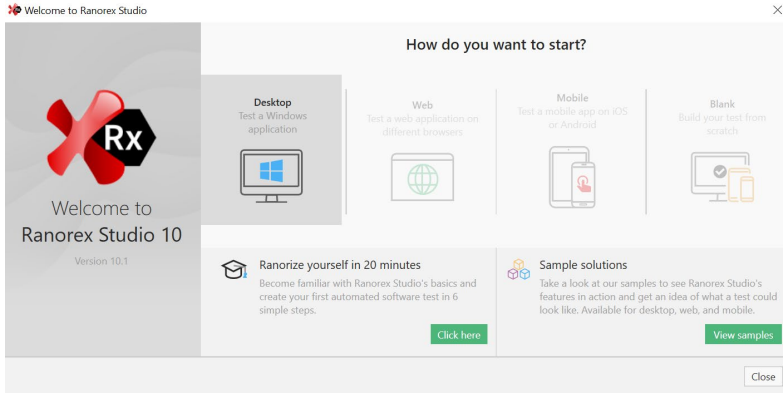
- Example 4: Creating a Test Suite for Paint Application on Your Location Machine:** In this example, we shall test the application Microsoft paint using Ranorex Studio. We will perform actions in the paint application and proceed to save the file as ‘test.bmp’ in the ‘Pictures’ Folder.

The steps taken are as follows:

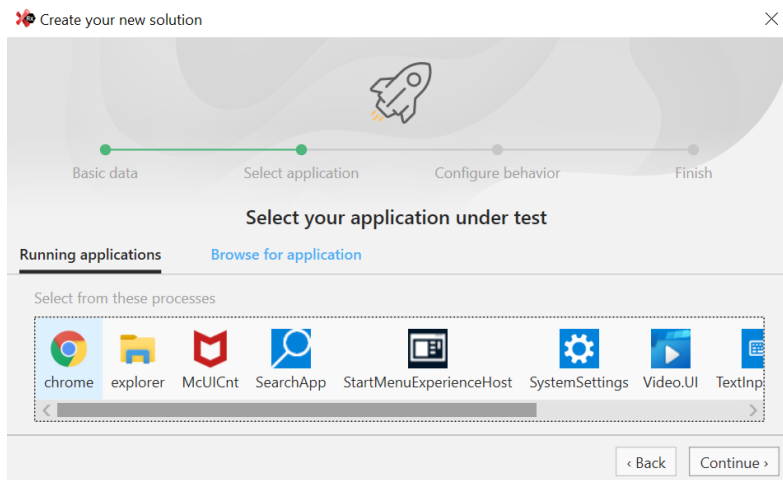
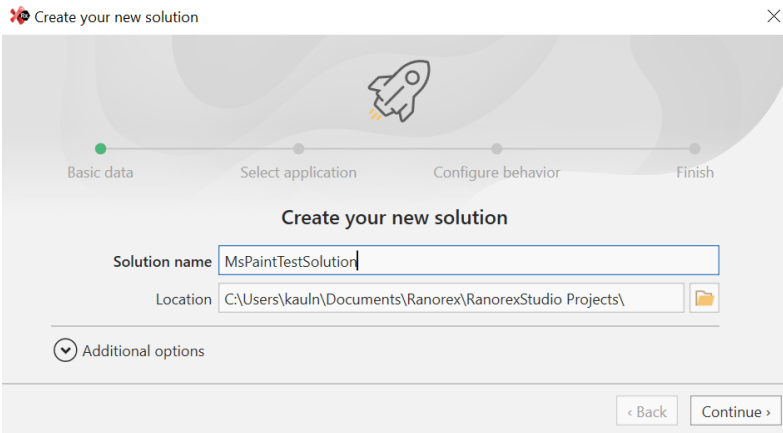
1. We create a new solution using the New Solution creation wizard.

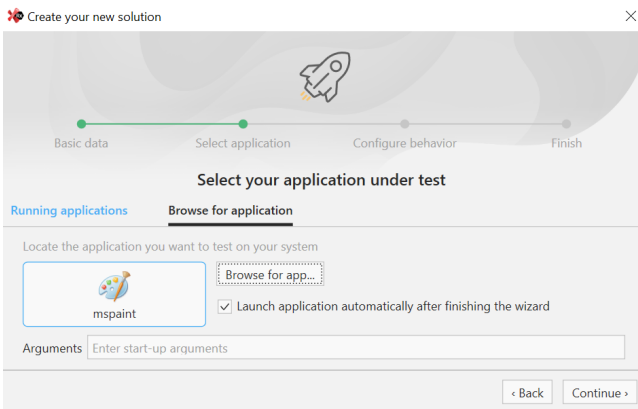
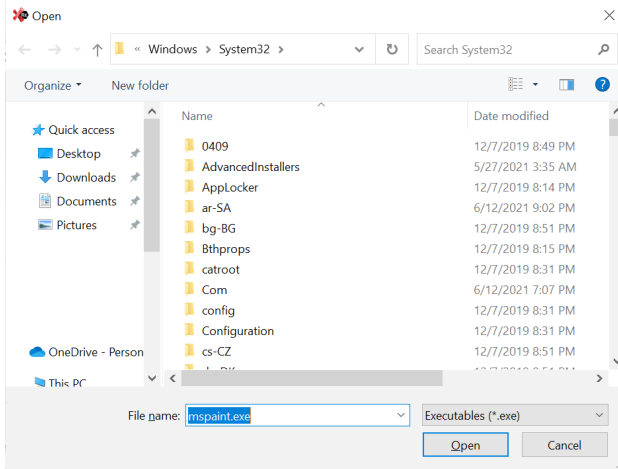
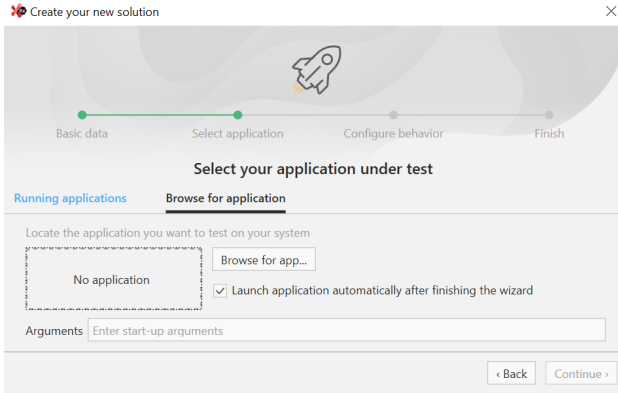


2. We choose the type of application to be ‘Desktop’ and provide the details of the test solution, such as the name and location.

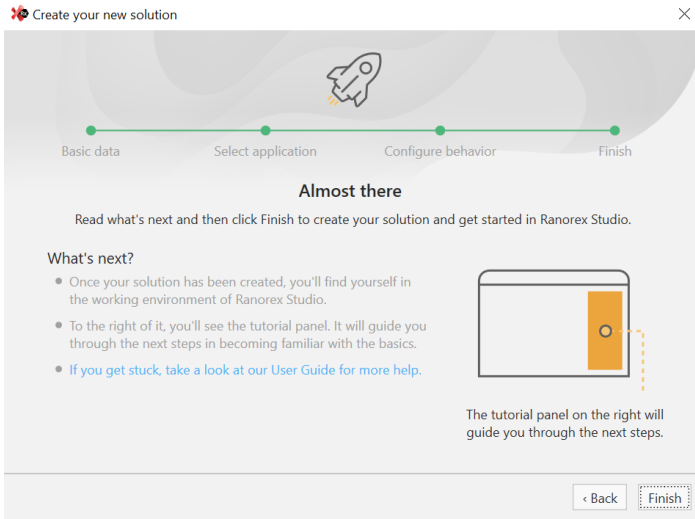
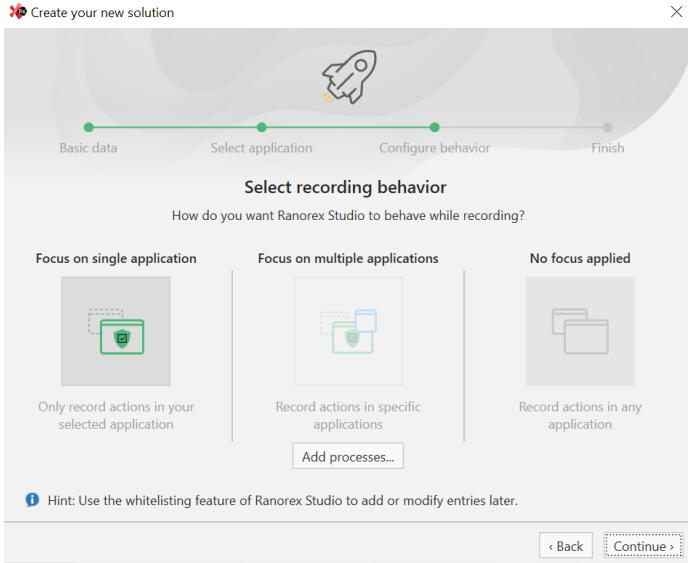


3. We choose the paint application in the next step.

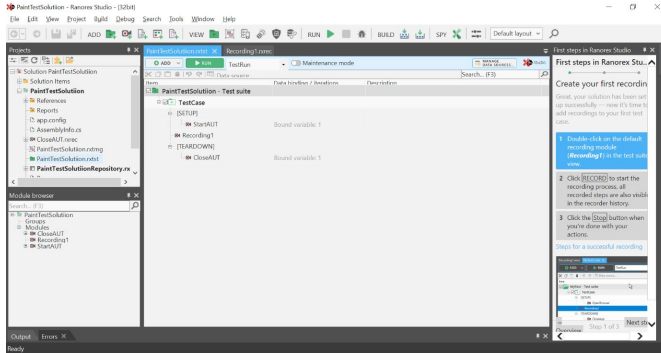




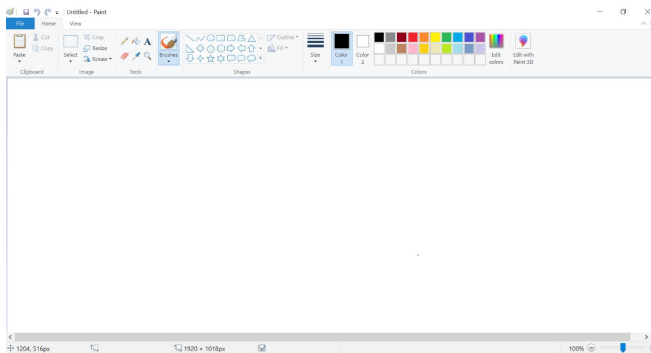
4. We choose the recording behavior and complete the setup as follows:



