ISSN 2085-4552

Comparing Karate Framework with Others for Automated Regression Testing: A Case Study of PT Fliptech Lentera Inspirasi Pertiwi

Afina Putri Dayanti[1], Tony Tony[2]
[1, 2] Department of Information Systems, Faculty of Information Technology, Tarumanagara University
Jakarta, Indonesia
[1]afina.825200049@stu.untar.ac.id, [2]tony@fti.untar.ac.id

Accepted 10 November 2023
Approved 19 April 2024

***Abstract*—In the rapidly evolving digital era, applications, and software systems increasingly rely on Application Programming Interfaces (APIs) to enable interaction, integration, and functionality extension. However, manual testing of APIs is often inefficient and challenging to reuse when changes occur. To address this, automation testing has become a more effective choice, where test scripts can verify and execute tests repeatedly, easily adapting to API changes. Essentially, automation testing plays a vital role in software maintenance, particularly in regression testing, which tests modified or upgraded software versions to ensure that their core functions remain unchanged and unaffected. One approach to automation testing is employing the Software Testing Life Cycle (STLC), which follows a systematic series of stages conducted by the testing team to ensure software product quality. This paper utilizes PT Fliptech Lentera Inspirasi Pertiwi's public API to conduct testing on 25 scenarios from two modules. The objective is to utilize the Karate Framework to conduct these automated regression tests, resulting in an impressively short testing duration, averaging only 42.645 seconds, or approximately 1.706 seconds per scenario. A comparison with the Behave framework, using the same scenarios but with differences in steps, reveals that Behave achieves a duration of 18.762 seconds, or 0.750 seconds per scenario, making it 127.295% faster than Karate. However, in terms of the number of steps, Behave covers only 188, while Karate includes 543. This means that Behave requires 0.100 seconds per step, while Karate necessitates 0.079 seconds per occurrence. Karate provides more detailed results by 188.830% per step or 26.582% in terms of step duration. The primary goal is to enhance testing efficiency, expedite issue identification and resolution, provide a clearer testing process, and potentially improve overall software quality.**

***Index Terms*—API; automation testing; Karate framework; regression testing; STLC.**

I. INTRODUCTION

In the era of advancing information technology, software continually undergoes development and refinement to align with ever-changing needs and the rapid progression of technology [1]. Each change applied to the software, whether it involves improvements, feature additions, or other modifications, has the potential to influence the overall performance and stability of the entire system. Consequently, the significance of regression testing has increasingly become an essential foundation. This type of testing ensures that any alterations do not disrupt the functionality that was previously operating smoothly. The primary objective of regression testing is to verify whether these changes have caused disruptions in pre-existing functions, with the purpose of mitigating the risk of potential system failures resulting from these modifications [2].

PT Fliptech Lentera Inspirasi Pertiwi, aka Flip, is an Indonesian fintech company established in 2015 [3]. As a prominent player in the software development industry, the company encounters challenges similar to those faced by its industry peers while managing its multifaceted operations. The developer team engaged in various projects and features is tasked with ensuring the stability of the system. However, manual regression testing consumes significant time and resources. In this context, the implementation of an automation testing tool through an Application Programming Interface (API) emerges as a practical solution. APIs find widespread application in the creation of distributed software systems featuring interconnected components. Despite their absence of a visible interface [4], APIs play a pivotal role in enabling machine-to-machine communication and serving as a means to foster interaction, integration, expansion, and data exchange among distinct software functions or entities [5].

This paper contains the following contributions. Firstly, it provides insights into the challenges faced by Flip and the importance of system stability in software development. Secondly, it sheds light on the resource-intensive nature of manual regression testing and advocates for the use of automation testing, which involves executing tests through specialized tools or software [6] via APIs, as a solution to these challenges.

Thirdly, it promotes the use of the Karate Framework, an automation testing tool that utilizes Gherkin syntax from Cucumber BDD (behavior-driven development). While Karate is based on Java, it does not always require advanced programming skills for basic software testing. Instead, it encourages a deeper understanding of Cucumber and specific framework development [7]. Further, this paper conducts a case study using Flip's public API, the Flip for Business Platform, to create a specialized tool for regression testing. Fourthly, it anticipates reduced time and resource requirements for issue detection following system changes. Although automation can accelerate the testing process, it is essential to ensure that the benefits outweigh the initial setup and maintenance expenses. Lastly, it emphasizes the potential enhancement in testing efficiency, accuracy, and coverage through an approach of automation testing for regression.

