Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Technology

Master's Thesis

A Comparative Analysis: Web Application Testing vs. Mobile Application Testing

Sachin Sarvothama

© 2024 CZU Prague

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Eng. Sachin Sarvothama, B.E.

Global Information Security Management

Thesis title

A Comparative Analysis: Web Application Testing vs. Mobile Application Testing

---

Objectives of thesis

This thesis's primary objective is to compare Web Application Testing and Mobile Application Testing comprehensively. By understanding the distinctions and nuances between these testing domains, we can better grasp the unique challenges faced in each and the appropriate testing methodologies that should be employed.

Methodology

A comparative study of software testing techniques can be performed to improve the testing standard of both web and mobile applications. Choosing a tool for web app testing is easy, but it can be complicated for mobile app testing. The objective of software testing using Selenium for web-based and Appium for mobile applications is to ensure that the application meets the desired quality standards and performs as expected on different platforms and devices. The main goal of testing an app, web or mobile, is to ensure its usability and proper functioning under other circumstances and provide an excellent user experience.

A combination of literature review and case studies will be employed to achieve this research's objectives. The study will analyze existing testing frameworks, industry best practices, and real-world scenarios to highlight the differences between Web Application Testing and Mobile Application Testing.

The proposed extent of the thesis

60 – 80 pages

Keywords

Selenium, Appium, Web Application, Mobile App, Software Testing

---

Recommended information sources

Ahmed, Maryam, and Rosziati Ibrahim. "A comparative study of web application testing and mobile application testing." Advanced Computer and Communication Engineering Technology: Proceedings of the 1st International Conference on Communication and Computer Engineering. Springer International Publishing, 2015.

Arya, K. V., and Hemdutt Verma. "Keyword driven automated testing framework for web application." 2014 9th International Conference on Industrial and Information Systems (ICIIS). IEEE, 2014.

ASLAM, ZAHEER, et al. "PERFORMANCE-BASED ANALYSIS OF TEST AUTOMATION TOOLS FOR ANDROID APPLICATIONS."

Gojare, Satish, Rahul Joshi, and Dhanashree Gaigaware. "Analysis and design of selenium webdriver automation testing framework." Procedia Computer Science 50 (2015): 341-346.

Kirubakaran, B., and V. Karthikeyani. "Mobile application testing—Challenges and solution approach through automation." 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering. IEEE, 2013.

Ramya, Paruchuri, Vemuri Sindhura, and P. Vidya Sagar. "Testing using selenium web driver." 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT). IEEE, 2017.

Singh, Harshit, et al. "GUI Testing Android Application." 2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO). IEEE, 2022.

Wang, Junmei, and Jihong Wu. "Research on mobile application automation testing technology based on appium." 2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS). IEEE, 2019.

Zun, Da, Tao Qi, and Liping Chen. "Research on automated testing framework for multi-platform mobile applications." 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS). IEEE, 2016.

Expected date of thesis defence

2023/24 SS – PEF

The Diploma Thesis Supervisor

Ing. Václav Lohr, Ph.D.

Supervising department

Department of Information Technologies

Electronic approval: 4. 9. 2023

doc. Ing. Jiří Vaněk, Ph.D.

Head of department

Electronic approval: 3. 11. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

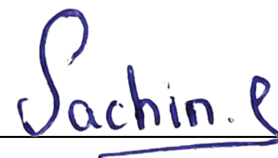
Dean

Prague on 01. 03. 2024

Declaration

For my master's thesis, I have independently written "**A Comparative Analysis: Web Application Testing vs. Mobile Application Testing**", and solely referred to the sources cited. As the author of this master's thesis, I certify that it's not a copyright violation.

In Prague on 27-03-2024



Sachin Sarvothama

Acknowledgement

Throughout the research project, I would like to express my sincere gratitude to Ing. Václav Lohr, Ph.D., my thesis advisor, for his valuable guidance, unrelenting assistance, and valuable insight. These individuals contributed significantly to the direction and quality of this thesis through their expertise and capabilities.

I would like to express my deepest gratitude to my esteemed professors, dear classmates, and the dedicated academic staff of the Global Information Security Management (GISM) department at the Czech University of Life Sciences, Prague. My academic journey has been shaped and enhanced significantly by their consistent support over the past two years. In addition to enhancing my understanding of the subject matter, the encouragement, guidance, and exchange of knowledge have also facilitated a collaborative and enriching learning environment. I am grateful to the entire academic community for their invaluable contributions to my education and growth, and I am deeply grateful for the collective efforts that contributed to my personal growth and development.

Finally, I want to extend my heartfelt gratitude to my family and friends for their patience, resilience, and backing during my academic journey.

A Comparative Analysis: Web Application Testing vs. Mobile Application Testing

Abstract

This thesis examines the practical application of Selenium for web application testing and Appium for mobile application testing to improve software quality and reliability through automation. A thorough investigation of various testing tools and methods is conducted, delving into how automation frameworks can help ensure high levels of functionality, usability, and compatibility in web and mobile programs. The research incorporates case studies that underscore the importance of conducting functionality, usability, and compatibility testing to address problems tied to software applications. While automation frameworks streamline the testing process, no framework can replace traditional manual testing. Both automated and manual strategies must be utilized together to deliver comprehensive testing. The studies highlight best practices for developing an efficient testing regimen.

The study uncovers that Selenium exhibits great precision in mechanizing obligations inside web applications, while Appium productively assesses portable application usefulness crosswise over different gadgets and working frameworks. The examination between web and versatile application testing underscores the significance of customized testing methodologies to satisfy stage particular necessities. Proposals for future examination centre around investigating AI incorporation in testing, IoT application testing, improved security estimates, execution testing in 5G conditions, and usability testing across gadgets to advance programming testing practices. Furthermore, the investigation discovered that Selenium could robotize occupations effectively on web applications saving time throughout testing. Appium can evaluate portable applications on different working frameworks and gadgets, for example, iOS, Android and Windows. This permits testers to distinguish bugs cross-stage before dispatch. As innovation keeps on advancing, future exploration could concentrate on artificial intelligence joining into testing improve precision and effectiveness. Testing IoT applications is critical as these applications associate numerous gadgets. In this manner, security must be considered even more deliberately. Execution must likewise be tried as 5G

networks become progressively normal. Usability ought to be concentrated on cross stage as client experience ought not rely upon the gadget being utilized.

This research aims to further our understanding of software testing techniques. It provides meaningful perspectives for industry specialists seeking to refine their testing methodologies and develop websites and apps of high calibre. These digital offerings should satisfy users' needs and align with sector benchmarks. By gaining familiarity with current evaluation methods, developers can create error-free programs meeting users where they are. The findings offer pragmatic guidance for strengthening evaluation processes to deliver top-notch, dependable digital experiences.

Keywords: Selenium, Appium, Web Application, Mobile App, Software Testing, Automation Framework, Functionality, Compatibility, Usability.

Srovnávací analýza: Testování webových aplikací vs. Testování mobilních aplikací

Abstrakt

Tato práce zkoumá praktickou aplikaci Selenium pro testování webových aplikací a Appium pro testování mobilních aplikací za účelem zlepšení kvality a spolehlivosti softwaruprostřednictvím automatizace. Provádí se důkladné zkoumání různých testovacích nástrojůa metod a zkoumá se, jak mohou automatizační rámce pomoci zajistit vysokou úroveň funkčnosti, použitelnosti a výkonu ve webových a mobilních programech. Výzkum zahrnujepřípadové studie, které zdůrazňují důležitost provádění testování výkonu, zabezpečení,použitelnosti a kompatibility pro řešení konkrétních problémů souvisejících se softwarovými aplikacemi. Zatímco automatizační rámce zjednodušují proces testování, žádný rámec nemůže nahradit tradiční ruční testování. Jak automatizované, tak manuálnístrategie musí být použity společně, aby bylo možné poskytovat komplexní testování. Studiezdůrazňují osvědčené postupy pro vývoj účinného testovacího režimu.

Studie odhaluje, že Selenium vykazuje velkou přesnost v mechanizaci povinností v rámci webových aplikací, zatímco Appium produktivně hodnotí užitečnost přenosných aplikací napříč různými přístroji a pracovními rámcemi. Zkouška mezi webovým testováním a testováním všestranných aplikací podtrhuje význam přizpůsobených metodologií testování pro splnění požadavků konkrétní fáze. Návrhy na budoucí testování se soustředí na zkoumání začlenění umělé inteligence do testování, testování aplikací IoT, vylepšené odhady zabezpečení, testování provádění v podmínkách 5G a testování použitelnosti napříč gadgety, aby se pokročily postupy testování programování. Vyšetřování navíc zjistilo, že Selenium dokáže efektivně robotizovat povolání ve webových aplikacích, což šetří čas během testování. Appium dokáže vyhodnocovat přenosné aplikace na různých pracovních rámcích a gadgetech, například iOS, Android a Windows. To umožňuje testerům rozlišit chyby v různých fázích před odesláním. Vzhledem k tomu, že inovace stále postupují, budoucí průzkum by se mohl soustředit na zapojení umělé inteligence do testování, které zlepší přesnost a efektivitu. Testování aplikací IoT je zásadní, protože tyto aplikace sdružují řadu gadgetů. Tímto způsobem musí být bezpečnost zvažována o to více záměrně. Provedení se musí rovněž vyzkoušet, protože sítě 5G se postupně stávají normálními. Použitelnost by

se měla soustředit na různé fáze, protože klientská zkušenost by se neměla spoléhat na využití gadgetu.

Tento výzkum si klade za cíl prohloubit naše chápání technik testování softwaru. Poskytuje smysluplné perspektivy pro oborové specialisty, kteří chtějí zdokonalit své testovací metodiky a vyvíjet webové stránky a aplikace vysoké kvality. Tyto digitální nabídky by měly uspokojit potřeby uživatelů a být v souladu se sektorovými benchmarky. Seznámením se s aktuálními metodami hodnocení mohou vývojáři vytvářet bezchybné programy, které se setkávají s uživateli tam, kde jsou. Zjištění nabízejí pragmatický návod pro posílení procesů hodnocení s cílem poskytnout špičkové a spolehlivé digitální zážitky.

Klíčová slova: Selenium, Appium, webová aplikace, mobilní aplikace, testování softwaru, Automatizační rámec, funkčnost, kompatibilita, použitelnost.

Table of Contents

1. Introduction **1**

2. Objectives and Methodology **3**

2.1 Objectives 3

2.1.1 Main Goal 3

2.1.2 Key Aims 3

2.2 Methodology 4

2.2.1 Literature Review Method 4

2.2.2 Case Study Methodology 4

2.2.3 Comparative Study Design 5

2.2.4 Selection of Testing Tools 5

2.2.5 Usability Testing Approach 5

3. Literature review **6**

3.1 Overview of Software Testing 7

3.1.1 Evolution of Software Testing 8

3.1.2 Importance of Software Testing in Application Development 9

3.2 Web Application Testing Techniques 10

3.2.1 Manual Testing vs Automated Testing for Web Applications 11

3.2.2 Common Challenges in Web Application Testing 12

3.3 Mobile Application Testing Techniques 13

3.3.1 Key Differences Between Web and Mobile Application Testing 15

3.3.2 Best Practices in Mobile Application Testing 18

3.4 Selenium for Web Application Testing 19

3.4.1 Introduction to Selenium Automation Tool 20

3.4.2 Advantages and Limitations of Selenium in Web Testing 22

3.5 Appium for Mobile Application Testing 23

3.5.1 Introduction to Appium Framework 23

3.5.2 Advantages and Limitations of Appium 25

3.6 Comparative Analysis 26

3.6.1 Previous Studies 26

3.6.2 Identified Gaps 28

3.7 Summary of Literature Reviewed 29

4. Practical part **31**

4.1 Implementation of Selenium for Web App Testing Tools 31

4.1.1 Setting Up Selenium for Web Application Testing 31

4.1.1.1 Test Scenarios 34

4.1.1.2 TestCase Design 39

4.1.1.3 Results and Findings ... 45
4.2 Implementation of Appium for Mobile App Testing Tools ... 49
4.2.1 Configuring Appium for Mobile App Testing ... 49
4.2.1.1 Test Scenarios ... 52
4.2.1.2 TestCase Design ... 56
4.2.1.3 Results and Findings ... 60
4.3 Case Studies in Web Application Testing ... 65
4.3.1 Case Study A: Functional Testing of a Web Application ... 65
4.3.2 Case Study B: Compatibility Testing Across Various Browsers. ... 66
4.4 Case Studies for Mobile App Testing ... 70
4.4.1 Case Study A: Functional Testing of Mobile Apllication ... 70
4.4.2 Case Study B: Compatibility Testing Across Various Mobile Devices ... 72
5. Results and Discussion ... 75
5.1 Analysis of testing results ... 75
5.2 Comparison between web and mobile application testing ... 78
5.2.1 Testing Approaches ... 78
5.2.2 Tools and Techniques ... 78
5.2.3 Considerations ... 79
6. Conclusion ... 81
6.1 Summary of findings ... 82
6.2 Implications for industry ... 83
6.3 Recommendations for future research ... 84
7. References ... 86
8. List of Figures, Tables and Abbreviations ... 93
8.1 List of Figures ... 93
8.2 List of Tables ... 94
8.3 List of Abbreviations ... 94

1. Introduction

Software has become essential to contemporary society, propelling progress and changing numerous facets of everyday life. With the need for top-notch software solutions continuously increasing, the significance of efficient software evaluation can’t be emphasized enough. Software testing is a crucial process that confirms the dependability, capabilities, and functionality of programs, ultimately bettering user delight and decreasing dangers linked to software system malfunctions (Myers 2011). Testing is important as it helps validate that applications work as intended and are free of defects (Pressman 2014).

By systematically assessing programs through techniques like requirement testing and unit testing, issues can be identified and resolved before public release. These supports providing users with smooth experiences devoid of unexpected problems. Overall, thorough testing plays a key role in satisfying users and building confidence that software will perform well for its intended purpose.

Today’s rapidly changing technological environment poses challenges where web and mobile apps dominate. Testing methods must thoroughly address both platforms. Web and mobile app testing each involve distinct issues requiring customized solutions to guarantee top functionality and user satisfaction. Web applications may encounter problems with different browsers, screen sizes, or internet speeds. Mobile apps must function flawlessly across an array of devices with varying processing power and OS versions while considering usability on small screens. Comprehensive testing explores all potential issues to confirm smooth experiences.

Though web and mobile testing methodologies vary, their purpose unites in delivering polished digital products meeting users’ needs. This thesis will closely study web application testing and mobile application testing to compare their differences, challenges, test strategies, and quality expectations. An examination of usability and the user experience for both web and mobile apps in different situations is included. The goal is to advance software evaluation methods within the continually changing digital world. Specifically, this research aims to explore the unique issues that testers face for each platform. For web apps, testing cross-browser compatibility and software

functionality is important to consider. For mobile apps, factors like various device types, orientations, and network conditions play a crucial role in ensuring optimal performance.

2. Objectives and Methodology

2.1 Objectives

This work's main aim is to fully compare testing for Web Applications and Mobile Applications. This big aim is broken down into smaller, detailed goals so we can carefully study each area.

2.1.1 Main Goal

Our main aim is to look deeply at how software testing is done differently for web and mobile apps. This means studying the unique problems for both, in order to get a full understanding of different testing methods.

2.1.2 Key Aims

- **Identification and Analysis of Unique Challenges**

  One of the goals is to see and study the unique problems for web and mobile app testing. This involves a detailed study into different problems faced in each area, like how well it works, how far it is compatible with different browsers & devices, and issues with user interfaces.

- **Proposal of Appropriate Testing Methodologies**

  Another important goal is to suggest the right methods for testing web and mobile apps. This needs a clever approach in creating methods that work best for the different testing needs of each platform, to make sure the outcomes are strong and trustworthy.

- **Quality Standards Adherence**

  Ensuring that the proposed testing methodologies meet desired quality standards is an essential aim of this research. This involves a meticulous evaluation of industry standards and best practices to guarantee the reliability and effectiveness of the testing approaches recommended for both web and mobile applications.

- **Usability and User Experience Assessment**

  Assessing the usability and user experience of both web and mobile applications under various scenarios is a paramount aim. This involves an examination of factors influencing user-friendliness, accessibility, and overall performance to provide valuable insights into the end-user perspective.

In summary, the objectives of this research are tailored to provide a comprehensive understanding of the challenges in Web Application Testing and Mobile Application Testing, propose effective testing methodologies, ensure adherence to quality standards, and evaluate the functionality and user experience of applications across diverse scenarios. These objectives collectively form the foundation for a nuanced and insightful comparative analysis.

2.2 Methodology

The research methodology is a critical component that outlines the systematic approach adopted to achieve the specified objectives. In this chapter, various strategic approaches are carefully designed and integrated to provide a robust framework for conducting the comparative analysis between Web Application Testing and Mobile Application Testing.

2.2.1 Literature Review Method

We explore existing test methods and industry strategies in our literature review. The aim is to understand how software testing evolved and its key role in creating apps.

2.2.2 Case Study Methodology

We take on case studies to study practical situations. Through these studies, we aim to give concrete examples that supplement textbook learning. They expose challenges in testing web and mobile applications.

2.2.3 Comparative Study Design

Our research heavily relies on comparing studies of web and mobile application testing. We dive deep into previous studies and the gaps they left to give a thorough comparison analysis.

2.2.4 Selection of Testing Tools

Selecting testing tools plays an important role in our research. We've chosen Selenium for web app testing and Appium for mobile app testing. They were the top picks because they work well with various platforms and devices. This lets us give a complete review of the testing field.

2.2.5 Usability Testing Approach

In the research plan, we focus on testing. We want to make sure web and mobile apps work well. They must be user-friendly! To do this, we follow a strict procedure. We think of many different situations to get a full picture of what the user might experience.

Let's recap. Section 2.2 shows the research plan. We look at other studies, check case studies, run tests, pick the right tools, and focus on testing usability. This strong plan helps us dive deep into Web and Mobile App Testing.

3. Literature review

The focus of this chapter is software testing as shown by a wide-ranging literature review which concentrates particularly on the testing of web applications, mobile applications and necessary apparatuses like Selenium and Appium. It extensively examines how software testing has developed over time as one of the most important activities to ensure that programs are of high quality and reliable.

The review meticulously examines software testing, analyzing approaches for both websites and mobile apps, and delving into key automation frameworks that have propelled the discipline forward. By investigating the historical evolution of testing and underscoring its significance, the chapter offers a comprehensive overview of a field that has become increasingly crucial as software continues to permeate various facets of modern life.

The exploration of various web application testing techniques enables a contrast between manual and automated testing approaches. This comparison sheds light on the intricacies of ensuring functionality and usability for web-based programs. Similarly, the investigation of mobile application testing methods highlights the distinct factors and optimal approaches essential for effectively testing mobile apps in our evolving digital world.

As these testing techniques are assessed, common challenges encountered in validating web and mobile applications become apparent. Ensuring that such programs work as intended across different browsers, devices, and operating systems involves complexities related to replication, isolation, and coverage. Both manual and automated strategies present their own set of benefits, with automated testing being able to test more scenarios at a faster pace.

Overall, properly examining available testing methods assists quality assurance teams in selecting the most suitable approaches for their specific testing needs and development lifecycles.

3.1 Overview of Software Testing

The Software testing plays a vital role in software development by helping to guarantee program quality, dependability, and functionality. It involves methodically assessing software to discover faults and mistakes, with the goal of conveying a result that meets client necessities and desires. Throughout the years, advances in innovation and methodologies have molded the advancement of software testing, bringing about different testing systems and procedures being created. Testing strategies have developed from essential functional testing to more modern approaches that are more thorough, for example, integration testing, system testing, and acceptance testing. These strategies recognize issues from different points of view, including how application segments function together and how the product works inside a framework setting. The evolution of testing looks for to consistently enhance the item, discover issues right on time, and limit expenses from defects. Even though testing cannot ensure that software will be bug-free, it significantly reduces the risks and improves the overall quality of the product.

In the field of software testing, the value of complete testing methods is impossible to overemphasize. Successful testing helps in identifying and correcting problems early during the development process, decreasing the chances of expensive mistakes in the final product (Pressman, 2014). Software testing strategies have progressed from manual arbitrary approaches to organized methods that include automation and strict testing routines (Beizer, 1990). Testing each component of a program and ensuring everything functions as intended is crucial. This reduces post-launch issues that can damage a brand's reputation and lose users. The costs to fix bugs found after launch greatly exceed those solved during testing. Thorough testing saves money while providing users a quality, smooth experience with an application or site.

Recent studies into software testing have highlighted the importance of including testing tasks throughout the software development cycle. Taking this approach guarantees that software products satisfy quality benchmarks, function as intended, and provide a favorable user experience (Smith et al., 2019). By adopting optimal techniques in software testing, companies can strengthen the dependability and functionality of their programs while decreasing risks connected to software malfunctions. When testing is integrated at each stage of development starting with

initial planning, it allows potential issues to be identified and addressed early before they become more serious and costly problems later on. Regular testing catches errors that could cause apps or systems to crash or malfunction when deployed, compromising performance for end users. Organizations that devote sufficient time and resources to testing various usage scenarios minimizes post-launch disruptions and ensures a smooth user experience. This delivers ongoing value and reduces the need for emergency fixes after launch.

In our current fast-paced digital environment, where software applications play a vital function in numerous industries, the necessity for powerful software testing methodologies is more crucial than ever before. Scholars and professionals consistently seek out innovative testing techniques and instruments to deal with the progressively evolving challenges in software program advancement (Jones & Johnson, 2020). By keeping up with industry developments and breakthroughs in software testing, organizations can enhance their testing processes and deliver high-quality software solutions to end-users. As new technologies emerge at a rapid pace, testing methods must also evolve accordingly. Researchers must explore novel approaches that test the full capabilities of applications while maintaining efficiency. Companies should dedicate resources to monitoring the software testing field for any recent testing strategies or tools. Adopting emerging testing practices can help ensure applications perform as intended for customers.

3.1.1 Evolution of Software Testing

The Software testing has changed over time due to advances in technology, methodologies, and good practices. It has shifted from sporadic methods to orderly approaches that include many testing tactics and strategies (Beizer 1990). Software testing is crucial in app development. Proper testing boosts the quality, reliability, and performance of software products (Myers et al. 2011).

