SELENIUM TOOL: AN EFFECTIVE AUTOMATION FRAMEWORK FOR WEB TESTING USING JAVA

Mr. Ankit Gupta
i-nurture education solutions pvt.ltd. (Asst. Prof. IT vertical),
ankitgupta@inurture.co.in

Abstract
Automation testing plays a vital role in modern software development by enhancing efficiency, reliability, and speed of the testing process. This research paper explores the field of automation testing, with a particular focus on utilizing the Java programming language for test automation. We discuss the benefits and challenges associated with automation testing, the key principles and best practices for implementing automation testing using Java, and various tools and frameworks available in the Java ecosystem. Additionally, this paper provides real-world examples and case studies to demonstrate the effectiveness of automation testing with Java in different scenarios. By the end of this research paper, readers will gain a deep understanding of automation testing using Java and its potential to streamline the software development lifecycle.
.

Keywords- automate, Restful, TestNg, Junit & CI/CD

Introduction

Automation testing has emerged as a crucial aspect of software development, enabling organizations to achieve higher efficiency, improved quality, and faster time-to-market for their applications. The use of automation testing frameworks and tools has significantly revolutionized the way software testing is conducted. One such powerful tool is Java, a versatile programming language that offers a rich set of libraries and frameworks for automation testing.

1.1 Background

Traditionally, manual testing has been the primary method for verifying software functionality. However, manual testing is time-consuming, error-prone, and lacks scalability, especially when dealing with complex and rapidly evolving software systems. Automation testing addresses these challenges by leveraging tools and frameworks to automate repetitive and labor-intensive testing tasks.

Java, being a widely adopted and robust programming language, provides an ideal platform for implementing automation testing. It offers a vast ecosystem of libraries and frameworks specifically designed for testing purposes, making it a popular choice among software testers and developers.

1.2 Objectives

The main objective of this research paper is to explore the use of Java in automation testing and provide a comprehensive understanding of its benefits, best practices, and tools. The specific objectives are as follows:

To highlight the importance and benefits of automation testing in the software development lifecycle.
To examine the reasons for choosing Java as the programming language for automation testing.
To identify the key principles and best practices for implementing automation testing using Java.
To explore the various automation testing frameworks available in the Java ecosystem.
To demonstrate how Java can be used for web, API, and mobile automation testing.
To provide practical examples and case studies showcasing the effectiveness of Java in automation testing.
To discuss the scope and limitations of automation testing using Java.

1.3 Scope

This research paper focuses on automation testing using the Java programming language. It covers a broad range of topics related to automation testing, including the benefits and challenges of automation testing, introduction to Java for automation testing, test automation frameworks in Java, web automation testing with Java using Selenium Web Driver, API automation testing with Java using REST-assured, mobile automation testing with Java using Appium, best practices for automation testing with Java, and relevant case studies and examples.

The scope of this paper encompasses the conceptual understanding of automation testing with Java, practical implementation techniques, and real-world scenarios. However, it does not cover advanced or specialized topics related to specific domains or advanced testing methodologies.
.

2. Literature Survey

2.1 Automation Testing Overview

Automation testing involves the use of software tools and frameworks to automate the execution of test cases and verify the functionality, performance, and reliability of software applications. It offers numerous advantages over manual testing, including increased test coverage, faster test execution, reduced human error, and improved test accuracy. This section provides an overview of the importance and benefits of automation testing, the challenges associated with it, and the strategies for implementing effective automation testing.

2.2 Importance and Benefits of Automation Testing

Automation testing plays a critical role in the software development lifecycle, offering several significant benefits:
a. Enhanced Efficiency: Automation testing allows for the execution of a large number of test cases in a shorter time span, significantly reducing the testing cycle duration. It helps identify defects and issues early in the development process, enabling faster bug fixes and enhancing overall development efficiency.

b. Improved Test Coverage: Automation testing enables comprehensive test coverage by executing a vast number of test cases, including repetitive and complex scenarios, which may not be feasible with manual testing alone. This leads to a higher level of software quality and ensures that critical functionalities are thoroughly tested.

c. Increased Reliability: Automated tests are consistent and reliable, eliminating the risk of human error that is inherent in manual testing. It ensures that test cases are executed precisely, improving the accuracy and reliability of test results.

d. Regression Testing: Automation testing is particularly valuable for regression testing, where previously tested functionalities are retested to ensure that new changes or fixes do not introduce new defects. Automation allows for the quick and efficient execution of regression tests, saving time and effort.

e. Cost Savings: While the initial investment in automation testing may be higher, the long-term benefits include significant cost savings. Automation reduces the need for manual testers, minimizes repetitive tasks, and decreases the time required for testing, resulting in overall cost reduction.