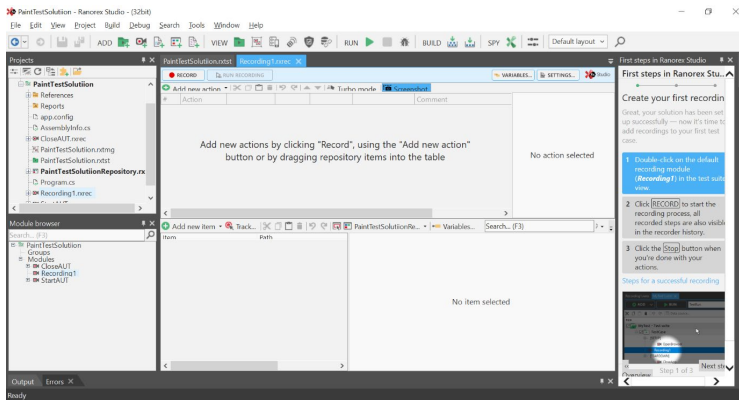
5. The new solution is now created with a Test suite containing a single test case which is made up of a setup and teardown step and an empty recording by default. This is shown below:



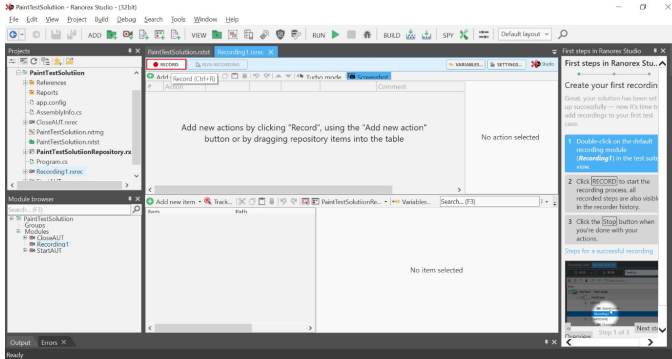
The paint application is opened on launch of the solution as well.



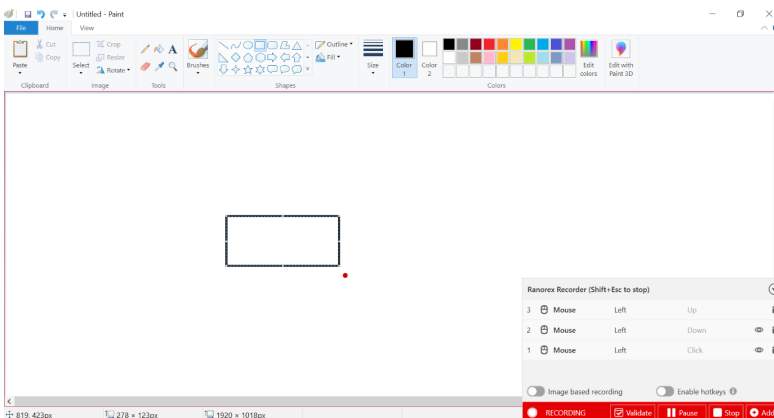
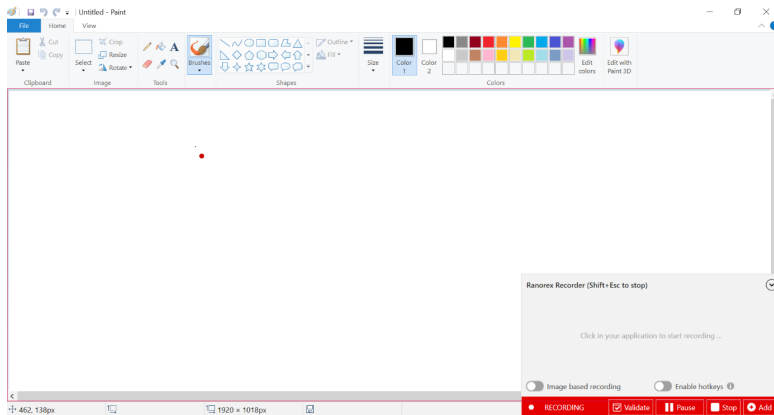
We open the empty recording file.



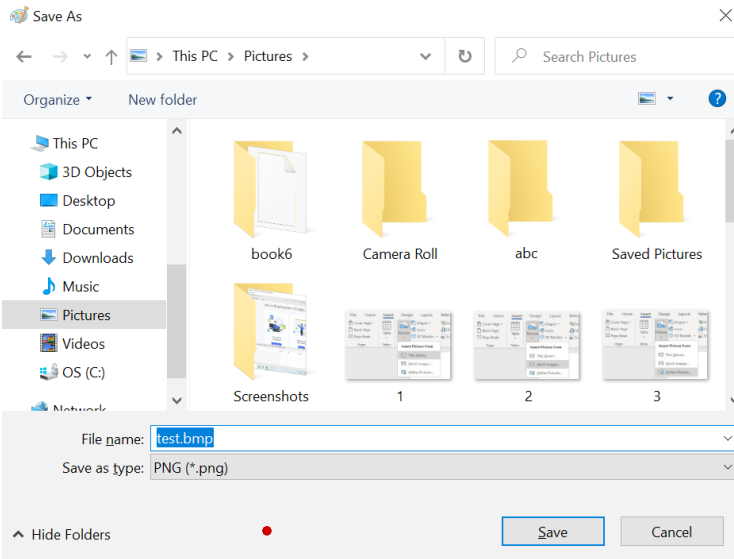
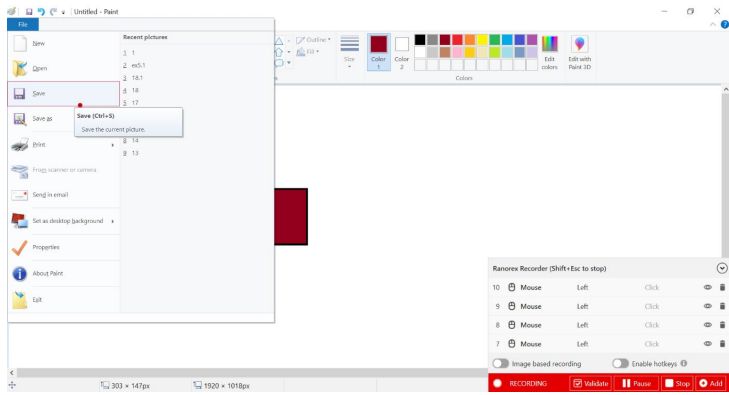
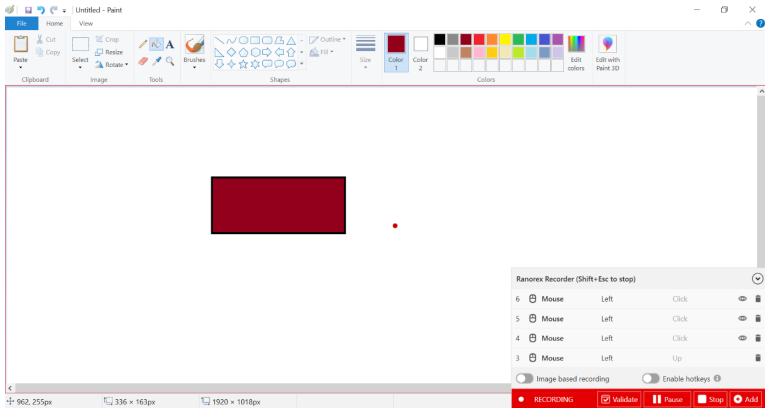
- We begin recording actions that we will perform in the paint application.



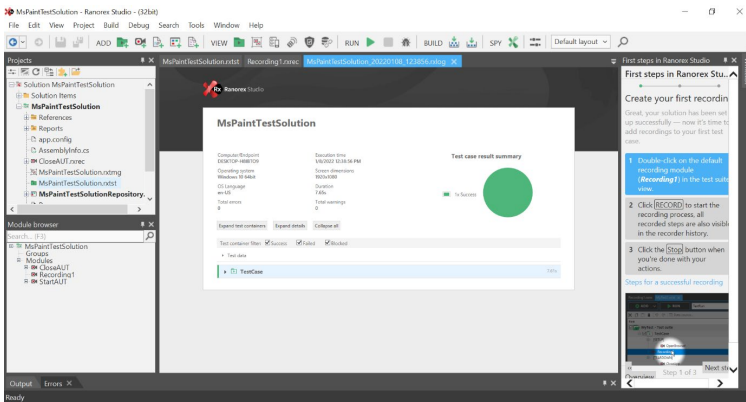
We proceed to create a simple shape in the application and save the file as 'test.bmp.' The Ranorex recorder located at the right-hand bottom corner of the screen starts recording our actions.



Note that each keystroke that we performed is recorded by the Ranorex recorder.



We run the test suite by click on the play icon located in the toolbar at the top.



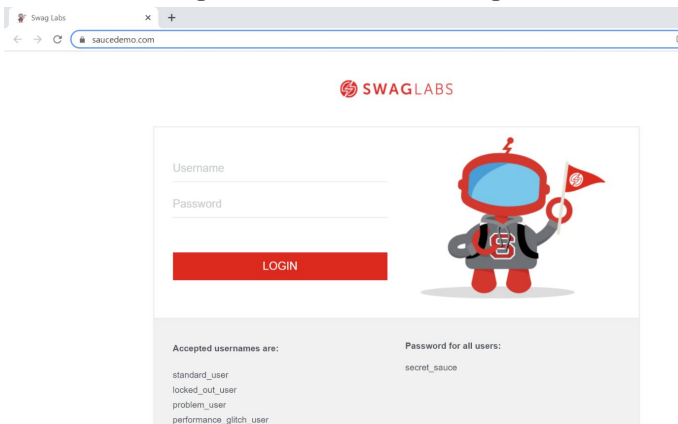
As visible from the screenshot above, the test suite runs successfully.

- **Example 5: Creating a Test Suite for a Demo Website:** In this example, we shall create a test solution for a web application.

