



Entwicklung einer Web-Testsuite am Beispiel von jExam

Michael Vivian Mboni Saha

michael_vivian.mboni_saha@tu-dresden.de

Studienrichtung: Bachelor Informatik

Matrikelnummer: 4757191

Immatrikulationsjahr: 2018

Bachelor-Arbeit

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

Betreuer

M.Sc. Ronny Marx

Betreuender Hochschullehrer

Prof. Dr. rer. nat habil. Uwe Aßmann

Eingereicht am: 29. Januar 2021

Inhaltsverzeichnis

Glossar	7
1. Einleitung	9
1.1. Motivation	9
1.2. Zielsetzung und Abgrenzung	10
1.3. Aufbau der Arbeit	11
2. Grundlagen	12
2.1. Ziele und Grenzen des Softwaretestens	12
2.2. Webanwendungen	14
2.3. Testumgebung	14
2.4. Testautomatisierung	15
2.5. Testsmethoden und -ansätze	16
2.5.1. Statisches Testen	16
2.5.2. Dynamisches Testen	17
2.6. Verschiedene Testarten	17
2.6.1. Funktionale Tests	17
2.6.2. Nichtfunktionale Tests	19
2.7. Verschiedene Schritte zur Erstellung einer Testsuite	20
3. Problemanalyse	22
3.1. Verwandte Arbeiten	22
3.2. Beschreibung von jExam	23
3.3. Anforderungen an die jExam-Testsuite	24
3.3.1. jExam UI-Tests	24
3.3.2. jExam Sicherheitstests	25
3.3.3. jExam Performancetests	26
3.3.4. Zusatzanforderungen	27
3.4. Zusammenfassung der Analyse	27
4. Implementierung der automatisierten Testinfrastruktur	28
4.1. Vorstellung der Testansatzes	28
4.2. Entwicklung von Sicherheitstests	29
4.2.1. OWASP	29
4.2.2. ZAP Proxy	35
4.2.3. Tests und Ergebnisse	37

4.3.	Entwicklung von Performancetests	40
4.3.1.	Apache JMeter	41
4.3.2.	Implementierung der Performancetests	42
4.4.	Entwicklung von UI-Tests	44
4.4.1.	Verwendete Werkzeuge	44
4.4.2.	Entwurfsmuster	46
4.4.3.	Implementierung der Testsuite	49
4.4.4.	Erreichte Ergebnisse	54
4.5.	Einrichtung der Docker-Infrastruktur	56
4.5.1.	Docker	56
4.5.2.	Docker jExam Infrastruktur	58
4.5.3.	Erreichte Ergebnisse	62
5.	Evaluation	63
5.1.	Erreichte Ziele	63
5.2.	Begrenzung und mögliche Lösungen	64
5.2.1.	Probleme beim UI-Testing	64
5.2.2.	Probleme bei der Initialisierung von Daten	64
5.2.3.	Leistungsprobleme	65
6.	Zusammenfassung und Ausblick	66
6.1.	Zusammenfassung	66
6.2.	Ausblick	67
	Literatur	68
A.	Appendix	71
A.1.	Dokumentation der jExam Testinfrastruktur	71
A.1.1.	Über die Testinfrastruktur	71
A.1.2.	Verwendete Technologien	71
A.1.3.	Einrichtung des Projekts	72
A.1.4.	jExam-Anwendungen starten	73
A.1.5.	Durchführung von Performancetests	73
A.1.6.	Durchführung von Sicherheitstests	74
A.1.7.	Durchführung von Seleniumtests	75
A.2.	Präsentation der grafischen Oberfläche der Berichtstools	77
A.3.	UML-Klassendiagramm und globale Struktur für einige Pakete der UI-Tests	78
A.4.	Einige Beispiele für Code Sources, die zum Verständnis beitragen . . .	79

