**Automation Testing Documentation**

**I. Automation Testing**

**1) What is automation testing?**

Test automation refers to the practice of using software tools and scripts to automate the execution of tests in software development and quality assurance processes. It involves creating scripts or test cases that can be run automatically, rather than requiring manual effort. Test automation aims to increase the efficiency and effectiveness of testing by reducing human errors, saving time, and improving test coverage.

**2) What is the framework architecture?**

There are some common testing automation frameworks:

- Linear Automation Framework:

+ Also referred to as a record-and-playback framework

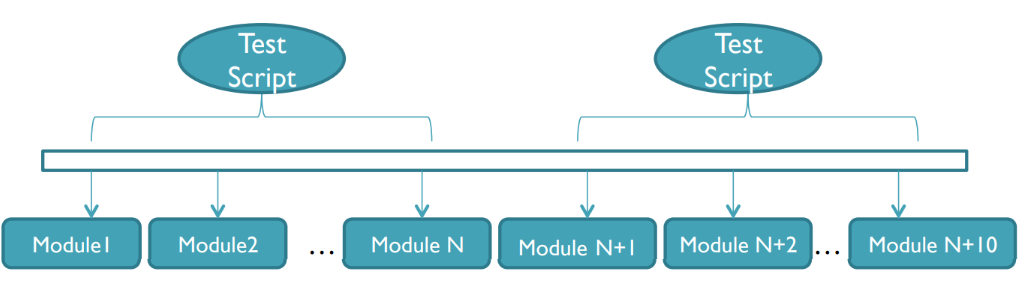
+ Testers do not need to write code to create functions and the steps are written in sequential order

+ Testers record each step (navigation, user input, checkpoints, etc.), play the script back automatically to conduct the test.

- Modular Based Testing Framework:

+ The framework builds an abstract layer for a component to hide that component from the rest of the application. Therefore, the changes made to the other part of the application do not affect that component.

+ Involve the creation of small, independent scripts that represent the modules of the application under test. These modules in turn are used in a hierarchical fashion to build large test cases.



**Figure 1:** Modular Based Testing Framework architecture

- Library Architecture Testing Framework:

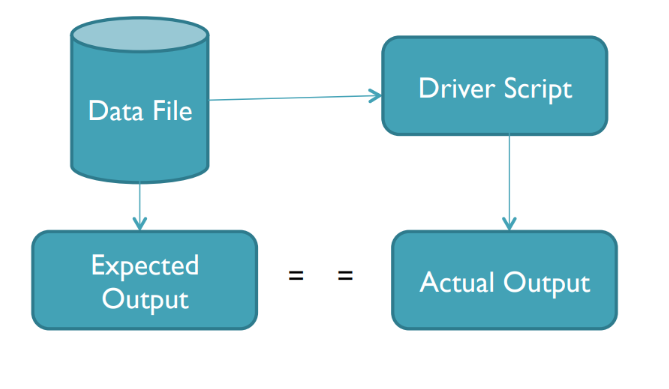
+ Based on Modular Testing Framework

+ Similar tasks within the scripts are identified and later grouped by function. These functions are kept in a library which can be called upon by the test scripts whenever needed.

- Data – Driven Testing Framework:

+ The test input and the expected output results are stored in a separate data file (normally in a tabular format) so that a single driver script can execute all the test cases with multiple sets of data.

+ The driver script contains navigation through the program, reading of the data files and logging of the test status information.