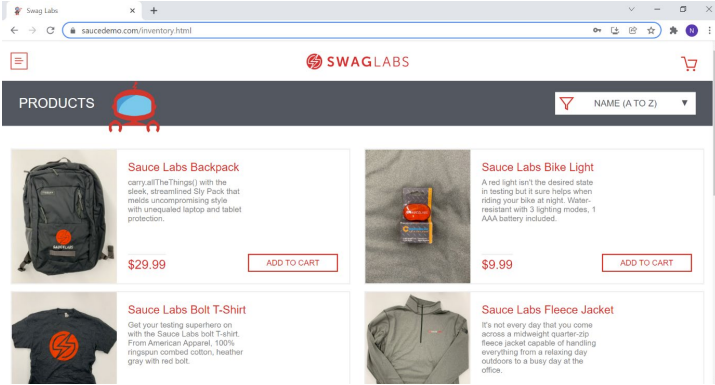
We create a Ranorex solution for an online demo web application using their solution creation wizard and create a test suite for it.

The steps are as follows:

1. We first access a demo website available online. For this exercise, we go to a sample demo website provided by SwagLabs: <https://www.saucedemo.com/> (Swag Labs, 2022).
2. We open the application and login to the demo application using one of the accepted usernames and the password.



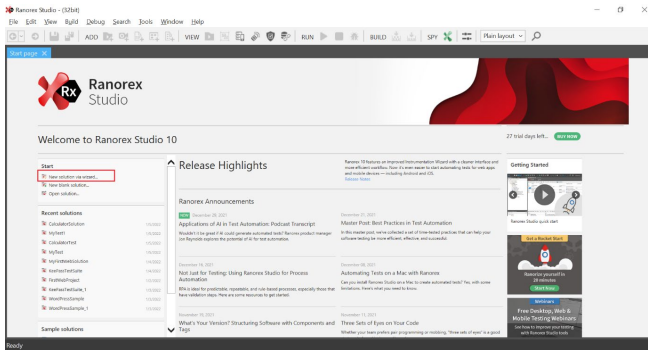
(Swag Labs, 2022)



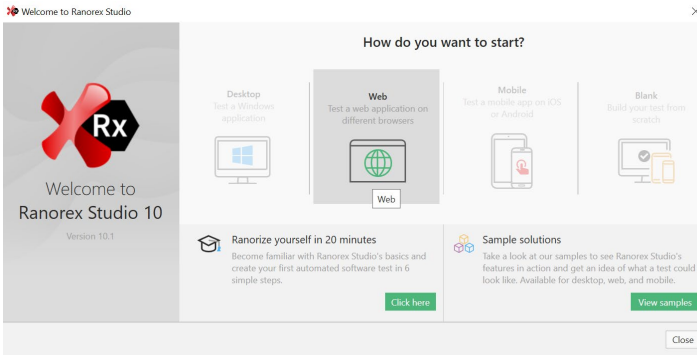
(Swag Labs, 2022)

As we can see, a sample set of products is available for purchase in the demo application.

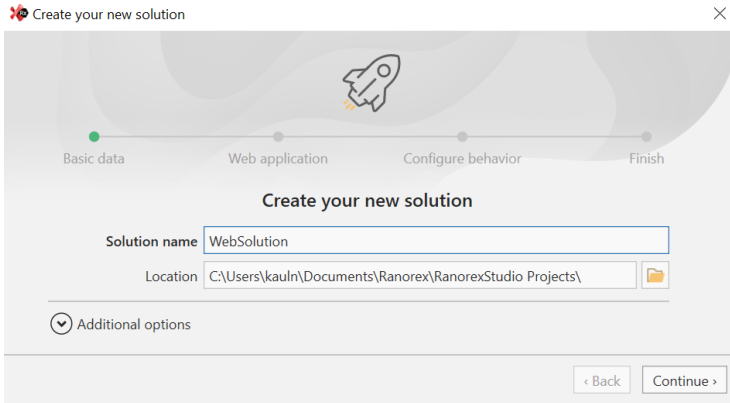
3. To test this demo website, we open Ranorex Studio and create a new Web Solution that tests this website as follows:



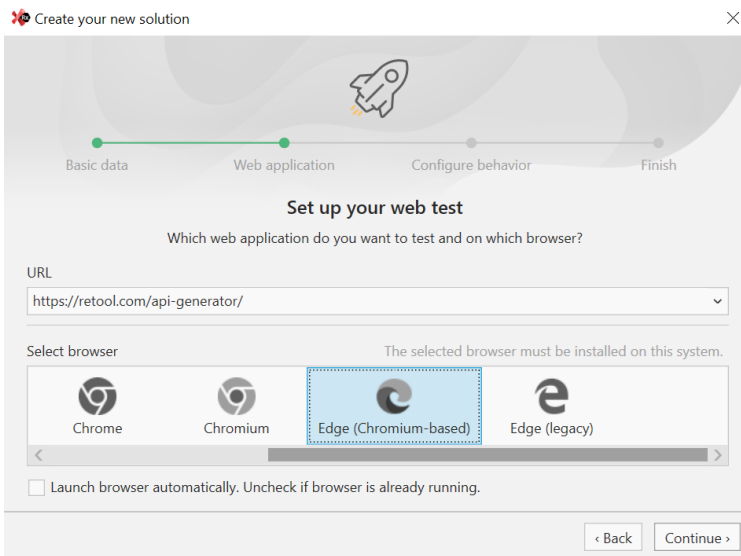
4. We select the Web solution as we are testing a web application.



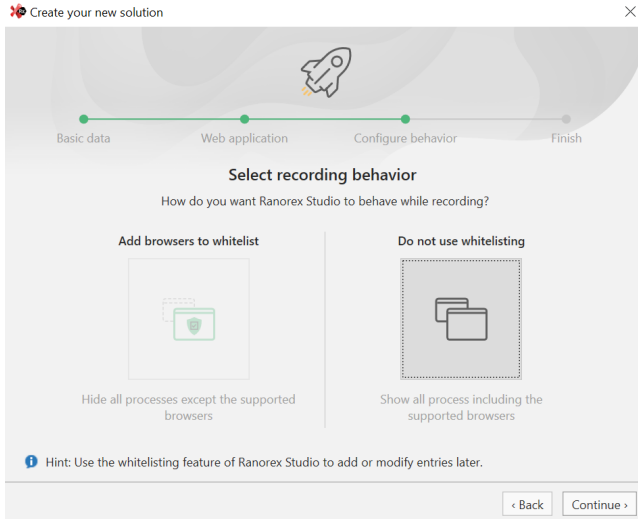
5. We proceed to provide the name of the solution and the location.



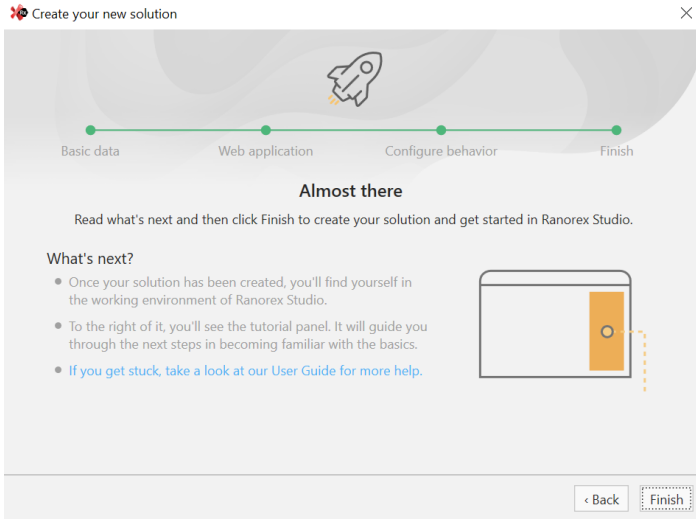
6. We set up the test by providing the URL of the web application and select the browser that we wish to use. In this case, we use the Edge browser.



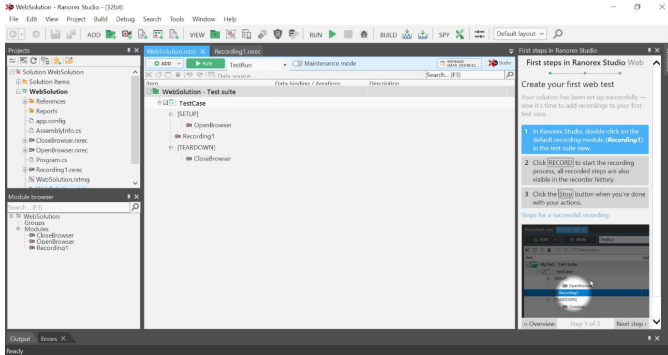
7. We then proceed to select the behavior of the way in which the recordings would take place. We choose not to use whitelisting in this example.



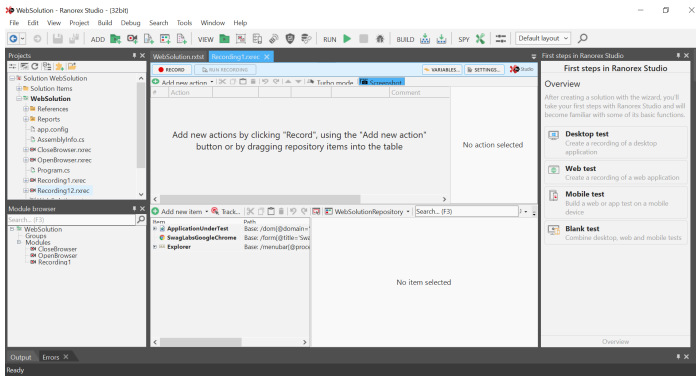
- 8. We finalize the solution and click on 'Finish' to complete the setup.