Abbildungsverzeichnis

2.1. Kosten für die Behebung von Fehlern (Bugs) in verschiedenen Phasen (vgl. [17])	13
2.2. Vorteile der Testautomatisierung (vgl. [8], S.31)	16
3.1. Funktionsweise eines WebAutomationTesting-Frameworks (vgl. [37]) .	23
3.2. Frontend Quelltext von jExam 2009 und jExam new	25
4.1. Deserialisierung Diagram (vgl. [24])	33
4.2. Grafische Benutzeroberfläche von Zap	36
4.3. Bericht nach der Ausführung einer Zap Baseline	40
4.4. Funktionsweise von JMeter	42
4.5. Darstellung der jExam LoginPage mit dem Page Object Model	47
4.6. Projektstruktur von jExam Page Object Model	48
4.7. Vom Initializer erzeugte CSV-Daten	49
4.8. UML Diagramm für die Java Page-Objekte	50
4.9. StartRegistration Funktion	52
4.10. Extent Report HTML Page	55
4.11. jExam Docker Network Darstellung	58
4.12. jExam Launcher Testservice	59
4.13. jExam UI Testservice	60
4.14. jExam Penetration Testservice	61
4.15. jExam Performance Testservice	61
A.1. Globale Dateistruktur der jExam-Testinfrastruktur	72
A.2. Beispiel für einen JMeter-Bericht	74
A.3. Beispiel für einen ZAP Proxy-Bericht	75
A.4. TestUser Java Class	75
A.5. Grafische Benutzeroberfläche von Jmeter	77
A.6. UML-Diagramm des Pakets systemTests	78
A.7. Genereller Struktur der UI-Tests	78
A.8. UML-Diagramm des Pakets pages	79

Tabellenverzeichnis

4.1. OWASP TOP 10 2017	29
----------------------------------	----

Abkürzungsverzeichnis

URL Uniform Resource Locator

W3C World Wide Web Consortium

GUI Graphical user interface

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

API Application User Interface

UI User Interface

GUI Graphical User Interface

SEO Search Engine Optimisation

OWASP Open Web Application Security Project

ZAP Zed Attack Proxy

XML eXtensible Markup Language

XEE XML External Entity

DTD Document Type Definition

DAST Dynamic Application Security Testing

ZAP Zed Attack Proxy

AJAX Asynchronous JavaScript And XML

Glossar

Backend bezeichnet in der Regel den funktionalen Teil eines digitalen Produkts wie beispielsweise Webseiten oder Apps und bezieht sich meist auf Client Server.. 14

Bottleneck In der Softwareentwicklung spricht man von einem Bottleneck (Engpass), wenn die Kapazität einer Anwendung oder eines Computersystems durch eine einzelne Komponente begrenzt wird, wie der Hals einer Flasche, der den gesamten Wasserfluss verlangsamt. Der Bottleneck (Engpass) hat den geringsten Durchsatz aller Teile des Transaktionspfads.. 19

Container standardisierte ausführbare Komponenten.. 56

crawlen Prozess der automatischen Entdeckung neuer Ressourcen in einer Anwendung. 37

CVE Common Vulnerabilities and Exposures auf Englisch, bezeichnet eine öffentliche Liste von IT-Sicherheitslücken. Wenn man von einer CVE spricht, bezieht man sich in der Regel auf eine Sicherheitslücke, der eine CVE-Kennung zugewiesen wurde.. 26

Debugging Debugging ist das Auffinden und Beheben von Fehlern in einem Programm, die den korrekten Betrieb einer Software verhindern.. 18

Frontend Benutzeroberfläche, die z. B. in Form einer grafischen Benutzeroberfläche (englisch graphical user interface, kurz GUI) oder mittels Bildschirmmasken implementiert sein kann.. 14, 24

Headless Headless bedeutet, dass die Selenium-Tests mit einem Headless-Browser ausgeführt werden. Er funktioniert wie ein typischer Browser, jedoch ohne Benutzeroberfläche, was ihn für automatisierte Tests hervorragend geeignet macht.. 55

jExam 2009 Alte Version von jExam. 10, 14, 19, 24, 43, 54, 58, 59, 61, 63–65

jExam New Neue Version von jExam, die im Moment in Entwicklung steht. 10, 19, 24, 43, 54, 58, 59, 61, 63–65