2.3 Challenges in Automation Testing

Despite the numerous benefits, automation testing also presents several challenges that need to be addressed.
a. Test Maintenance: As applications evolve and change, automated tests require regular maintenance to keep them up to date. Any modifications to the application may impact the existing automated tests, requiring updates or modifications to maintain their effectiveness.

b. Test Data Management: Managing test data is crucial for automation testing. Test data should be representative of real-world scenarios and should be maintained efficiently to avoid data-related issues during test execution.

c. Test Environment Configuration: Setting up and configuring the test environment, including software, hardware, and network configurations, can be complex. Ensuring consistent and reliable test environments across different stages of testing can be challenging.

d. Tool Selection: Choosing the right automation testing tool that aligns with the project requirements and supports the desired functionalities is critical. The selection process involves evaluating factors such as ease of use, compatibility, scalability, and available support and resources.

2.4 Automation Testing Strategies.

To implement effective automation testing, organizations should consider the following strategies.

a. Test Case Selection: Not all test cases are suitable for automation. Organizations should prioritize the selection of test cases that are repetitive, time-consuming, and critical to the application's functionality. This ensures optimal utilization of automation resources.

b. Framework Selection: Choosing an appropriate automation testing framework is essential for creating a structured and maintainable test suite. Frameworks provide predefined structures, libraries, and guidelines that help in efficient test creation, execution, and maintenance.

c. Continuous Integration: Integrating automation testing into the continuous integration and delivery (CI/CD) pipeline enables automated tests to be executed regularly, ensuring quick feedback on the application's quality and reducing the risk of regression.

d. Collaboration and Communication: Collaboration between developers, testers, and stakeholders is vital for successful automation testing. Regular communication helps identify testing requirements, clarify expectations, and resolve issues efficiently.

3. Introduction to Java for Automation Testing:
Java is a versatile and widely adopted programming language that offers a range of features and libraries suitable for automation testing. This section provides an introduction to Java for automation testing, highlighting the reasons for choosing Java, fundamental concepts of Java programming relevant to test automation, and key Java libraries and frameworks commonly used in automation testing.

3.1 Why Choose Java for Automation Testing?
There are several compelling reasons to choose Java for automation testing:
a. Platform Independence: Java is platform-independent, meaning that tests written in Java can be executed on different operating systems, such as Windows, macOS, and Linux. This feature makes Java a flexible choice for cross-platform testing.
b. Rich Ecosystem: Java has a vast ecosystem of libraries, frameworks, and tools specifically designed for test automation. These resources provide a wide range of functionalities and support for various aspects of automation testing, including web testing, API testing, mobile testing, and more.

c. Object-Oriented Programming (OOP) Paradigm: Java follows the OOP paradigm, which facilitates modular, reusable, and maintainable code. This allows testers to create robust and scalable automation frameworks that can be easily extended and adapted as per the evolving testing requirements.

d. Integration with Testing Frameworks: Java seamlessly integrates with popular testing frameworks like JUnit and TestNG, which provide powerful assertion capabilities, test execution management, and reporting functionalities. This integration enhances the efficiency and effectiveness of automation testing.

e. Community Support: Java has a large and active community of developers and testers who actively contribute to the development of automation testing tools, libraries, and frameworks. This community support ensures regular updates, bug fixes, and extensive documentation, making it easier for testers to find resources and solutions for their automation needs.

3.2 Java Programming Fundamentals for Test Automation

To effectively utilize Java for automation testing, it is essential to understand the following fundamental concepts.

a. Syntax and Data Types: Familiarize yourself with Java syntax, such as variable declaration, data types (e.g., int, String, boolean), operators, control structures (if-else, loops), and exception handling.

b. Object-Oriented Programming (OOP): Understand key OOP concepts, including classes, objects, inheritance, polymorphism, and encapsulation. OOP principles allow for the creation of reusable and maintainable automation code.

c. Java Libraries: Gain knowledge of commonly used Java libraries for automation testing, such as Selenium Web Driver for web testing, REST-assured for API testing, and Appium for mobile testing. Understanding how to use these libraries enhances automation capabilities.

d. File Handling and Input/output Operations: Learn how to read and write data from files using Java's File and IO classes.