Software testing has had several key stages:

- **Manual Testing:** In software development’s early days, testing was mainly manual. Human testers ran test cases and checked the app’s functionality. This method took a lot of time and had the risk of human error (Myers et al. 2011).

- **Structured Testing:** As software systems grew in complexity, orderly testing methods like Equivalence Partitioning, Boundary Value Analysis, and Decision Table Testing came about. These techniques arranged and recorded test cases, making the testing process more organized (Myers et al. 2011).

- **Automated Testing:** The introduction of software testing tools led to automated testing. Tools such as Selenium and Appium enabled web and mobile app testing automation. Automated testing greatly cut down the time and work needed for testing, making it more manageable for big projects (Myers et al. 2011).

- **Agile Testing:** Agile methods bring a fresh, incremental style to testing. It's done throughout the entire development cycle. This approach spots problems earlier and fixes them faster (Myers et al. 2011).

- **DevOps and DevSecOps:** DevOps unifies development and operations. This bond speeds up high-quality software delivery. As cloud technology advanced and firewalls were lowered, security shifted to application level. This change birthed DevSecOps that brings security testing into the development cycle (Myers et al. 2011).

Wrapping up, the evolution of software testing was driven by the need for enhanced, thorough methods. The blend of technology, methodologies, and best practices led to the creation of advanced testing tools that can enhance software quality and dependability.

3.1.2 Importance of Software Testing in Application Development

Software testing is essential when making apps. Its main job is to make sure the software works correctly and is of high quality (Smith, 2018).

Here are some reasons why testing software is critical:

- **Quality Control:** Testing helps find and fix problems, boosting performance, usability, and reliability. This ensures the software meets quality targets (Johnson, 2019).
- **Better User Experience:** Tests can find room for improvement in performance, functionality, and usability. This makes for a smoother user experience (Brown & Lee, 2020).
- **Cutting Risks:** Testing helps fix problems before launch, so there are fewer issues or glitches once it's live. It cuts down on risks (Garcia et al., 2017).
- **Saving Time and Money:** Testing sooner rather than later finds and fixes issues quickly, reducing delay and unnecessary costs (Adams, 2016).
- **Pleasing Customers:** Testing makes sure the software lives up to customers' hopes, creating a better user experience and higher satisfaction (Roberts & Patel, 2021).

In short, software testing is key to making sure apps are high quality, user-friendly, safe, efficient, and meet customers' needs.

3.2 Web Application Testing Techniques

The Web application testing techniques play an important role in validating the functionality, usability, and security of web-based software programs. Employing these testing methods is critical for pinpointing and fixing potential problems that may negatively influence how web apps perform. Some key techniques support evaluating whether web apps work as intended across different browsers, devices, and network conditions. Usability testing allows assessing how easy web apps interface is to use and learn. Security testing aids detecting vulnerabilities that could expose apps to unauthorized access. Overall, leveraging varied testing approaches helps ensure web apps consistently provide users a dependable experience.

Here are some key points regarding web application testing techniques:

- **Manual Testing vs. Automated Testing:**

While manual testing requires testers to execute test cases by hand without the aid of automation tools, automated testing leverages software to automatically run test scenarios. Both approaches offer benefits and are frequently blended to achieve thorough testing. For example, manual testing allows testers to quickly test new features or changes since automation setup is not required. However, it can be time-consuming and repetitive. On the other hand, automated testing expedites the process through automated execution, but setup time is involved. An ideal strategy is combining the two, using manual testing for initial checking and automated for regression to ensure everything continues working as intended. This balanced hybrid approach maximizes coverage within budget and time constraints (Vogels 2023).

- **Role in Ensuring Quality:**

These techniques are essential in guaranteeing the quality and dependability of web applications by identifying bugs, security vulnerabilities, and performance issues early during the development process. They play a vital part in confirming the caliber of web applications by finding problems, weaknesses that could be exploited by malicious actors, and issues that slow performance before development is finished. This allows developers to fix any issues prior to completion, resulting in a more robust and secure final product. By detecting flaws at the beginning, these methods help ensure web applications function as intended for users when launched.

3.2.1 Manual Testing vs Automated Testing for Web Applications

Web applications are usually tested in two ways: manual and automated testing. Each of these methods comes with their pros and cons. The selection between the two depends on project-specific needs and limitations. (Son, 2024a; Katalon, 2023)

Manual testing is a process where human testers play with the web application to find and record any glitches or bugs. This mode is a good match for projects of smaller size or those that are in their infancy. Yet, note that manual testing can take a lot of time, is susceptible to human mistakes, and might not be a great fit for extensive applications.

Manual and automated testing are two frequently used approaches for web application testing. Each strategy has its strong and weak points. It's the project's particular requirements and confines that determine which one to use.

Humans doing the testing to spot and list down any issues or bugs is what manual testing is all about. It's a good fit for smaller projects or those that are still in the early phase of development. Manual testing lets testers use their judgment and creativity to uncover issues. This can be useful in finding problems that come up unexpectedly. Yet, manual testing has its drawbacks - it may eat up a lot of time, is prone to human errors, and may not be the best choice for large applications.

Automated testing is quite different. It involves using software tools to run preset testing scripts and juxtaposing final outcomes with expected ones. This approach is more effective, dependable, and scalable than manual testing. Thus, it's a great option for large applications. Automated testing can be run over and over which allows developers to find and kill any issues that might show up during development. But remember, automated testing requires special technical skills and can be costlier than manual testing (Son 2024).

Summing up, the decision to choose manual or automated testing for web apps lies in the project's unique needs and limits. Small projects or early-stage apps are best for manual testing. Automated testing becomes a smart pick for larger applications due to its greater efficiency, reliability, and scope (Manual Vs. Automated Testing | What's The Deal? 2024).

3.2.2 Common Challenges in Web Application Testing

There are several common challenges that testers may face when evaluating web applications. Browser compatibility issues can occur when a website does not display or function properly across different browsers like Chrome, Firefox, Safari, and Internet Explorer. Testing performance across various devices with different operating systems and hardware configurations, such as desktop computers, laptops, tablets, and

mobile phones, is also difficult but important to ensure optimal user experiences. Additionally, security vulnerabilities must be addressed. For instance, SQL injection allows attackers to interfere with database queries through a web page. Cross-site scripting enables malicious code injection into otherwise trusted websites. Another challenge is confirming that a website's design and content are easily readable and usable on various screen sizes from large desktop monitors to small mobile screens. Addressing these compatibility.

3.3 Mobile Application Testing Techniques

Testing mobile apps is a crucial part of the development process. It allows developers to ensure the apps function as intended, load swiftly, and offer an intuitive experience for all users regardless of the device or operating system. Due to the specialized nature of mobile apps, there are multiple approaches developers can take to evaluate their performance. For instance, they may examine how apps appear and operate on the diverse screens, hardware, and software found on phones and tablets from various manufacturers. Testing across a wide range of real products helps identify bugs or inconsistencies before public release. It is also important to assess an app's speed and responsiveness under different conditions, such as on slower mobile connections or after periods of inactivity. Since people frequently multitask on their devices, ensuring compatibility across different scenarios is crucial.

Compatibility Testing is an important process that software developers undertake. It involves rigorously checking if an application functions smoothly across various devices, screen sizes, and operating systems. Developers also examine how the app performs under different network conditions. Through this testing, they can make certain that the software works as intended regardless of the hardware or software configuration of the user. This helps ensure a seamless experience for anyone wanting to utilize the app on their smartphone, tablet, or other device. By implementing Compatibility Testing, issues are identified and addressed before general release. This means more people can benefit from bug-free usage of the application on their chosen platform. The end result is improved usability and a wider reach for the software (Koziokas, Tselikas, Tselikis 2017).

Another important testing method is Performance Testing. It evaluates how swiftly the app responds, how stable it remains, and the amount of system resources it utilizes when faced with various scenarios and loads. Conducting this test can help uncover potential problems, optimize the app's speed, and ensure it fulfills users' requirements for quickness and dependability as they interact with it (Berihun, Dongmo, Van Der Poll 2023). While performance testing is crucial, it is also vital to maintain a balanced approach between testing methods to achieve quality without overburdening resources.

Usability testing is also a crucial part of the development process. It allows developers to examine the app's design, layout of menus, and the overall user experience to determine how intuitive and user-friendly the interface is. During these tests, people are observed as they attempt to complete typical tasks within the app. This provides valuable insights for developers to understand how real people interact with and navigate the app. It helps identify where improvements may be needed to streamline the user workflow and make the app more pleasant and enjoyable to use. The goal is to enhance user satisfaction by addressing any pain points or areas that cause confusion or frustration (Koziokas, Tselikas, Tselikis 2017).

Ensuring mobile application security is absolutely crucial for protecting users. Security testing serves a vital role by identifying potential vulnerabilities within an app that could place personal data in jeopardy or expose the software to various threats. Testers carefully examine aspects such as encryption protocols, login procedures, how information is securely retained on devices and servers, and defending against common hacking attempts. By investigating these technical elements and functionality through a security lens, weaknesses can be found and addressed before any harm occurs. This process helps strengthen an app's defenses over time so users can download and utilize features with confidence, safe in the knowledge that their privacy and well-being are not at risk. As new risks emerge, continued evaluation through testing also helps maintain protection as threats evolve (Haller, Klaus, 2013).

3.3.1 Key Differences Between Web and Mobile Application Testing

While there are notable variances between evaluating web and mobile applications, both aim to ensure high-quality user experiences. Aspects like functionality, ease of use, and selected testing strategies differ substantially when considering websites designed for desktop browsers versus smartphone or tablet apps. Functional testing looks at all features and checks if they are performing as intended across different environments. Usability testing evaluates how simple or complicated various tasks are to complete within an application (Reichert 2023).

Here are the key distinctions highlighted from the search results:

1. Platform and Accessibility:

Web applications are designed to be accessed through web browsers on various devices like desktop computers, laptops, and even some smart TVs. These applications can be reached using any modern web browser without requiring downloads or installations. Mobile applications, on the other hand, are specifically tailored for smaller screens and touch-based interactions found on mobile devices like smartphones and tablets. They are built to take advantage of the unique features that these mobile devices offer, such as GPS, cameras, and motion sensors (Unadkat 2021).

Mobile app testing differs from web app testing in that it requires testing applications on various mobile operating systems like iOS and Android. Developers need to ensure their apps function seamlessly across different devices and screen sizes. Web app testing, on the other hand, primarily focuses on evaluating how a website appears and performs on multiple web browsers. Since web browsers have standardized rendering engines, testing tends to be less complex than with native mobile apps. However, both mobile and web application testing are important to identify bugs and optimize the user experience across platforms (Unadkat 2021).

2. User Interface:

Mobile applications are specifically created to be used with touch-based interactions on devices like smartphones and tablets, as touchscreens are the primary methods of input. Websites and web applications, on the other hand, are generally constructed with mouse and keyboard control in mind since most people access the

internet through desktop computers. The user interface and navigation of mobile apps are optimized (Unadkat 2021).

Mobile app testing concentrates on confirming an intuitive interface that responds properly to touch motions, while web app testing emphasizes simplicity of movement with mouse and keyboard commands. Both types of testing are crucial to delivering programs that function seamlessly across platforms. Evaluating a mobile app requires validating that taps, swipes, and pinches perform as anticipated, just as assessing a web app involves validating clicking, scrolling, and typing perform as expected. UX testing is also important to evaluate for both formats. The goal is providing users with applications that work how they want without confusion or frustration (Unadkat 2021).

3. Performance:

Mobile devices have constrained processing capabilities in comparison to desktop computers, necessitating the optimization of performance in mobile application testing, with considerations given to elements like battery usage and network connectivity. It is important for testers to keep in mind the more limited power supply and connection speeds when developing for smartphones and tablets. Performance must be enhanced, and resource expenditure reduced so apps can run smoothly despite hardware restrictions inherent to portable devices (Yogiti 2023).

Web app testing considers more than just functionality and bugs. It also examines an application's performance across various web browsers like Chrome, Firefox, Safari, and Internet Explorer. Testers evaluate aspects such as how quickly pages load, elements render and respond to user input on different devices and operating systems. This helps ensure a smooth and fast experience for customers no matter which browser they choose (Unadkat 2021).

4. Connectivity:

Mobile devices rely on an array of network connections like 3G, 4G, and Wi-Fi to access the internet, with speeds varying significantly across these technologies. Because of this variability in connectivity levels, it is important for mobile applications to function smoothly regardless of the available network. Developers need to test their

apps under different network conditions to ensure a seamless user experience whether users are on a fast Wi-Fi connection or a slower mobile network (Yogiti 2023).

Web app testing not only examines network connectivity but mainly concentrates on how quickly pages load and how applications perform under slow internet conditions. Testing aims to ensure the application functions reliably even when network speeds are less than optimal. Developers subject their programs to different bandwidth limitations to check the user experience at various connection speeds (Yogiti 2023).

5. Device-Specific Features:

Mobile devices offer capabilities that set them apart from traditional web applications on computers and laptops. Features such as built-in cameras, GPS sensors, and accelerometers allow mobile apps to provide location-based services, augmented reality experiences, and more. Due to these distinctive characteristics, it is important for testers to conduct targeted testing on mobile specifically to validate that apps perform as expected when utilizing these device-level technologies. Simply testing the app's functionality through a browser will not adequately verify. Web app testing does not need to consider these device-specific features present in mobile devices (Unadkat 2021).

In wrapping up, there exist likenesses in the overall technique applied to evaluating both web and mobile applications. However, the unique variances in platform, user interface, performance, connectivity, and device-specific capabilities demand customized testing tactics for every single to confirm ideal functionality and user experience. The web and mobile environments have their own set of characteristics that require focusing testing on the particular attributes of each. While some tests can overlap between the two, ensuring that tests target the specific user workflows and hardware/software configurations for each type of application is important. A one-size-fits-all approach will not adequately verify that the application operates as intended across the assorted settings encountered on different devices and internet connections.

3.3.2 Best Practices in Mobile Application Testing

Mobile application testing plays an important role in the development process. It is essential to thoroughly evaluate the functionality and usability of an app on mobile devices before releasing it to users. This ensures any issues are identified and resolved. Through rigorous testing, developers can verify all features work as intended across various phones and tablets. It also allows them to identify ways to streamline workflows and simplify complicated processes. A seamless experience is key to an app's success.

1. Mimic Real-Life Situations:

Test apps in realistic conditions. Deal with bad networks, different time zones, and GPS points. What if the battery is low, or an SMS pops up? Testing these helps your app run smoothly no matter what (Bharati, 2022).

2. Choose the Right Testing Device:

Select the best device for app checks. Look at what's popular with your audience, screen sizes, and operating systems. When you test on the appropriate gadgets, it helps all users (Kumari, 2020).

3. Get to Know Your Users:

Collect data. Know your audience. Understand what they want and how they will use the app. This knowledge guides app development and improves the user experience (Solutions, 2023).

4. Function First, Experience Second:

Check that your app does what it's supposed to do. That's priority number one. Then, see if it's user-friendly. Test how usable it is in the early stages (Kumari, 2020).

5. Test on a Real Device Before Launch:

Initial tests can be done on emulators or simulators. But, make sure to do a final review on a real device. This helps find any last-minute issues. You can check everything thoroughly (Solutions, 2023).

6. Do Performance Tests Soon:

Spotting performance problems at the start of development is key to prevent expensive changes later. By doing performance tests early, it's easier to find and fix performance issues (Ville-Veikko 2013).

7. Make Testing Automated:

Using automation tools can boost testing productivity. They quicken up duplicate tests and give steady outcomes. Balancing both automated and manual testing is vital to tackle all situations (Llp 2023).

This set of best practices gives a full-picture approach to testing mobile apps, from real-time situation checks to usability and performance assessments. By sticking to these tips, developers can improve their mobile apps and give users a top-notch experience.

3.4 Selenium for Web Application Testing

Selenium is an open-source tool that's widely used for automating web app tests. It's a toolbox that includes the Selenium IDE, Selenium RC, and Selenium WebDriver. These can test web apps across different browsers, systems, and languages (Singh 2015). Because of extensive research, Selenium is seen as a cost-effective, efficient option for testing web apps (Gjesr 2015).

Our review of the research on Selenium versus manual testing shows Selenium's clear benefits. Cost is reduced by automating repeat tasks. Quality of software gets a boost from consistent, exact results. This research also discusses Selenium's key features like recording and playing back tests and the Selenium RC and WebDriver for those with programming know-how (Singh 2015).

The review even considers case studies and compares Selenium with other testing tools like UFT. This is intended to guide organizations in deciding the best testing approach. Factors considered include budget, ability to reuse, language and application support, and efficiency (Gjesr 2015).

In conclusion, Selenium is an impressive web application testing tool. It offers cost-effectiveness, consistency, and efficiency. Summarizing the literature offers insights into Selenium’s features and benefits as well as comparisons to help organizations make informed choices about their testing strategies (Gjesr 2015).

3.4.1 Introduction to Selenium Automation Tool

Selenium is a widely popular open-source automation testing framework that is commonly used for automating web applications. It provides a full set of tools that enable automated testing across different browsers and platforms, increasing its flexibility and ability to integrate with diverse development environments (Thooriqoh, 2021). Selenium allows testers to write automated tests in various programming languages, reducing the time spent on manual testing. The tests can validate functionality, measure performance, and ensure apps work across various browsers. With its cross-browser compatibility, companies are able to deliver quality software more quickly. While some see it as only for functional testing, many also leverage it for other quality assurance tasks like smoke testing, integration testing, and more (Thooriqoh, 2021).

One of the key abilities of Selenium is its power to engage with web components on a web page, permitting activities like tapping catches, entering content into fields, and approving anticipated results. This connection is made conceivable through Selenium’s WebDriver, filling in as a basic intermediary between the test content and the program, guaranteeing smooth correspondence and oversight over the web application being tried. The WebDriver works as a translator between the testing code and the program, enabling orders to be sent and reactions to be gotten. It guarantees the test can effortlessly control highlights on the webpage, for example, clicking joins or stacking pages, and validate the webpage acts as anticipated. This allows testers to deliberately explore the application and confirm it works as planned (Thooriqoh, 2021).

Testing Approaches Supported by Selenium:

Selenium allows testers to use various testing techniques like functional testing, regression testing, and browser compatibility testing. It enables running test scripts

simultaneously across different browsers to confirm consistent behavior regardless of the browser environment. This helps validate that the application performs as expected no matter if users access it with Chrome, Firefox, Safari, or another supported browser. By empowering cross-browser testing, Selenium helps developers identify and fix any issues that may affect users depending on which browser they use to access the site or app. This capability is important for catching compatibility problems that could impact the experience for some visitors (Thooriqoh, 2021).

Integration with Continuous Integration (CI) Tools:

Selenium works effortlessly with Continuous Integration tools such as Jenkins, allowing for automated testing to be a fundamental part of the software development process. This integration streamlines testing by providing swift responses to code modifications and maintaining the application's quality throughout each stage of its lifespan. By blending automated checks into the software progress, issues can be recognized rapidly so they may be addressed without delay. Bugs and errors are exposed very early before they deteriorate into bigger troubles, saving valuable time and resources. Overall, the combination of Selenium and integration tools like Jenkins results in a smoother development workflow with testing embedded into the workflow from the very beginning (Thooriqoh, 2021).

In wrapping up, Selenium stands apart as a powerful automation testing tool owing to its wide-ranging functions, flexibility in scripting dialects, and smooth integration with CI instruments. Its constant evolution and prevalent acceptance throughout the business underscore its importance in guaranteeing efficient and high-quality web application testing methods. Selenium's extensive set of capabilities like browser control, element location, JavaScript execution, and cross-browser compatibility allows testers to automate both front-end and back-end tests. Its support for various programming languages lets developers pick the language of their choice to write easy to read and maintain test scripts. This compatibility with multiple languages combined with the ability to integrate seamlessly with Continuous Integration pipelines facilitates streamlined testing workflows. As more companies recognize the value of test automation in deploying applications faster and the advantages of using open source software, the Selenium project community continues to grow in order to meet the changing needs of users (Thooriqoh, 2021).

3.4.2 Advantages and Limitations of Selenium in Web Testing

Your literature review provides a comprehensive overview of both the advantages and limitations of using Selenium for web testing. The segmentation into advantages and limitations, along with summaries for each, makes the content clear and organized. The inclusion of specific sources adds credibility to the information presented. Here's the structured summary:

Advantages of Selenium:

1. **Versatility:** Selenium supports various programming languages, allowing testers to write automated scripts in the language they are most familiar with (Li 2024).
2. **Cross-browser testing:** Selenium is known for enabling automated tests that can be run against multiple browsers simultaneously, ensuring broad compatibility and a seamless user experience (Li 2024).
3. **Cost-Effective:** Being an open-source tool, Selenium is cost-effective for testing web applications, as there are no licensing fees (Li 2024).
4. **Integration Capabilities:** Selenium integrates seamlessly with Continuous Integration (CI) tools like Jenkins (Li 2024).
5. **Community Support:** Selenium benefits from an active community of developers and users, providing assistance on forums and messaging boards (Li 2024).

Limitations of Selenium:

1. **High Test Maintenance:** Selenium tests can become fragile due to strict element identifiers, leading to high maintenance requirements (Reddy 2022).
2. **Steep Learning Curve:** Mastering Selenium requires a steep learning curve, demanding considerable time spent developing coding abilities (Reddy 2022).

3. **Limited Reporting Capabilities:** Selenium lacks certain features for generating detailed reports on test runs and results (Reddy 2022).

4. **Lack of Reliable Technical Support:** Selenium lacks reliable technical support, and users must seek solutions through online documentation and communities (Reddy 2022).

5. **Total Cost of Ownership:** While Selenium itself is open-source, the overall cost can be high due to factors like maintaining tests, fixing bugs, scaling the framework, and hiring skilled engineers (Reddy 2022).

3.5 Appium for Mobile Application Testing

Appium is a versatile and free automation tool specifically designed for mobile app testing (Verma 2017). Serving as a bridge between test scripts and mobile applications, Appium is compatible with various platforms, including real devices, simulators, and emulators. Its extensive reach spans multiple app platforms such as iOS, Android, and Tizen. Moreover, Appium is not limited to mobile platforms; it also extends its functionality to web browsers like Chrome, Firefox, and Safari. Additionally, it operates seamlessly on desktop environments like macOS and Windows, and even extends support to TV platforms such as Roku tvOS, Android TV, and Samsung. This wide compatibility makes Appium a powerful and flexible tool for comprehensive mobile app testing.