Produktion Produktionsumgebung ist ein Begriff, der vor allem von Entwicklern verwendet wird, um die Umgebung zu beschreiben, in der Software und andere Produkte tatsächlich für ihre beabsichtigte Verwendung durch Endbenutzer in Betrieb genommen werden.. 14, 19

Selenese Programmiersprache, die zum Schreiben von Selenium-Befehlen verwendet wird. Diese Selenium-Befehle werden dann zum Testen von Webanwendungen verwendet. Basierend auf den HTML-Tags der UI-Elemente kann man deren Existenz überprüfen. Befehle helfen Selenium zu verstehen, welche Aktionen oder Operationen ausgeführt werden sollen.. 44

Test-Suite Die Zusammenstellung (Aggregation) mehrerer Testfälle für den Test einer Komponente oder eines Systems, bei der Nachbedingungen des einen Tests als Vorbedingungen des folgenden Tests genutzt werden können.. 13

1. Einleitung

Das erste Kapitel dieser Arbeit befasst sich mit der grundlegenden Motivation und definiert dann das allgemeine Problem, das gelöst werden soll. Es folgt eine Erläuterung der Zielsetzung und zum Schluss wird der Aufbau der Arbeit im Detail beschrieben.

1.1. Motivation

Die meisten Entwickler finden Testen langweilig und denken, dass Testen den Entwicklungsprozess verlangsamt. Nach Murugesan wird es oft nur zweitrangig berücksichtigt, insbesondere in den letzten und entscheidenden Phasen des Softwareentwicklungsprozesses (vgl. [20], S.112). Jedoch ist diese Herangehensweise fehlerbehaftet und kostspielig. "Testing is not a PIT, it is a LADDER ...!" (Anand vgl. [3], S.02). Nach dieser Auffassung von Anand geht es beim Testen vielmehr darum, Fehler und Situationen zu vermeiden, die den Entwicklungsprozess einer Anwendung verlangsamen könnten. Dadurch wird die Qualität der Software erhöht und die zeitlichen Vorgaben werden eingehalten. Dies reduziert die Produktionskosten.

An der Technischen Universität Dresden steht den Studierenden der Informatik eine Plattform für die Anmeldung zu Lehrveranstaltungen und Prüfungen sowie für die Bekanntgabe der Prüfungsergebnisse zur Verfügung. Es handelt sich um ein studentisches Projekt, das im Laufe der Zeit mit relativ wenig Personal und einer hohen Personalfuktuation entwickelt wurde. Das erschwert die Wartung der Plattform und hat das Entwicklerteam dazu gezwungen, sich auf Entwicklungs- und Wartungsaufgaben zu konzentrieren. Aus diesem Grund liegt die Plattform im Moment ohne automatische Tests vor.

Die Entwicklung der Plattform in ihrer jetzigen Form basiert auf Technologien aus dem Jahr 2009 und früher. Aufgrund der zwischenzeitlichen Entwicklung dieser Kern-technologien sowie der potenziellen Sicherheitsrisiken, die in den älteren Versionen vorhanden sein könnten, besteht ein Bedarf an Weiterentwicklung. Diese Arbeit wird derzeit von einer studentischen Hilfskraft aus der jExam-Gruppe durchgeführt.

jExam ist jedoch eine kritische Plattform für Studenten, zum einen, weil sie vertrauliche Informationen über Studenten und ihr Studium enthält, und zum anderen, weil

1. Einleitung

sie zu Beginn des Semesters für die Wahl der Fächer und am Ende des Semesters für die Anmeldung zu Prüfungen unerlässlich ist. Um möglichen Fehlern vorzubeugen und eine gute Wartung der Plattform zu gewährleisten, ist es notwendig, zu testen.

1.2. Zielsetzung und Abgrenzung

Wie Azeem Uddin sagte: "Testen bedeutet herauszufinden, wie gut etwas funktioniert" (vgl. [3], S.02). Die aktuelle Version von jExam (die nun als **jExam 2009** beschrieben wird) und die neue Version (**jExam New**), die derzeit entwickelt wird, laufen ohne Tests. Das Testen von jExam 2009 wird dazu beitragen, die Plattform besser zu warten, die Sicherheit zu erhöhen, die Qualität der Anwendungen zu verbessern und mögliche Fehler zu vermeiden (vgl. [29], S.21), während auf die Inbetriebnahme von jExam New noch aussteht.