- 9. On creation of the web solution, a basic test suite with a single test case containing the setup, an empty recording and teardown module is created as shown below:

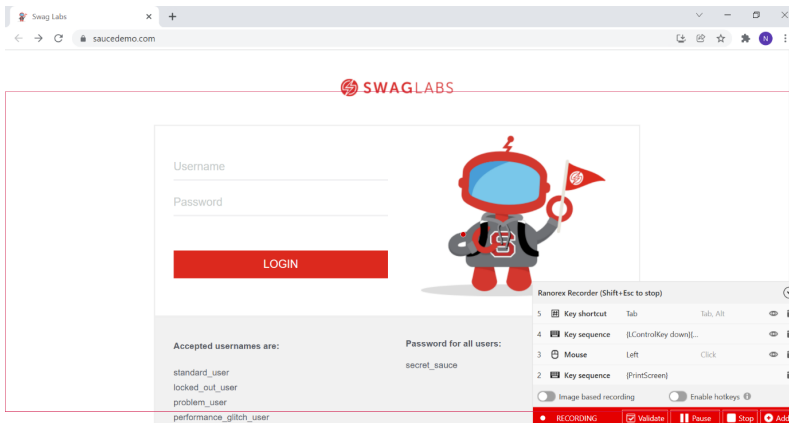


We proceed to view the empty recording file as shown below:



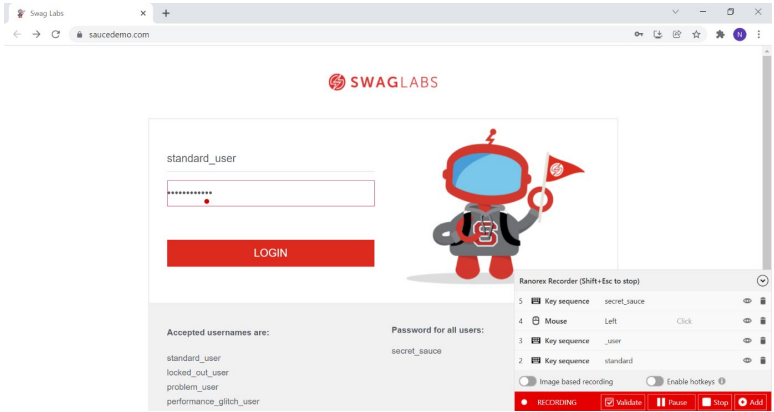
10. We begin recording by clicking on the 'Record' button and start recording activities that we perform on the demo web application that we are testing.

The web application is launched, and the login page opens.



(Swag Labs, 2022)

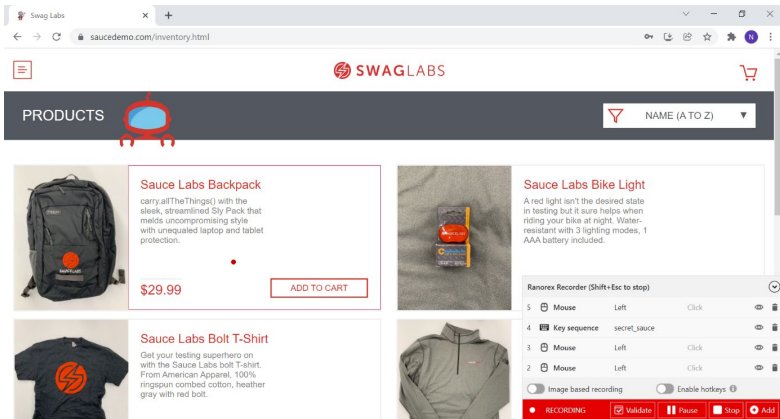
We choose the username ‘standard_user’ from the list of accepted usernames and enter the password provided on the demo site.



(Swag Labs, 2022)

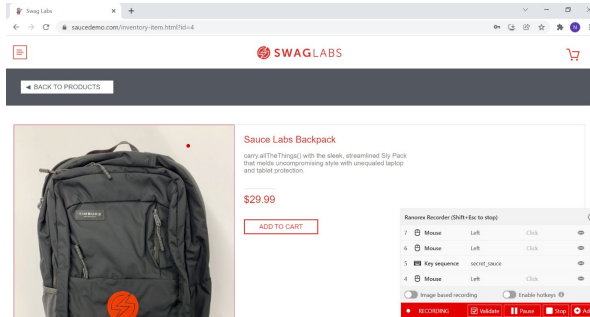
As can be seen from the Ranorex recorder visible in the right-hand bottom corner, each keystroke and action are being recorded.

On logging into the website, the following page with a list of products appears.



(Swag Labs, 2022)

For the test case recording, we select the first option to see it in detail.

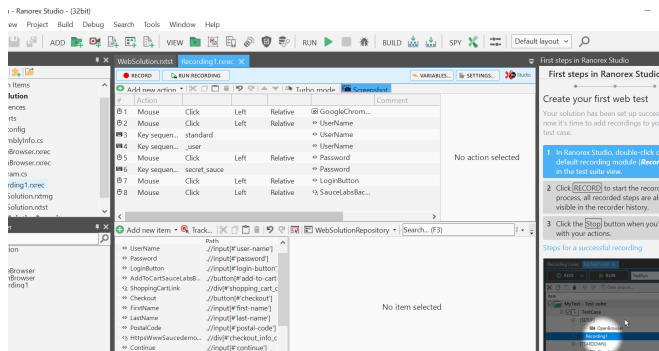


(Swag Labs, 2022)

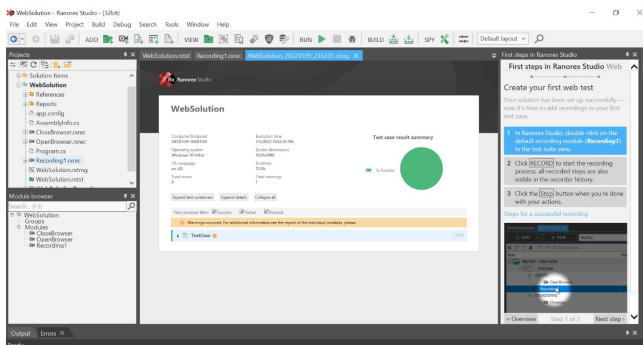
The item we selected appears in detail as clear in the screenshot above.

We stop the recording at this point.

When we stop recording, we can see that the file 'Recording1.rxrec' is now updated with all the actions that we performed.



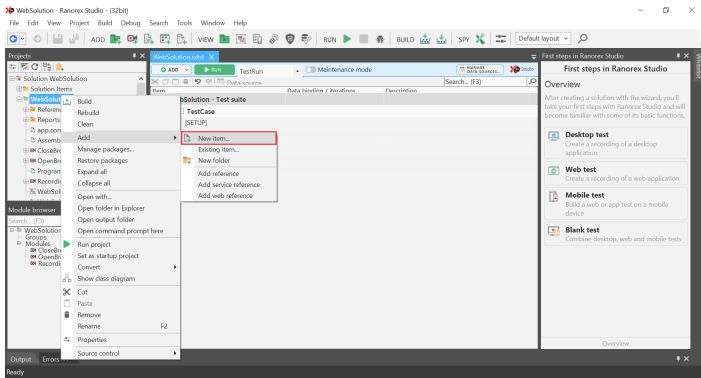
- We proceed to execute the test suite with the test case that we just recorded clicking on the run button (green play icon) at the top.



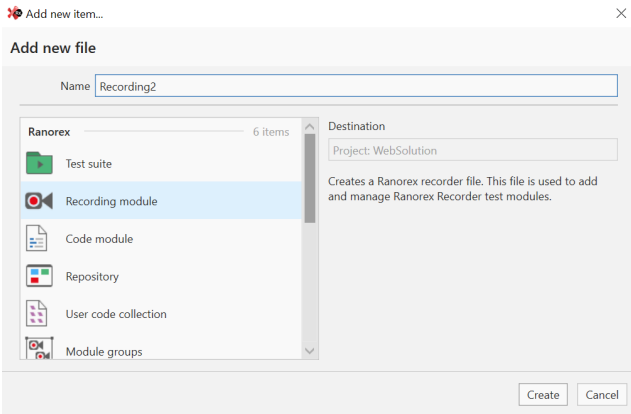
As seen from the test results summary above, the test suite is executed successfully.

12. Now, we proceed to create a new test case that runs the scenario of an incorrect or locked out user login. In this test case, we will try to login using a username that is ‘locked out’ of the demo system and cannot login in.

To create a new test case, we go to the Projects tab, right click on the WebSolution project name, and go to Add → New Item.

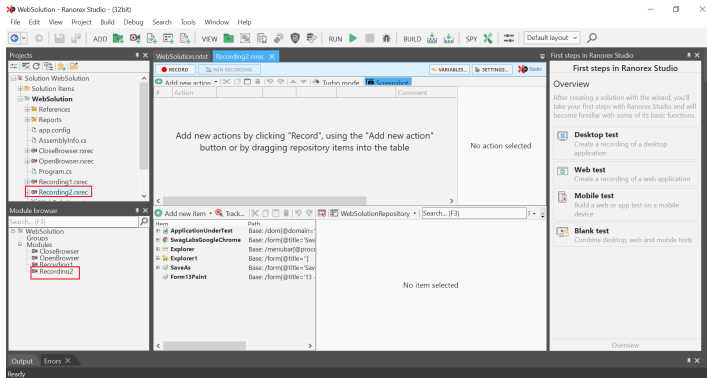


The Add new item dialog appears and we choose ‘Recording module.’

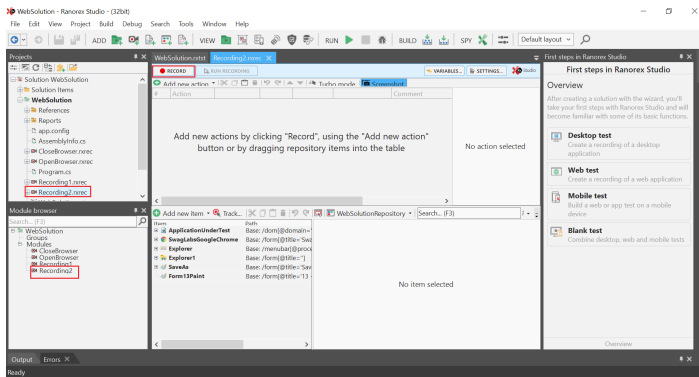


We name it as ‘Recording2’ for the purpose of this example, but ideally meaningful names should be given to each recording based on the test case that is being recorded.

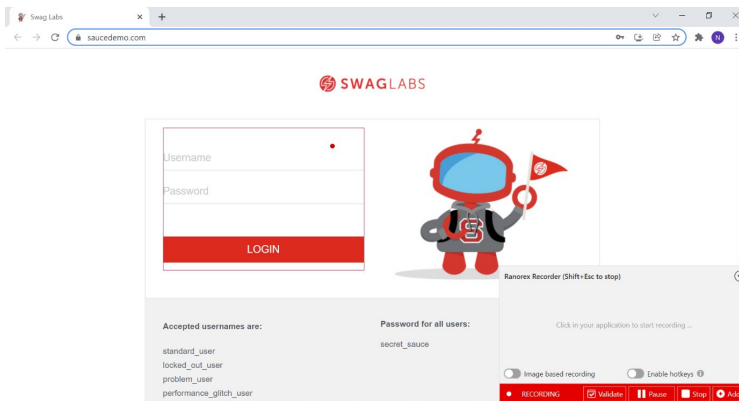
We click on ‘Create’ to create the new recording. It is added to project as demonstrated by the screenshot below:



- We proceed to start recording events in the newly created 'Recording2.rrec' file by clicking on the record button.