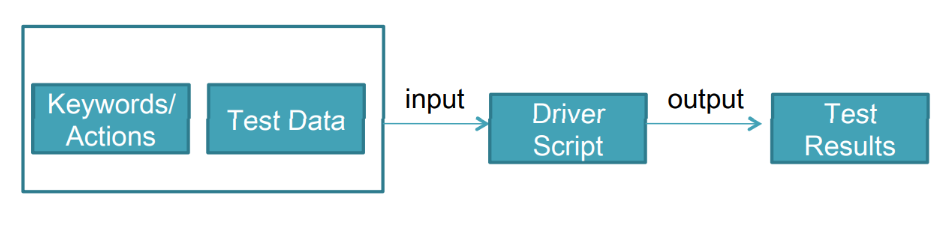


**Figure 2:** Data – Driven Testing Framework architecture

- Keyword – Driven Testing Framework

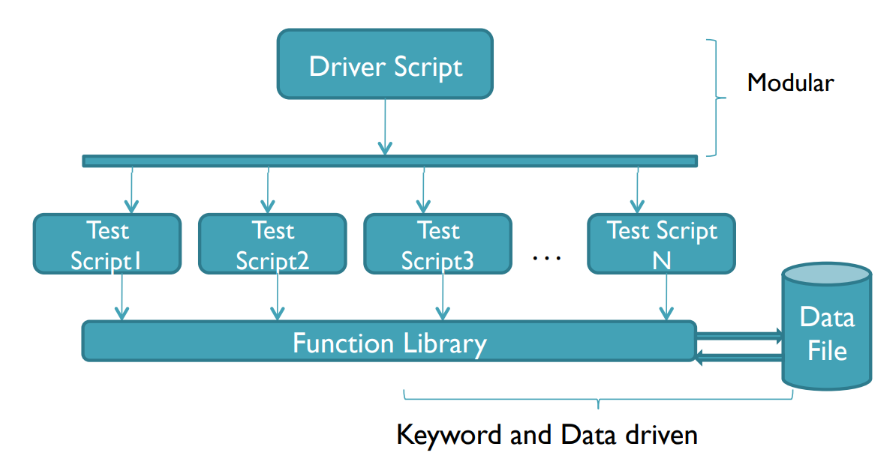
+ An application independent framework utilizing data tables and self-explanatory keywords to explain the actions to be performed on the application under test.

+ The test data and the directives (called keywords) telling what to do which in the test scripts are put in the external input data file.



**Figure 3:** Keyword – Driven Testing Framework architecture

- Hybrid Testing Framework: A combination of any of the previously mentioned frameworks to leverage the advantages of some and mitigate the weakness of others



**Figure 4:** Hybrid Testing Framework using the combination of Modular Based, Keyword and Data Driven Testing Framework

**3) What is Selenium Framework?**

Selenium Framework is a suite of automation testing tools based on JavaScript framework. It could run the tests directly on the target browser, drive the interactions on the required web page and rerun them without any manual input. It eliminates repetitive manual testing that consumes lots of time and effort.

**4) What is Selenium IDE?**

Selenium IDE is an open – source record / run tool that a test case developer uses to develop Selenium Test cases. Selenium IDE is an easy – to – use tool from the Selenium Test Suite and can even be used by someone new to developing automated test cases for their web applications.

**5) What is Selenium WebDriver?**

Selenium WebDriver is the core component of Selenium which provides a programming interface for driving the web browsers. It allows us to write tests in different programming languages to interact with the web elements, simulate user interactions and perform assertions.

**6) What are key features and benefits of Automation Testing?**

- Key features:

+ Executed programmatically

+ Repeatable

+ Test ends with a validation

+ Stable enough to be used in CI / CD

+ Easy to understand

+ Requires as little maintenance as possible

+ Test is independent and can run in parallel with all other tests

- Benefits:

+ Cost efficiency

+ Improve test accuracy

+ Improve test coverage

+ Maximize ROI (Return on Investment)

+ Fast Feedback loop

+ Better Allocation of Resources

+ Detects bugs earlier

+ Test at Scale

**7) Why choose Selenium Framework (Benefits of Selenium Framework) ?**

+ Allow running tests across various web browsers like Chrome, Firefox, Safari, IE, Opera, Edge.

+ Open - source, free to use and distributed with no upfront cost.

+ Support multiple programming languages: C#, Java, Ruby, Python and more.

+ Can integrate with various testing frameworks like JUnit, TestNG, NUnit and more.

+ Easy customization and extensibility to meet specific testing needs.

+ Test scripts can be executed on any operating system.

+ Allow parallel test execution, reduce overall testing time, enabling faster feedback and quicker releases.

+ Allow running tests repeatably.

**II. Selenium Automation Framework**

**1) Selenium IDE**

**a) Architecture and Workflow:**

A diagram of a computer program

Description automatically generated with medium confidence

**Figure 5:** Selenium IDE Architecture

A diagram of a chemical reaction

Description automatically generated

**Figure 6:** Selenium IDE Workflow

A diagram of a process

Description automatically generated

**Figure 7:** Create and Execute Test Workflow

**b) Main features:**

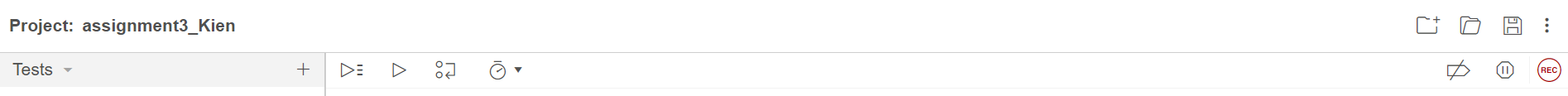
+ File Options at the beginning: File Options allows users to create a new project (with or without recording a test) and open an existing project.

A screenshot of a computer

Description automatically generated

**Figure 8:** Selenium IDE after opening the extension

+ Toolbar: The toolbar contains buttons for controlling the execution of the test cases/suites, including a step feature for debugging the test cases. An input for user to input the URL link for testing and search tests. The right-most button (the one with the red color with REC inside) is the record button. File Options also appears in the Toolbar.