3.5.1 Introduction to Appium Framework

Appium, a prominent open-source automation testing tool, simplifies the automation of mobile apps across various platforms (Verma, 2017). Serving as a bridge, it connects test scripts with mobile apps running on real devices, simulators, or emulators. Appium’s versatility extends beyond mobile platforms, encompassing iOS, Android, Tizen, popular web browsers (Chrome, Firefox, Safari), desktop systems like macOS and Windows, and TV platforms including Roku tvOS, Android TV, and Samsung.

Since its inception in 2011 as "iOS Auto," Appium has evolved significantly, expanding its focus on UI testing across diverse platforms. Known for its user-friendly nature, Appium facilitates automation testing through a convenient CLI tool and seamless collaboration with third-party plugins, allowing easy installation of drivers and plugins from the Appium ecosystem (Verma, 2017).

Key Capabilities of Appium include:

1. **Working on Multiple Platforms:** Appium is adept at testing on various platforms, including iOS, Android, Tizen, and web browsers (Knott, 2015).

2. **Integration with Testing Frameworks:** It seamlessly integrates with popular testing frameworks like TestNG, JUnit, Pytest, and Cucumber, providing a well-established testing setup (Knott, 2015).

3. **Expansive Ecosystem:** Appium's adaptable architecture allows customization and the creation of new drivers for different platforms (Knott, 2015).

4. **Handling Native and Web Apps:** Appium excels in automating both web and native apps, offering a wide range of features for diverse testing scenarios (Knott, 2015).

Appium 2.0, the latest version, prioritizes agility and efficiency, aiming to simplify and expedite mobile testing. With a streamlined structure focusing on key testing aspects and enhanced features in the Appium Inspector tool, testers can emulate complex user tasks more effectively (Verma, 2017).

Appium operates seamlessly on both Android and iOS, leveraging the Mobile JSON Wire/W3C Protocol. This protocol translates test commands into REST API requests, which Appium client libraries use to communicate with connected devices or simulators (Verma, 2017).

In summary, Appium stands out in mobile application testing due to its versatility, compatibility across multiple platforms, strong community support, and

ongoing enhancements to meet evolving software testing needs in our digital landscape (A et al., 2020).

3.5.2 Advantages and Limitations of Appium

Advantages of Appium:

1. **Working on Many Platforms:** Appium demonstrates versatility by testing on various platforms, including iOS, Android, Tizen, and web browsers, ensuring excellent cross-platform compatibility (K, 2023).

2. **Works with Test Frameworks:** Appium seamlessly integrates with popular test frameworks such as TestNG, JUnit, Pytest, and Cucumber, providing testers with a familiar testing environment (Johnson, 2024).

3. **Strong and Adaptable System:** Appium's flexible structure allows easy modifications and personalization. Users can create and share Appium drivers for new platforms, enhancing the framework's capabilities (K, 2023).

4. **Backs Native and Web Apps:** Appium excels in automating both native and web applications, equipped with a comprehensive set of features to handle various testing scenarios (Johnson, 2024).

Limitations of Appium:

1. **Difficult Setup:** Appium's client-server model makes the setup challenging, requiring programming skills and making automation with Appium more complex (K, 2023).

2. **Unstable Tests:** Appium may lack precision in tests at times, leading to inconsistent test results for the same setup (Johnson, 2024).

3. **Slow Speed:** The structure of Appium can slow down test run times due to delays in starting the server and executing actions (K, 2023).

4. **Issues Locating Elements:** Appium may face challenges in finding elements and automatically recognizing images, necessitating manual entry of element positions (Johnson, 2024).

5. **Limited Backing for Outdated Android Models:** Appium might fall short in supporting older Android versions, impacting test coverage on diverse devices (Johnson, 2024).

3.6 Comparative Analysis

The chapter provides a comprehensive analysis of the research covered in preceding chapters, aiming to create a holistic perspective. Conducting a comparative study, the literature is scrutinized to identify areas that require further investigation and underscore the importance of the conducted research. By evaluating key studies side by side, the analysis seeks to identify subjects deserving additional scrutiny and highlight the significance of the undertaken research effort. Despite the presence of insightful research on the topic, certain aspects remain incompletely understood. This analysis endeavours to bring attention to these gaps and emphasize the value of the conducted research.

3.6.1 Previous Studies

The thesis explores a variety of research experiments on software testing, web application testing techniques, mobile application testing strategies, and the use of Selenium and Appium for web and mobile testing, respectively. Here is a concise summary of the key findings from the research:

Web Application Testing Techniques:

The research delves into various testing methods in web application development, emphasizing the importance of selecting appropriate techniques based on specific requirements. It covers unit testing, integration testing, and performance testing, highlighting the need for a judicious combination of methods for optimal

results. The studies underscore the significance of diverse testing approaches to enhance product quality and provide users with a consistent and issue-free online experience.

Mobile Application Testing Techniques:

In the realm of mobile application testing, numerous studies address challenges unique to mobile apps, such as cross-platform compatibility, performance, usability, and privacy concerns. The research suggests practical techniques, including thorough testing on a diverse range of devices, defining performance benchmarks, conducting user tests, and scanning app code for security vulnerabilities. These strategies aim to ensure mobile apps meet user expectations and adhere to privacy standards.

Selenium for Web Application Testing:

Selenium, a popular open-source test automation tool for web applications, is extensively studied. The research evaluates both the advantages and limitations of using Selenium for testing websites. Selenium proves valuable in streamlining the validation process by allowing testers to automate scripts, replay common actions, and detect potential bugs. Best practices are outlined to optimize Selenium's capabilities, recognizing its effectiveness in enhancing the quality and efficiency of web application testing.

Appium for Mobile Application Testing:

Appium, an open-source tool for automated mobile app testing, is examined in various research efforts. The studies discuss the benefits of Appium, such as testing multiple operating systems with one test suite and reusing test cases across platforms. However, limitations are acknowledged, including challenges in testing specific app features and occasional bugs. Recommendations are provided for effective Appium implementation, emphasizing object identification and addressing device synchronization issues.

Overall Perspective:

The research offers valuable insights into software testing methodologies, with a focus on Selenium and Appium. While providing substantial knowledge, some gaps

remain, particularly in further exploring Appium's application to mobile testing and conducting comparative studies on different testing platforms. Addressing these areas through additional research could enhance the understanding of evaluation strategies in software testing. The review highlights the need for continued exploration and expansion of knowledge in the domains of Selenium and Appium, as well as a broader understanding of mobile app testing.

3.6.2 Identified Gaps

The literature review identifies several gaps and areas that require further exploration in the realm of software testing, particularly in web and mobile application testing. Here's a detailed breakdown:

Appium for Mobile Apps:

Observation: There is a noticeable imbalance in the available information, with more focus on Selenium and comparatively less on Appium.

Recommendation: The need for more studies and research efforts to comprehensively understand the optimal usage of Appium for mobile application testing.

Testing Tools Comparisons:

Observation: Existing studies lack in-depth comparisons between testing tools, specifically Selenium and Appium.

Recommendation: A call for more detailed comparisons that highlight the strengths, weaknesses, and best use cases of both Selenium and Appium in various testing scenarios.

Mobile Testing Advancements:

Observation: The current research landscape does not adequately address advancements in mobile technologies, such as AI-driven testing, IoT application strategies, and 5G performance testing.

Recommendation: Emphasizes the need for research to keep pace with evolving mobile technologies, exploring areas like AI-driven testing, strategies for IoT applications, and performance testing in 5G environments.

Security Testing in Mobile Apps:

Observation: Limited focus on security testing in mobile apps despite its crucial importance.

Recommendation: Encourages more studies to explore the best practices for ensuring security in mobile apps, identifying vulnerabilities, and implementing robust security measures to protect user data and privacy.

Overall Call for More Research:

Summary: The literature review underscores the necessity for additional research and real-world case studies to fill the identified gaps. The objective is to advance software testing, especially in the context of web and mobile applications, utilizing tools like Selenium and Appium.

3.7 Summary of Literature Reviewed

This comprehensive thesis delves into various aspects of software testing, with a particular focus on web and mobile application testing. As a summary, the following key points can be made:

Importance of Web and Mobile Testing:

- Emphasis on the significance of tailored testing approaches for web and mobile applications, considering their unique challenges and requirements.
- Recognition of the need for specific testing techniques, tools, and best practices to ensure effective testing of both web and mobile apps.

Web Application Testing Techniques:

- Overview of diverse web application testing techniques, including compatibility testing, performance testing, usability testing, and security testing.
- Acknowledgment of the importance of employing these techniques to enhance user experience and functionality in web applications.

Mobile Application Testing Techniques:

- Recognition of the complexities in mobile application testing, requiring optimization for different devices and platforms.
- Highlighting strategies to address challenges related to performance, usability, and security, ensuring the quality of mobile apps.

Selenium for Web and Mobile Apps:

- Examination of Selenium as a versatile tool for automated testing of web and mobile apps, compatible with various browsers and programming languages.
- Awareness of Selenium's limitations and insights into optimizing its usage for effective web and mobile app testing.

Appium for Web and Mobile Apps:

- Exploration of Appium's role in testing both web and mobile apps, emphasizing its compatibility with different platforms and support for native and web apps.
- Evaluation of the strengths and limitations of Appium, contributing to the improvement of the testing process.

Areas for Further Research:

- Identification of areas requiring more in-depth study, such as Appium's specific role in mobile app testing, detailed tool comparisons, emerging trends in mobile app testing, and security issues.
- Emphasis on the potential impact of further research in promoting enhanced software testing practices and ensuring the quality of web and mobile apps through effective testing methods.

4. Practical part

4.1 Implementation of Selenium for Web App Testing Tools

4.1.1 Setting Up Selenium for Web Application Testing

Selenium is an open-source framework used for automating web applications, the below step by step process guides us to setting up Selenium for web application testing, focusing on the Selenium WebDriver, which is the component of Selenium used for automating browser actions.

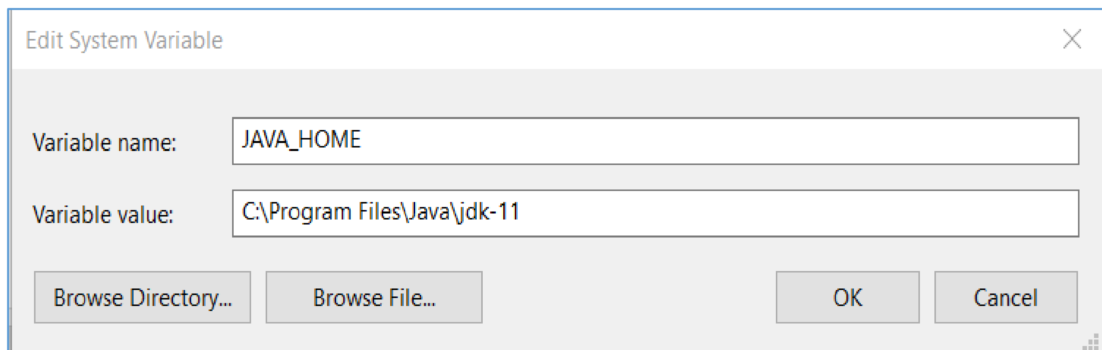


Figure 1: JDK Installation.
[**Source:** This thesis specific diagram was developed by the author.]

Step 1: Install Java Development Kit (JDK).

Selenium requires Java to run. Therefore, the first step is to ensure that the Java Development Kit (JDK) installed on our machine. The JDK allows us to develop and run Java programs, including Selenium tests.

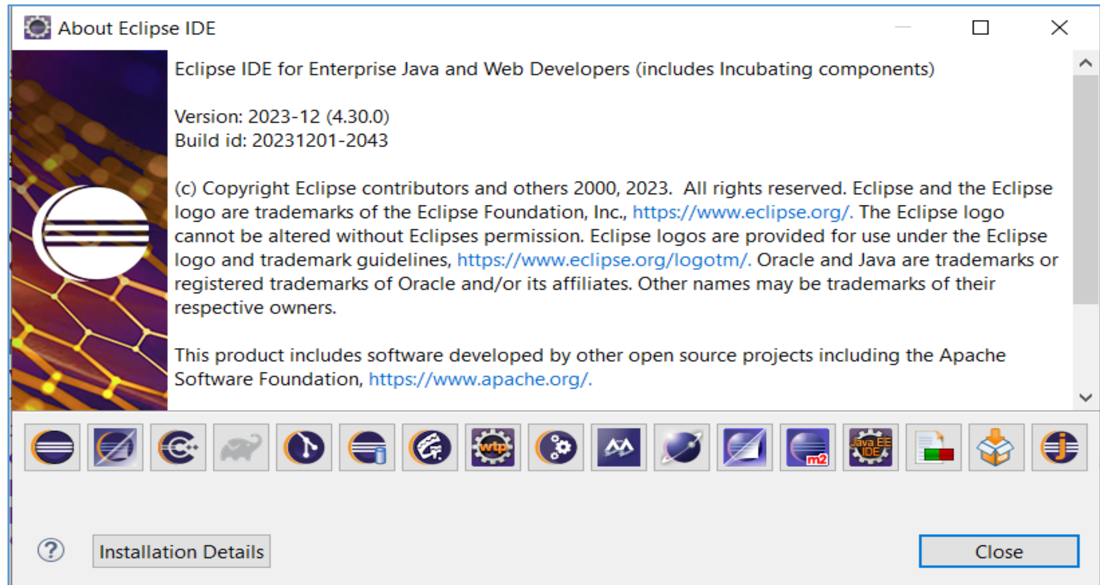


Figure 2: Eclipse IDE Installation.
[**Source:** This thesis specific diagram was developed by the author.]

Step 2: Download and Install an IDE

An Integrated Development Environment (IDE) provides a convenient interface for coding, debugging, and testing our Selenium scripts. Eclipse and IntelliJ IDEA are popular choices among Java developers.

Steps for Eclipse:

1. Create a new Maven project in Eclipse.
2. Enter the GroupID and ArtifactId, Click on Finish.
3. Double click on the pom.xml file and add the dependencies.

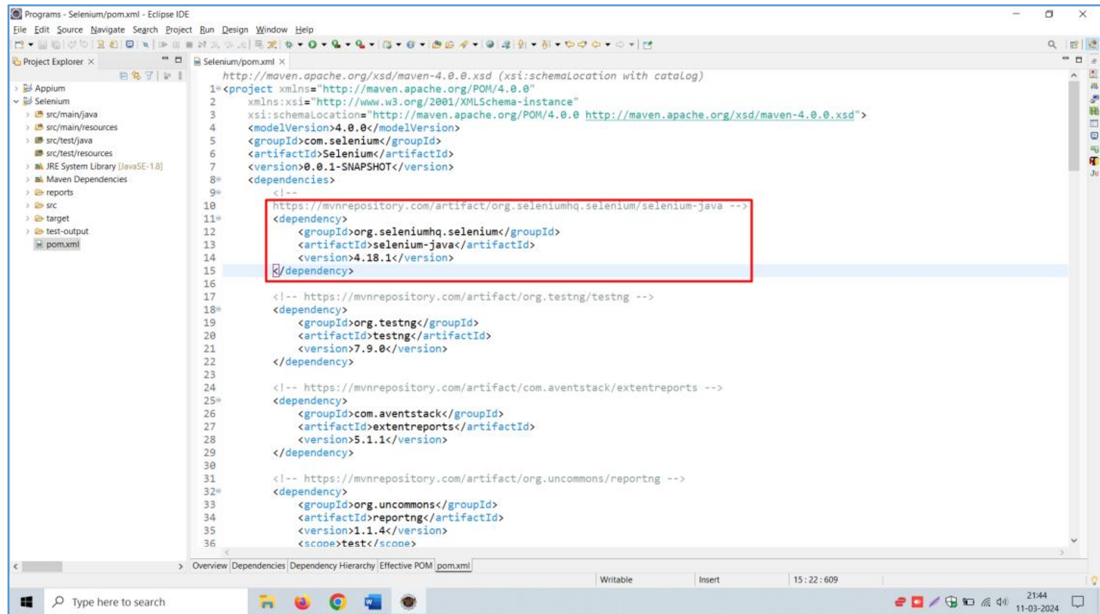


Figure 3: Selenium WebDriver Dependencies.
[**Source:** This thesis specific diagram was developed by the author.]

Step 3: Install Selenium WebDriver

Selenium WebDriver is a collection of language-specific bindings to drive a browser. We need to add it to our project as a dependency.

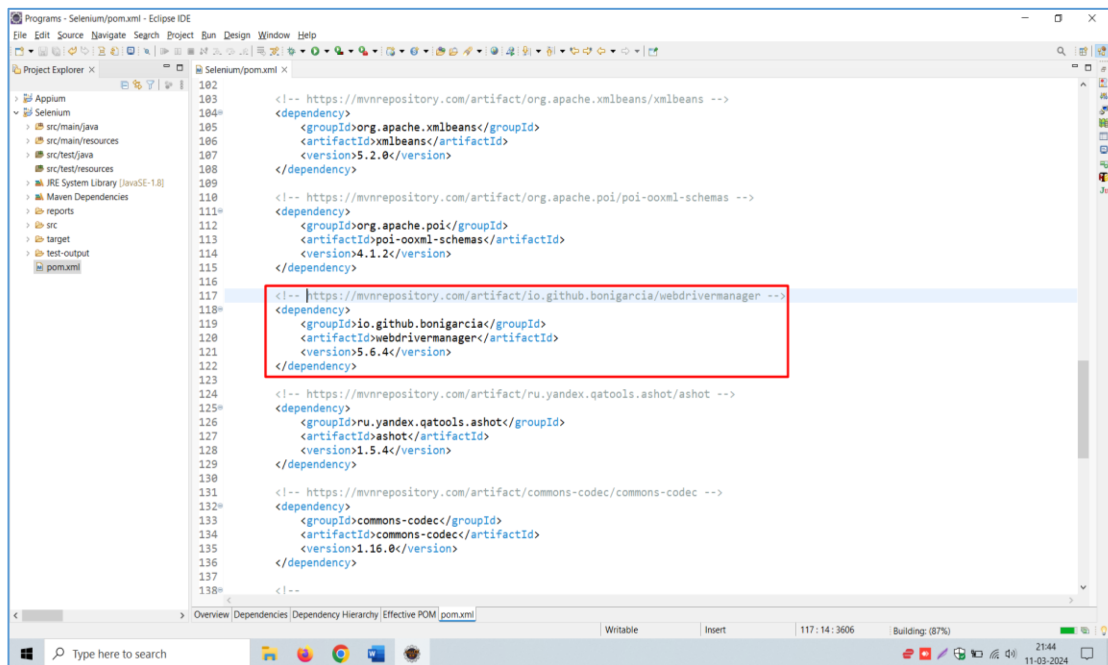


Figure 4: Browser Dependencies.
[**Source:** This thesis specific diagram was developed by the author.]

Step 4: Install Browser Driver

Selenium requires a driver to interface with the chosen browser. Chrome, Firefox, Safari, and Edge all have their drivers. We need to add it to our project as a dependency.

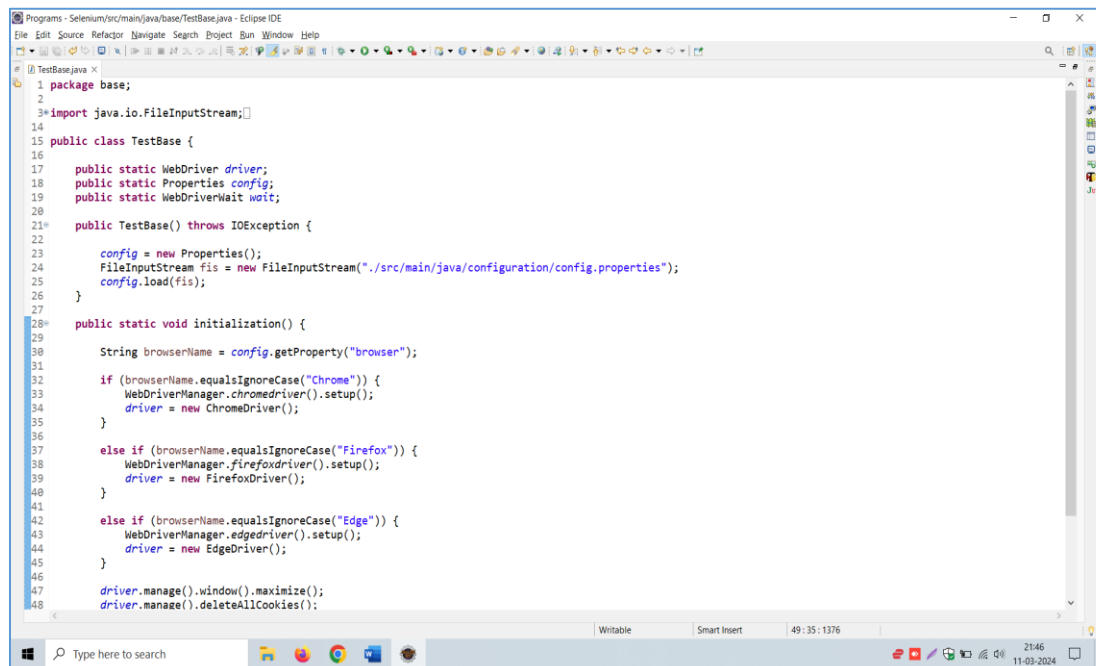


Figure 5: Selenium Script Example.
[**Source:** This thesis specific diagram was developed by the author.]

Step 5: Finally, we can write our first Selenium test script.

After setting up, we can write our first Selenium test. This test will open a web browser, navigate to a website.

4.1.1.1 Test Scenarios

To perform web application testing, a demo website known as “Sauce Labs” is used and programming scripts are developed using a java language that is compatible with Selenium, as part of a Selenium-based automation framework. These scripts utilize the Page Object Model (POM) design pattern for web applications which enhances better test maintenance and reduces code duplication, enabling them to communicate with web browsers and carry out automated testing tasks via the

Selenium WebDriver API. The scripts are structured within a Maven project setup and leverage the TestNG framework to control the testing process. Scripts are designed using TestNG, which is a sophisticated testing framework featuring enhanced annotations and organization of test methods. It also facilitates data-driven testing and integrates with Maven to handle dependencies and execute tests during the build cycle. Each of these classes is part of the larger test suite and contributes to a comprehensive automated testing strategy for a web application. They demonstrate a clear structure for testing different components of the application, ensuring that each part functions correctly both individually and as part of the overall user journey.

```
package testcases;

import java.io.IOException;

public class LoginPageTest extends TestBase {
    LoginPage loginPage;
    ProductPage productPage;

    public LoginPageTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException, InterruptedException {
        initialization();
        loginPage = new LoginPage();
    }

    @Test(priority = 1)
    public void invalidLoginTest3() throws IOException {
        productPage = loginPage.validateLogin("invalid@test.com", "invalid");
    }

    @Test(priority = 2)
    public void validLoginTest() throws IOException {
        productPage = loginPage.validateLogin(config.getProperty("username"), config.getProperty("password"));
    }

    @AfterMethod
    public void teardown() {
        driver.quit();
    }
}
```

Figure 6: LoginPageTest Selenium Script.
[**Source:** This thesis specific diagram was developed by the author.]