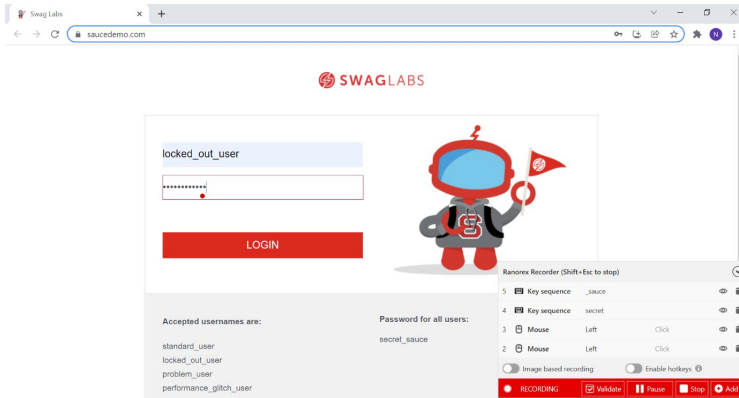
The recording starts and the demo website is launched as follows:



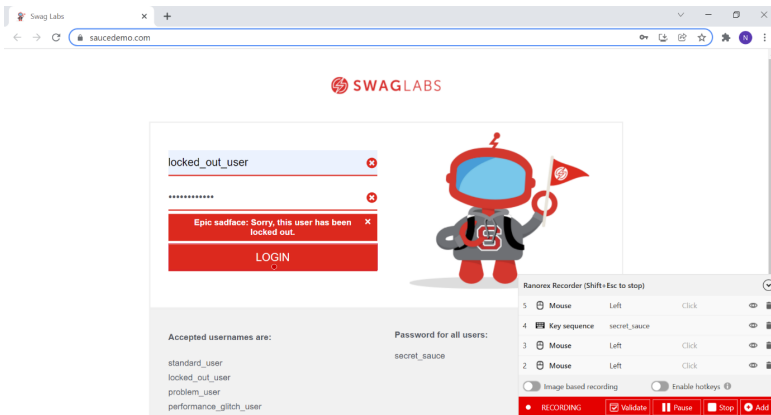
(Swag Labs, 2022)

Note that the Ranorex recorder appears in the right-hand bottom corner and starts recording every action that we do on the application under test.

We login using the test username that fails on login, which is the ‘locked_out_user’ and the password ‘secret_sauce.’



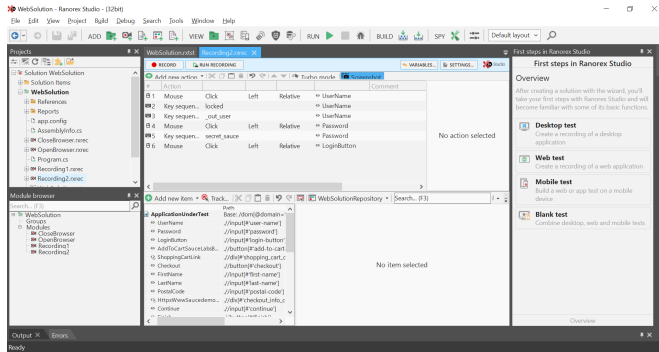
When we click on the ‘LOGIN’ button, we get an error saying that user has been locked out.



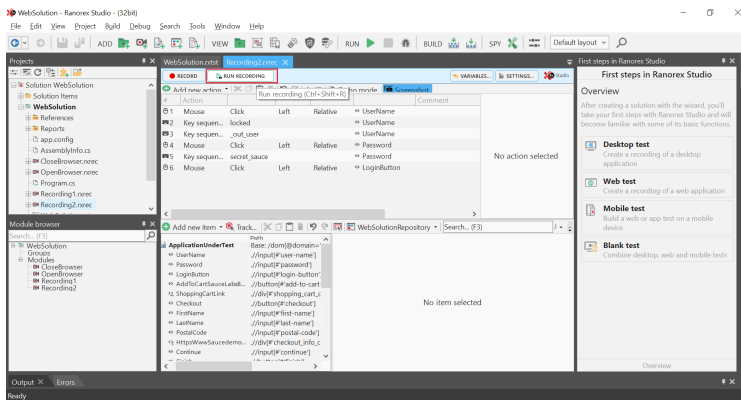
(Swag Labs, 2022)

This action concludes our recording and hence we press the ‘stop’ button on the Ranorex recorder.

As visible from the screenshot below, the actions we performed are recorded in newly created recording file.

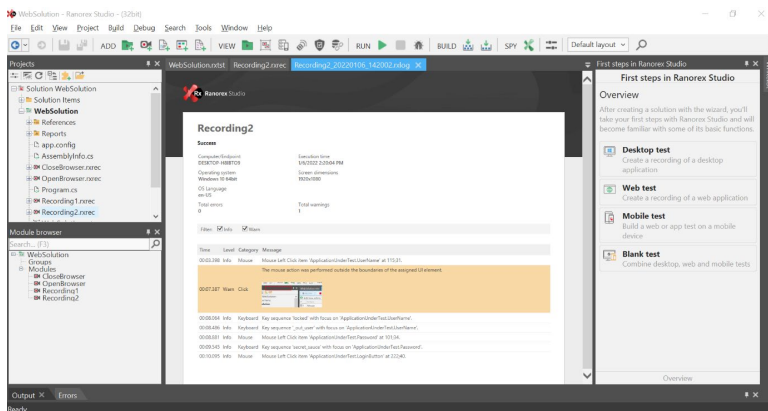


14. We can test the recording independently of other recordings by clicking on the 'Run Recording' button as shown below:



15. On running the recording, we get a result summary just like on executing the test suite.

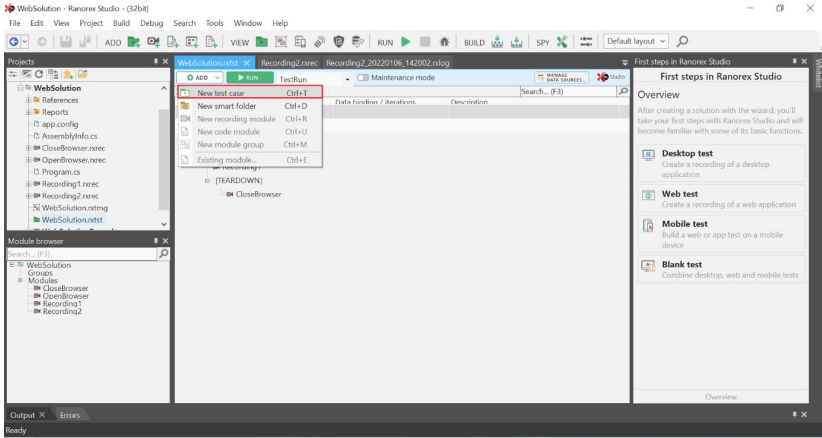
The following report provides us details of the execution of recording2 and we can see that it has executed successfully.



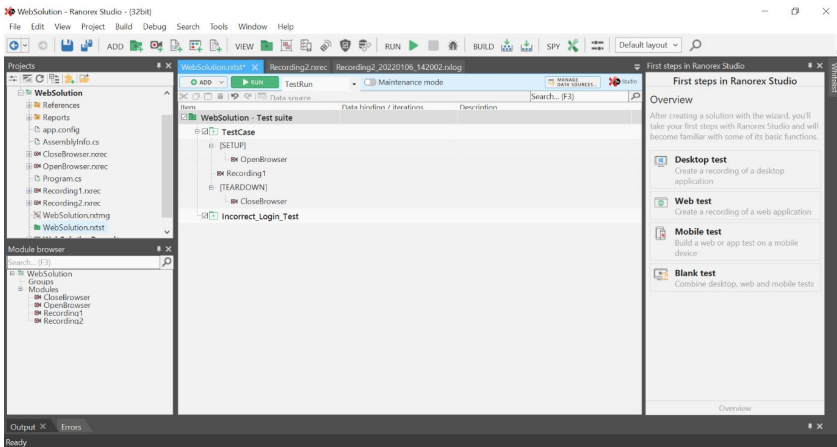
16. We now proceed to add this newly created recording to a new test case which is added to the test suite.

This is done as follows:

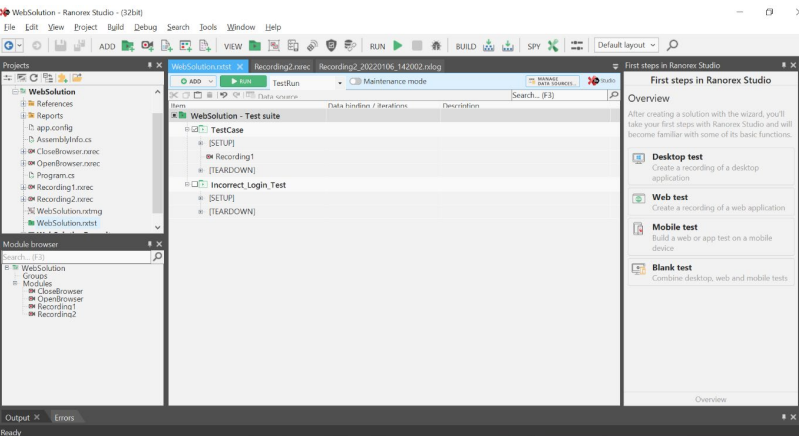
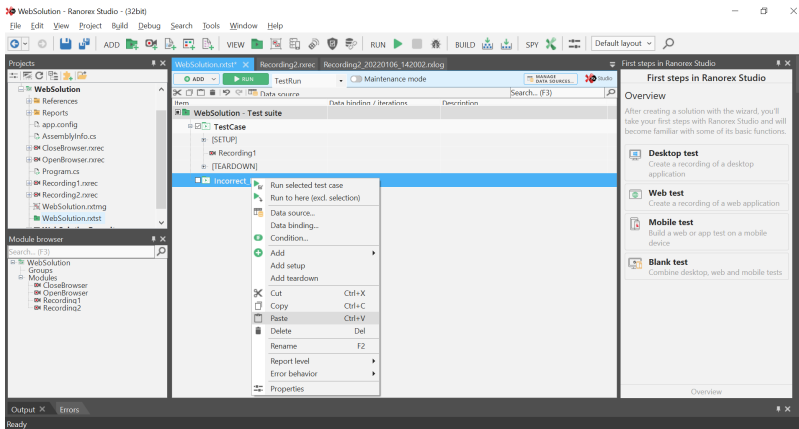
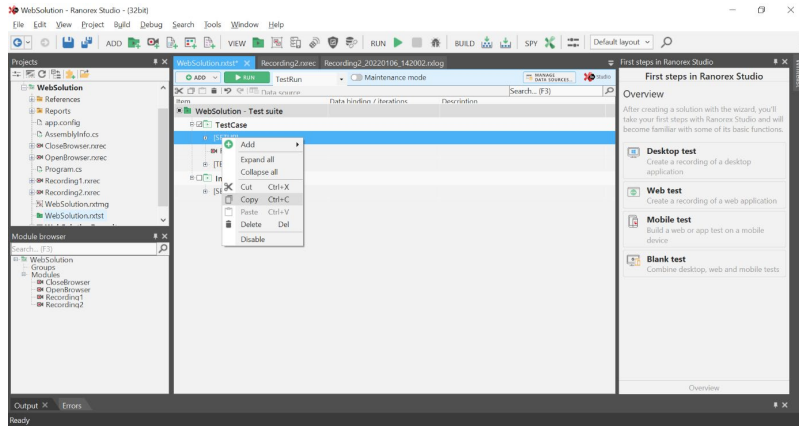
First, we navigate to the test suite file and click on ‘Add’ and then go to ‘New Test Case.’