Beide Versionen werden genau die gleichen Funktionalitäten haben, da die neue Version (jExam New) nur eine Kopie der alten Version mit modernen Technologien ist. So kann eine Testinfrastruktur geschaffen werden, die mit beiden Versionen kompatibel ist. Damit besteht die Möglichkeit zu testen, ob jExam New genau die gleichen Funktionalitäten wie jExam 2009 hat und gleichzeitig alle Vorteile einer getesteten Webanwendung zu haben.

Wegen des Mangels an Arbeitskräften im Entwicklungsteam ist es notwendig, eine Infrastruktur zu schaffen, um das Schreiben zu erleichtern und die Entwicklungszeit der Tests zu beschleunigen. Diese Arbeit zielt darauf ab, das jExam-Entwicklungsteam bei der Wartung und schnellen Funktionsprüfung der Plattform zu unterstützen. Dafür müssen die folgenden Ziele umgesetzt werden:

1. Entwicklung einer Testsuite für die Neuentwicklung von jExam-Web. Dabei sollten mindestens folgende Funktionen abgedeckt sein:
 - a) Login
 - b) Registrierung
 - c) Abruf der Noten in der Übersicht und als pdf
 - d) Einschreibung in Prüfungen
 - e) Einschreibung in Lehrveranstaltungen
 - f) Einschreibung in Seminargruppen
2. Hohe Wartbarkeit der Testsuite
3. Bereitstellung einer automatisierten Infrastruktur
4. Erweiterung der Testsuite um Sicherheitstests
5. Erweiterung der Testsuite um Performancetests
6. Dokumentation

1.3. Aufbau der Arbeit

Diese Arbeit ist in fünf Teile gegliedert. Zunächst wird das zum Verständnis der Arbeit erforderliche Grundwissen erläutert. Darin werden verschiedene Konzepte beschrieben, darunter Webanwendungen und die Grundlagen des Softwaretestens, in denen grundlegende Fragen definiert werden. Außerdem erfolgt die Vorstellung von verschiedenen Testarten und -methoden.

Dann folgt Kapitel drei, das der Beschreibung der zu testenden Plattformen gewidmet ist, gefolgt von einer Phase der Analyse der Probleme, die die Entwicklung von Tests erschweren und sogar die Verwendung bestimmter Werkzeuge ausschließen können. Am Ende desselben Kapitels werden verschiedene Lösungsvorschläge erörtert.

Als nächstes kommt in Kapitel vier der praktische Teil, in dem die verschiedenen Technologien und Werkzeuge beschrieben werden, die während der Implementierung verwendet wurden, wobei die Gründe für ihre Verwendung dargelegt werden, gefolgt von einer detaillierten Beschreibung des Entwicklungsprozesses und der verwendeten Architektur (Entwurfsmuster, Softwarearchitektur ...)

Die Ergebnisse des entwickelten Tools werden in Kapitel fünf vorgestellt und im Hinblick auf verschiedene Aspekte wie Leistung und Fehler evaluiert. Schließlich werden in Kapitel sechs Perspektiven für die Erweiterung der vorgestellten Arbeit sowie die vollständige Dokumentation des Tools vorgestellt.

2. Grundlagen

In diesem Kapitel werden die notwendigen Definitionen und Methoden erläutert, die für das Verständnis der Arbeit von Bedeutung sind. Zunächst werden die Ziele und Grenzen des Softwaretestens definiert. Dann folgt eine kurze Einführung in den Begriff der Webanwendung und die Erklärung des Konzeptes der Testumgebung. Um mehr über die Vorteile der Automatisierung zu erfahren, wird im nächsten Teil das Konzept der Testautomatisierung erläutert, gefolgt von der Vorstellung verschiedener Testmethoden und -ansätze. Das Konzept des Testens im Bereich der Softwareentwicklung ist umfangreich und kann in dieser Arbeit nicht vollständig behandelt werden. Daher werden einige Testarten am Ende vorgestellt.