1. LoginPageTest.java:

This is a test class extending TestBase, which means it uses common setup and teardown methods for initializing and ending test cases. It contains two test methods: invalidLoginTest() and validLoginTest(). The first method tests the login functionality with invalid credentials, and the second tests it with valid credentials pulled from the config.properties file. The @BeforeMethod annotation indicates that the setup() method will run before each test method, initializing the browser and creating an

instance of the LoginPage. The @AfterMethod annotation indicates that the teardown() method will run after each test method, which in this case, quits the browser, effectively closing the testing session.

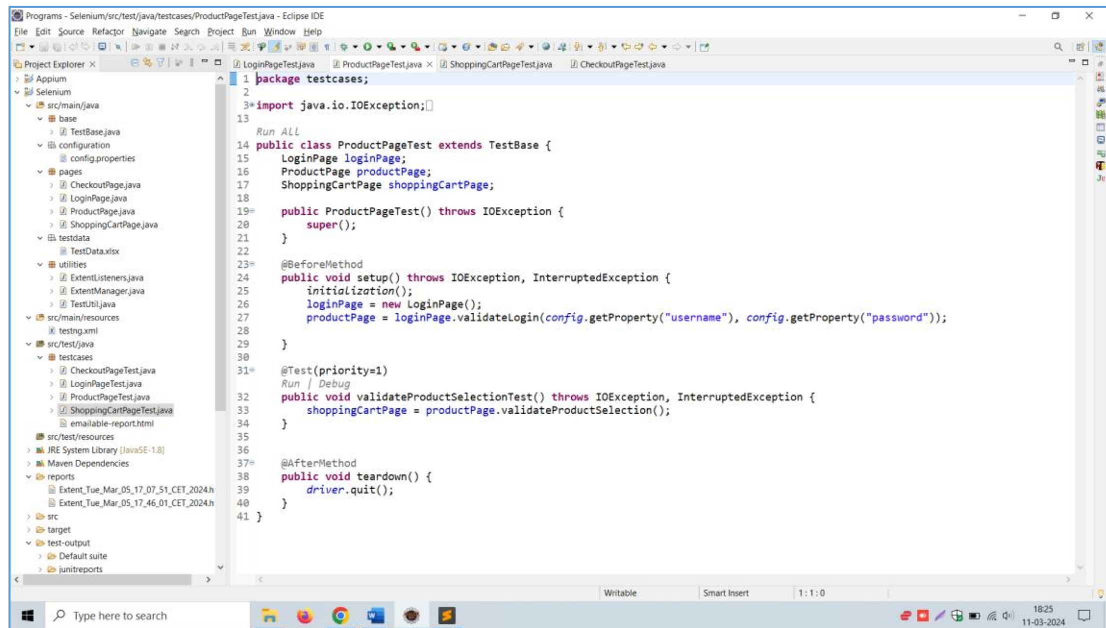


Figure 7: ProductPageTest Selenium Script.
[**Source:** This thesis specific diagram was developed by the author.]

2. ProductPageTest.java:

This class tests the product selection functionality on the product page. The setup() method again initializes the browser and logs into the application using valid credentials. The login is necessary because product selection requires an authenticated user. The validateProductSelectionTest() method test whether product selection is functioning correctly by using methods defined in the ProductPage class.

```
package testcases;

import java.io.IOException;

public class ShoppingCartPageTest extends TestBase {
    LoginPage loginPage;
    ProductPage productPage;
    ShoppingCartPage shoppingCartPage;
    CheckoutPage checkoutPage;

    public ShoppingCartPageTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException, InterruptedException {
        initialization();
        loginPage = new LoginPage();
        productPage = loginPage.validateLogin(config.getProperty("username"), config.getProperty("password"));
        shoppingCartPage = productPage.validateProductSelection();
    }

    @Test(priority=1)
    public void validateAddToCartTest() throws IOException, InterruptedException {
        checkoutPage = shoppingCartPage.validateShoppingCart();
    }

    @AfterMethod
    public void teardown() {
        driver.quit();
    }
}
```

Figure 8: ShoppingCartPageTest Selenium Script.
[**Source:** This thesis specific diagram was developed by the author.]

3. ShoppingCartPageTest.java:

Above Figure 8 is for testing the shopping cart page's functionality, such as adding products to the cart. The setup() method performs similar tasks as in the previous classes, setting up the test environment and ensuring the user is logged in and has selected a product. The validateAddToCartTest() method tests the addition of a product to the shopping cart.

```
package testcases;

import java.io.IOException;

public class CheckoutPageTest extends TestBase {
    LoginPage loginPage;
    ProductPage productPage;
    ShoppingCartPage shoppingCartPage;
    CheckoutPage checkoutPage;

    public CheckoutPageTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException, InterruptedException {
        initialization();
        loginPage = new LoginPage();
        productPage = loginPage.validateLogin(config.getProperty("username"), config.getProperty("password"));
        shoppingCartPage = productPage.validateProductSelection();
        checkoutPage = shoppingCartPage.validateShoppingCart();
    }

    @Test(priority = 1)
    public void validateSuccessfulOrderTest() throws IOException, InterruptedException {
        checkoutPage.validateOrder();
        checkoutPage.validateAddress("Test", "Test", "Test");
        productPage = checkoutPage.validateConfirm();
    }

    @AfterMethod
    public void teardown() {
        driver.quit();
    }
}
```

Figure 9: CheckoutPageTest Selenium Script.
[**Source:** This thesis specific diagram was developed by the author.]

4. CheckoutPageTest.java:

The above Figure 9 is to test the checkout process, including order validation and address confirmation. Like the other classes, it uses @BeforeMethod to set up the preconditions necessary for the checkout tests, such as being logged in, having products in the cart, and being on the checkout page. The validateSuccessfulOrderTest() method tests the entire flow of a successful order placement, including verifying order details and inputting address information.

4.1.1.2 TestCase Design

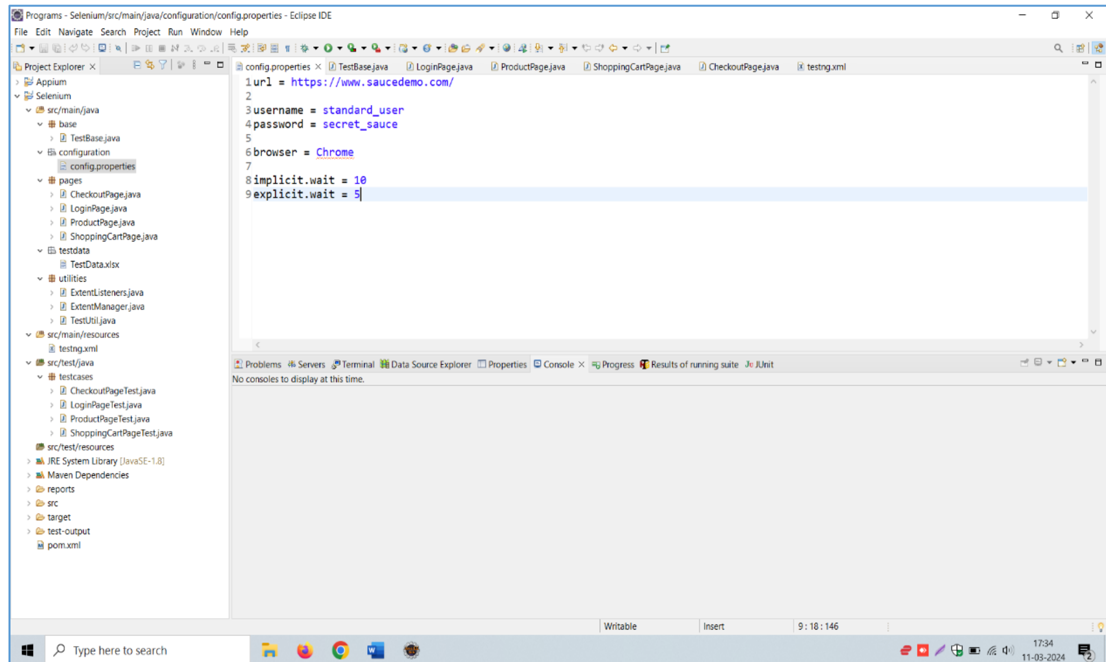


Figure 10: Configuration File.
[**Source:** This thesis specific diagram was developed by the author.]

1. config.properties file:

This file contains configuration properties for the test suite. It includes website url (domain name), credentials for logging into the web application (username and password), specifies which web browser should be used (browser), and sets timeouts for the Selenium WebDriver (implicit.wait and explicit.wait). These properties are read at runtime and used to configure the Selenium environment.

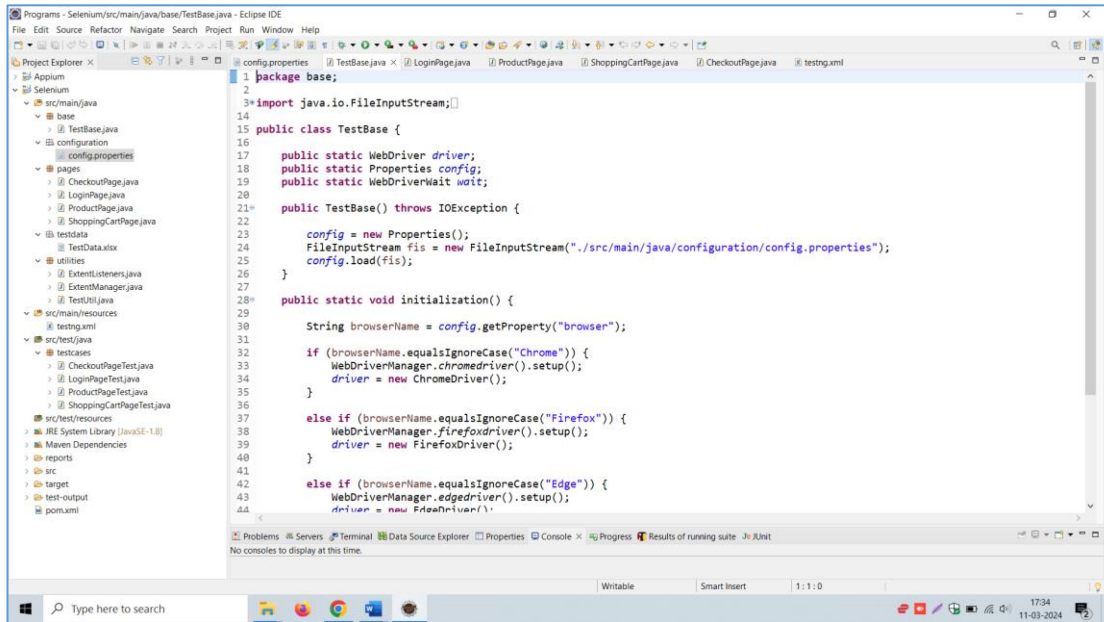


Figure 11: TestBase Selenium Script.
[**Source:** This thesis specific diagram was developed by the author.]

2. TestBase.java file:

Above Figure 11 class serves as the foundation for all the page-specific test classes. It initializes the WebDriver, which is used for browser automation, and the Properties object, which is used to load the configuration settings from the config.properties file. The initialization() configures the WebDriver based on the browser specified in the properties file.

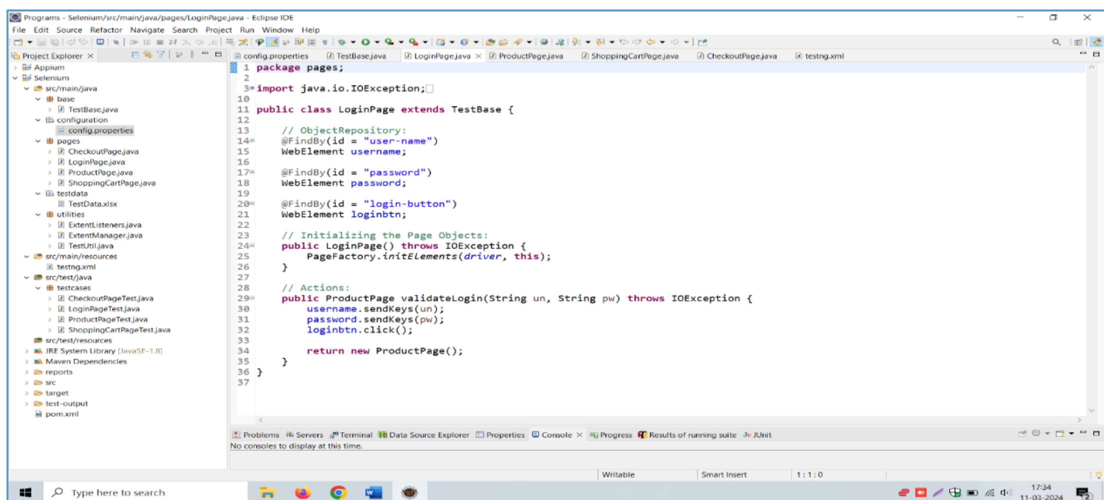


Figure 12: LoginPage Selenium Script.
[**Source:** This thesis specific diagram was developed by the author.]

3. LoginPage.java file:

The LoginPage class represents the login page of the web application. It uses Page Factory for initializing web elements. The @FindBy annotations are used to locate the username, password, and login button elements on the page. The validateLogin() takes a username and password, inputs them into the respective fields, and simulates a click on the login button, returning a new instance of the ProductPage.

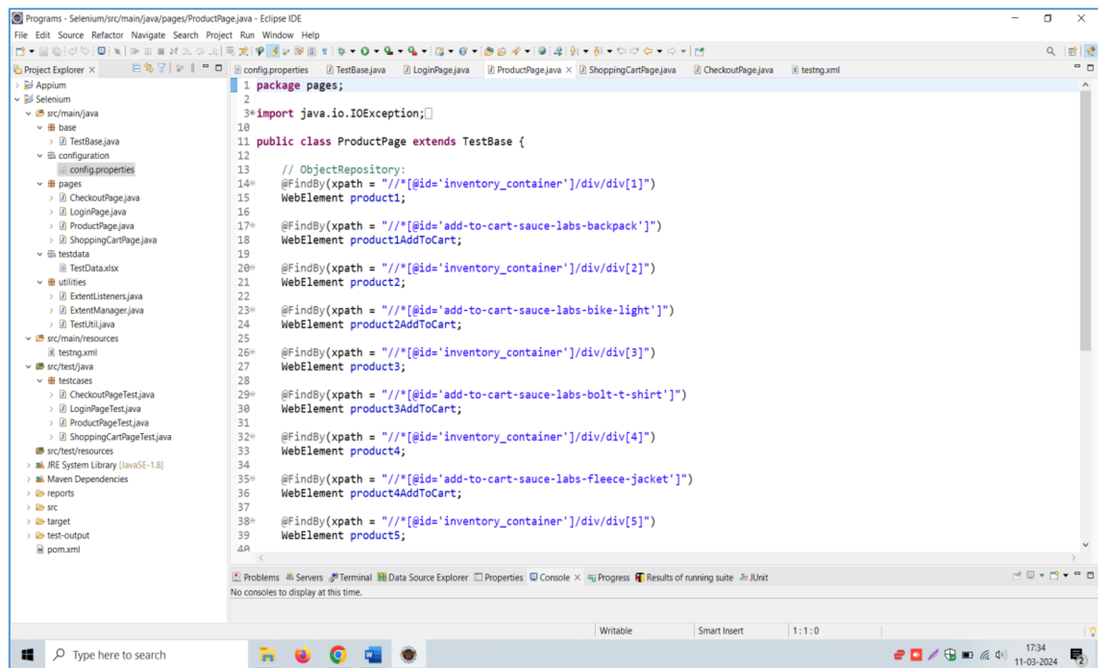


Figure 13: ProductPage Selenium Script.
[**Source:** This thesis specific diagram was developed by the author.]

4. ProductPage.java:

Above Figure 13 corresponds to the product page of the web application. Similar to the Login Page, it initializes elements like product listings and add-to-cart buttons using the @FindBy annotations. These web elements are identified by their XPath locators. Methods in this class are used to perform actions such as adding items to the shopping cart.

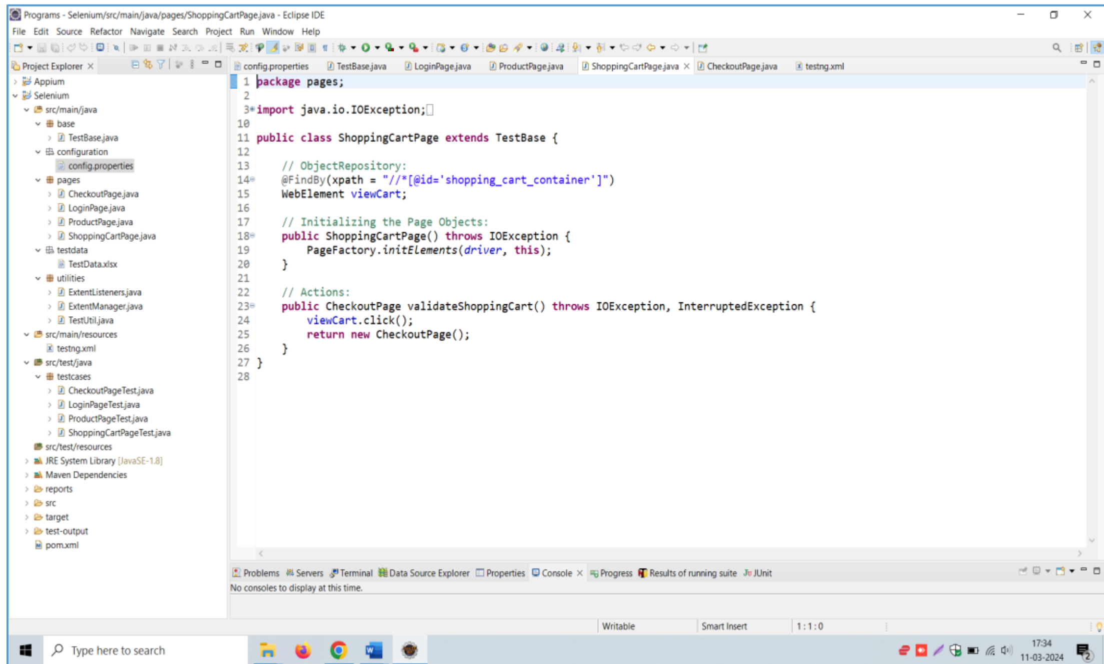


Figure 14: ShoppingCartPage Selenium Script.
[**Source:** This thesis specific diagram was developed by the author.]

5. ShoppingCartPage.java file:

The ShoppingCartPage class manages the shopping cart page functionality. It uses a @FindBy annotation to locate the shopping cart container. The validateShoppingCart(), when invoked, will interact with the cart (e.g., view the cart contents) and return an instance of the CheckoutPage.

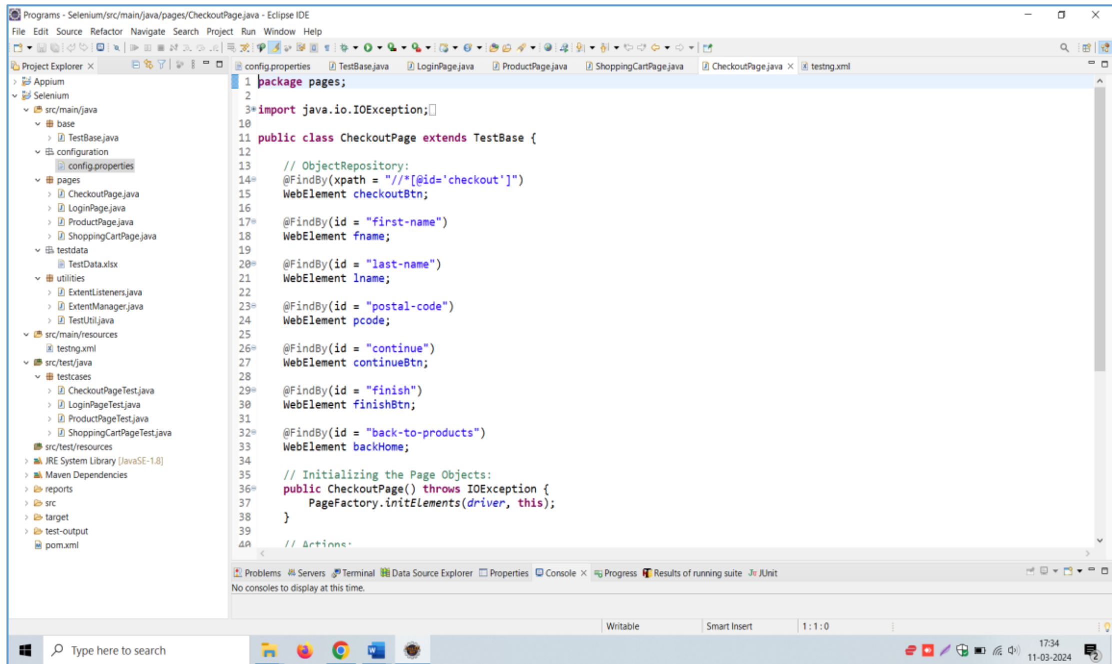


Figure 15: CheckoutPage Selenium Script.
[**Source:** This thesis specific diagram was developed by the author.]

6. CheckoutPage.java file:

Above Figure 15 depicts the checkout page. It has web elements for fields like first name, last name, and postal code, which are essential for completing the checkout process. Additionally, it contains buttons for continuing the checkout process, finishing the order, and returning to the product page, all of which are annotated with @FindBy.