**Figure 9:** Selenium IDE Toolbar

+ Test Case pane and Editor:

The Test Case pane shows all the tests user created in the project.

In Test mode, the user can view existing tests (if any) and create a test in the project. For each test, the user can choose to duplicate, rename, delete, or export it to different programming language and frameworks.

In Test Suite mode, users can view an existing test suite (if any) and create a test suite in the project. For each test suite, users can choose to add tests, rename, delete, change settings while running the test suite, or export the test suite.

Executing mode is used to view all the running tests in the project.

The script is displayed in the editor, with an input for user to type in an URL for testing.

The Command, Target, Value and Description entry fields display the currently selected command along with its parameters. These are entry fields where users can modify the currently selected command.

A screenshot of a computer

Description automatically generated

**Figure 10:** Test Case pane and Editor

+ Log/Reference pane:

The bottom pane is used for two different functions: Log and Reference, depending on which tab is selected.

*Log:* When user runs the test case, error messages and information messages are displayed in this pane automatically, even if user does not first select the Log tab. These messages are often useful for test case debugging. The Clear button is used for clearing the Log.

A white background with a black border

Description automatically generated with medium confidence

**Figure 11:** Log pane

*Reference:* The Reference tab is the default selection when entering or modifying Selenese (Selenium commands) commands and parameters. The Reference pane will display documentation of the current command. When entering or modifying commands, it is critically important to ensure that the parameters specified in the Target and Value fields match those specified in the parameter list in the Reference pane. The number of parameters provided must match the number specified, the order of parameters provided must match the order specified, and the type of parameters provided must match the type specified. If there is a mismatch in any of these three areas, the command will not run correctly.

*A screen shot of a computer

Description automatically generated*

**Figure 12:** Reference pane

+ Export to different programming languages:

Selenium supports many different programming languages like Python (Pytest), Java, Ruby, C#. Just right – click on the Test Case (or Test Suite) that user wants and choose Export. Then choose a suitable programming language (with its corresponding framework). Selenium can also export the test to run on Selenium Server Grid

A screenshot of a computer

Description automatically generated

**Figure 13:** Export the Test Case to different programming languages and frameworks

**c) Pros and Cons:**

- Pros:

+ Selenium IDE is very easy to use.

+ It has the capability to convert the test to different programming languages such as Java, Ruby, Python, etc.

+ Programming language experience is not required for Selenium IDE (only know some basic HTML, CSS, JS)

+ Selenium IDE provides Logging capabilities using file login plug-in.

+ In Selenium IDE, user can debug and set breakpoints.

+ Support multiple browsers like Chrome, Edge, Firefox, Safari

+ Free to use

+ User friendly

- Cons:

+ Not supporting iteration

+ Does not support testing many features like verify upload and download file, database testing

+ Testing mouse moving and scrolling is not supported

+ Cannot create test report, user can only see it in Log pane

+ Lack of professional support

**2) Selenium WebDriver**

**a) Architecture:**

**A diagram of a computer program

Description automatically generated with medium confidence**

**Figure 14:** Selenium WebDriver architecture

- Automation Code / Client Library:

Everything starts from writing automation code in a client library provided by Selenium and as anyone associated with testing would know, this list is endless. Selenium is popular for offering support for all the major programming languages through its bindings to write the WebDriver code in. These are then wrapped around the library to create browser-specific code.

- JSON Wire Protocol over HTTP Client:

Once the test cases are constructed, the data needs to be communicated to the browser driver in some way. In Selenium WebDriver, this is achieved by following JSON (JavaScript Object Notation) Wire Protocol with HTTP as the communication handler. The actual internal work happens with RESTful Web Services and the tester’s system works as a client. This gets connected to the HTTP server provided by the browser driver in their library.

Since the communication is established in a client-server infrastructure, it is referred to as a “request-response” system or “command-response” system in the Selenium WebDriver world.

- Browser Driver:

The actual browser we use in our daily lives does not let the end-user know its internal working or code structure. Even if a tester would want to perform simple actions such as selecting an element through a locator, they won’t be able to do that directly. To accomplish this, an intermediary is required that can understand the intent of an automation script and perform that action on the actual browser. This intermediary is a browser driver. A browser driver communicates with both Selenium and the browser to perform the test execution. Since the working of a browser is confidential, browser drivers are therefore not generic and are developed and provided by the browser developers only. Therefore, for each browser, the testers may require one browser driver with the same version as the installed browser. From the tester’s end, the Selenium scripts will always target a browser driver rather than a browser.