The rest of this paper is organized as follows. Section II discusses related works to the research while Section III presents the research's results and discussion. The details of our solution and its performance are described in Section IV. Finally, Section V concludes the paper and provides future research directions.

II. RELATED WORKS

To the best of our knowledge, there has been no prior work addressing a problem similar to ours, except for the study conducted by Gidvarowart et al. [8]. In their work, the authors explored automated API testing using the Karate Framework and presented a case study of an online assessment web application demonstrating reduced testing time per iteration in contrast to manual testing and comparisons with other frameworks. Our approach extends to a comparison with the Behave framework, closely associated with the Python programming language and commonly identified using the search terms "bdd" and "behave" [9]. We opted for this framework because Behave incorporates a concept similar to Karate, namely BDD, and our aim is to compare them to identify more efficient regression testing automation.

When juxtaposing our work with other related studies, several distinct patterns emerge. Putri [10] applied regression automation testing using Katalon Studio for the "Teman Diabetes" mobile app, citing the perceived limitations of manual regression testing. In contrast, this work centers on regression testing for Flip for Business's API, presenting a different context. Directed attention, Yutia [11] highlighted automated functional API testing using the Robot framework for KALcare.com, emphasizing the efficiency gains brought about by automation over manual methods. In this proposed approach, the STLC methodology is applied to API regression testing, with the Karate framework harnessed for test execution. Additionally, automated load and performance testing for DiTenun's API were extensively explored by Barus et al. [12], while this work specifically highlights regression automation testing executed after system changes. Lastly, automation testing challenges in the context of Hospital Management Systems were discussed by Saputra and Stefanie [13], with this work predominantly focusing on API testing, where the Karate framework serves as the primary automation testing tool, contributing to the growing body of research on automation testing within various contexts.

III. METHODOLOGY

Testing is a process with its own stages, even though it's an integral part of the Software Development Life Cycle (SDLC) [14], see Fig. 1. Software Testing Life Cycle (STLC) refers to a systematic series of stages run by the testing team to test the software product. In essence, STLC constitutes the testing phase embedded within the SDLC, running in parallel but with its distinct cycle [15].

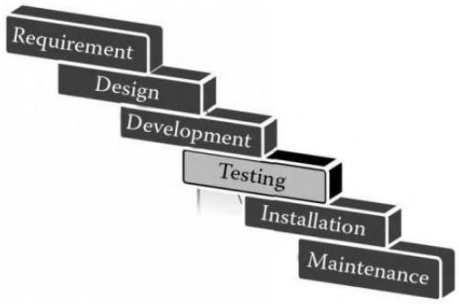


Fig 1. Testing Phase in Software Development Life Cycle [16]

In the design process, STLC approach serves as the primary methodology. Although the implementation of testing may vary depending on each SDLC's specific approach, the key steps in each software STLC remain consistent. This STLC approach provides a structured framework for governing all software testing stages, much like SDLC, and consists of stages (see Fig. 2) [16]. Each phase is designed to enhance the quality of the product [17] and is characterized by well-defined entry and exit criteria, activities, and associated deliverables [18].

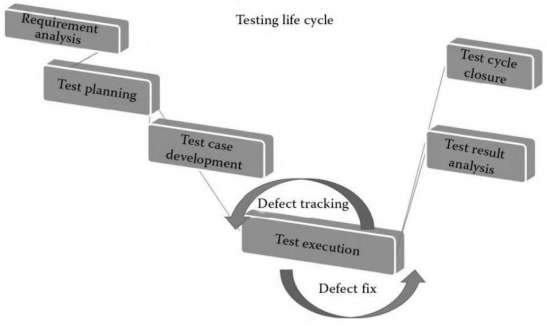


Fig 2. Stages of the Software Testing Life Cycle [16]

1) Requirements Analysis: This phase involves identifying the target, goals, scope, and testing approach to be taken. The plan will provide detailed explanations of how regression automation testing will be conducted, including resource allocation and testing schedules.
2) Test Planning: Analyzing testing requirements in detail. This includes an analysis of functional, non-functional features, and relevant testing scenarios for selected API features.
3) Test Case Development: Designing testing scenarios and automation testing scripts based on the requirements analysis results. The testing plan includes test steps, test data, and the test environment.
4) Test Execution (defect tracking and fixing): Involves creating and executing automation testing scripts according to the plan. Automation testing tools are developed using API concepts to test selected features. After obtaining test results, if defects are found, the next step is to return them to this phase for further analysis. Every bug and error in the API will be thoroughly analyzed. Afterward, corrective actions and updates will be implemented to address these defects.
5) Test Result Analysis: A post-conditional process involving data collection from end-users. After testing execution, the team evaluates the results of regression testing.
6) Test Cycle Closure: Discussion and evaluation of testing artifacts to identify strategies to be applied in the future, using the experience gained from the completed regression testing cycle. The goal is to reduce process constraints in subsequent testing cycles and share best practices for similar projects in the future.