This is useful for handling test data, configuration files, and generating test reports.

3.3 Key Java Libraries and Frameworks for Automation Testing.
Java offers a wide range of libraries and frameworks that are instrumental in automation testing.

a. Selenium Web Driver: Selenium is a popular open-source automation testing framework that provides a robust API for web testing. Selenium Web Driver, a part of the Selenium suite, allows testers to interact with web elements, simulate user actions, and perform verifications in web applications.

b. TestNG and JUnit: TestNG and JUnit are widely used testing frameworks for Java that facilitate test execution, assertion management, test grouping, parallel test execution, and reporting. They provide essential functionalities for organizing and running automated tests efficiently.

c. REST-assured: REST-assured is a Java library specifically designed for API testing. It simplifies the testing of Restful APIs by providing a domain-specific language for constructing HTTP requests, handling responses, and asserting API behaviors.

d. Cucumber: Cucumber is a Behavior-Driven Development (BDD) framework.

4. Test Automation Frameworks in Java:

Test automation frameworks provide a structured and organized approach to automation testing, enabling efficient test creation, execution, and maintenance. This section introduces various test automation frameworks available in the Java ecosystem, including data-driven testing with Apache POI, behavior-driven development with Cucumber, the Page Object Model (POM) design pattern, and the usage of TestNG and JUnit for test execution.

4.1 Introduction to Test Automation Frameworks:
Test automation frameworks offer a systematic approach to automation testing by providing a set of guidelines, practices, and tools for test automation. These frameworks aim to enhance test maintainability, reusability, and scalability, allowing testers to build robust and efficient automated test suites. They provide a standardized structure for test code, enabling easier collaboration between team members and facilitating test management and reporting.

4.2 Data-Driven Testing with Apache POI:
Data-driven testing is a technique that involves executing the same test case with multiple sets of test data. Apache POI is a Java library that provides support for reading and writing Microsoft Excel files. It can be used effectively for data-driven testing in automation frameworks. Testers can create data sheets in Excel, store test data in them, and retrieve data during test execution using Apache POI. This approach allows for easy management and modification of test data without modifying the test code.

4.3 Behavior-Driven Development with Cucumber:
Cucumber is a popular behavior-driven development (BDD) framework that promotes collaboration between business stakeholders, developers, and testers. It allows tests to be written in a plain-text format called Gherkin, which is easily understandable by all stakeholders. Cucumber utilizes natural language constructs such as Given-When-Then to describe test scenarios. The Gherkin syntax is then mapped to automation code written in Java using step definitions. Cucumber facilitates clear communication, enhances test readability, and promotes a shared understanding of requirements.

4.4 Page Object Model (POM) Design Pattern:
The Page Object Model (POM) is a design pattern commonly used in automation testing to enhance test maintainability and reusability. It involves creating a separate class for each web page or component in the application under test. These classes encapsulate the page's elements and related actions, providing a high-level abstraction of the user interface. By using the POM design pattern, testers can easily update and maintain the automation code when there are changes to the application's UI, reducing code duplication and improving test stability.

4.5 TestNG and JUnit for Test Execution:
TestNG and JUnit are widely used testing frameworks for Java that provide essential functionalities for test execution and management. These frameworks facilitate the creation and organization of test cases, the definition of test suites, and the execution of tests in a controlled manner. They offer built-in assertion mechanisms, test grouping, parallel test execution, and powerful reporting capabilities. TestNG and JUnit integrate seamlessly with popular automation tools and libraries, making them suitable for various automation testing scenarios.

5. Web Automation Testing with Java:
Web automation testing using Java and Selenium Web Driver provides a powerful combination for automating web application testing. This section focuses on the integration of Selenium Web Driver with Java, techniques for locating web elements,

handling interactions and assertions, and test reporting and logging using Java.