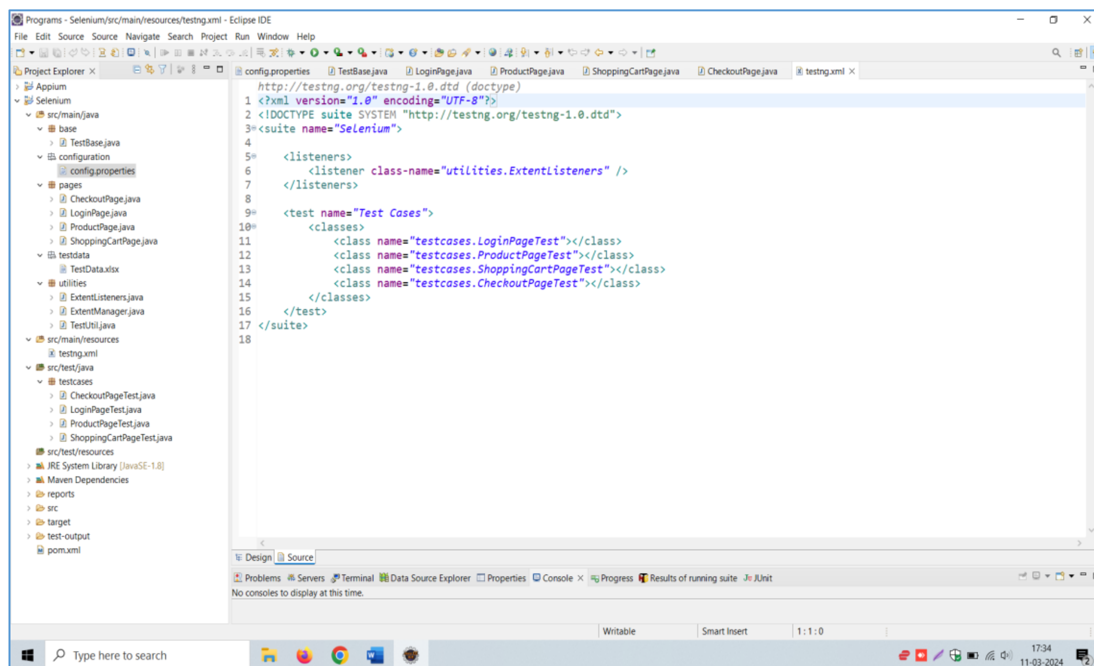


Figure 16: Selenium TestNG Xml file.
[**Source:** This thesis specific diagram was developed by the author.]

7. testng.xml file:

The testng.xml is a configuration file for the TestNG framework. It specifies which classes contain test cases that should be executed. The file includes references to listener classes that are used for reporting purpose and specifies the suite of tests to run. Each <class> element within the <test> tag corresponds to a test class that contains one or more test methods.

In Selenium automation framework, the above components work together to automate the testing of a web application. The config.properties file holds the environment setup. TestBase provides the common setup and teardown functionality for the tests. Each page class represents a page within the web application, encapsulating the elements and actions on that page. Finally, testng.xml is used to manage and run the test suite, leveraging TestNG’s capabilities for grouping, sequencing, and parallel execution of tests.

4.1.1.3 Results and Findings

1. Functionality Testing Findings:

Login Functionality:

The LoginPage class shows that functionality tests are written to validate user authentication. Test findings include whether correct credentials allow access and incorrect ones are denied.

Product Selection Functionality:

The ProductPage class indicates tests are performed for product selection and adding items to the cart. Functionality findings cover the accuracy of product details, the responsiveness of the add-to-cart action and updating of the cart count.

Shopping Cart Functionality:

Through the ShoppingCartPage class, functionality tests check the cart's ability to display selected items, update quantities, and remove items. The findings shows if the shopping cart correctly calculates totals and retains items upon session refresh or login/logout cycles.

Checkout Functionality:

CheckoutPage class shows that tests included form submission, input validation, and navigation to a successful order completion. Test findings focused on form validations, mandatory field checks, and the accuracy of the final summary before order placement.

2. Compatibility Testing Findings:

Browser Compatibility:

The TestBase class initializes different web browsers based on the property file configuration, suggesting that compatibility tests across Chrome, Firefox, and Edge were performed. Findings include how consistently the web application functions across these browsers.

3. Implicit and Explicit Waits:

The use of implicit and explicit waits convey testing for page load times and element availability across environments and network conditions. Compatibility findings include differences in load times and how well the application handles dynamic content or AJAX-loaded elements across browsers.

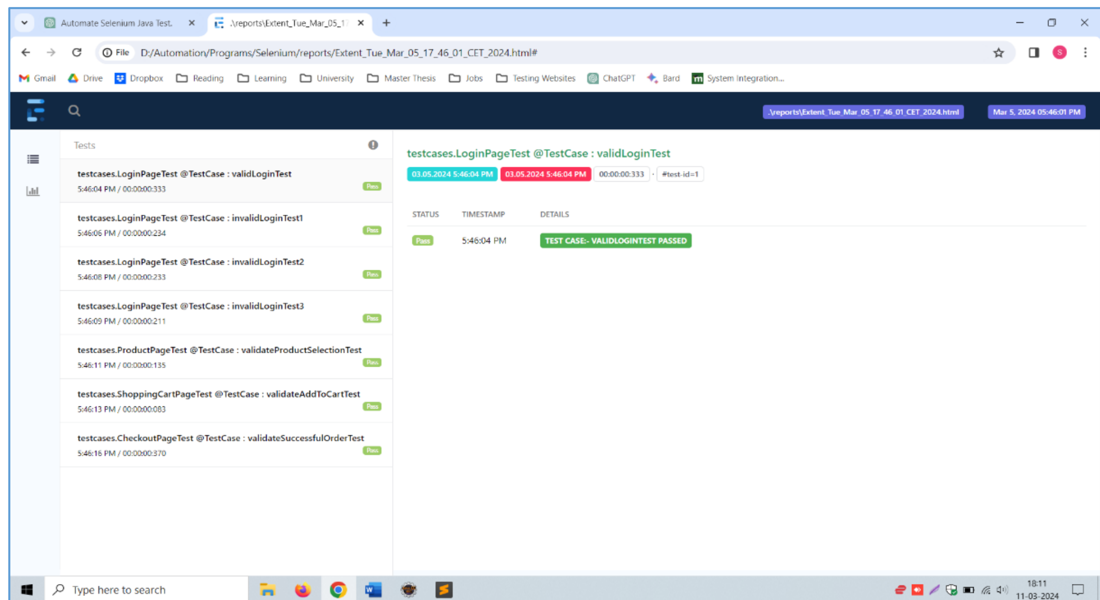


Figure 17: Selenium Extent Report.
[**Source:** This thesis specific diagram was developed by the author.]

4. Extent Test Report:

The above Figure 17 details the status of each test case executed. Tests from LoginPageTest are listed, including validLoginTest and invalidLoginTests, all of which passed. These tests validate both successful and unsuccessful login attempts. Other tests like validateProductSelectionTest, validateAddToCartTest, and validateSuccessfulOrderTest from ProductPageTest, ShoppingCartPageTest, and CheckoutPageTest, respectively, also passed. These test the functionality of product selection, view shoppingcart, and the checkout process. The timestamp and duration for each test are logged, indicating performance metrics and test efficiency.

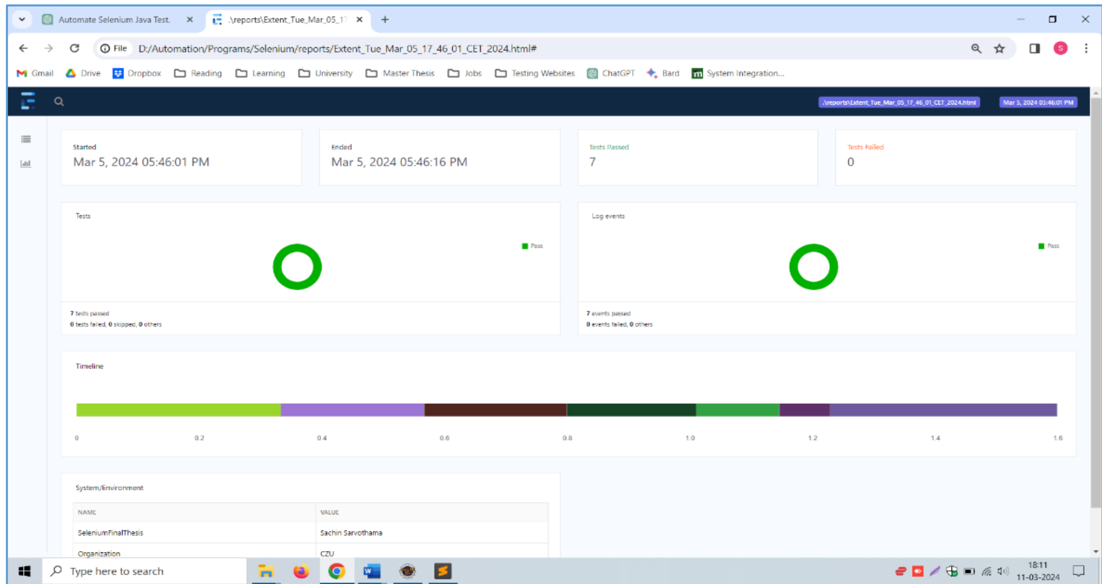


Figure 18: Selenium Extent Report Summary.
[**Source:** This thesis specific diagram was developed by the author.]

5. Summary of Extent Test Report:

The above Figure 18 summary snapshot depicts an overview of the test execution results. It explains that all 7 tests have passed, with a graphical representation (a large green circle) to quickly convey the success rate. Start and end times for the test suite are shown, along with a duration, which can be important for tracking how long the testing process takes.

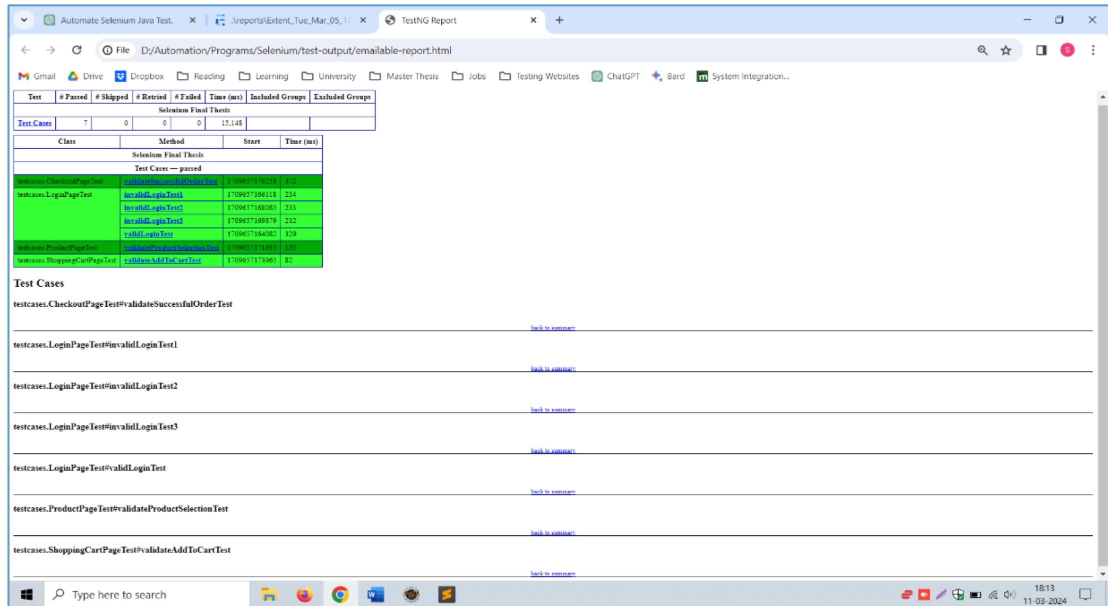


Figure 19: Selenium TestNG Report.
[**Source:** This thesis specific diagram was developed by the author.]

6. TestNG Report Summary:

The above Figure 19 is a TestNG generated report, provides a tabular view of the tests executed, including columns for the number of passed, skipped, and failed tests. Below, there are details of each test method, along with their start time and duration. The last section breaks down the test cases by their respective classes and methods, showing individual results, which in this case are all green, indicating a pass.

All the above Results and Findings, Extent Test Report, Extent Test Report Summary and TestNG Report Summary indicates that the test suite successfully executed without any failures. This suggests both functional correctness of the web application under test in different scenarios and compatibility across the different environments and browsers are covered by execution of the required test scripts. These reports are valuable for stakeholders to understand the robustness and reliability of the application, and for the development and QA teams to identify and resolve any issues early in the development cycle.

4.2 Implementation of Appium for Mobile App Testing Tools

4.2.1 Configuring Appium for Mobile App Testing

Appium, a trending tool in Mobile Automation Testing Technology, supports automated testing for native, hybrid, and web applications. Its capabilities extend to automation tests on simulators (iOS), emulators (Android), and physical devices (both Android and iOS). Here are the steps for configuring Appium for mobile app testing.

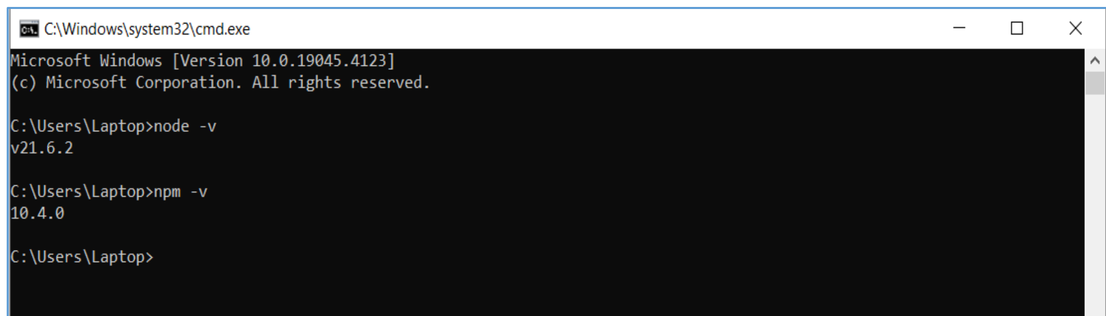


Figure 20: Node.js and NPM Installation.
[**Source:** This thesis specific diagram was developed by the author.]

Step 1: Install Node.js and NPM.

Appium is a server written in Node.js. Installing Node.js automatically comes with npm (Node Package Manager), which is required to install Appium.

1. Download the Node.js installer from the official Node.js website.
2. Run the installer and install both Node.js and npm.
3. Verify the installation by opening a command prompt and run node -v and npm -v. This displays the installed versions of Node.js and npm.

```
C:\Windows\system32\cmd.exe - "node"  "C:\Users\Laptop\AppData\Roaming\npm\\node_modules\appium\index.js"
Microsoft Windows [Version 10.0.19045.4123]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Laptop>appium
[Appium] Welcome to Appium v2.5.1
[Appium] The autodetected Appium home path: C:\Users\Laptop\.appium
[Appium] Attempting to load driver uiautomator2...
[Appium] Requiring driver at C:\Users\Laptop\.appium\node_modules\appium-uiautomator2-driver\build\index.js
[Appium] AndroidUiautomator2Driver has been successfully loaded in 2.208s
[Appium] Appium REST http interface listener started on http://0.0.0.0:4723
[Appium] You can provide the following URLs in your client code to connect to this server:
[Appium]        http://192.168.78.226:4723/
[Appium]        http://127.0.0.1:4723/ (only accessible from the same host)
[Appium] Available drivers:
[Appium]   - uiautomator2@3.0.1 (automationName 'UiAutomator2')
[Appium] No plugins have been installed. Use the "appium plugin" command to install the one(s) you want to use.
```

Figure 21: Appium Installation.
[**Source:** This thesis specific diagram was developed by the author.]

Step 2: Install Appium

After the Node.js and npm installed.

1. Open a command prompt.
2. Run the command npm install -g appium to install Appium.
3. Verify the installation with appium, which will start the Appium server.

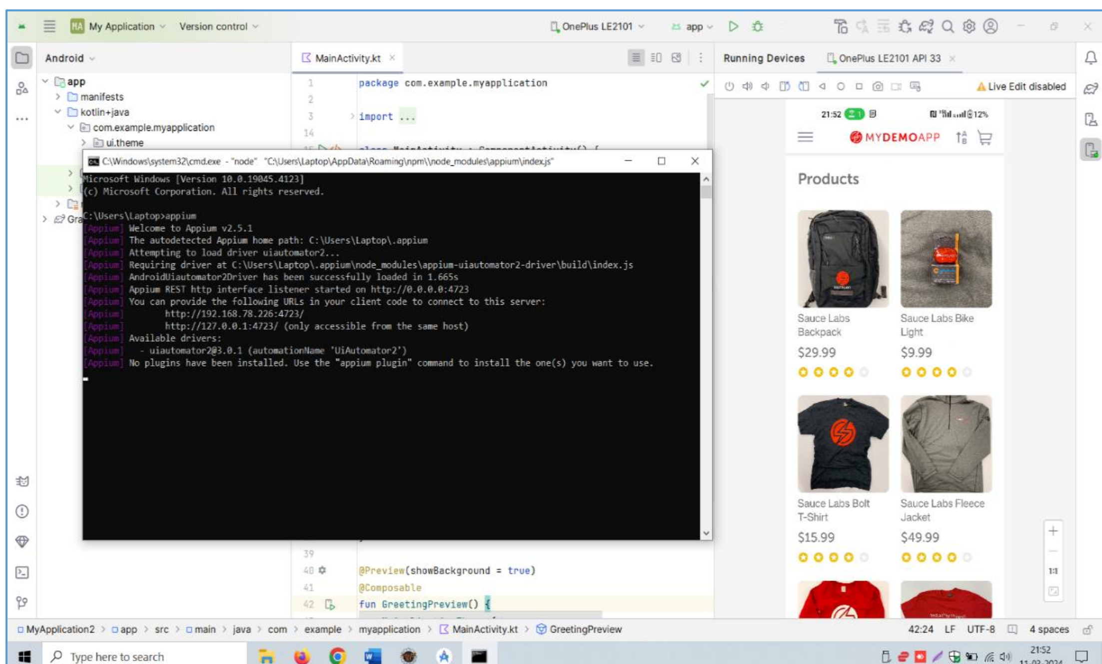


Figure 22: Android Studio Emulator.
[**Source:** This thesis specific diagram was developed by the author.]

Step 3: Install Android Studio (for Android Testing)

To test Android applications, we need the Android SDK and emulators, which are part of Android Studio.

1. Download Android Studio from the official Android Studio website.
2. Run the installer.
3. Set the ANDROID_HOME environment variable to your Android SDK location.
4. Add the Android SDK platform-tools directory to PATH variable.

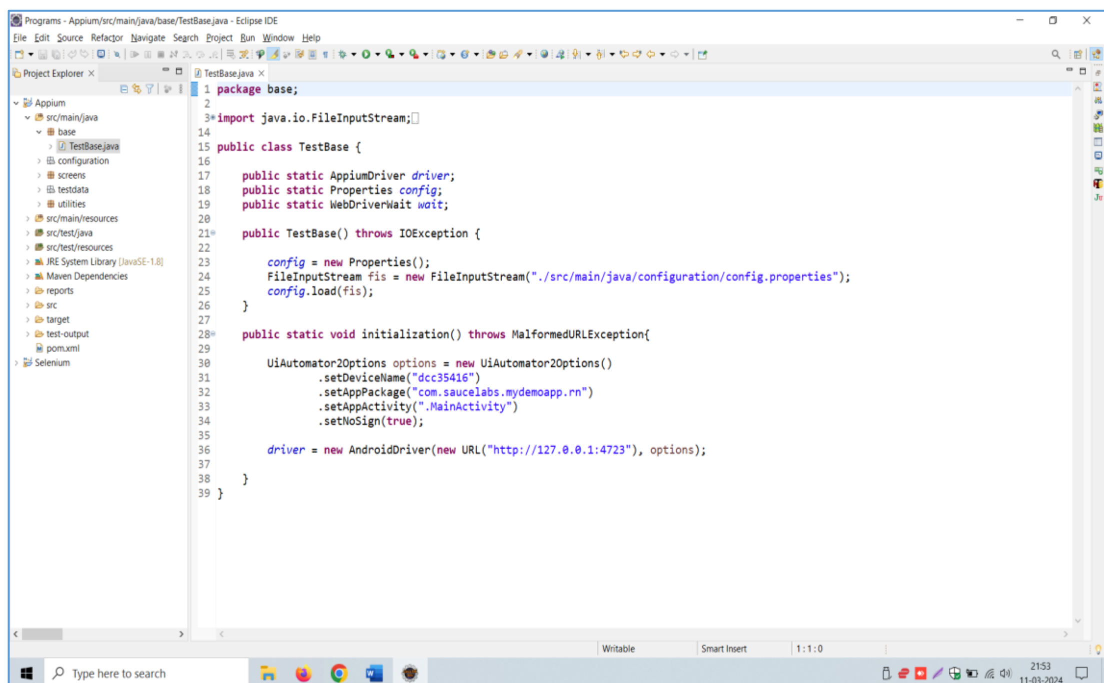


Figure 23: Appium Script Example.
[**Source:** This thesis specific diagram was developed by the author.]

Step 4: Writing and Running First Test

With everything set up, we can run our first mobile application test using Appium. We need to create a test script in a language supported by Appium, such as Java.

4.2.1.1 Test Scenarios

In the realm of mobile application testing, ensuring that an app functions correctly across a range of devices and operating systems is paramount. Appium, an open-source test automation framework, has become a cornerstone tool for testers worldwide due to its flexibility in supporting both Android and iOS platforms. A series of Java classes is designed for an Appium test suite to automate functionality and compatibility testing of a mobile application. The TestBase class acts as the foundation, setting up the Appium environment and ensuring tests run against the specified device configurations. This setup is crucial for compatibility testing as it defines the parameters for which devices and OS versions the tests will execute. Subsequent classes, such as LoginPageTest, ProductPageTest, ShoppingCartPageTest, and CheckoutPageTest, are structured following the Page Object Model (POM). This model enhances test maintenance and reduces code duplication by encapsulating the properties and behaviors of the application pages within dedicated classes.