IV. Result and Discussion

The Institute of Electrical and Electronics Engineers (IEEE) defines a process as the actions required to carry out a task or as a written description of those actions, as in documented testing procedures. From this perspective, it can be concluded that "the testing process" is the "actions necessary to carry out testing" or the "approach used in conducting testing" [19]. In the design of this process, we refer to the STLC methodology to create a structured overview of the API automation process, as shown in Fig. 3, where several key stages are detailed in the form of a flowchart. Commencing with "Start", the process navigates through stages such as analyzing requirements, selecting APIs, and conducting manual API inspections. Subsequently, the automation structure is aligned with the folder hierarchy, and testing scripts are created or adjusted. The results of automation testing are analyzed; if the outcome is "no", manual testing is revisited, while "yes" results lead to presenting the outcomes. System artifact evaluation ensures a thorough review, and the process loops back for continuous testing. The cycle concludes with the "End" phase, marking the conclusion of the API automation testing workflow.

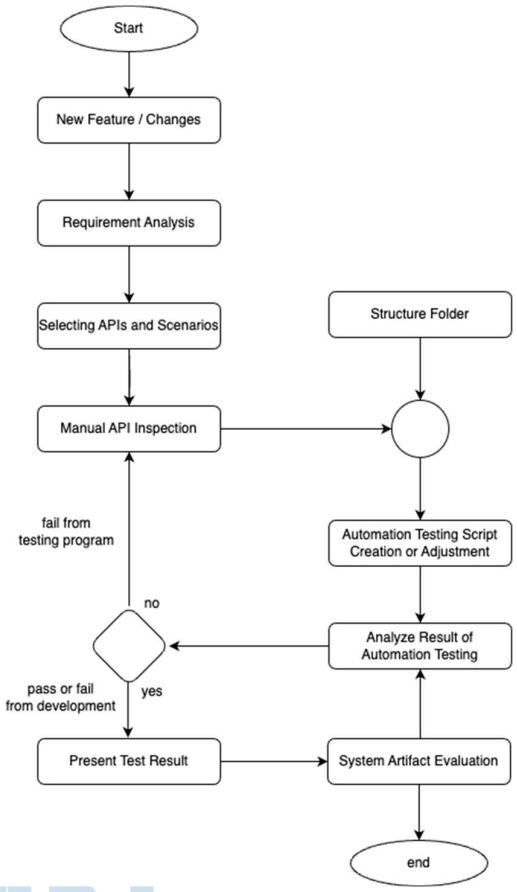


Fig 3. Flowchart of Automation Testing Process

A. Folder Structure

In the context of designing tests using Karate, it is different from Java code development that follows conventions like com.mycompany.foo.bar and results in nested sub-folders. The Karate documentation actually suggests having a folder structure with only one or two levels, where the folder names clearly identify the resource, entity, or API under test. However, in this design, we choose to adopt an automation structure customized to the company's needs (see Fig. 4).

Initially, all files are placed in the **src-test** folder. The **src-test-java** folder is used to store all automation files, including the Karate runner for execution and HTML report generation. The service folder is assigned for storing scenarios (.feature), the config folder for global configuration, the spec folder for payload requests, and the utils folder for utility data. Meanwhile, the **src-test-resource** folder is designated for application configuration files.

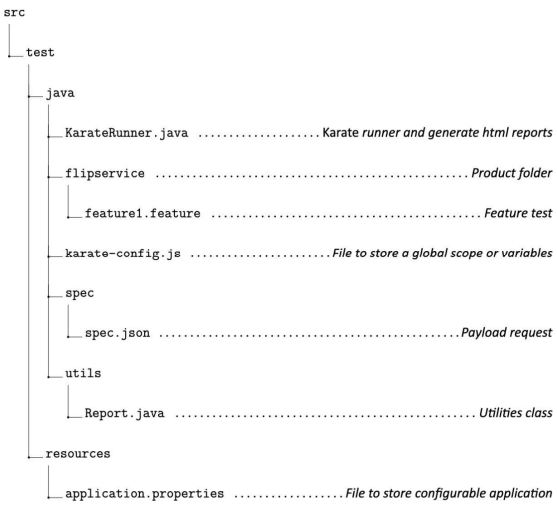


Fig 4. Folder Structure

B. Requirements Analysis

The process commences with the identification and analysis of the requirements and specifications of Flip for Business APIs. The required data is acquired through an interview with one of the Test Engineers from the Business and Solution Team responsible for Flip for Business. This data-gathering process involves collecting information about the relevant API usage, analyzing the API's flow, and identifying its integration with the database.