5.1 Selenium Web Driver and Java Integration:
Selenium Web Driver is a popular open-source automation framework used for web application testing. It provides a rich set of APIs to interact with web elements, simulate user actions, and perform verifications. Selenium Web Driver can be seamlessly integrated with Java to leverage the language's robustness and flexibility.
To integrate Selenium Web Driver with Java, testers need to set up the Selenium Web Driver Java bindings, commonly referred to as Selenium Java. These bindings provide the necessary APIs and dependencies to interact with the browser and automate web testing. Testers can configure their Java development environment to include the Selenium Java bindings and start writing automation scripts using Java.

5.2 Locating Web Elements using Selenium and Java:
To interact with web elements, testers need to locate them on the web page. Selenium Web Driver provides various methods to locate web elements based on different criteria such as ID, class name, CSS selectors, XPath, and more.
In Java, testers can utilize the Web Driver’s find Element () and find Elements () methods to locate web elements. They can employ different locator strategies provided by Selenium, such as By.id(), By.className(), By.cssSelector(), By.xpath(), and others, to identify and interact with web elements accurately.

5.3 Handling Interactions and Assertions in Web Testing:
Web automation testing involves performing interactions with web elements and verifying their behavior. Selenium Web Driver in combination with Java offers methods for handling various interactions such as clicking buttons, entering text, selecting options from dropdowns, and handling alerts and pop-ups.
Additionally, testers can employ assertions to verify expected outcomes in web testing. Java's assertion capabilities, coupled with Web Driver’s methods for retrieving element attributes and text content, allow for efficient verification of web element properties, presence, visibility, and more.

5.4 Test Reporting and Logging with Java:
Test reporting and logging are essential aspects of automation testing. Java provides several libraries and frameworks for generating detailed test reports and logging relevant information during test execution.
TestNG and JUnit, the popular testing frameworks in Java, offer built-in reporting capabilities that generate comprehensive HTML reports with detailed test results, including test case status, execution time, and failure/success details. These reports facilitate test result analysis and provide insights into the application's quality.
Moreover, Java's logging frameworks such as Log4j and SLF4J enable testers to log important information, errors, and debug messages during test execution. Logging helps in troubleshooting, identifying issues, and monitoring the automation process. By leveraging Java's integration with Selenium Web Driver, testers can automate web testing effectively. They can locate web elements using various strategies, interact with elements, assert expected behaviors, and generate informative test reports and logs. This combination of Java and Selenium Web Driver empowers testers to
Conduct comprehensive and efficient web automation testing.

In conclusion, utilizing test automation frameworks in Java allows for efficient and maintainable automation testing. Data-driven testing with Apache POI enables testers to execute tests with multiple data sets, while behavior-driven development with Cucumber enhances collaboration and test readability. The Page Object Model design pattern improves test maintainability and reusability. TestNG and JUnit provide robust frameworks for test execution and management, ensuring efficient and controlled test execution. By leveraging these frameworks, automation testers can streamline their testing processes and achieve reliable and scalable automation testing solutions.

6 .Best Practices for Automation Testing with Java:

Automation testing with Java requires adherence to certain best practices to ensure efficient test design, code organization, continuous integration, error handling, and test environment management. This section outlines the recommended best practices for automation testing with Java.

6.1 Test Design and Planning:
a. Test Coverage: Design test cases that cover the critical functionalities of the application under test. Prioritize the test cases based on their importance and impact on the system.
b. Test Case Independence: Ensure that each test case is independent of others and can be executed in isolation. This allows for better test traceability and reduces dependencies between test cases.
c. Test Data Management: Develop a strategy for managing test data effectively. Separate test data from test code, utilize data-driven testing techniques, and consider using external data sources or tools to generate test data.

6.2 Test Code Organization and Maintainability:
a. Modularity and Reusability: Organize test code into modular units that can be easily reused across test suites. Utilize the Page Object Model (POM) design pattern to create reusable components for interacting with web elements.
b. Code Comments and Documentation: Add comments to explain the purpose and functionality of the test code. Document any assumptions, limitations, or special considerations to facilitate collaboration and understanding among team members.
c. Maintainable Test Structure: Structure the test code in a logical and intuitive manner. Use appropriate naming conventions for classes, methods, and variables. Consider using packages and directories to categorize tests and related resources.