A new test case is added, and we rename it to ‘Incorrect_Login_Test’ as shown:

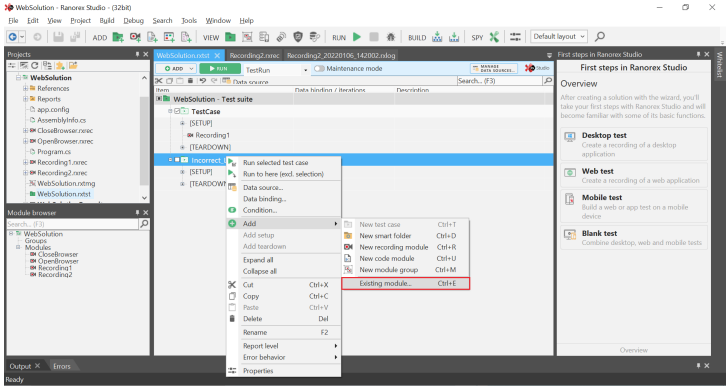


We proceed to adding the setup and teardown modules to the newly added test case by copy-pasting them from the previous test case as follows:



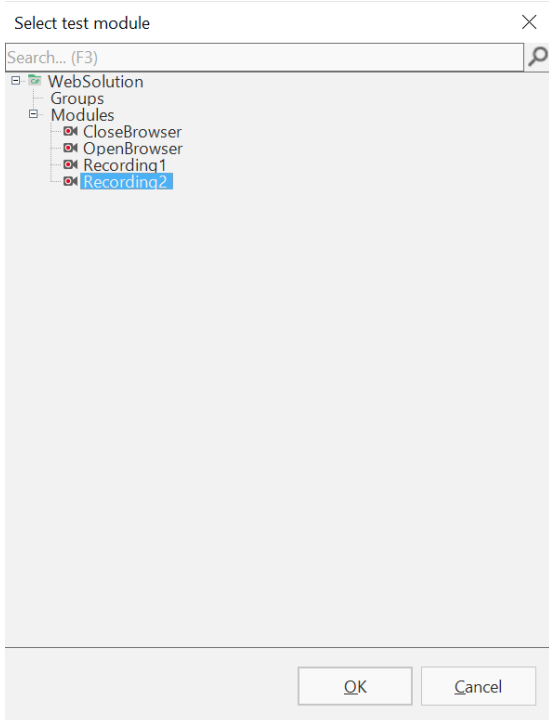
As clearly visible from the screenshot above, the new test case 'Incorrect_Login_Test' now has a setup and teardown module.

We continue and add ‘Recording2’ to this test case by right-clicking on the test case name and going to Add → Existing module...



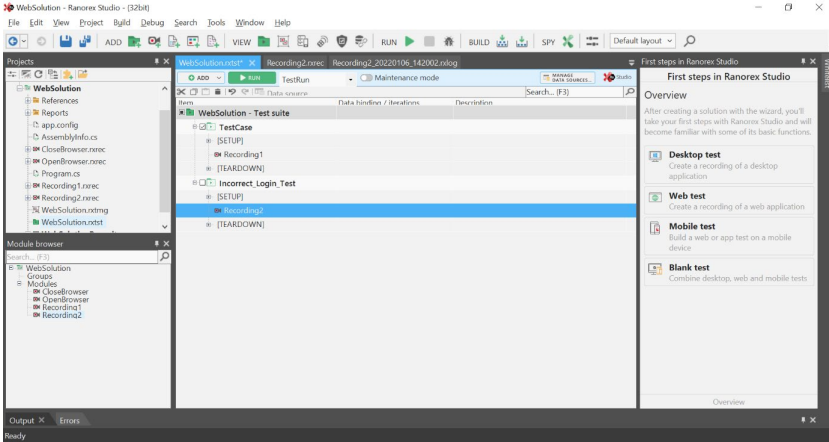
We select the option ‘Existing module’ as we wish to choose Recording2, which we have already created in the previous steps.

The ‘Select Test Module’ window appears, and we select Recording2 from the list of modules.

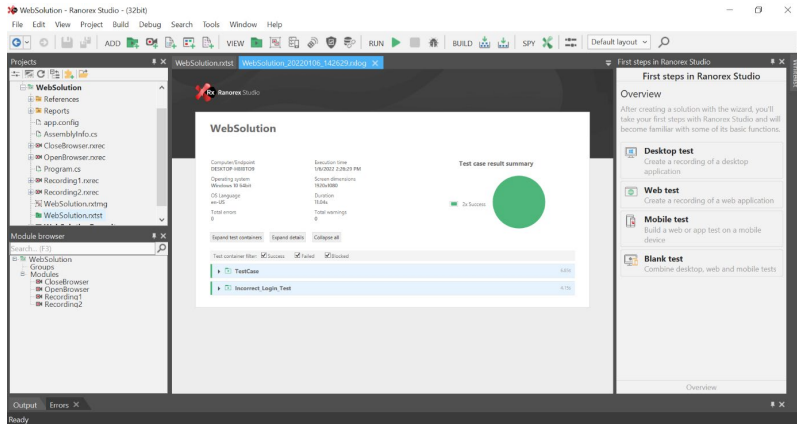


We click on ‘OK’ to add the recording to the newly created test case.

As can be seen in the image below, the newly created test case now has Recording2 in it.

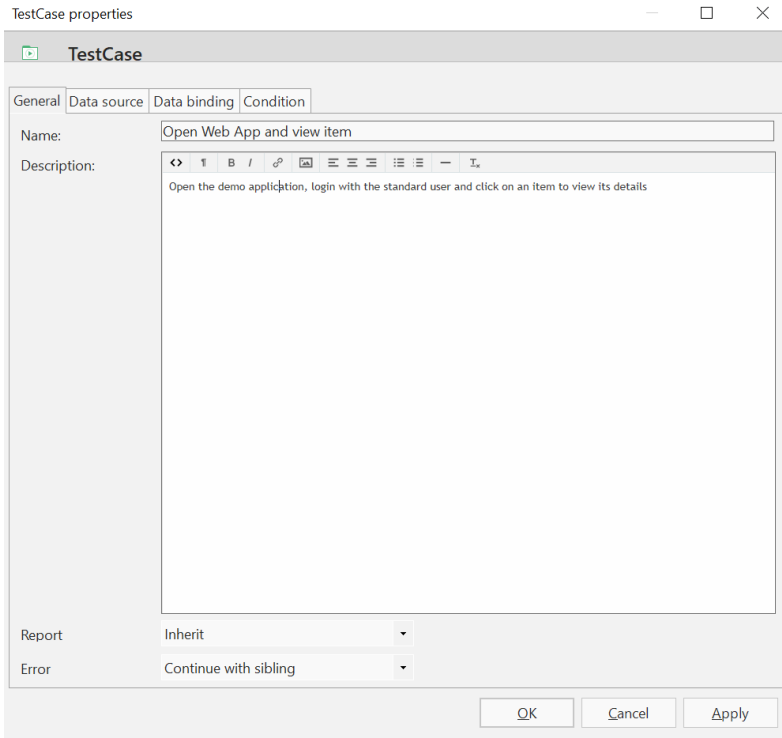
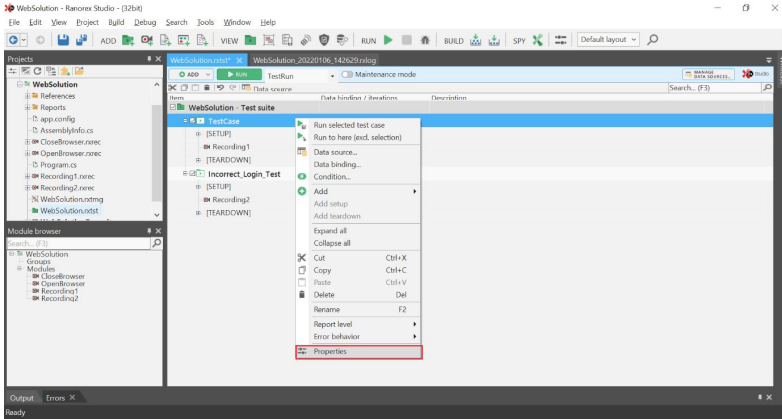