2.1. Ziele und Grenzen des Softwaretestens

Die Standarddefinition des Testens nach dem ANSI/IEEE 1059-Standard besagt, dass Testen der Prozess der Analyse eines Softwareobjekts ist, um Unterschiede zwischen bestehenden und erforderlichen Bedingungen (d.h. Defekte/Fehler/Bugs) zu erkennen und die Eigenschaften des Softwareobjekts zu bewerten (vgl. [31], S.07). Das Softwaretesten ist daher eine Methode, um zu überprüfen, ob das Softwareprodukt den erwarteten Anforderungen entspricht und um sicherzustellen, dass es frei von Fehlern ist.

Im 21. Jahrhundert ist der Einsatz von Software und Anwendungen weit verbreitet und in allen Bereichen vertreten. Die Gesamtmenge der weltweit erstellten, erfassten, kopierten und verbrauchten Daten ist laut Statista (vgl. [32]) bis 2020 rasant auf 64,2 Zettabyte (2^{70}) angestiegen. Die Nutzer vertrauen ihre sensiblen und privaten Daten Plattformen an, deren Aufgabe ist es, sie zu schützen. Testen ist wichtig, weil Softwarefehler teuer oder sogar gefährlich sein können. Sie können finanzielle und menschliche Verluste verursachen, und die Geschichte ist voll von solchen Beispielen:

1. Im Mai 1996 führte ein Softwarefehler dazu, dass den Konten von 823 Kunden einer großen US-Bank 920 Millionen US-Dollar gutgeschrieben wurden (vgl. [9]).
2. Der Airbus A300 der China Airlines stürzte am 26. April aufgrund eines Softwarefehlers ab, 1994, bei dem 264 Unschuldige ums Leben kamen (vgl. [35]).

Das Testen einer Anwendung hat viele Vorteile. Zu den wichtigsten gehören die folgenden:

Kosteneffektivität: Das rechtzeitige Testen eines IT-Projekts hilft auf lange Sicht Geld zu sparen. Wenn die Fehler bereits in der frühen Phase des Softwaretests entdeckt werden, kostet es weniger, sie zu beheben. Es ist besser, mit dem Testen früher zu beginnen und es in jeder Phase des Lebenszyklus der Softwareentwicklung einzuführen. Regelmäßige Tests sind erforderlich (vgl. [17], S.53), um sicherzustellen, dass die Anwendung gemäß den Anforderungen entwickelt wird.

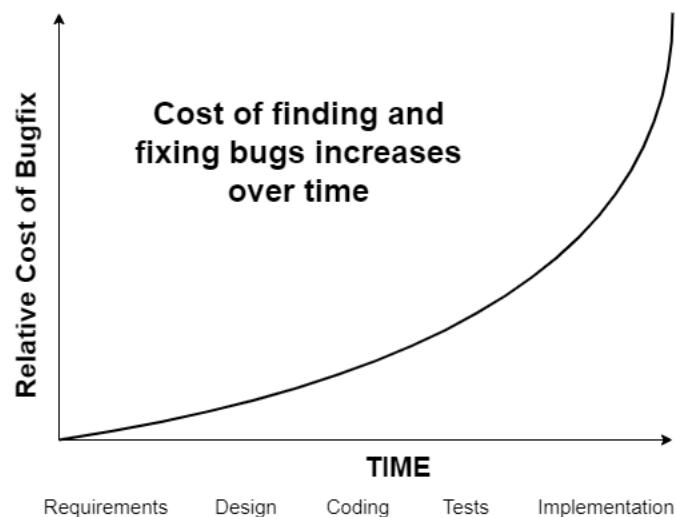


Abbildung 2.1. Kosten für die Behebung von Fehlern (Bugs) in verschiedenen Phasen (vgl. [17])

Erhöhung der Sicherheit: Sicherheit ist der anfälligste und sensibelste Teil des Softwaretestens. Durch Testen wird sichergestellt, dass die Anwendung über einen minimalen Schutz verfügt. Testen hilft dabei, Risiken und Probleme früher zu beseitigen. So wird die Anwendung für Nutzer attraktiv, die vertrauenswürdige Produkte suchen (vgl. [29], S.09).