6.3 Continuous Integration and Test Execution:
a. Version Control: Utilize a version control system (e.g., Git) to manage the test code and collaborate with team members effectively. Maintain a clean and organized repository structure with proper branching and merging strategies.
b. Continuous Integration (CI): Integrate automation tests with CI tools (e.g., Jenkins, Bamboo) to automate the test execution process. Configure CI pipelines to trigger tests automatically on code commits and generate test reports.
c. Parallel Test Execution: Utilize parallel test execution capabilities provided by frameworks like TestNG or JUnit to optimize test execution time. Distribute tests across multiple threads or machines to run them concurrently.

6.4 Error Handling and Reporting:

a. Proper Exception Handling: Implement appropriate exception handling mechanisms to handle and report errors gracefully during test execution. Use try-catch blocks to catch exceptions and handle failures effectively.
b. Logging and Reporting: Utilize logging frameworks (e.g., Log4j, SLF4J) to capture relevant information, errors, and debug messages during test execution. Generate detailed test reports that provide comprehensive information about test results and failures.
c. Failure Analysis: Analyze test failures and errors to identify root causes. Maintain a systematic approach for debugging and troubleshooting failed tests to resolve issues promptly.

6.5 Test Environment Configuration and Management:
a. Versioned Dependencies: Manage dependencies, including automation frameworks, libraries, and tools, using dependency management tools like Maven or Gradle. Ensure consistent and versioned dependencies across different environments.
b. Configuration Management: Centralize the configuration settings required for test execution, such as URLs, credentials, and environment-specific settings. Use configuration files or external sources to store and manage these settings.
c. Environment Provisioning: Automate the process of setting up and configuring test environments. Use infrastructure-as-code tools (e.g., Ansible, Docker) to create reproducible and consistent test environments.
By following these best practices, testers can establish a solid foundation for automation testing with Java. They can achieve improved test coverage, maintainable test code, efficient test execution, accurate error reporting, and well-managed test environments. These practices contribute to the overall effectiveness and success of automation testing efforts.

7. Drawbacks of manual testing using selenium in Java

While Selenium is primarily known for its automation capabilities, it can also be used for manual testing. However, there are some drawbacks to consider when using Selenium for manual testing:

Lack of Human Judgment: Manual testing involves the tester's critical thinking, intuition, and judgment to identify potential issues. When using Selenium for manual testing, these human factors can be limited. Selenium focuses on executing predefined test scripts and may not have the ability to identify subtle issues or make subjective judgments that a human tester can.

Time-Consuming Test Case Execution: Manual testing with Selenium can be time-consuming compared to traditional manual testing methods. Testers need to interact with the application manually, navigate through various screens, and perform test steps using Selenium's APIs. This process can be slower and less efficient than directly interacting with the application in a manual testing environment.

Limited Exploration and Ad hoc Testing: Manual testing using Selenium may restrict exploratory testing and ad hoc testing practices. Selenium scripts are usually designed to follow predefined test scenarios, leaving less room for spontaneous exploration of the application. Exploratory testing, where testers actively explore the software and test different areas, may not be as effective when using Selenium for manual testing.