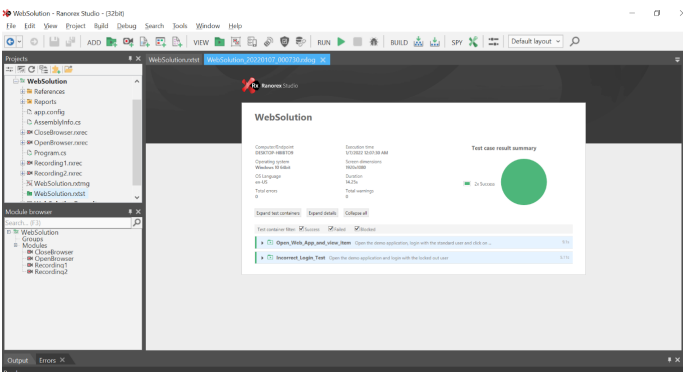
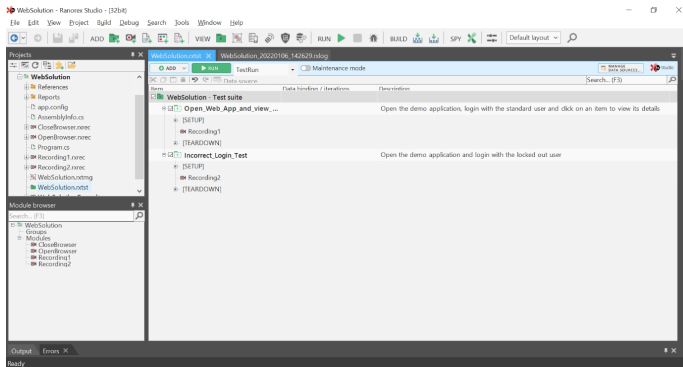
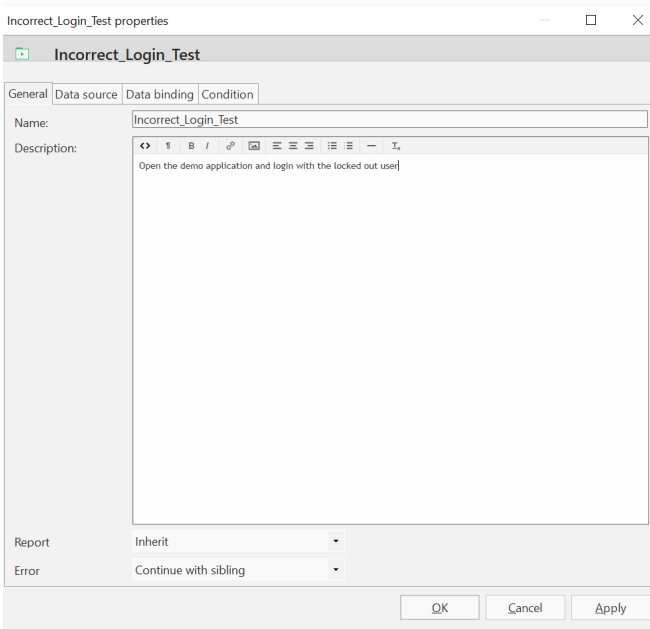
Now that we have completed the second test case, we save the test suite file and execute it by clicking on the play icon located in the toolbar.



As can be seen from the results above, the test suite containing the following two test cases has been executed with success:

- Opening the demo website, logging in and clicking on an item to view its details; and
 - Opening the demo website and logging in with a locked-out user.
17. As the name of the first test case is not very meaningful and does not accurately describe the test case, we update its name and description and then re-run the test suite.





The test suite runs successfully as shown in the screenshot above.

BIBLIOGRAPHY

1. (2017). Ranorex. <https://www.ranorex.com/> (accessed on 24 March 2022).
2. *[WS] Contains String*, (2022). <https://Docs.katalon.com>. <https://docs.katalon.com/katalon-studio/docs/ws-contains-string.html> (accessed on 24 March 2022).
3. *[WS] Send Request*, (2022). <https://Docs.katalon.com>. <https://docs.katalon.com/katalon-studio/docs/ws-send-request.html> (accessed on 24 March 2022).
4. *[WS] Verify Element Property Value*, (2022). <https://Docs.katalon.com>. <https://docs.katalon.com/katalon-studio/docs/ws-verify-element-property-value.html> (accessed on 24 March 2022).
5. *[WS] Verify Elements Count*, (2022). <https://Docs.katalon.com>. <https://docs.katalon.com/katalon-studio/docs/ws-verify-elements-count.html#description> (accessed on 24 March 2022).
6. *[WS] Verify Response Status Code*, (2022). <https://Docs.katalon.com>. <https://docs.katalon.com/katalon-studio/docs/ws-verify-response-status-code.html> (accessed on 24 March 2022).
7. Ammann, P., & Offutt, J., (2016). *Introduction to software testing*. Cambridge University Press.
8. Antunes, N., & Vieira, M., (2012). Defending against web application vulnerabilities. *Computer*, 45(02), 66–72.

9. Bangare, S. L., Borse, S., Bangare, P. S., & Nandedkar, S., (2012). Automated API testing approach. *International Journal of Engineering Science and Technology*, 4(2).
10. Beizer, B., (1995). *Black-Box Testing: Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, Inc.
11. Beizer, B., (2003). *Software Testing Techniques*. Dreamtech Press.
12. Collins, E., Dias-Neto, A., & De Lucena, Jr. V. F., (2012). Strategies for agile software testing automation: An industrial experience. In: *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops* (pp. 440–445). IEEE.
13. *Create Your First Test*, (2022). <https://docs.katalon.com>. <https://docs.katalon.com/katalon-recorder/docs/automate-scenarios.html> (accessed on 24 March 2022).
14. *Cross Browser Automation Testing Using Watir*, (2019). LambdaTest. <https://www.lambdatest.com/blog/cross-browser-automation-testing-using-watir/> (accessed on 24 March 2022).
15. Dustin, E., Rashka, J., & Paul, J., (1999). *Automated Software Testing: Introduction, Management, and Performance*. Addison-Wesley Professional.
16. Ereiz, Z., (2019). Automating web application testing using katalon studio. *Zbornik Radova Međunarodne Naučne Konferencije o Digitalnoj Ekonomiji DIEC*, 2(2), 87–97.
17. Groover, M. P., (2020). *Automation*. Encyclopedia Britannica. <https://www.britannica.com/technology/automation> (accessed on 24 March 2022).
18. <https://docs.katalon.com>. (2022). *[WS] Send Request*. [online] Available at: <https://docs.katalon.com/katalon-studio/docs/ws-send-request.html#returns> (accessed on 24 March 2022).
19. Humble, J., & Farley, D., (2010). *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Pearson Education.
20. *Introduction to Custom Keywords*, (2022). <https://Docs.katalon.com>. <https://docs.katalon.com/katalon-studio/docs/introduction-to-custom-keywords.html#create-a-package> (accessed on 24 March 2022).
21. Joy, B., Steele, G., Gosling, J., & Bracha, G., (2000). *The Java Language Specification*.

22. Jsonplaceholder.typicode.com. (2022). *JSONPlaceholder – Free Fake REST API*. [online] Available at: <https://jsonplaceholder.typicode.com/> (accessed on 24 March 2022).
23. Katalon Solution, (2022). *Katalon | Simplify Web, API, Mobile, Desktop Automated Tests*. [online] Available at: <https://www.katalon.com/> (accessed on 24 March 2022).
24. Khan, M. E., & Khan, F., (2014). Importance of software testing in software development life cycle. *International Journal of Computer Science Issues (IJCSI)*, 11(2), 120.
25. Limaye, M. G., (2009). *Software Testing*. Tata McGraw-Hill Education.
26. Mäntylä, M. V., Adams, B., Khomh, F., Engström, E., & Petersen, K., (2015). On rapid releases and software testing: A case study and a semi-systematic literature review. *Empirical Software Engineering*, 20(5), 1384–1425.
27. McGraw, G., & Hovemeyer, D., (1996). *Untangling the Woven Web: Testing Web-Based Software*, 1, 1–8.
28. Mills, H. D., Dyer, M., & Linger, R. C., (1987). *Cleanroom Software Engineering*.
29. Module: Watir, (2022). *RubyDoc.info: Documenting RubyGems, Stdlib, and GitHub Projects*. <https://www.rubydoc.info/gems/watir-webdriver/Watir> (accessed on 24 March 2022).
30. Pargal, D., (2020). *Gemfile and Gemfile. Lock in Ruby*. Davalpargal. <https://medium.com/never-hop-on-the-bandwagon/gemfile-and-gemfile-lock-in-ruby-65adc918b856> (accessed on 24 March 2022).
31. Peischl, B., Ramler, R., Ziebermayr, T., Mohacsi, S., & Preschern, C., (2011). Requirements and solutions for tool integration in software test automation. In: *Proc. of the 3rd International Conference on Advances in System Testing and Validation Lifecycle* (pp. 71–77).
32. Polo, M., Reales, P., Piattini, M., & Ebert, C., (2013). Test automation. *IEEE Software*, 30(1), 84–89.
33. Potter, B., & McGraw, G., (2004). Software security testing. *IEEE Security & Privacy*, 2(5), 81–85.
34. Project, W., (2009). *Watir Project*. Watir.com. <http://watir.com/> (accessed on 24 March 2022).
35. Pugh, W., & Ayewah, N., (2007). Unit testing concurrent software. In: *Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering* (pp. 513–516).

36. Renfer, G. F., (1962). Automatic program testing. In: *Proceedings of the 3rd Conference of the Computing and Data Processing Society of Canada*.
37. RubyMine: The Ruby on Rails IDE by JetBrains, (2022). *JetBrains*. <https://www.jetbrains.com/ruby/> (accessed on 24 March 2022).
38. Singh, S. K., & Singh, A., (2012). *Software Testing*. Vandana Publications.
39. Sublime Text, (2000). *Sublime Text – A Sophisticated Text Editor for Code, Markup and Prose*. Sublimetext.com. <https://www.sublimetext.com/> (accessed on 24 March 2022).
40. *Swag Labs*, (2022). www.saucedemo.com. <https://www.saucedemo.com/> (accessed on 24 March 2022).
41. *Test Automation, 20 Years After*, (2021). Agile Alliance. <https://www.agilealliance.org/resources/experience-reports/test-automation-20-years-after/> (accessed on 24 March 2022).
42. TestComplete, (2022). *Software Testing Tools Guide*. Retrieved from: <http://www.testingtoolsguide.net/tools/testcomplete/> (accessed on 24 March 2022).
43. *User Guide | Ranorex Help Center*, (2021). Ranorex. <https://www.ranorex.com/help/latest/> (accessed on 24 March 2022).
44. Watir Project, W., (2009). *Waiting*. Watir.com. Retrieved from: <http://watir.com/guides/waiting/> (accessed on 24 March 2022).