Produktqualität: Sie ist eine wesentliche Voraussetzung für jedes Softwareprodukt. Zu den sechs Gruppen von Software-Qualitätsindikatoren in der ISO-Norm 9126 (vgl. [2]) gehört die Wartbarkeit, zu der auch die Untergruppe Testbarkeit gehört. Durch Testen kann die Qualität einer Anwendung sowie ihre Wartbarkeit erhöht werden und so wird dem Kunden sichergestellt, dass ein Qualitätsprodukt geliefert wird.

Aus diesen Gründen ist das Testen ein integraler Bestandteil des Softwareentwicklungsprozesses, jedoch hat es Grenzen. Testen dient nur dazu, das Vorhandensein von potentiellen Fehlern aufzudecken. Aber es kann nicht sicherstellen, dass die Software keine Fehler oder Bugs enthält (vgl. [17], S.55). Dazu können Tests nicht nachweisen, dass ein Produkt unter allen Bedingungen richtig funktioniert, sondern nur, dass es unter bestimmten Bedingungen nicht richtig funktioniert (vgl. [17], S.56).

Da das Ziel dieser Arbeit darin besteht, eine Test-Suite für eine Webanwendung einzurichten, ist es wichtig zu definieren, was mit Webanwendung eigentlich gemeint ist.

2.2. Webanwendungen

Das weite Feld der Softwareentwicklung umfasst auch die Entwicklung von Webanwendungen. Lange Zeit wurden Anwendungen als kompakte, installierbare Programme verkauft. Dabei handelt es sich um die so genannten klassischen Anwendungen oder Computeranwendungen, die lokal auf einem Computer, Mobiltelefon oder Tablet installiert werden müssen. Im Gegensatz zu herkömmlichen Anwendungen werden Webanwendungen nicht lokal auf dem Gerät des Nutzers installiert, sondern auf einem Server, so dass sie über eine bestimmte URL zugänglich sind.

Nach Kappel et al. ist eine Webanwendung ein Softwaresystem, das auf den Spezifikationen des World Wide Web Consortiums (W3C) basiert und Webressourcen bereitstellt, die über eine Benutzeroberfläche wie einen Webbrowser genutzt werden können (vgl. [15], S.02). Für den Betrieb einer Webanwendung werden mehrere Computersystemkomponenten benötigt. Erstens das Frontend, das die grafische Oberfläche bezeichnet, mit der der Benutzer interagiert, gefolgt vom Backend, das die Logik der Anwendung enthält (z.B. Datenbanken, Cloud).

Der Benutzer greift auf die Webanwendung über einen Computer zu, der als Client bezeichnet wird. Der Client sendet eine oder mehrere Anfragen über das Internet oder Intranet via HTTP oder HTTPS an einen anderen Computer (Server), auf dem die Webanwendung läuft. Der Server nimmt dann die HTTP-Anfragen entgegen und verarbeitet sie. Je nach Anfrage werden die angeforderten Daten entweder aus der Datenbank abgerufen oder gespeichert. Die verarbeiteten Daten werden dann vom Server in einer entsprechenden Antwort (HTTP-Response) an den Client zurückgesendet und im Webbrowser angezeigt. Dies ist die grundsätzliche Funktion von Webanwendungen.

Um Webanwendungen zu testen, ist es wichtig, eine Testumgebung zu schaffen. Diese ermöglicht es dem Tester, alle möglichen Schwachstellen in einer Webanwendung zu untersuchen, ohne die Anwendung zu gefährden, wenn sie bereits in Produktion ist.