C. Selection APIs and scenario

The selection of APIs and scenarios involves a process that encompasses defining use cases, endpoints, actions, and test scenarios. Within Flip for Business, there are numerous use cases to choose from. However, in this study, only two modules will be selected, i.e., Money Transfer and Special Money Transfer [20] because these modules are among the most frequently used and represent the core of Flip's business operations, thus significantly contributing to Flip for Business' revenue. In each of the selected modules, there are four endpoints. The Money Transfer module has a total of 15 scenarios, while the Special Money Transfer module has a total of 10 scenarios. Therefore, for this research, the combination of these two modules results in a total of 25 scenarios. For more detailed information, the definition of the scope is based on the chosen use cases, as exemplified in Table I and Table II.

TABLE I. SCENARIO OF MONEY TRANSFER

No	Action	Endpoint	Scenario
1	POST	https://bigflip.id/api/v3/disbursement	Should success create disbursement with one beneficiary email
			Should return error create disbursement params required
			Should return error create disbursement only number
			Should return error create disbursement amount minimum
			Should return error create disbursement amount maximum
			Should return error create disbursement invalid bank code
			Should return error create disbursement max email
			Should return error create disbursement invalid email
2	GET	https://bigflip.id/api/v3/disbursement?pagination=pagination&page=page&sort=sort&attribute=value	Should success get all disbursement
			Should success get all disbursement with pagination
			Should success get all disbursement filter by status
			Should success get all disbursement filter by bank
			Should success get all disbursement filter by created from
3	GET	https://bigflip.id/api/v3/get-disbursement?idempotency-key=idempotencykey	Should success get disbursement by idempotency key
4	GET	https://bigflip.id/api/v3/get-disbursement?id=id	Should success get disbursement by id

TABLE II. SCENARIO OF SPECIAL MONEY TRANSFER

No	Action	Endpoint	Scenario
1	POST	https://bigflip.id/api/v3/special-disbursement	Should success create special disbursement for do mestic transfer
			Should success create special disbursement for foreign inbound transfer
			Should return error create special disbursement params required
			Should return error create special disbursement only number
			Should return error create special disbursement amount minimum
			Should return error create special disbursement amount maximum
			Should return error create disbursement invalid bank code
2	GET	https://bigflip.id/api/v2/disbursement/city-list	Should success get city list
3	GET	https://bigflip.id/api/v2/disbursement/country-list	Should success get country list
4	GET	https://bigflip.id/api/v2/disbursement/city-country-list	Should success get city and country list

D. Manual API Inspection

Before delving into the automation testing phase, it is highly advantageous to initiate the process with a manual API inspection. This initial step entails using software tools like Postman, which enable testers to manually interact with the API. By doing so, testers gain valuable insights into how the API functions and communicates. This manual exploration serves as a foundation for the subsequent phases and ensures a comprehensive understanding of the API's behavior. This phase is illustrated in Fig. 5, showcasing how manual testing aids in comprehending the intricacies of API interactions.

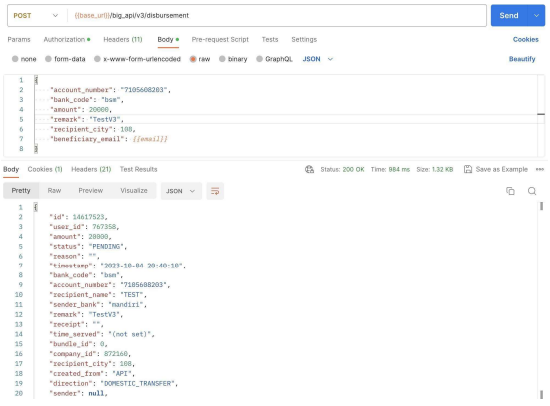


Fig 5. Testing Manually with Postman

E. Automation Script Creation or Adjustment

Before proceeding with the creation or modification of automation scripts, it's crucial to fulfill the prerequisites for system implementation, including software, hardware, personnel, installation procedures, and user guides. This phase is a pivotal point in the system's life cycle as it initiates the solution's deployment in the production environment. Proper software installation on prepared hardware, especially with the presence of personnel like test engineers, is vital. The installation process should ensure optimal system component functionality, meet performance standards, and provide user manuals for accurate system utilization.