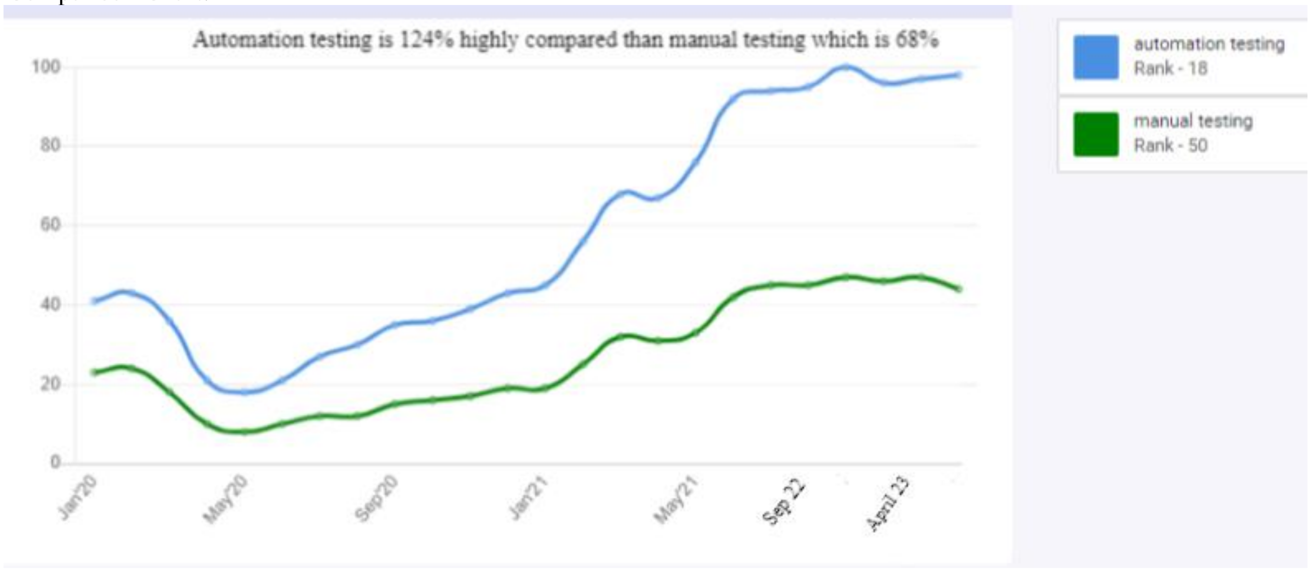
Difficulty in Dynamic and Unpredictable Testing Scenarios: Selenium's primary strength lies in its ability to automate repetitive test scenarios. However, when it comes to manual testing, particularly in dynamic and unpredictable scenarios, Selenium may face challenges. Adapting and responding to changes in real-time or testing complex scenarios that require immediate decision-making can be cumbersome when relying solely on Selenium.

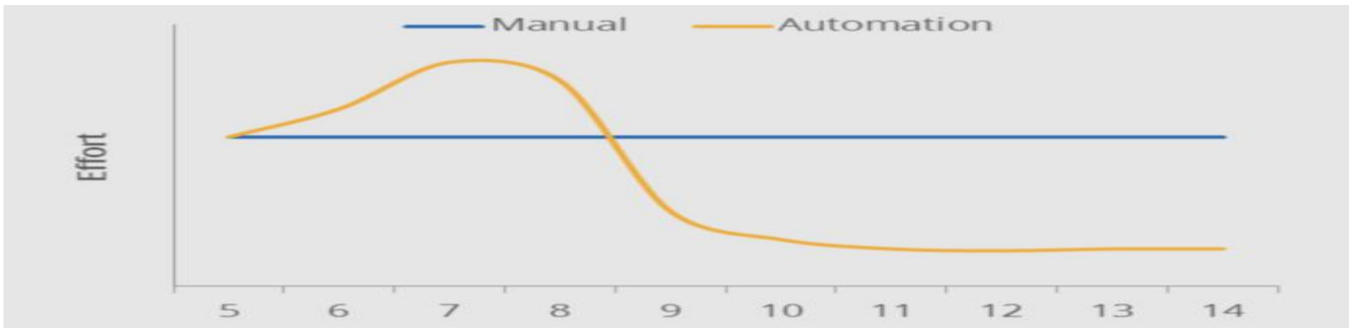
Limited Collaboration and Communication: Manual testing often involves effective collaboration and communication between testers, developers, and other stakeholders. When using Selenium for manual testing, the collaborative aspect may be limited. Testers need to manually document and communicate issues rather than directly discussing them with other team members, which can slow down the feedback loop and hinder efficient problem-solving.

Skill Dependency and Learning Curve: While Selenium provides a user-friendly interface for manual testing, it still requires testers to have knowledge of Selenium's features, APIs, and test script creation. Testers need to learn Selenium's syntax, commands, and object locators, which can add complexity and a learning curve to manual testing using Selenium.

Maintenance Overhead: Test scripts developed for manual testing using Selenium may require maintenance as the application evolves or changes over time. Any updates or modifications to the application's user interface or functionality may necessitate corresponding changes in the test scripts. Testers need to invest effort in maintaining and updating the test scripts, which can be time-consuming.