2.3. Testumgebung

Im Moment ist die alte Version von jExam (jExam 2009) in Produktion, weil sie stabil und funktional ist. Die Entwickler des Teams sorgen dafür, dass die Funktionen der Anwendung ordnungsgemäß funktionieren. Dies ist bereits eine Art manueller Test. Die neue Version, die derzeit entwickelt wird, wurde noch nicht vollständig getestet. Daher ist sie von unbekannter Qualität und kann nicht als stabil für die Produktion gelten. Sie sollte nicht in die Produktion aufgenommen werden, damit die Benutzer sie testen können. Wenn die Entwickler dies täten, würden sie Schwachstellen aufdecken, die von böswilligen Personen genutzt werden könnten, um die Sicherheit der Nutzerdaten zu gefährden. Bevor eine Anwendung in Produktion geht, sollte sie getestet werden. Da dies nicht in der Produktion geschehen sollte, ist es notwendig, eine Umgebung zu schaffen, in der die Anwendung getestet werden kann.

Laut Everett eine Testumgebung ist eine Einrichtung, die Hardware, Instrumente, Simulatoren, Software-Tools und andere unterstützende Elemente enthält, die für die Durchführung eines Tests erforderlich sind, d.h. sie ermöglicht es, die Anwendung zu

testen, ohne den Kunden zu beeinträchtigen (vgl. [29], S. 150-152). Die Einrichtung einer Testumgebung hat eine Reihe von Vorteilen :

1. Die Nutzer nicht mit einer Anwendung von zweifelhafter Qualität zur Verfügung stellen. Ein Qualitätsabfall ist oft sehr nachteilig für eine Anwendung. Die Nutzer könnten sich beschweren, dass die alte Version besser ist und die ganze Arbeit der Entwickler, die die neue Version entwickelt haben, abwerten.
2. Die Einbeziehung aller möglichen Szenarien in die Tests. Das erhöht die Zuverlässigkeit der Software und hilft auch dabei, fehlende Implementierungen der Software zu finden. In der Produktion wäre dies eine komplizierte Aufgabe, da sich die Daten und die Anwendung ständig ändern.
3. Die Senkung der Produktionskosten durch frühzeitige Erkennung von Fehlern.

Testumgebungen sind für jede Softwareentwicklung notwendig. Sie ermöglichen es, das Produkt zu testen, bevor es auf den Markt kommt, ohne ein Risiko für die Produktion einzugehen. Wenn sie jedoch wirklich wirksam sein sollen, dürfen sie die Entwicklung nicht behindern, sondern müssen auch zuverlässig sein. Aus diesem Grund muss diesen Umgebungen besondere Aufmerksamkeit geschenkt werden. Die endgültige Qualität des Produkts hängt stark von der Qualität der Testumgebung ab (vgl. [29], S. 152).

2.4. Testautomatisierung

Nach Sharma et al. ist die Testautomatisierung eine Technik, bei der Skripte geschrieben werden, um einen manuellen Testprozess zu automatisieren (vgl. [28], S.909). Dabei werden vordefinierte Tests durchgeführt, um verschiedene Funktionen in einer Anwendung zu überprüfen. Dies bedeutet, dass Tools und Testskripte verwendet werden, um verschiedene Zustände zu erzeugen und Daten vorzubereiten. Anschließend wird eine Reihe von Schritten ausgeführt, um ein Szenario zu validieren. Auf diese Weise können die Tester feststellen, ob eine Anwendung wie erwartet funktioniert oder nicht. Entwicklungs- und Testteams entscheiden sich aus mehreren Gründen für die Testautomatisierung. Zu den wichtigsten gehören:

Zeit: Manuelle Tests sind langsam und können mit vielen Entwicklungsprozessen nicht mithalten (vgl. [28], S.910).

Kosten: Manuelle Tests sind ressourcenintensiv und kostspielig (vgl. [28], S.910).

Genauigkeit: Manuelle Tests sind bei der Durchführung sich wiederholender Aufgaben fehleranfällig. Umgekehrt verringert die Automatisierung die Wahrscheinlichkeit dieser Fehler (vgl. [28], S.910).

Umfang: Bei der Durchführung komplexer Iterationen ist es schwierig, sich auf manuelle Tests zu verlassen (vgl. [28], S.910).