Once the prerequisites for system implementation have been satisfied and gaining a comprehensive understanding of the API's requests and responses through manual methods, the next phase involves the transformation of this knowledge into automated tests. This step is essential for achieving efficiency and repeatability in the testing process. Automated tests are designed to mimic the interactions that were previously tested manually. The creation or adjustment of automation scripts allows for the seamless execution of these tests, as demonstrated in Fig. 6. Essentially, these scripts function as a comprehensive set of directives meticulously guiding the testing framework, effectively streamlining and optimizing the entire testing process for enhanced accuracy and efficiency.

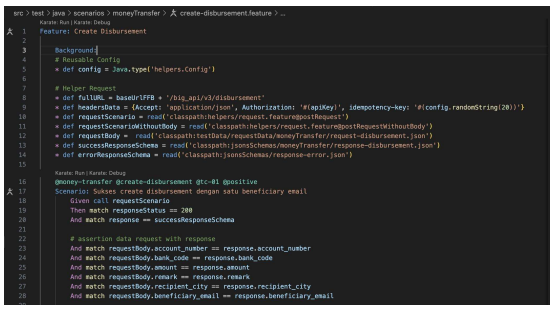


Fig 6. Transitioning from Manual Testing to Automation Testing

F. *Analyze result of automation testing*

In this phase, the focus shifts towards analyzing the outcomes of the automation testing process. After the automated tests have been executed as illustrated in Fig. 7, the results obtained need to be comprehensively examined and assessed. This entails scrutinizing the data, logs, and metrics generated during the testing process. One of the primary goals is to identify any anomalies, errors, or issues that might have surfaced during the automation testing. It's crucial to thoroughly evaluate the collected data to gain insights into the performance and behavior of the tested API. This phase serves as a critical checkpoint for quality assurance and ensures that the automated tests have been carried out effectively.

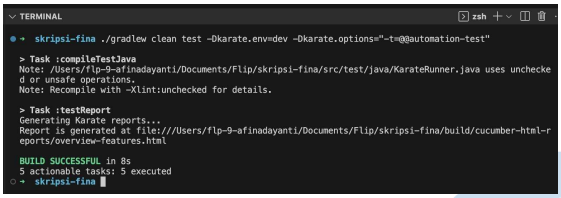
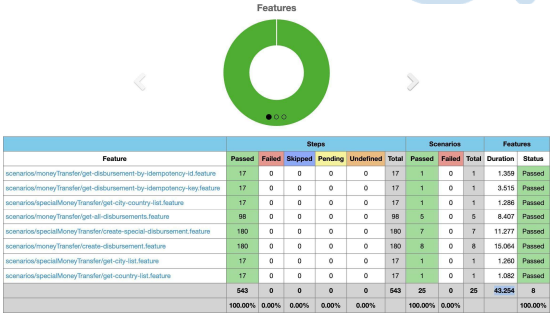


Fig 7. Automation Testing Execution

G. *Present test result*

The subsequent phase involves generating an HTML-formatted report to present the test results in a structured and informative manner. This comprehensive report accounts for various aspects of the testing process, including executed test cases, their outcomes, identified issues or defects, and performance metrics. This structured report plays a crucial role in facilitating in-depth test analysis, offering stakeholders a clear overview of the API's behavior and areas that need attention. Referenced as Fig. 8, it aids in problem identification and necessary improvements.



	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
scenarios/moneyTransfer/get-disbursement-by-idempotency-id.feature	17	0	0	0	0	17	1	0	1	1.359	Passed
scenarios/moneyTransfer/get-disbursement-by-idempotency-key.feature	17	0	0	0	0	17	1	0	1	3.515	Passed
scenarios/specialMoneyTransfer/get-city-country-list.feature	17	0	0	0	0	17	1	0	1	1.286	Passed
scenarios/moneyTransfer/get-all-disbursements.feature	98	0	0	0	0	98	5	0	5	8.407	Passed
scenarios/specialMoneyTransfer/create-special-disbursement.feature	180	0	0	0	0	180	7	0	7	11.277	Passed
scenarios/moneyTransfer/create-disbursement.feature	180	0	0	0	0	180	8	0	8	15.064	Passed
scenarios/specialMoneyTransfer/get-city-list.feature	17	0	0	0	0	17	1	0	1	1.260	Passed
scenarios/specialMoneyTransfer/get-country-list.feature	17	0	0	0	0	17	1	0	1	1.082	Passed
	543	0	0	0	0	543	25	0	25	43.254	8
	100.00%	0.00%	0.00%	0.00%	0.00%		100.00%	0.00%			100.00%