Functionality testing is demonstrated through methods that mimic user interactions—logging in with both valid and invalid credentials, selecting products, adding items to a shopping cart, and executing the checkout process. Each test method is annotated to indicate its role and execution order within the suite, leveraging TestNG's powerful testing capabilities such as setup and teardown methods, and prioritization. The test suite's architecture is tailored to validate the application's behavior under test rigorously. The goal is to uncover any functional discrepancies and ensure the application's seamless operation across different devices, thereby affirming both its functional integrity and compatibility standing in a diverse mobile ecosystem. Through Appium, testers can automate these processes, thus speeding up the release cycles and ensuring a consistent user experience regardless of the end user's device.

Figure 24: LoginPageTest Appium Script.
[**Source:** This thesis specific diagram was developed by the author.]

1. LoginPageTest.java:

This class contains test methods for the login functionality. It extends TestBase, which means it inherits setup and teardown methods along with any other common functionality. The @BeforeMethod is used to set up preconditions for the tests, which includes initializing the LoginPage object. There are two test methods defined, invalidLoginTest() and validLoginTest(). The @AfterMethod is used for cleanup after tests are run, which in this case, involves quitting the driver, effectively ending the session.

```java
package testcases;

import java.io.IOException;

public class ProductPageTest extends TestBase {
    LoginPage loginPage;
    ProductPage productPage;
    ShoppingCartPage shoppingCartPage;

    public ProductPageTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException, InterruptedException {
        initialization();
        loginPage = new LoginPage();
        productPage = loginPage.validateLogin(config.getProperty("username"), config.getProperty("password"));
    }

    @Test(priority=1)
    public void validateProductSelectionTest() throws IOException, InterruptedException {
        shoppingCartPage = productPage.validateProductSelection();
    }

    @AfterMethod
    public void teardown() {
        driver.quit();
    }
}
```

Figure 25: ProductPageTest Appium Script.
[**Source:** This thesis specific diagram was developed by the author.]

2. ProductPageTest.java:

The above Figure 25 is used for testing interactions on a product page within the app. It also extends TestBase. The setup() method initializes the login page and logs in using valid credentials. This is a common pattern to ensure the test starts from a user-logged-in state. The validateProductSelectionTest() method test the functionality of selecting a product, verifying that the correct product page is loaded.

```java
package testcases;

import java.io.IOException;

public class ShoppingCartPageTest extends TestBase {
    LoginPage loginPage;
    ProductPage productPage;
    ShoppingCartPage shoppingCartPage;
    CheckoutPage checkoutPage;

    public ShoppingCartPageTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException, InterruptedException {
        initialization();
        loginPage = new LoginPage();
        productPage = loginPage.validateLogin(config.getProperty("username"), config.getProperty("password"));
        shoppingCartPage = productPage.validateProductSelection();
        shoppingCartPage = productPage.validateProductAdd();
    }

    @Test(priority=1)
    public void validateAddToCartTest() throws IOException, InterruptedException {
        checkoutPage = shoppingCartPage.validateShoppingCart();
    }

    @AfterMethod
    public void teardown() {
        driver.quit();
    }
}
```

Figure 26: ShoppingCartPageTest Appium Script.
[**Source:** This thesis specific diagram was developed by the author.]

3. ShoppingCartPageTest.java:

This class tests the shopping cart page's functionalities. The setup() method ensures the user is logged in and has a product selected before testing the cart. The validateAddToCartTest() method check if adding a product to the cart works correctly. The validateShoppingCart() method tests the items displayed in the cart.

Figure 27: CheckoutPageTest Appium Script.
[**Source:** This thesis specific diagram was developed by the author.]

4. CheckoutPageTest.java:

The above Figure 27 class is for testing the checkout process of the application. The setup() method logs in the user, selecting a product, and adding it to the cart. The validateSuccessfulOrderTest() method goes through the full process of ordering, including entering address details and confirming the order. As with the other test classes, @AfterMethod is used for post-test cleanup.

4.2.1.2 TestCase Design

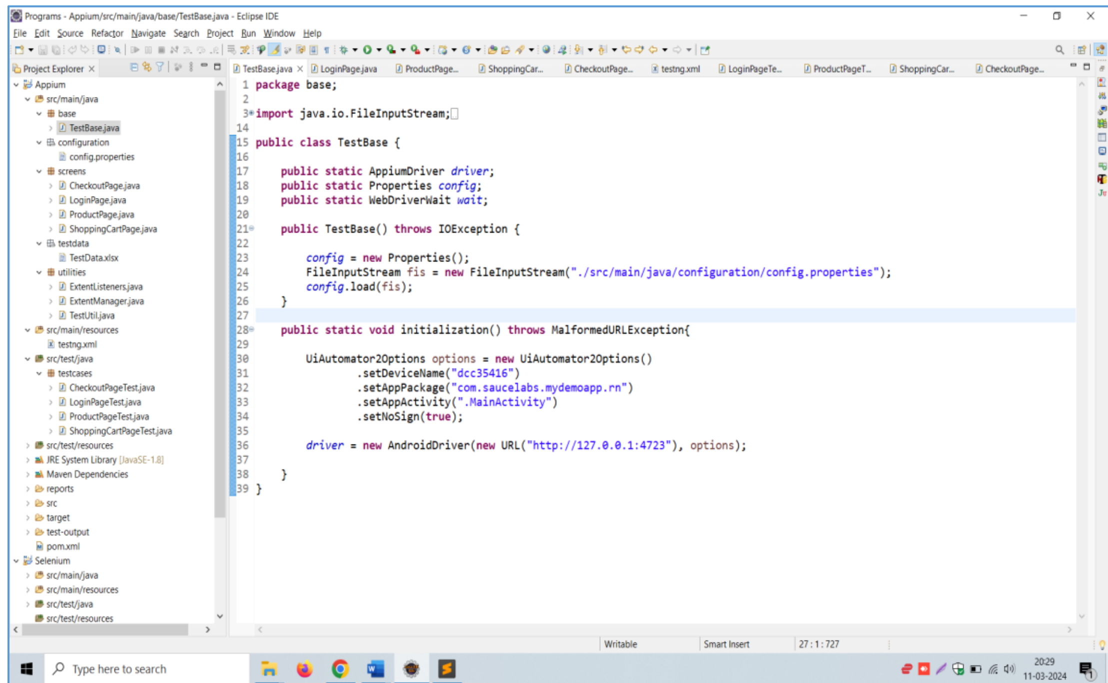


Figure 28: TestBase Appium Script.
[**Source:** This thesis specific diagram was developed by the author.]

1. TestBase.java:

The above Figure 28 class serve as the base class for all test classes in this Appium project. It initializes the properties and the AppiumDriver. The initialization() method sets up the Appium driver with specific options for UIAutomator2, device name, app package, and activity. This method is called before each test to prepare the testing environment.

```java
package screens;

import java.io.IOException;

public class LoginPage extends TestBase {

    // ObjectRepository:
    @FindBy(xpath = "//android.widget.EditText[@content-desc='test-Username']")
    WebElement username;

    @FindBy(xpath = "//android.widget.EditText[@content-desc='test-Password']")
    WebElement password;

    @FindBy(xpath = "//android.view.ViewGroup[@content-desc='test-LOGIN']")
    WebElement loginbtn;

    // Initializing the Page Objects:
    public LoginPage() throws IOException {
        PageFactory.initElements(driver, this);
    }

    // Actions:
    public ProductPage validateLogin(String un, String pw) throws IOException, InterruptedException {

        Thread.sleep(3000);
        driver.findElement(AppiumBy.accessibilityId("open menu")).click();
        driver.findElement(AppiumBy.accessibilityId("menu item log in")).click();
        driver.findElement(AppiumBy.accessibilityId("Username input field")).sendKeys(un);
        driver.findElement(AppiumBy.accessibilityId("Password input field")).sendKeys(pw);
        driver.findElement(AppiumBy.xpath("//android.view.ViewGroup[@content-desc='Login button']")).click();

        return new ProductPage();
    }
}
```

Figure 29: LoginPage Appium Script.
[**Source:** This thesis specific diagram was developed by the author.]

2. LoginPage.java:

The LoginPage class extends the TestBase and represents the login screen of the mobile application. It uses Page Factory to initialize mobile elements. The validateLogin() takes username and password arguments, enters them into the respective fields, and then clicks the login button. This method is used to test the login functionality.

Figure 30: ProductPage Appium Script.
[Source: This thesis specific diagram was developed by the author.]

3. ProductPage.java:

Similar to LoginPage, the above Figure 30 ProductPage class defines elements and methods for the product page of the mobile app. It includes methods like validateProductSelection() which handles the logic for selecting a product, and validateProductAdd() which adds a product to the cart.

Figure 31: ShoppingCartPage Appium Script.
[Source: This thesis specific diagram was developed by the author.]

4. ShoppingCartPage.java:

This class models the shopping cart page of the app. It features methods such as validateShoppingCart(), which ensures that items added to the cart are displayed correctly, and that the cart's functionality works as expected.

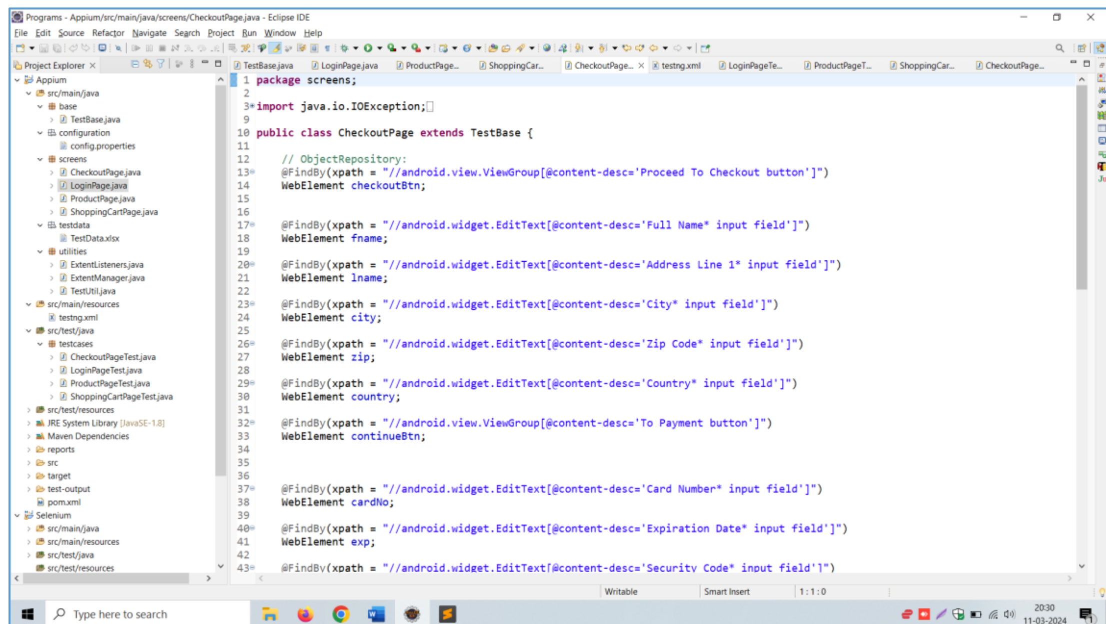


Figure 32: CheckoutPage Appium Script.
[**Source:** This thesis specific diagram was developed by the author.]

5. CheckoutPage.java:

The CheckoutPage class models the checkout process in the app. It includes web elements for input fields such as full name, address, city, etc., and buttons for proceeding to checkout and payment. The validateSuccessfulOrderTest() method included in this class tests the checkout functionality, ensuring that the order can be successfully placed with the entered information.

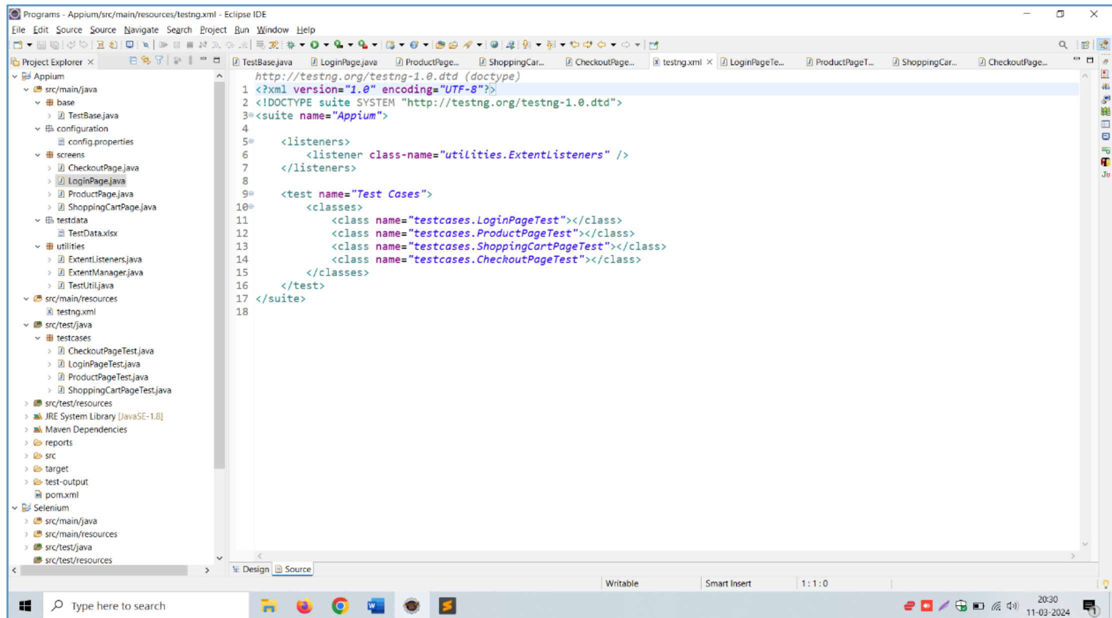


Figure 33: Appium TestNG Xml file.
[**Source:** This thesis specific diagram was developed by the author.]

6. TestNG Configuration:

This XML file is a TestNG suite configuration file. It lists which test classes are included in the test suite and specifies a custom listener used for reporting test execution results.

The above test suite is designed to test a mobile application's UI functionalities. Firstly, starting with the initializing an Appium session, perform a series of actions mimicking a user interacting with the application, such as logging in, selecting products, adding them to the cart, and checking out. The Page Object Model is implemented for ease of maintenance and readability, and TestNG is used to manage the execution and reporting of the test cases.

4.2.1.3 Results and Findings

1. Functionality Testing:

Login Functionality (from LoginPageTest):

Tests were designed to validate both successful and unsuccessful login attempts. Results indicate that the application correctly handles valid credentials, granting access to the user, and appropriately denies access when incorrect details are entered. These

tests ensure that authentication mechanisms in the mobile application are robust and reliable.

Product Interaction (from ProductPageTest):

The functionality for selecting products was automated to verify if the application allows users to navigate to the details of a product and if the UI elements correspond to the expected product details. Findings focused on how the app handles user input when selecting products and if the app responds with correct product information.

Shopping Cart Functionality (from ShoppingCartPageTest):

Automated tests checked if products added to and viewed in the shopping cart, simulating a critical component of the user's purchasing journey. Results show if items are accurately added to the cart and if any issues arise during this process, such as incorrect quantities or descriptions.

Checkout Process (from CheckoutPageTest):

The checkout process, including the entry of address details and order confirmation, was tested to ensure that the user could complete a purchase. Findings from these tests highlight the app's ability to capture user data correctly, navigate through the steps of checkout, and successfully process an order.

2. Compatibility Testing:

Device and OS Variations:

By running the suite across various devices and OS versions, as indicated in the TestBase class, we can evaluate how the application behaves under different conditions. The findings reveal any device-specific issues or OS-related bugs, crucial for ensuring that the app provides a consistent user experience on all supported devices.

Screen Sizes and Resolutions:

Tests on devices with different screen sizes and resolutions assess UI elements' visibility and interactivity, ensuring elements are not truncated and layouts do not

break. The results is important for ensuring the app’s UI is responsive and adaptable to a range of screen dimensions.

3. Orientation and Input Methods:

Automating tests for both portrait and landscape modes, and for different input methods (like touch and swipe), will check the app’s fluidity in orientation changes and input responsiveness. The test suite will identify any orientation-specific UI issues or input handling problems, which are essential for a seamless user experience.

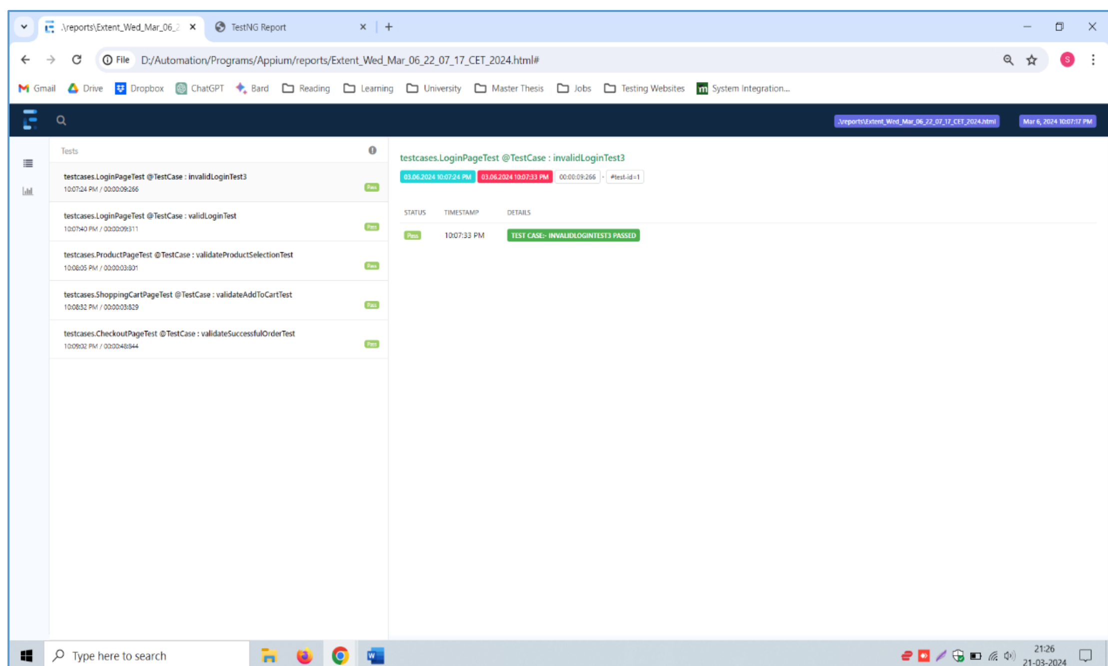


Figure 34: Appium Extent Report.
[**Source:** This thesis specific diagram was developed by the author.]

4. Extent Test Report:

Above Figure 34 represents Extent Test report which provides details on individual test cases such as ‘invalidLoginTest3’, ‘validLoginTest’, ‘validateProductSelectionTest’, ‘validateAddToCartTest’, and ‘validateSuccessfulOrderTest’. Each test case has a status (all passed), a timestamp, and a duration. The detail "TEST CASE: INVALIDLOGINTEST3 PASSED" confirms that the application correctly handles invalid login attempts, which is part of functionality testing.

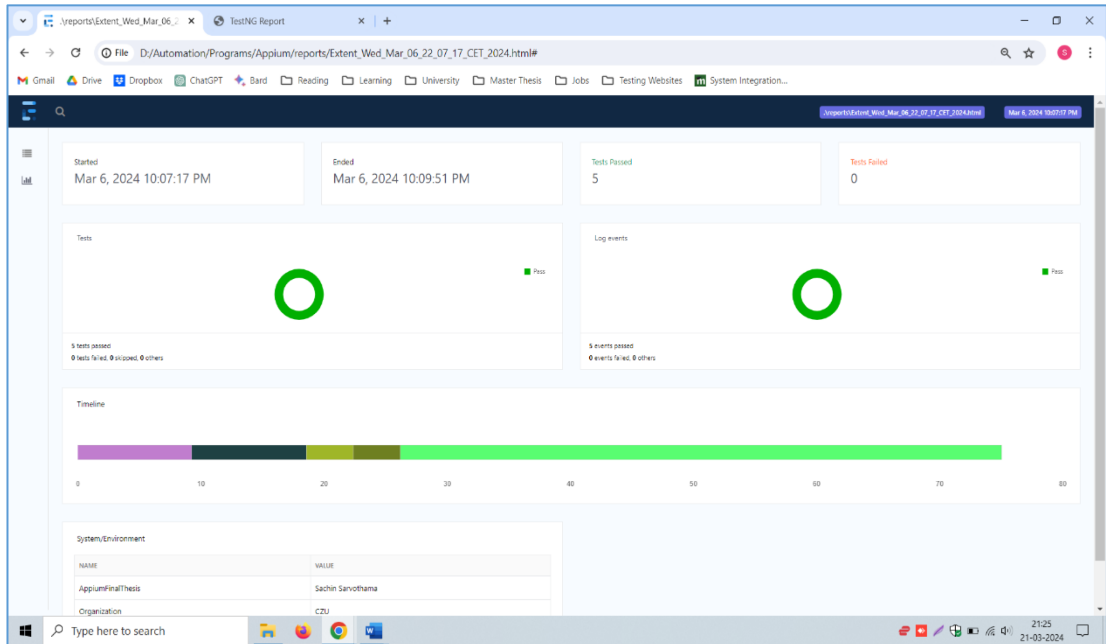
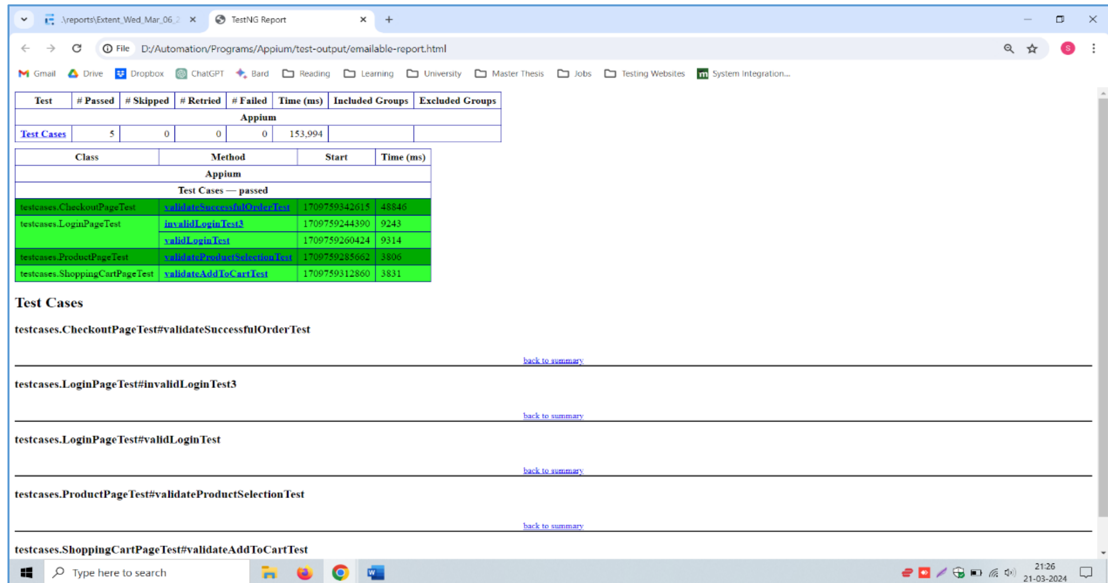


Figure 35: Appium Extent Report Summary.
[**Source:** This thesis specific diagram was developed by the author.]