Comparison Chart:





8. Conclusion:

8.1 Summary of Key Findings:
In this research paper, we explored the use of Java for automation testing and discussed various aspects related to its integration with automation tools and frameworks. We highlighted the importance and benefits of automation testing and identified the challenges faced in automation testing. We also discussed strategies and best practices for automation testing with Java. We found that Java is a popular choice for automation testing due to its platform independence, rich ecosystem, support for object-oriented programming, integration with testing frameworks, and active community support. Java provides a solid foundation for building robust and scalable automation frameworks.
We explored the fundamental concepts of Java programming relevant to automation testing, such as syntax, data types, object-oriented programming, and file handling. We also identified key Java libraries and frameworks, including Selenium Web Driver,

TestNG, JUnit, REST-assured, and Cucumber that are commonly used for automation testing.
We delved into web automation testing with Java, covering topics such as Selenium Web Driver and Java integration, locating web elements, handling interactions and assertions, and test reporting and logging with Java. These concepts enable testers to effectively automate web testing tasks using Java.
Furthermore, we discussed best practices for automation testing with Java, emphasizing the importance of test design and planning, test code organization and maintainability, continuous integration and test execution, error handling and reporting, and test environment configuration and management. Following these best practices ensures efficient and scalable automation testing solutions.

8.2 Future Trends in Automation Testing with Java:
The field of automation testing continues to evolve, and there are several future trends to watch out for in the context of Java:
a. Artificial Intelligence and Machine Learning: Integration of AI and ML techniques in automation testing can lead to smarter test case generation, intelligent test data management, and self-healing tests. Java libraries and frameworks will likely provide support for incorporating AI and ML capabilities into automation workflows.

b. Robotic Process Automation (RPA): RPA involves automating repetitive tasks performed by humans. The integration of RPA with Java for automation testing can lead to more efficient and comprehensive test automation across different systems and applications.

c. Cloud-Based Testing: Cloud computing offers scalability, flexibility, and cost-effectiveness in setting up test environments. Java-based automation frameworks are expected to provide seamless integration with cloud-based testing platforms and services, enabling testers to execute tests in distributed and scalable environments.

d. IoT and Mobile Testing: With the increasing adoption of IoT devices and mobile applications, automation testing frameworks in Java will likely offer enhanced support for IoT and mobile testing, including simulators, emulators, and device farms.

e. DevOps and Continuous Testing: The integration of automation testing with DevOps practices will continue to gain importance. Java-based automation frameworks will evolve to support continuous integration, continuous delivery, and continuous testing, enabling faster feedback loops and improved software quality.

In conclusion, automation testing with Java provides a robust and flexible solution for organizations aiming to streamline their testing processes and improve software quality. By leveraging Java's extensive libraries, frameworks, and best practices, testers can build scalable and maintainable automation frameworks. Embracing future trends in automation testing will further enhance the effectiveness and efficiency of Java-based automation testing solutions.

References

[1] Garaga, R., & Tirumala, R. (2017). An automated framework for web application testing. International Journal of Scientific Research in Computer Science, Engineering, and Information Technology (IJSRCSEIT), 2(2), 241-246.
[2] Mishra, R., & Rath, S. K. (2019). Automation Testing of Web Applications using Selenium WebDriver and TestNG Framework. In Proceedings of the 3rd International Conference on Internet of Things and Connected Technologies (ICIoTCT) (pp. 1489-1494). Springer, Singapore.
[3] Lim, C. C., & Ng, T. K. (2018). Automated web testing using the Page Object Model design pattern. Journal of Information Systems Engineering & Management, 3(2), 17.
[4] Acharya, A., & Chaurasia, V. (2016). A Comparative Study of Selenium and Sahi for Web Testing. International Journal of Computer Science and Mobile Computing, 5(1), 224-232.
[5] Mishra, R., & Rath, S. K. (2018). Web Application Testing using Selenium WebDriver and Java. International Journal of Advanced Research in Computer Science, 9(3), 272-278.
[6] Rathi, N., & Mohapatra, S. (2017). Automation of Web Application Testing Using Selenium WebDriver with Java. International Journal of Computer Science and Engineering, 5(12), 266-269.
[7] Zhou, C., & Yao, Z. (2017). Research on automation testing of web application based on Selenium WebDriver. In 2017 International Conference on Information Science and Communications Technologies (ICISCT) (pp. 422-425). IEEE.