Fig 8. Automation Testing Execution

In the context of the initial selection of 25 scenarios, covering 8 endpoints across 2 previously chosen modules, the process of automation script creation and refinement resulted in a total of 543 occurrences. These scenarios underwent 5 consecutive trial runs, as seen in the variability of results in Table III, which are influenced by CPU and memory resources. The calculated average duration of 42.645 seconds implies that each scenario takes approximately 1.706 seconds to execute, equivalent to 0.079 seconds per occurrence.

TABLE IV. AUTOMATION TESTING ACROSS 10 TEST RUNS

Execute	Duration Result
1	43.580 s
2	42.693 s
3	45.216 s
4	40.311 s
5	43.254 s
6	42.111 s
7	42.918 s
8	41.289 s
9	43.246 s
10	41.832 s
Average	**42.645 s**

$$\text{Average Duration} = \frac{\text{Total Duration}}{\text{Trial Runs}} = \frac{426.45}{10} \approx 42.645\text{s}$$

$$\text{Average Scenario} = \frac{\text{Average Duration}}{\text{Total Scenario}} = \frac{42.645}{25} \approx 1.706\text{s}$$

$$\text{Average Occ.} = \frac{\text{Average Duration}}{\text{Total Occurrences}} = \frac{42.645}{543} \approx 0.079\text{s}$$

In addition to reporting test results in the .html format, there is also the possibility of integration, such as using the Slack platform. Through this integration, it becomes feasible to automatically send notifications when testing failures occur. For example, the integrated output within Slack, as depicted in Fig. 9, will provide immediate notifications to the development team when there are issues that require immediate attention during testing. Integrations like this ensure that information regarding testing problems

can be rapidly received by developers, allowing for more effective and timely responses to potential issues that may arise during the testing process.

Following the presentation of test results, a comprehensive evaluation of the results or report will be conducted. If any defects or issues are detected during this assessment, the testing process might regress to the manual testing phase to further investigate and resolve these problems. However, if no defects are found, the implementation will progress to the evaluation of system artifacts, ensuring the overall robustness and quality of the system.

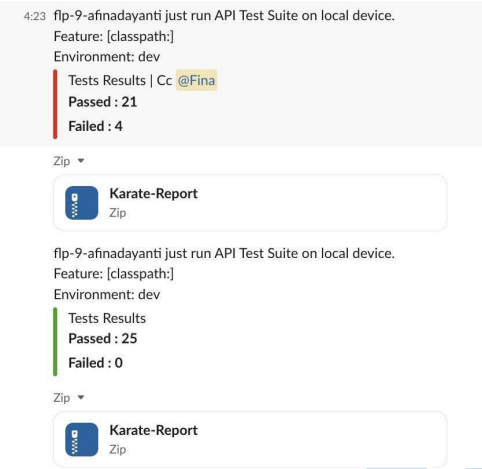


Fig 9. Automation Testing Report Integration with Slack

H. *System artifact evaluation*

The discussion and evaluation of testing artifacts aim to identify strategies that will be applied in the future, leveraging the experience gained from the on-going regression testing cycle. The goal is to minimize process constraints in the next testing cycle and share best practices for similar projects in the future.

I. *Comparison with Other Framework*

For the purpose of comparing the duration of test results, the author employed the behave framework. As detailed in Table IV, it became apparent that after conducting 10 test runs using the same 25 scenarios, encompassing 8 endpoints across 2 previously selected modules, with a slight difference in the number of steps (precisely 188 steps, as shown in Fig. 10), Behave achieved an average duration of 18.762 seconds. This suggests that each scenario takes roughly 0.750 seconds to execute, which translates to approximately 0.100 seconds per step.