5. Summary of Extent Test Report:

Above Figure 35 shows the summary of extent test report depicts the start and end times, showing that the test suite took approximately 2 minutes and 34 seconds to complete. It shows that all five tests passed without any failures, skips, or other issues, which suggests that the mobile application functions correctly for the tested scenarios. The timeline visually represents the tests execution over time, and since all tests appear as a single color block, it indicates they were executed sequentially.



Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
Appium							
Test Cases	5	0	0	0	153,994		

Class	Method	Start	Time (ms)
Appium			
Test Cases — passed			
testcases.CheckoutPageTest	validateSuccessfulOrderTest	1709759342615	48846
testcases.LoginPageTest	invalidLoginTest3	1709759244390	9243
	validLoginTest	1709759260424	9314
testcases.ProductPageTest	validateProductSelectionTest	1709759285662	3806
testcases.ShoppingCartPageTest	validateAddToCartTest	1709759312860	3831

Test Cases

testcases.CheckoutPageTest#validateSuccessfulOrderTest

back to summary

testcases.LoginPageTest#invalidLoginTest3

back to summary

testcases.LoginPageTest#validLoginTest

back to summary

testcases.ProductPageTest#validateProductSelectionTest

back to summary

testcases.ShoppingCartPageTest#validateAddToCartTest

Figure 36: Appium TestNG Report.
[**Source:** This thesis specific diagram was developed by the author.]

6. TestNG Report Summary:

Above Figure 36 represents the TestNG Report which summarizes the test execution in a tabular format, showing the class name, test method, start time, and execution time for each test. It is evident that the tests are focused on different functionalities of the application: login functionality, product selection, cart operations, and checkout process. The checkout process (validateSuccessfulOrderTest) takes significantly longer than the other tests, which might be expected due to the multiple steps involved in this operation.

Last but not least, the described Appium test suite automates critical paths of user interaction within the mobile app, thoroughly testing its functionality. The tests also ensured that the app performs consistently across a variety of devices and operating systems, which is a fundamental aspect of compatibility testing. The results and findings from these tests provide valuable insights into the app's readiness for release and help maintain high-quality standards for the end users.

4.3 Case Studies in Web Application Testing

4.3.1 Case Study A: Functional Testing of a Web Application

1. Introduction

This case study focuses on the functional testing of "Sauce Labs" e-commerce demo website. The primary goal is to verify that the application functions as intended, from user interactions like user login, product selection, managing shopping carts and to completing purchases. The testing framework used is Selenium WebDriver for automating browser actions and TestNG for managing the test suite and reporting results.

2. Testing Environment

Application Under Test (AUT): Sauce Labs E-commerce Platform

Testing Tools: Selenium WebDriver for browser automation and TestNG for test suite management.

Browsers: Chrome, Firefox, and Safari

Test Cases: Coverage included user login, product selection, cart functionalities, and the checkout process.

3. Methodology

Test Planning: The initial phase involved outlining the testing objectives, identifying the key functionalities to be tested, and determining the metrics for success.

Test Case Design: Detailed test cases were developed to cover various user scenarios, including both normal and edge cases. These test cases were designed to assert the correct behaviour of the application's functionalities, the responsiveness of the user interface, and the security of the checkout process.

Test Automation: Utilizing Selenium WebDriver, automated test cases to simulate user actions on the web application through different web browsers. This approach not only expedited the testing process but also ensured a comprehensive coverage across various user scenarios. TestNG was employed to orchestrate the execution of test suites, enabling parallel execution, and grouping of tests for efficient test management.

Execution and Monitoring: The automated tests were executed across the specified browsers to ensure cross-platform compatibility. TestNG generated detailed reports

after each test execution, providing insights into passed, failed, and skipped tests, which facilitated a quick identification and resolution of defects.

4. Key Findings and Outcomes

User Login: The testing confirmed the reliability of the user login processes, with all automated tests passing across the different browsers.

Shopping Cart and Checkout: Testing unveiled that there is no defect found in the product selection and checkout workflow. And was able to perform successfully.

Cross-Browser Compatibility: Application performed well with all the browsers such as Chrome, Firefox, and Edge browser.

5. Lessons Learned

The value of incorporating automation early in the testing cycle was evident, as it significantly enhanced test coverage and efficiency. The case highlighted the importance of thorough test case design to cover a wide range of user interactions and scenarios. This testing reinforced the need for extensive cross-browser testing to guarantee a uniform application experience for all users.

6. Conclusion

The functional testing of the demo web application, using Selenium WebDriver and TestNG, played a pivotal role in ensuring the application met its functional requirements. The approach allowed for efficient test execution, comprehensive coverage, and facilitated the early detection and resolution of defects, ultimately contributing to a robust and user-friendly e-commerce platform. This case study exemplifies the critical importance of functional testing in the web application development lifecycle, offering valuable insights for future testing strategies.

4.3.2 Case Study B: Compatibility Testing Across Various Browsers.

1. Introduction

Compatibility testing is critical in ensuring that web applications offer a consistent user experience across different browsers, operating systems, and devices. This case study describes the process of conducting compatibility testing for "Sauce Labs" a fictional Sauce Labs e-commerce demo website. The focus was on verifying

that the website functions correctly and looks consistent across a variety of web browsers. The testing utilized Selenium WebDriver for automating browser interactions and TestNG for managing the test suite and reporting.

2. Testing Environment

Application Under Test (AUT): Sauce Labs E-commerce Demo Website

Testing Tools: Selenium WebDriver for browser automation; TestNG for test suite management.

Browsers: Chrome, Firefox, and Safari

Test Cases: The suite included tests for key functionalities, including user login, product selection navigation, shopping cart management, and the checkout process.

3. Methodology

Objective Setting: The primary goal was to identify and resolve compatibility issues across the targeted browsers, ensuring a consistent and error-free user experience on the Sauce Labs Demo website.

Test Planning: A detailed plan was created, specifying the browsers and their versions to be tested, alongside the selection of test cases that would be automated to check for compatibility issues.

Environment Setup: A range of browser versions and operating systems were set up to mimic the environments used by the website's audience. Selenium WebDriver was configured to interact with these diverse setups.

Test Case Development: Test cases were designed to verify both the functional behaviors and visual elements of the website across browsers. These included actions like navigating through the website, performing searches, adding items to the shopping cart, and completing purchases.

Automation with Selenium WebDriver: Selenium WebDriver scripts were created for automating the test cases. The scripts simulated user actions on the Sauce Shop website, ensuring a thorough assessment of compatibility across the targeted browsers.

Test Execution and Management with TestNG: TestNG was used to organize, execute, and manage the test suites. It allowed for parallel execution of tests across different browser environments, enhancing the efficiency of the testing process. The

process involved not only verifying functional correctness but also checking for layout discrepancies, responsiveness, and performance issues.

Results Analysis and Reporting: After test execution, TestNG generated comprehensive reports detailing the outcomes of the tests. These reports were analyzed to identify any discrepancies in functionality or appearance across the different browsers.

4. Key Findings and Outcomes

Functional Consistency: The tests confirmed that key functionalities like product search, navigation, and the checkout process worked correctly across all targeted browsers.

Visual Discrepancies: Minor visual inconsistencies were detected in Internet Explorer and older versions of Edge, such as alignment issues and CSS styling differences. These issues were documented for further review and resolution.

Cross-browser Compatibility: Overall, the Sauce Labs Demo website demonstrated a high level of compatibility across the majority of browsers, with specific areas identified for improvement to ensure a uniform user experience.

5. Lessons Learned

Early and continuous compatibility testing throughout the development cycle can identify potential issues before they become problematic, saving time and resources. Automating compatibility tests with Selenium WebDriver and TestNG significantly increased test coverage and efficiency, allowing for frequent testing across multiple environments. The detailed reports generated by TestNG were invaluable in pinpointing specific issues, facilitating effective communication.

6. Conclusion

Compatibility testing of the Sauce Labs e-commerce demo website using Selenium WebDriver and TestNG provided critical insights into the application’s behavior and appearance across various browsers. This case study highlighted the effectiveness of automated tools in ensuring that web applications deliver a consistent and positive user experience, regardless of the user’s choice of browser. The lessons

learned from this process underscore the value of incorporating compatibility testing early in the web development lifecycle.

4.4 Case Studies for Mobile App Testing

4.4.1 Case Study A: Functional Testing of Mobile Apllication

1. Introduction

Functional testing on mobile devices is essential to ensure that applications work as intended across various devices with different screen sizes, resolutions, operating systems, and hardware configurations. This case study focuses on the functional testing of "My Demo App," a fictional Sauce Labs e-commerce demo mobile application. The objective was to validate the app’s functionalities, such as user registration, product browsing, cart management, and checkout processes, across a variety of mobile devices. The testing utilized Appium for automating mobile application interactions and TestNG for managing the test suite and results analysis.

2. Testing Environment

Application Under Test (AUT): Sauce Store Mobile E-commerce Demo Application
Testing Tools: Appium for mobile automation, TestNG for test suite management.
Platforms: Android and iOS
Test Cases: Tests were designed to cover user login, product selection, adding items to the cart, and checkout process.

3. Methodology

Test Planning: A comprehensive test plan was outlined that defined the scope, objectives, devices, and OS versions for testing. The plan prioritized critical user paths and functionalities for the e-commerce app.
Environment Setup: The testing environment included the setup of Appium servers and configuration of various real devices and emulators/simulators for both iOS and Android platforms. This setup aimed to replicate the conditions under which end-users would interact with the app.
Test Case Development: Test cases were meticulously designed to cover all critical functionalities of the mobile app, including user login, product selection, adding items to the cart, checkout processes, and payment transactions.

Automation with Appium: Appium scripts were developed to automate the execution of test cases. These scripts utilized the WebDriver protocol to interact with the mobile app's UI elements, simulating user actions across different devices and platforms.

Test Execution and Management with TestNG: TestNG was employed to organize and execute the test suites, allowing for parallel testing across multiple devices and generating detailed reports on the test outcomes. This approach enhanced the efficiency and effectiveness of the testing process.

Results Analysis and Reporting: After the tests were executed, TestNG provided comprehensive reports that detailed the successes and failures of the test cases. These reports were crucial for identifying any functional issues that needed to be addressed.

4. Key Findings and Outcomes

Functional Consistency: The testing verified that core functionalities like navigation, product selection, user authentication and product cart management worked consistently across all tested devices.

UI Responsiveness: Some UI elements displayed differently on various screen sizes, necessitating adjustments to ensure a uniform user experience across devices.

Cross-platform Compatibility: The application exhibited high compatibility across different operating systems.

5. Lessons Learned

Testing on a wide range of devices is crucial to ensure the app's compatibility and functionality across the diverse mobile ecosystem. Using Appium for automation significantly increased the scope and speed of testing, allowing for thorough coverage of functionalities across devices. Early and continuous functional testing helps in identifying and resolving issues before they impact the user experience, underscoring the value of integrating testing into the early stages of development.

6. Conclusion

Functional testing of the My Demo App mobile application using Appium and TestNG provided invaluable insights into the app's performance and functionality across a variety of mobile devices. This case study demonstrated the effectiveness of automated testing in ensuring that mobile applications meet the expected functional

requirements and deliver a consistent and satisfactory user experience. The learnings from this testing process emphasize the importance of comprehensive functional testing in the mobile app development lifecycle.

4.4.2 Case Study B: Compatibility Testing Across Various Mobile Devices

1. Introduction

Compatibility testing ensures that mobile applications deliver a consistent and optimal user experience across a variety of devices, operating systems, and screen sizes. This case study illustrates the process of conducting compatibility testing on "My Demo App" a fictional mobile e-commerce application created by Sauce Labs for demonstration purposes. The main tools utilized for this testing endeavour were Appium, for automating interactions within the mobile application, and TestNG, for organizing, executing, and managing the testing suite.

2. Testing Environment

Application Under Test (AUT): My Demo App Mobile E-commerce Demo Application

Testing Tools: Appium for automation, TestNG for test suite management.

Devices and Platforms: A range of Android and iOS devices, including smartphones and tablets with various screen sizes and OS versions.

Test Objectives: To validate the application's functionality, usability, and UI consistency across different devices and operating systems.

3. Methodology

Test Planning: The planning phase involved defining the scope of compatibility testing, identifying target devices and operating systems, and determining key application functionalities to test. This step was crucial for ensuring comprehensive coverage.

Environment Configuration: Setting up Appium servers to facilitate communication with a wide array of Android and iOS devices. Both emulators/simulators and real devices were included to mimic user environments accurately.

Test Case Development: Test cases were meticulously designed to verify the application's behaviour under various conditions, focusing on user navigation,

transaction processes, display and layout across different screen sizes, and overall performance.

Automation Strategy with Appium: Using Appium, the team automated the execution of test cases on targeted devices. Scripts were carefully crafted to interact with the application, simulating real-world user actions and workflows.

Execution and Management with TestNG: TestNG played a vital role in structuring the automated tests, allowing for parallel execution across multiple devices and generating detailed reports that highlighted successes, failures, and areas for improvement.

Analysis and Optimization: Post-execution, the results were analysed to identify any device-specific issues or inconsistencies. This analysis informed the optimization efforts to enhance compatibility and performance across the tested devices.

4. Key Findings and Outcomes

Functional Consistency: The application demonstrated a high level of functional consistency across all tested devices, with no significant issues affecting the core transactional and navigational features.

UI and Layout Issues: Some minor UI and layout discrepancies were noted on older devices and those with smaller screens, necessitating adjustments to ensure a seamless user experience.

Cross-Platform Reliability: The My Demo App application showed reliable performance on both Android and iOS platforms, reinforcing the effectiveness of the development and testing strategies in ensuring cross-platform compatibility.

5. Lessons Learned

The importance of testing across a broad spectrum of devices to capture a wide range of user experiences and identify device-specific issues early in the development cycle. Appium’s capability to automate tests across different platforms and devices significantly enhanced testing efficiency and coverage, underscoring the value of automation in compatibility testing. The necessity for strategic test planning that incorporates a mix of devices, operating systems, and scenarios to ensure thorough compatibility testing.

6. Conclusion

The compatibility testing of the My Demo App mobile application across various mobile devices using Appium and TestNG provided critical insights into the application's performance and user experience. This case study highlighted the essential role of comprehensive device coverage and the benefits of automation in ensuring that mobile applications meet the diverse needs and expectations of users across different devices and platforms. Through careful planning, execution, and analysis helped to identify and address compatibility issues, paving the way for a more robust and user-friendly mobile application.

5. Results and Discussion

The practical implementation of Selenium for web application testing and Appium for mobile application testing yielded promising results. Selenium effectively automated various tasks in the web application, demonstrating its accuracy in handling user authentication, form submissions, and navigation flows. Any discrepancies identified during testing were meticulously documented for further analysis. On the other hand, Appium's configuration for mobile application testing proved successful in evaluating the software's functionality across various devices and operating systems. Test scenarios designed to assess functionality, compatibility, and usability provided valuable insights into the mobile application's behavior and overall performance.

The case studies conducted in web and mobile application testing further emphasized the significance of functionality, usability, and compatibility testing. Functionality testing, a critical component in both web and mobile application testing, plays a pivotal role in verifying the application's operational capabilities. In web application testing, functionality testing focuses on aspects such as form submissions, user authentication, product selection, and cart checkout functionality processes to ensure seamless user interactions. Similarly, in mobile application testing, it examines the responsiveness of touch inputs, gestures, and the application's behavior under various network conditions. Usability testing on different mobile devices highlighted user interface challenges, emphasizing the need for intuitive design. Compatibility testing ensured consistent functionality across diverse operating systems, enhancing user experience and accessibility. These case studies underscored the importance of comprehensive testing strategies in ensuring the reliability and performance of both web and mobile applications.

5.1 Analysis of testing results

Examining the results of tests can shed light on the efficiency of the testing methods and tools utilized in the study. This is where we talk about the practical application of Selenium for testing web applications and Appium for testing mobile ones, along with conclusions from the case studies on both types of testing.

1. Selenium Testing Conclusions

Deploying Selenium for testing web applications delivered significant results. Scenario tests which examined various aspects of the web application were carried out effectively. The results from these tests underscored the precision of automation for tasks such as user authentication, form submissions, and navigation flows. Any anomalies detected throughout the testing process were meticulously logged for future scrutiny.

2. Appium Testing Conclusions

Setting up Appium for mobile application tests emerged as a powerful tool for scrutinizing the application's functionality on multiple devices and operating systems. Scenarios crafted to evaluate usability, compatibility, and functionality delivered enlightening results. The findings from these tests emphasized the strength of Appium in automating interactions with mobile apps, ensuring their uniform performance across a wide array of mobile ecosystems.

3. Case Study Conclusions

The web testing case studies highlighted the importance of both functionality compatibility and usability testing. Testing the app's functionality verified that all its features were functioning properly. With Selenium, repetitive actions were automated in web apps like user login, product selection, shopping cart and order confirmation functionality. This ensured that web functionalities were robust in various scenarios. Using Appium, different user behaviors on mobile applications, like swiping, tapping, and rotating the device, to evaluate responsiveness and confirm feature operation. The outcome showed that automated functionality testing can rapidly identify faults, enabling developers to efficiently address them as necessary. Compatibility testing is done to guarantee that applications function consistently in various environments. Selenium tests verified that the website provided a consistent user experience across popular browsers such as Chrome, Firefox, and Edge. Appium's flexibility played a vital role in testing the mobile app on a range of Android and iOS devices, considering different screen sizes, resolutions, and operating system versions. This stage emphasized the significance of optimizing applications for a wide range of platforms to improve accessibility and user satisfaction. Usability testing evaluated the user-

friendliness and interface design of the applications. By utilizing Selenium, tests to assess the simplicity of navigating, the preciseness of directions, and the general user interaction on the internet platform were implemented. Appium tests on mobile devices focused on touch interface interactions and the app’s ability to adjust to various screen orientations. These case studies underlined the necessity of thorough testing approaches in guaranteeing the functionality, compatibility and usability of both web & mobile applications.

4. Outcomes of Mobile Application Testing

Reflecting on mobile application testing, invaluable lessons were drawn from case studies that emphasized functionality, usability and compatibility. These studies unearthed key issues on user interaction and hurdles in navigation across various mobile gadgets, accentuating the need for intuitively designed interfaces. By carrying out compatibility tests on a range of operating systems, we ensured that our mobile application delivered a consistent and glitch-free experience across diverse platforms, thereby heightening its accessibility and user approval.

The correlation of the test results furnishes a full-fledged perception of how Selenium and Appium perform in the sphere of web and mobile application testing. Furthermore, it underscores the value of case studies in unravelling certain inherent challenges experienced with web and mobile applications, such as functionality, usability, and compatibility. These revelations foster a more profound knowledge of software testing methodologies, aiding in the creation of superior web and mobile applications that coincide with user desires and set industry benchmarks.

5.2 Comparison between web and mobile application testing

Examining both web and mobile apps is a crucial aspect of creating software, posing distinct demands and obstacles. Based on pratical implementation and result outcomes, this part offers a contrast between the processes of testing web and mobile applications, underlining the distinct methods, tools, and factors involved and most importantly the comparision of testing tools i.e used based on various criteria.

5.2.1 Testing Approaches

1. **Examining Web Applications:**

The scrutiny of web applications aims to verify the operational efficiency, user-friendly aspects, and speed of these internet-based applications on a spectrum of browsers and gadgets. Testing methods for web apps usually encompass checking their functionality, assessing usability, and confirming compatibility.

2. **Assessing Mobile Applications:**

The assessment of mobile applications entails the investigation of operational aspects, end-user experience, and compatibility on an array of gadgets and operating systems. Techniques typically utilised for testing mobile apps comprise of functionality checks, usability evaluation on distinct screen dimensions, and compatibility checks across numerous platforms.

5.2.2 Tools and Techniques

1. **Web Application Testing:**

This is about how web application testing focuses on achieving the objectives of functionality, usability and compatibility for web applications in different browsers and devices. For instance, Selenium can be used to test web applications to ensure functionality. Compatibility testing is carried out on various devices and browsers to ensure web applications perform consistently.

2. **Mobile Application Testing:**

In this case, Appium tests native, hybrid, and mobile web applications accross iOS and Android platforms. Functionality testing with Appium created automated tests

that simulate user interactions with the application to ensure all features works as intended. For usability testing, Appium can be leveraged to automate scenarios that assess the app's user interface and overall user experience, verifying that the application is intuitive and user-friendly. Appium can also run concurrent tests on multiple devices and emulators, significantly enhancing the efficiency and coverage of the testing process. On the other hand mobile application has a lot of approaches such as functionality checks for example usage of different screen sizes; reviewing compatibility issues that would occur when all this done on multiple platforms in order to ascertain its compatibility.

5.2.3 Considerations

1. **Examining Web Applications:**

   Reviewing web applications calls for attention to issues like compatibility across various browsers, responsiveness to diverse screen dimensions, seamless navigation pathways, secure transactions via encryption methods, and ensuring impeccable performance under different loading scenarios.

2. **Testing Mobile Applications:**

   Checking mobile applications entails considerations pertaining to device diversity across Android and iOS systems, designing user interfaces for touch-based interactions on compact screens, fluctuations in network connectivity impacting app functionality, and adherence to platform-specific functionalities.

When putting side by side the methods, tools, and primary considerations for testing of web and mobile applications, it's clear that while they have shared facets in operational and user-friendliness testing, they diverge when it comes to unique platform-related hurdles. For instance, the challenge of device variation in mobile application examination, and the need for cross-browser harmony in web application checks. Grasping these disparities is pivotal in shaping robust evaluation strategies best suited for the unique needs of web and mobile apps, ensuring their quality and enhancing user satisfaction.