- Browser:

A browser, with respect to the architecture, is the actual browser we use for browsing the internet. Once the browser driver understands the JSON sent, it communicates it to these browsers over HTTP using an internal HTTP server. The response is received using the same protocol and communicated to the client, hence completing the cycle.

*Workflows summarize:*

- A tester  **writes an automation test script** targeting a specific browser driver.

- When they click the “Run” button, **the script is converted to the API** format with data in JSON format.

- The browser driver receives the data and if validation is successful, it  **communicates those actions to the browser** via HTTP.

- If the validation is rejected, **the errors are communicated back** to the client.

- Once the browser initializes, the driver **performs the actions one by one**, hence performing the testing like a manual tester through automation.

- The **commands are sent through HTTP** and the response is received via the same protocol to the driver.

- Once all the actions are performed, the  **browser shuts down** and the driver communicates the results to the client.

**b) Pros and Cons:**

- Pros:

+ Open – source

+ Cross – browser and cross – platform compatible

+ Enhances re – usability

+ Provides Selenium integration

+ Parallelism support

+ No server required

+ Consumes fewer resources

+ Easy to learn and implement

+ Large community support

+ Support for multiple programming languages

- Cons:

+ Limited Support for Desktop Applications

+ No Built-in Reporting

+ Learning Curve

+ Handling Dynamic Elements

+ No Image Testing

+ No support for CAPTCHA and Barcode Readers.

**III. Design a Test Cases using Selenium IDE and Selenium WebDriver**

**1) Test Case and Test Scenarios:**

**a) Test Environment:**

A screenshot of a computer

Description automatically generated

**Figure 15:** Test Environment

**b) Test Sign in:**

**A white sheet with black text

Description automatically generatedA white sheet with black text

Description automatically generatedA screenshot of a computer

Description automatically generatedA white sheet with black text

Description automatically generated**

**Figure 16, 17, 18, 19:** Sign in Scenarios

**c) Test Sign out:**

**A white and blue box with black text

Description automatically generated**

**Figure 20:** Sign out Scenario

**c) Library Interaction:**

A white grid with black text

Description automatically generated

**Figure 21:** Library interaction Scenarios

**2) Design Test Scripts using Selenium IDE and Selenium WebDriver**

The Test Cases structure in Selenium IDE

A screenshot of a computer

Description automatically generated

**Figure 22:** Selenium IDE with 3 Test Suites

File ‘daiichi\_test.side’ is included in the following GitLab repository: [instech / Intern-RnD / automation-test · GitLab (dai-ichi-life.com.vn)](https://git.dai-ichi-life.com.vn/instech/intern-rnd/automation-test) (branch autotest)

To run the python project, follow the instructions written in file README.md

**IV. Add logging to track test execution, generate test reports**

**a) Generate our own logging:**

File logger.py contains the logging tool to track execution of the test. It has two handlers: one handler logs to stdout and one logs to file.

Each test module (define in testCases folder) contains a logger to notify us that which method is running (print to console), and if an exception is caught, log will be sent to file

**b) Generate Test report**

Module pytest-html supports us in generating html report after testing

In main.py, --html=’./report/report.html’ is an argument used to generate html report to the report folder. User can change the directory of the html file

If unittest is used instead of pytest, user can use third – party libraries like HTMLTestRunner or using noses (pytest is also acceptable since pytest is compatible with unittest)

**V. Push Notifications to channel in Microsoft Teams**

- Module requirements:

+ pymsteams (if not installed successfully, implement using urllib3 and json instead, see the file utils/msteams\_notification.py)

+ adaptivecardbuilder: to build an Adaptive Card (can use dictionary but the code will be hard to maintain)

+ html2image: to render HTML file into PNG image

**1) msteams\_notification.py**

In this file, two classes called TeamsWebhookException and ConnectorCard will be created. TeamsWebhookException inherited from Exception, while ConnectorCard is used to connect to Webhook URL and send our Adaptive Card using POST method.

**2) main.py**

**a) generateImgFromHTML**

A method used to convert a HTML and CSS file into PNG image. This method will return a string representing the result image after encoding to base64 format

**b) main**

An entry point of the entire project. The main function has an argv parameter contains all parameters of the pytest command (except --html=./report/report.html argument)

Our Adaptive Card will be created and sent to Teams channel in this method

If argv is an empty list, all the Test Cases will be run (144 Test Cases), so it is better to specify the test module directory, or only some methods and classes using nodeid (-v argument is required if the test module is not at the same directory as main.py)

Some examples of valid arguments:

- -v ./testCases/test\_sign\_in.py::TestSignIn::test\_sign\_in\_success (run test\_sign\_in\_success method)

- ./testCases/test\_library.py (run all the test methods in test\_library.py, -v is optional)

A screenshot of a computer

Description automatically generated

**Figure 23:** A test report was pushed to Microsoft Teams channel