INDEX

A

Acceptance testing 20
AddEntry recording 209, 210
address field 178
Agile delivery process 10
API request 50, 59, 76, 81
Application Programming Interface (API) 22
Application Programming Interface (API) Testing 22
authorization 32, 49
Automated software testing 10
Automated testing 10, 11
Automatic program testing 11
Automation 10, 12, 14, 15, 16, 17
automation logs 214
Automation testing 2
Automation tool 10, 14, 24
Automobile industry 10

B

browser driver 119
business email 192

C

calculator application 218, 222, 224, 225
chrome browser 120, 157
chromedriver 119, 120, 122
ChromeDriver website 120
chrome web page 123
code 26, 35, 36, 50
CodeModules 207
coding languages 192
command line 129, 132, 133, 139, 148, 157
command prompt 117, 127, 133, 141, 152, 161, 165, 174
confirmation alert dialog 159
Contains String 57, 58, 61, 62
Continue evaluation 198
Continuous delivery model 10
credentials 214, 215
credentials file 214
cross-browser testing 192
crud actions 168
Customer satisfaction 3, 5
Customer's confidence 6
custom keyword 98, 103

D

data-driven testing 192
 Delete 154, 156, 159, 179
 deleteEntry recording 210
 delete link 154, 158, 163, 172, 179
 demonstration 37, 38
 Desktop 206, 218, 220, 229
 desktop applications 192
 Desktop Automated Tests 26

E

Edge 112, 124, 125, 129, 130
 Edge browser 239
 Edit 114, 149, 153, 177
 edited record 158
 Element Property Value 70, 73, 78, 80, 95
 email address 192
 empty 146, 164, 170
 endpoint 31, 74
 End User License Agreement 195
 entry 53, 54, 55, 57, 58, 67, 68, 71, 72, 79, 80, 91, 92, 94, 96, 108
 error message 164
 execute 33, 35, 58, 59, 60, 62, 65, 82, 86, 87
 execution time 218
 Existing module 250
 expectedStatusCode 56, 91

F

File view 199
 Financial markets 4
 Firefox 105, 112
 Functional testing 16, 20

G

gems 117, 118, 127
 Get Element Property Value 70

getUsers 54, 58
 getUsersResponse 56, 58, 68
 Google chrome 130, 134
 google search bar 106
 google search page 130

H

header 32, 33
 headers tab 33
 Help menu 38
 html 50, 103
 HTTP header tab 32

I

icon 205, 208, 211, 237, 243, 251
 Input 67, 71, 72, 90, 91, 96
 Input dialog 67
 Input Window 71
 installation 26, 29, 114, 117, 120, 124, 126
 installation process 197
 Integration test 18
 Intercom Chat window 205
 Internet Explorer 112

J

Java 192
 JavaScript 50
 JSONPlaceholder 50
 JSON property 71

K

katalon.exe 29
 Katalon Studio 25, 26, 27, 28, 30, 38, 66, 70, 75, 81, 82, 84, 98, 101, 103
 Keepass data 208
 keepass database 217
 keyword-driven testing 192

Keywords 52, 98

L

Leanne Graham 42, 58
 Load New Content 86
 Load Service Function 84, 85
 locator parameter 71, 81
 log viewer section 36
 Log Viewer tab 59

M

machine details 218
 Manually repeating tests 11
 Manual testing 2
 Microsoft 124, 125, 126, 130, 136
 Mobile 26, 103
 mobile applications 192
 module browser 199, 207, 209
 Module View 199
 MSI installer package 193
 multiple records 168

N

New entry 52
 New Web Service Keyword 66
 Non-functional parameters 17
 Non-functional tests 17

O

Object 54, 59, 75, 82, 107, 108
 object-oriented programming 106
 object-oriented scripting language
 Ruby 125
 official website 114
 OpenApi2 36
 Open-source library 112
 operation 86
 Output 54, 80

P

package 29, 98, 99, 100, 101
 paint application 229, 230, 233
 parallel testing 192
 parameters 33, 52, 58, 62, 71, 91,
 96, 103
 Performance test 20
 Personal Information Form 143
 pre-defined link 37
 PrintHello 90
 Privet Drive Street 151
 program 35, 52
 project 30, 38, 39, 40, 48, 59, 64,
 100
 Project Section 50
 Project View 199

Q

Quick Guide menu 38
 Quick Guide screen 30

R

Ranorex Recorder 200, 201, 226
 Ranorex repository 201
 Ranorex Spy 200, 201
 Ranorex Studio 191, 192, 197, 198,
 199, 200, 201, 204, 206, 220,
 229, 238
 Ranorex Test Suite Runner 201
 Ranorex website 192
 Recorder 200
 Recorder activity table 200
 Record Web 104
 record window 225
 recycle bin 215, 217
 regression testing 192
 Regression testing 16, 17, 19, 24
 Request message 86

responseObject 56, 91
 ResponseObject parameter 58
 returned data 71
 Rigorous testing 4
 Ruby 112, 114, 118, 120, 125, 185
 Ruby code 112, 118
 ruby file 127, 133, 137, 150, 161,
 174, 175, 182
 Ruby interpreter 112
 Ruby Interpreter 114
 Ruby language 112, 114

S

Safari 112
 screen 33, 36, 58, 59
 screenshot 33, 35, 37, 54, 60, 68,
 70, 73, 87, 88, 90, 93, 96, 100,
 103
 script view 35
 Security tests 19
 Select Test Module 250
 selenium web driver 119
 Send request 89
 Send Request 53, 61, 78, 79, 91
 Service Endpoint 87
 Smoke Test 18
 Software community 2
 Software development life cycle 2
 Software product 3, 4, 5, 6, 7
 Software system 2, 4, 6
 Software Testing 1, 2
 source files 36
 standalone applications 192
 StartAndLogin 207
 straightforward 114
 Street address 150, 151
 Sublime Text 143
 Submit 147, 154, 166
 Supplementary costs 12

T

tab 34, 36, 77, 85, 88
 Tanorex Spy 200
 TDD (test driven development) 17
 teardown module 240, 249
 test1.rb file content 131
 Test automation 10, 11
 Test case 30, 34, 36, 50, 51, 52, 53,
 54, 55, 58, 59, 60, 61, 62, 63,
 66, 67, 69, 70, 71, 72, 73, 77,
 78, 79, 80, 82, 88, 89, 94, 95,
 96, 97, 102, 103, 104, 107,
 108, 109
 test/dummy data 38
 Test parameters 52
 Tests Explorer 98
 test suite 34, 35, 36, 59, 63, 64, 65,
 66
 toolbar 205, 211, 237, 251

U

Unit testing 16, 17, 22
 useRegex 58
 User interface (UI) 18

V

validateEntry recording 210
 validation errors 166
 variable 54, 56, 58, 68, 71, 78, 80,
 89, 90, 91, 93
 Verify Element Property Value 73,
 78, 80
 Verify Elements Count 66, 67
 Verify Element Text 92
 verify response status code 55

W

Watir 111, 112, 114, 118, 122, 123,
 125, 126, 129, 130, 135, 138,

- 139, 143, 149, 159, 170, 177
 - watir package 118
 - watir test 119, 151, 180
 - web 38, 39, 44, 54, 55, 57, 66, 73, 77, 80, 82, 83, 84, 85, 88, 89, 90, 92, 93
 - web applications 192
 - Web Application Testing 112
 - web browser 112, 120, 130
 - Web Browser 104
 - webdriver 119, 124
 - Web Record Dialog 106
 - web server 53
 - Web Service keywords 53
 - Web Service Keywords 52, 55, 57
 - web services 38, 44, 84
 - Web solution 238
 - whitelisting 203, 239
 - window 37, 40, 50, 51, 63, 64, 82, 83
 - windows machine 114
 - wizard 202, 203, 223, 224, 229, 237
- X**
- XML 50
- Z**
- zip file 192

Implementing Automated Software Testing

Software systems are an essential component of our day-to-day lives. Software has morphed into one of our most basic necessities. We depend on software to accomplish routine tasks and activities in our lives. The impact that software has made in this world is enormous. Software applications are engaged on a large scale in both essential and non-essential sectors across the world. The immensity of the involvement and impact of software in the world is enormous. Software applications and systems help in completing complex tasks in an easier and cost-effective way which has improved the quality of life of millions of people.

The relevance of software will only increase in the future. Software will continue to increase in complexity as it is going to be used to solve the biggest problems faced by the world. It is a proven fact that as the complexity of software rises, the challenges associated with software testing and software maintenance will subsequently rise too. Complex software solutions created to solve complicated problems are destined to have an element of complexity in the process of testing as well. It is a known fact that there is a direct relation between the quality of a software system and testing. Effective testing is thought of as a measure of efficiency and quality of software. Testing is a vital non-skippable step in the software development lifecycle.

The field of software testing has grown considerably since its origin in the early 1900s. Testing of software helps instill confidence in the quality of the software to its users and stakeholders. It is advisable that software be evaluated and tested thoroughly as it is vulnerable to a range of potential attacks that may be of a malicious nature. Comprehensive testing and evaluation of any software solution is paramount to identifying vulnerabilities in the system and shield it from potential attacks. The lack of quality testing is one of prime reasons for the failure of software systems resulting in significant losses for the stakeholders, clients, and users. One can assume with certainty that software testing is a determining factor in the success of a software solution.

Over the last few decades, the field of software testing has grown exponentially. A branch of software testing namely automation testing has helped reshape the way in which testing is done. Automation testing is the exercise of executing tests automatically in a repetitive manner. This branch of study concentrates on execution of repetitive tests, management of the test data and the utilization of the results to ameliorate and improve the quality of software. In this book, we will investigate the automation testing field within software testing. The advantages of this type of testing will be discussed in brief. Further, different software solutions that perform automation testing will be presented and examined in this book. The implementation of different automated software systems will be presented in detail. Detailed practical implementations of automated software applications covering different types of testing scenarios have been provided in this book. This book assumes that the audience is familiar with the basic concepts of software testing.



Neha Kaul is an experienced software consultant and technical author currently residing in Sydney, Australia. She is the author of five technical books: Object Oriented Programming with Java, Logging Frameworks in Java, Applications of Data Mining in Engineering, Management and Medicine, Software Security: Building Secure Software Applications and Analytic Methods of Systems and Software Testing. She received her double Master's Degree in Computer and Communication Networks and Information Technology from Telecom SudParis and University Paris-Saclay in 2016. She is a recipient of the prestigious Telecom Scholarship for Excellence provided by Fondation Telecom, France. She received the Bachelor of Engineering degree in Computer Engineering from the University of Pune, India in 2011. From 2011 to 2014, she was employed as a Software Engineer with Geometric Ltd, Pune, India. Her major avenues of research include Advanced Java/J2EE frameworks, Automation Testing, Software Quality, Software Security, Logging Frameworks, Project Management and Leadership.