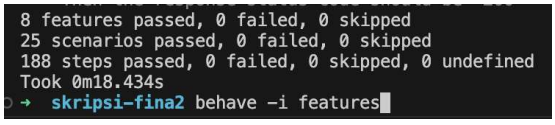


Fig 10. Execution of Automation Testing Using Behave Framework

TABLE IV. COMPARING THE DURATION RESULTS OF AUTOMATION TESTING FRAMEWORKS

Execute	Duration Result Karate	Duration Result Behave
1	43.580 s	19.215 s
2	42.693 s	18.551 s
3	45.216 s	18.293 s
4	40.311 s	18.176 s
5	43.254 s	19.385 s
6	42.111 s	18.898 s
7	42.918 s	18.972 s
8	41.289 s	17.978 s
9	43.246 s	20.180 s
10	41.832 s	17.969 s
Average	**42.645 s**	**18.762 s**

$$\text{Average Duration} = \frac{\text{Total Duration}}{\text{Trial Runs}} = \frac{187.617}{10} \approx 18.762\text{s}$$

$$\text{Average Scenario} = \frac{\text{Average Duration}}{\text{Total Scenario}} = \frac{18.762}{25} \approx 0.750\text{s}$$

$$\text{Average Step} = \frac{\text{Average Duration}}{\text{Total Occurrences}} = \frac{18.762}{188} \approx 0.100\text{s}$$

In evaluating the merits of these two options, we must consider testing objectives and priorities. Utilizing formulas to calculate the percentage increase or decrease can measure the extent of changes in a value [21]. In this context, it is necessary to identify the differences between the values of Karate and Behave first. The resulting difference is then divided by the value of Karate or Behave, depending on whether we are looking for a percentage increase or decrease. The result is then multiplied by 100 to convert it into a percentage.

$$\text{Percent decrease (\%)} = \frac{\text{Original Value} - \text{New Value}}{\text{Original Value}} \times 100\%$$

$$\text{Percent increase (\%)} = \frac{\text{New Value - Original Value}}{\text{Original Value}} \times 100\%$$

Behave boasts a faster execution time, completing tests in 18.762 seconds, compared to Karate's 42.645 seconds. This results in Behave reducing execution time by 127.295%, making it more efficient in terms of time and resource utilization compared to Karate. It's important to emphasize that this difference in duration cannot be attributed to a single factor. Instead, it's influenced by various variables, including test complexity, the testing environment, parallel execution, hardware and resource disparities, optimization, tool-specific factors, and more, all of which collectively contribute to these variations.

Percentage difference between the average duration of Karate and Behave:

$$\text{Percent decrease (\%)} = \frac{\text{Behave - Karate}}{} \times 100\% = \frac{18.762 - 42.645}{18.762} \times 100\% \approx -127{,}294\%$$

Moreover, shifting the focus to the number of steps, Behave employs 188 steps, whereas Karate uses 543 steps. This implies that utilizes's per-step duration is 0.100 seconds, while Karate's per-occurrence duration is 0.079 seconds. Consequently, Karate holds a 188.830% advantage in providing more detailed and comprehensive insights into the behavior of the tested software, particularly when considering step duration, where Karate outperforms Behave by 26.582%. Therefore, when deciding between these options, it's essential to consider the trade-off between execution speed and the depth of analysis while considering specific testing requirements and objectives.

Percentage difference between Karate and Behave steps:

$$\text{Percent decrease (\%)} = \frac{\text{Behave} - \text{Karate}}{\text{Behave}} \times 100\% = \frac{188 - 543}{188} \times 100\% \approx -188.830\%$$

Percentage difference between the duration of Karate and Behave steps:

$$\text{Percent increase (\%)} = \frac{\text{Behave} - \text{Karate}}{\text{Karate}} \times 100\% = \frac{0.100 - 0.079}{0.079} \times 100\% \approx 26.582\%$$

CONCLUSION

This paper utilizes the Karate Framework as an automation tool for testing to investigate the use of the public Flip for Business API during the development process. The execution of 25 scenarios selected from two modules resulted in an impressively short testing duration of only 42.645 seconds, which translates to approximately 1.706 seconds per scenario. This reduction in testing time is a significant improvement over manual methods, leading to substantial time savings of several minutes per test. Additionally, when compared with the Behave framework using the same scenarios but with differences in steps, Behave achieved 18.762 seconds, or 0.750 seconds per scenario, making it 127.295% faster than Karate. However, when considering the number of steps, Behave only covers 188 steps, while Karate includes 543 steps. This means that Behave requires 0.100 seconds per step, while Karate requires 0.079 seconds per occurrence. Karate provides more detailed results by 188.830% per step or 26.582% in terms of step duration. Therefore, the choice between Behave and Karate depends on your primary testing objectives. Since the main goal of this paper is efficiency to obtain results as quickly as possible and in-depth analysis, Karate will be the preferable choice. This acceleration in the testing process contributes to faster development cycles, ensuring consistent API quality with each modification. Additionally, the quality assurance report verifies the online assessment system's quality and deployment readiness. Thorough preparation is essential for seamless parallel testing, avoiding conflicts and overlaps in test cases. An immediate area of future work involves integrating this procedure into CI/CD (Continuous Deployment or Continuous Delivery) solutions, to speed up release cycles and address potential issues during code integration.