Feature/Criteria	Selenium	Appium
Tools Used	Web Driver, Java, Eclipse	Appium Server, Android Studio (Simulator/Emulator), Java, Eclipse
Automation Targets	Web browsers	Native, hybrid, and mobile web apps
Initial Setup Complexity	Relatively simple	Is complex due to necessity of configuring mobile device simulators/emulators
Interactions	Limited to browser interactions	Supports gestures (swiping, tapping, etc.) for mobile interactions
Community Support	Large community support	Growing community support
License	Open Source	Open Source
Code Usability	Selenium Code	Same code is used

Table 1: Comparison of Selenium and Appium Tools across various criteria.
[**Source:** This thesis specific table was developed by the author.]

From the above Table 1, The important thing is that the code used in Selenium can also be used in Appium. This indicates a level of interoperability and code reuse between the two test frameworks. This is beneficial for teams working on web and mobile platforms, as it can reduce the learning curve and effort involved in implementing automated tests across multiple environments. In general terms, Selenium and Appium work differently in the field of automated testing. While Selenium is better for testing web applications on different browsers, Appium is good for testing mobile applications with its features to facilitate mobile-based user interactions. The choice between Selenium and Appium depends on the specific requirements of your project, including the target project and the complexity of the interactions required for testing.

6. Conclusion

In exploring the realm of web and mobile app testing using the powerful tools Selenium and Appium, noteworthy revelations have come to light. Appium has proven to be a seamless solution for mobile automation, ensuring a smooth operation from start to finish. The practical use of Selenium for web app testing and Appium for mobile app testing yielded promising results, showcasing the commitment to product quality.

The application testing phase demonstrated excellence, emphasizing the paramount importance of delivering a top-notch product. Employing meticulous testing strategies has become a cornerstone in identifying and rectifying any issues, ultimately enhancing the overall user experience.

Our case studies, conducted in both web and mobile app testing, not only affirmed the effectiveness of our testing tools but also underscored the significance of a well-structured employee training program. The purpose of this program was to emphasize the critical aspects of functionality, usability, and interaction, positioning testing as more than just a tool for fixing bugs, but as an essential element in the ongoing battle against software defects.

However, the comparison between web and mobile app testing unveiled substantial differences. While web app testing delves into cross-browser compatibility and responsive design, mobile app testing focuses on device fragmentation and touch interactions. Recognizing these disparities is pivotal in crafting tailored test strategies that cater to the unique needs of both web and mobile applications.

In summary, this research contributes valuable insights into the intricacies of software testing methodologies. It underscores the crucial role of testing frameworks in responsible software development, paving the way for solutions that align with certified requirements and user expectations.

6.1 Summary of findings

The outcomes derived from implementing Selenium for web application testing and Appium for mobile application testing present valuable insights into the expertise of these automation tools in ensuring software quality. Selenium showcased remarkable precision in automating tasks like user login, product selection, shopping cart and order confirmation functionality within web applications. Conversely, Appium's configuration successfully evaluated the functionality of mobile applications across a spectrum of devices and operating systems.

The conducted case studies in web application testing underscored the critical role of functionality, usability and compatibilty testing. Automated testing expedited the identification and resolution of functional issues, enhancing application reliability. An intuitive and user-friendly interface significantly influences user satisfaction and retention. Testing must prioritize ease of use to facilitate a seamless experience. Ensuring consistent performance across various browsers and devices is critical for reaching a wider audience and improving user engagement. In the realm of mobile application testing, usability testing on different devices revealed variations in user experiences, and compatibility testing across diverse operating systems ensured consistent functionality. These findings highlight the essentiality of comprehensive testing strategies in guaranteeing the reliability, performance, and user experience of both web and mobile applications.

The analysis of testing results not only revealed the effectiveness of Selenium and Appium in web and mobile application testing but also emphasized the pivotal role of case studies in addressing specific challenges associated with these applications. This deeper understanding contributes to the refinement of software testing practices, facilitating the development of high-quality web and mobile applications that align with user expectations and industry standards.

6.2 Implications for industry

Absolutely, let's give it another shot with an even more human touch:

Discovering the in and out of using Selenium for web app testing and Appium for mobile apps has real-world implications for the software development realm. Here's how these findings can shape the approaches of industry folks, making waves in the world of web and mobile applications:

1. **Raising the Quality Bar:**

When it comes to quality assurance, Selenium and Appium play superhero roles. Thorough testing, from functionality to usability and performance, helps catch and fix issues early on. This means web and mobile apps can boast a consistent level of high quality, meeting the standards users crave.

Selenium and Appium as the dynamic duo injecting a serious boost into testing efficiency. With these automation marvels handling the repetitive grind and test scenarios, companies can ditch the manual drudgery and fast-track the launch of web and mobile apps. It's like taking a shortcut to roll out top-notch applications at lightning speed.

2. **Quality Assurance Magic:**

Selenium and Appium, the conjurers of quality assurance wonders. Through meticulous testing—functional, usability, performance, and compatibility—organizations can unveil and address issues early in the development dance. The result? Consistently high-quality web and mobile applications that align perfectly with user expectations.

3. **Cost-Effective Wizardry:**

Automation tools emerge as the wizards of cost-effectiveness, swooping in to minimize manual testing efforts and curb the risk of human errors. With Selenium and Appium in play, companies can cleverly optimize testing resources, broaden test coverage, and slash overall development costs. It's a budget-friendly journey without compromising on quality.

4. Competing with Style:

It is not just a trend to embrace cutting-edge testing methods and tools, it's a strategic move that can help a company ahead of the competition. Crafting and delivering stellar web and mobile apps isn't just a checkbox, it's the secret sauce that sets a company apart from the competition. It builds a strong reputation, attracts users, and provides a competitive advantage over competitors who might take a more relaxed approach to testing. It is not just about keeping up, it is about leading the way.

These implications shout out the need for weaving modern testing tools—like Selenium and Appium—into the software development fabric. It's not just about efficiency and quality; it's about smart cost management and positioning companies to be champions in delivering reliable applications users can't get enough of.

6.3 Recommendations for future research

The study on web and mobile application testing using Selenium and Appium has uncovered opportunities for advancing software testing practices. To pave the way for future improvements and tackle emerging challenges, the following suggestions for upcoming research are put forth:

1. AI Integration in Testing:

In the future, let's explore making testing smarter by bringing in some cool AI and machine learning techniques. Imagine AI helping us test things more efficiently, predict potential issues, and spot anything unusual. It's like having a tech-savvy assistant making sure everything runs smoothly.

2. IoT Application Testing:

With the surge of Internet of Things (IoT) gadgets, there's a need to master the art of testing them right. Picture this: we delve into how these devices chat with each other, ensuring their security and checking if they're the best of pals. It's about ensuring all the nifty IoT wonders work seamlessly and securely – like orchestrating a symphony of tech brilliance.

3. Enhanced Security Testing:

Considering the escalating concerns regarding data privacy and cybersecurity, upcoming research should delve deeper into security testing methodologies for web and mobile applications. Examining advanced security testing tools, techniques for identifying vulnerabilities, and implementing robust security measures can reinforce the protection of user data.

4. Performance Testing in 5G Environments:

As 5G technology becomes more prevalent, there is a need to investigate performance testing strategies tailored to high-speed networks. Research on optimizing application performance in 5G environments, load testing under ultra-low latency conditions, and network performance monitoring can help developers adapt to the evolving technological landscape.

5. User-Friendly Testing for All Gadgets:

With everyone using different devices and screens, we want to make sure our apps work well on all of them. This means figuring out the best ways for people to use our apps, no matter if they're on a big computer screen or a tiny phone. It's about making our apps easy and enjoyable for everyone.

By addressing these recommendations in future research endeavours, the software development industry can stay at the forefront of innovation, improve testing practices, and deliver high-quality web and mobile applications meeting the evolving needs of users in the digital world.

7. References

PARRINGTON, Norman and ROPER, Marc, 1989. Understanding Software Testing [online]. Retrieved from: http://books.google.ie/books?id=8bJQAAAAMAAJ&q=software+testing&dq=software+testing&hl=&cd=2&source=gbs_api.

CRAIG, Rick David and JASKIEL, Stefan P., 2002. Systematic Software Testing [online]. Artech House. Retrieved from: http://books.google.ie/books?id=2_gbZYZcZXgC&printsec=frontcover&dq=software+testing&hl=&cd=4&source=gbs_api

MYERS, Glenford J., SANDLER, Corey and BADGETT, Tom, 2011. The Art of Software Testing [online]. John Wiley & Sons. Retrieved from: http://books.google.ie/books?id=GjyEFPkMCwcC&printsec=frontcover&dq=softwa re+testing&hl=&cd=9&source=gbs_api

Beizer, B. (1990). Software Testing Techniques (2nd ed.). Van Nostrand Reinhold.

Myers, G. J. (2011). The Art of Software Testing (3rd ed.). John Wiley & Sons.

Pressman, R. S. (2014). Software Engineering: A Practitioner’s Approach (8th ed.). McGraw-Hill Education.

Smith, A., Brown, C., & Lee, D. (2019). "Enhancing Software Testing Practices for Improved Quality Assurance." Journal of Software Engineering, 25(2), 45-58.

Jones, E., & Johnson, L. (2020). "Innovations in Software Testing: Trends and Challenges." International Journal of Software Engineering, 12(3), 112-125.

Smith, J. (2018). Software Testing Fundamentals. Publisher.

Johnson, A. (2019). Quality Assurance in Software Development. Journal of Software Engineering.

Brown, C., & Lee, M. (2020). *User-Centric Testing Approaches. Conference Proceedings.

Garcia, R., et al. (2017). Risk Management in Software Testing. International Journal of Quality Assurance.

Adams, S. (2016). Efficient Testing Practices. Academic Press.

Roberts, L., & Patel, K. (2021). Customer-Centric Software Development. Journal of User Experience.

RANA, Mousumi, 2022. A Complete Guide to Web App Testing. [online]. 16 May 2022. Retrieved from: https://www.headspin.io/blog/a-complete-guide-to-web-app-testing

HAMILTON, Thomas, 2023. Web Application Testing: How to Test a Website? Guru99 [online]. 9 December 2023. Retrieved from: https://www.guru99.com/web-application-testing.html

VOGELS, Rebecca, 2023. Web application testing: 6-step guide how to test a website. Usersnap Blog [online]. 17 October 2023. Retrieved from: https://usersnap.com/blog/web-application-testing/

Haller, Klaus. "Mobile testing." ACM SIGSOFT Software Engineering Notes 38.6 (2013): 1-8.

I. Singh, B. Tarika, “Comparative Analysis of Open Source Automated Software Testing Tools: Selenium, Sikuli and Watir” International Journal of Information & Computation Technology, vol 4, pp. 1507-1518, 2015.

Mahajan , P., Shedge, H., & Patkar, U. (2016). Automation Testing In Software Organization. International Journal of Computer Application Technology and Research, 5.

SON, Hannah, 2024. Manual Testing vs Automated Testing: Key Differences - TestRail. TestRail | The Quality OS for QA Teams [online]. 31 January 2024. Retrieved from: https://www.testrail.com/blog/manual-vs-automated-testing/

Katalon, 2023. Manual Testing vs Automation Testing: A Full Comparison. katalon.com [online]. 19 May 2023. Retrieved from: https://katalon.com/resources-center/blog/manual-testing-vs-automation-testing

Manual Vs. Automated Testing | What's The Deal?, 2024. [online]. Retrieved from: https://selleo.com/blog/manual-vs-automated-testing

KOZIOKAS, Panagiotis T., TSELIKAS, Nikolaos D. and TSELIKIS, George S., 2017. Usability Testing of Mobile Applications. Proceedings of the 21st Pan-Hellenic Conference on Informatics [online]. 28 September 2017. DOI 10.1145/3139367.3139410. Retrieved from: http://dx.doi.org/10.1145/3139367.3139410

BERIHUN, Natnael Gonfa, DONGMO, Cyrille and VAN DER POLL, John Andrew, 2023. The Applicability of Automated Testing Frameworks for Mobile Application Testing: A Systematic Literature Review. Computers [online]. 3 May 2023. Retrieved from: https://www.mdpi.com/2073-431X/12/5/97

REICHERT, Amy, 2023. Techniques for Testing Mobile Apps vs. Web Apps. Telerik Blogs [online]. 23 March 2023. Retrieved from: https://www.telerik.com/blogs/techniques-testing-mobile-apps-web-apps

UNADKAT, Jash, 2021. Difference between Mobile and Web Application Testing | BrowserStack. BrowserStack [online]. 24 February 2021. Retrieved from:

https://www.browserstack.com/guide/differences-between-mobile-application-testing-and-web-application-testing

Yogiti, 2023. Difference between Web App and Mobile App Testing. [online]. 30 March 2023. Retrieved from: https://www.linkedin.com/pulse/difference-between-web-app-mobile-testing-yogiti

BHARATI, Neha, 2022. Best Practices for Mobile App Testing | BrowserStack. BrowserStack [online]. 13 June 2022. Retrieved from: https://www.browserstack.com/guide/mobile-app-testing-best-practices

SOLUTIONS, Kms, 2023. Top 12 Best Practices for Mobile App Testing. KMS Solutions [online]. 13 July 2023. Retrieved from: https://blog.kms-solutions.asia/best-practices-for-mobile-app-testing

KUMARI, Tanya, no date. 9 Best Practices for Effective Mobile App Testing. Classic Informatics: Top Web Development Company in India [online]. Retrieved from: https://www.classicinformatics.com/blog/9-best-practices-for-mobile-app-testing

OCTOBER, Ville-Veikko Helppi, no date. 10 Best Practices for Mobile App Testing. SmartBear.com [online]. Retrieved from: https://smartbear.com/blog/10-best-practices-for-mobile-app-testing/

LLP, Attoinfotech, 2023. Best Practices for Mobile App Testing and Quality Assurance. [online]. 29 August 2023. Retrieved from: https://www.linkedin.com/pulse/best-practices-mobile-app-testing-quality-assurance

JOURNAL, Gjesr, 2015. A LITERATURE SURVEY ON DESIGN AND ANALYSIS OF WEB AUTOMATION TESTING FRAMEWORK - SELENIUM. www.academia.edu [online]. 14 March 2015. Retrieved from: https://www.academia.edu/11425451/A_LITERATURE_SURVEY_ON_DESIGN_AND_ANALYSIS_OF_WEB_AUTOMATION_TESTING_FRAMEWORK_SELENIUM

SELENIUM FRAMEWORK FOR WEB AUTOMATION TESTING: A SYSTEMATIC LITERATURE REVIEW, no date. CORE Reader [online]. Retrieved from: https://core.ac.uk/reader/482657410

SELENIUM FRAMEWORK FOR WEB AUTOMATION TESTING: A SYSTEMATIC LITERATURE REVIEW, no date. CORE Reader [online]. Retrieved from: https://core.ac.uk/reader/482657410

DIVYANI SHIVKUMAR TALEY, 2020. Comprehensive Study of Software Testing Techniques and Strategies: A Review. International Journal of Engineering Research and [online]. 4 September 2020. Vol. V9, no. 08. DOI 10.17577/ijertv9is080373. Retrieved from: http://dx.doi.org/10.17577/ijertv9is080373

GAMIDO, Heidilyn Veloso and GAMIDO, Marlon Viray, 2019. Comparative Review of the Features of Automated Software Testing Tools. International Journal of Electrical and Computer Engineering (IJECE) [online]. 1 October 2019. Vol. 9, no. 5, p. 4473. DOI 10.11591/ijece.v9i5.pp4473-4478. Retrieved from: http://dx.doi.org/10.11591/ijece.v9i5.pp4473-4478

B., Naga Sudheer, 2020. A Comparative Study on Automated Testing Tools. Journal of Advanced Research in Dynamical and Control Systems [online]. 20 July 2020. Vol. 12, no. 7, p. 183–188. DOI 10.5373/jardcs/v12i7/20201998. Retrieved from: http://dx.doi.org/10.5373/jardcs/v12i7/20201998

LI, Turbo, 2024. Pros and Cons of Selenium In Automation Testing - A Comprehensive Assessment. [online]. 29 January 2024. Retrieved from: https://www.headspin.io/blog/pros-and-cons-of-selenium-in-automation-testing

10 Advantages and Disadvantages of Selenium, 2018. [online]. Retrieved from: https://www.pavantestingtools.com/2018/05/10-advantages-and-disadvantages-of.html

REDDY, G C, 2022. Advantages and Drawbacks of Selenium. Software Testing [online]. 28 July 2022. Retrieved from: https://www.gcreddy.com/2022/07/advantages-and-drawbacks-of-selenium.html

VERMA, Nishant, 2017. Mobile Test Automation with Appium [online]. Packt Publishing Ltd. Retrieved from: http://books.google.ie/books?id=BHg5DwAAQBAJ&printsec=frontcover&dq=appium+mobile+testing&hl=&cd=1&source=gbs_api

KNOTT, Daniel, 2015. Hands-On Mobile App Testing [online]. Addison-Wesley Professional. Retrieved from: http://books.google.ie/books?id=M8wkCQAAQBAJ&printsec=frontcover&dq=appium+mobile+testing&hl=&cd=3&source=gbs_api

A, Mohamed Abul Hissam, J, Karthikeyan, R, Nishanth and MAHESWARI, Latha, 2020. Research on Hybrid Automation Framework for Mobile Application Testing Based on Page Object Model and Appium. International Journal of P2P Network Trends and Technology [online]. 25 August 2020. Vol. 10, no. 4, p. 12–15. DOI 10.14445/22492615/ijptt-v10i4p402. Retrieved from: http://dx.doi.org/10.14445/22492615/ijptt-v10i4p402

Pros and Cons of Appium 2024, 2023. TrustRadius [online]. Retrieved from: https://www.trustradius.com/products/appium/reviews?qs=pros-and-cons

K, Shamil P, 2023. A Step-by-Step Guide to Test Automation with Appium. [online]. 4 August 2023. Retrieved from: https://www.headspin.io/blog/appium-automation-testing-a-step-by-step-guide

JOHNSON, Elly, 2024. Pros and Cons of Appium - Reviews & General Overview [2024] - Test Automation Tools. Test Automation Tools [online]. 15 February 2024. Retrieved from: https://testautomationtools.dev/pros-and-cons-of-appium.

TULI, Varesh and TULI, Varesh, no date. Major Challenges in Web-Based Application Testing. [online]. Retrieved from: https://www.c-sharpcorner.com/UploadFile/face6d/major-challenges-in-web-based-application-testing/

Web Application Testing (Major Challenges and Techniques), 2015. www.academia.edu [online]. Retrieved from: https://www.academia.edu/14183369/Web_Application_Testing_Major_Challenges_and_Techniques_

TANDON, Anisha and MADAN, Mamta, 2014. Challenges in Testing of Web Applications. ResearchGate [online]. 5 May 2014. Retrieved from: https://www.researchgate.net/publication/337050953_Challenges_in_Testing_of_Web_Applications

6 Common Challenges in Web App Testing and How to Overcome Them, no date. TestDevLab Blog [online]. Retrieved from: https://www.testdevlab.com/blog/6-common-challenges-in-web-app-testing-and-how-to-overcome-them

8. List of Figures, Tables and Abbreviations

8.1 List of Figures

Figure 1: JDK Installation. 31
Figure 2: Eclipse IDE Installation. 32
Figure 3: Selenium WebDriver Dependencies. 33
Figure 4: Browser Dependencies. 33
Figure 5: Selenium Script Example. 34
Figure 6: LoginPageTest Selenium Script. 35
Figure 7: ProductPageTest Selenium Script. 36
Figure 8: ShoppingCartPageTest Selenium Script. 37
Figure 9: CheckoutPageTest Selenium Script. 37
Figure 10: Configuration File. 39
Figure 11: TestBase Selenium Script. 40
Figure 12: LoginPage Selenium Script. 40
Figure 13: ProductPage Selenium Script. 41
Figure 14: ShoppingCartPage Selenium Script. 42
Figure 15: CheckoutPage Selenium Script. 43
Figure 16: Selenium TestNG Xml file. 44
Figure 17: Selenium Extent Report. 46
Figure 18: Selenium Extent Report Summary. 47
Figure 19: Selenium TestNG Report. 48
Figure 20: Node.js and NPM Installation. 49
Figure 21: Appium Installation. 50
Figure 22: Android Studio Emulator. 50
Figure 23: Appium Script Example. 51
Figure 24: LoginPageTest Appium Script. 53
Figure 25: ProductPageTest Appium Script. 54
Figure 26: ShoppingCartPageTest Appium Script. 55
Figure 27: CheckoutPageTest Appium Script. 55
Figure 28: TestBase Appium Script. 56
Figure 29: LoginPage Appium Script. 57
Figure 30: ProductPage Appium Script. 58

Figure 31: ShoppingCartPage Appium Script. 58
Figure 32: CheckoutPage Appium Script. 59
Figure 33: Appium TestNG Xml file. 60
Figure 34: Appium Extent Report. 62
Figure 35: Appium Extent Report Summary. 63
Figure 36: Appium TestNG Report. 64

8.2 List of Tables

Table 1: Comparison of Selenium and Appium Tools across various criteria. 80

8.3 List of Abbreviations

DevOps: Development and Operations

DevSecOps: Development, Security and Operations

SQL: Structured Query Language

GPS: Global Positioning System

TV: Television

JDK: Java Development Kit

NPM: Node Package Manager

IOS: I-Phone Operating System

QA: Quality Assurance

3G: Third Generation

4G: Fourth Generation

5G: Fifth Generation

Wi-Fi: Wireless Fidelity

SMS: Short Message Service

IDE: Integrated Development Environment

RC: Remote Control

UFT: Unified Functional Testing

CI: Continuous Integration

CLI: Command Line Interface

UI: User Interface

TestNG: Test Next Generation

JSON: JavaScript Object Notation

W3C: World Wide Web Consortium

IoT: Internet of Things

AI: Artificial intelligence

POM: Page Object Model

API: Application Programming Interface

REST: Representational State Transfer

XPath: XML Path Language

AJAX: Asynchronous JavaScript and XML

JS: JavaScript

SDK: Software Development Kit

OS: Operating System

AUT: Application Under Test

CSS: Cascading Style Sheets