ACKNOWLEDGEMENT

We acknowledge the support from Muhamad Rizal Indrabayu as the Engineering Manager, Henry Suryawirawan as the Vice President of Engineering, and Dwina Apriliasari as Corporate Communications at PT Fliptech Lentera Inspirasi Pertiwi.

REFERENCES

[1] J. A. Gerding, B. W. Brooks, E. Landeen, and more, "Identifying needs for advancing the profession and workforce in environmental health," American Journal of Public Health. 2020 Mar, 110(3):288-94.

[2] G. Blokdyk, Regression Testing: A Complete Guide - 2019 Edition. Emereo Pty Limited, 2019.

[3] Flip. (2023) Tentang flip. Accessed on August 10, 2023. [Online]. Available: https://flip.id/tentang-flip

[4] M. Biehl, API Architecture. CreateSpace Independent Publishing Platform, May 2015.

[5] AWS. (2023) What is an api? Accessed on 7 August 2023. [Online]. Available: https://aws.amazon.com/id/what-is/api/

[6] M. Baumgartner, T. Steirer, M.-F. Wendland, S. Gwihs, R. Seidl, and more, Test Automation Fundamentals: A Study Guide for the Certified Test Automation Engineer Exam – Advanced Level Specialist – ISTQB® Compliant. dpunkt.verlag, August 30 2022.

[7] P. A. Chaubal, Mastering Behavior-Driven Development Using Cucumber. BPB Publications, August 2021.

[8] S. Gidvarowart, A. Suchato, D. Wanvarie, N. Pratanwanich, and N. Tuaycharoen, "Automated api testing with karate framework: A case study of an online assessment

web application," in 2023 20th International Joint Conference on Computer Science and Software Engineering (JC- SSE). Phitsanulok, Thailand: IEEE, June 28 - July 01 2023.

[9] T. Storer and R. Bob, "Behave nicely! automatic generation of code for behaviour driven development test suites," in 2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM), 2019, pp. 228–237.

[10] Y. F. Putri, "Automation regression testing pada aplikasi teman diabetes dengan menggunakan metode black box testing," Ph.D. dissertation, Universitas Atma Jaya Yogyakarta, 2020.

[11] S. N. Yutia, "Automated functional testing pada api menggunakan keyword driven framework," Journal of Informatics and Communication Technology (JICT), vol. 3, no. 1, pp. 65–78, 2021.

[12] A. C. Barus, J. Harungguan, and E. Manulu, "Pengujian api website untuk perbaikan performansi aplikasi ditenun," Journal of Applied Technology and Informatics Indonesia, vol. 1, no. 2, pp. 14–21, 2021.

[13] B. D. Saputra and A. Stefanie, "Automation testing api, android, dan website menggunakan serenity bdd pada software sistem manajemen rumah sakit," Jurnal Ilmiah Wahana Pendidikan, vol. 9, no. 10, pp. 114–126, 2023.

[14] S. Desai and A. Srivastava, Software Testing. Phi Learning, January 30 2016.

[15] G. Singh, "A study on software testing life cycle in software engineering," Int. J. Manag. IT, vol, vol. 9.

[16] A. S. Mahfuz, Software Quality Assurance: Integrating Testing, Security, and Audit. CRC Press, April 27 2016, ebook.

[17] A. Anand and A. Uddin, "Importance of software testing in the process of software development," International Journal for Scientific Research and Development, vol. 12, no. 6, 2019.

[18] A. Nordeen, Learn Software Testing in 24 Hours: Definitive Guide to Learn Software Testing for Beginners. Guru99, October 31 2020.

[19] R. Drabick, Best Practices for the Formal Software Testing Process: A Menu of Testing Tasks. Pearson Education, July 15 2013.

[20] Flip. (2023) Flip for business. Accessed on August 10, 2023. [Online]. Available: https://flip.id/business

[21] D. Sharma. (2023) How to calculate percentage increase: Formula & examples. Updated on August 1, 2023. Accessed on November 20, 2023. [Online]. Available: https://www.indeed.com/career-advice/career-development/percent-increase-formula