Laut dem Global Quality Report profitieren Unternehmen auf unterschiedliche Weise von automatisierten Tests. Rund 60% der Unternehmen gaben an, dass sich die Fähigkeit zur Erkennung von Anwendungsfehlern durch eine höhere Testabdeckung verbessert hat (vgl. [8], S.30). Weitere 57% stellten fest, dass die Wiederverwendung von

Benefits of test automation

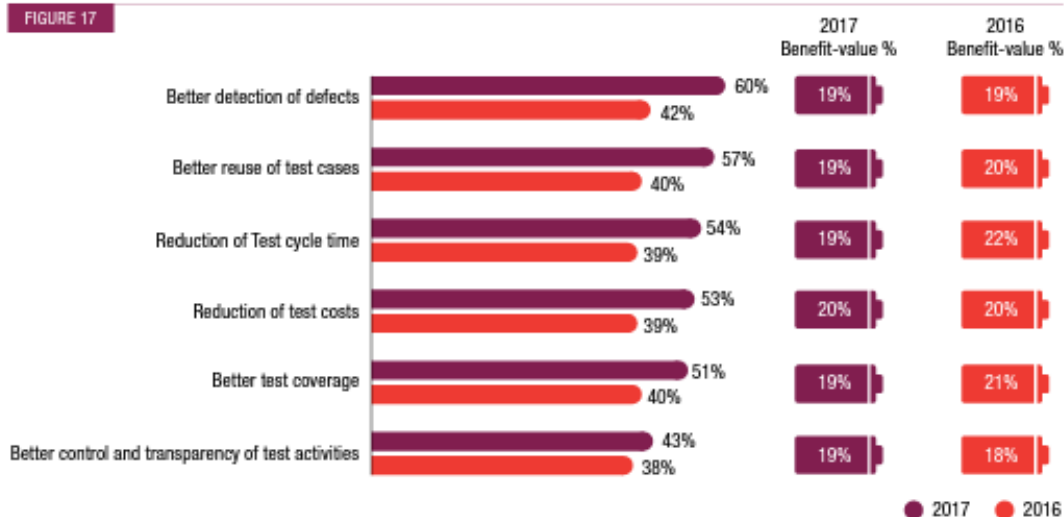


Abbildung 2.2. Vorteile der Testautomatisierung (vgl. [8], S.31)

Testfällen nach dem Einsatz der Automatisierung zunahm. Gleichzeitig verzeichneten 54% eine Verringerung des Zeitaufwands für Testzyklen (vgl. [8], S.31).

In den folgenden Abschnitten werden die verschiedenen Arten von Tests behandelt. Diese Tests können sowohl manuell als auch automatisch durchgeführt werden. Da der Schwerpunkt dieser Arbeit auf der Automatisierung liegt, wird nur dieser Aspekt betrachtet.

2.5. Testsmethoden und -ansätze

Softwareobjekte können auf unterschiedliche Weise getestet werden. Generell wird zwischen statischen und dynamischen Tests unterschieden. In den folgenden Fällen werden beide Konzepte in den folgenden Abschnitten ausführlicher dargestellt.

2.5.1. Statisches Testen

Statisches Testen ist eine Softwaretestmethode, bei der ein Programm zusammen mit den zugehörigen Dokumenten untersucht wird, ohne dass das Programm ausgeführt werden muss. Konkret besteht der Prozess aus der Prüfung schriftlicher Dokumente, die insgesamt einen Überblick über die zu testende Softwareanwendung geben. Zu den geprüften Dokumenten gehören Anforderungsspezifikationen, Designdokumente, Benutzerdokumente, Webseiteninhalte, Quellcode, Testfälle, Testdaten und Testskripte, Benutzerdokumente, Spezifikations- und Matrixdokumente. Statische Tests erleichtern die Kommunikation zwischen den Teams und vermitteln einen besseren Eindruck von den Qualitätsproblemen in der Software. Es reduziert nicht nur die Kosten in den frühen Entwicklungsphasen (in Bezug auf die Menge an Arbeit, die neu gemacht werden muss, um eventuelle Fehler zu korrigieren), sondern auch die Entwicklungszeit.

2.5.2. Dynamisches Testen

Dynamisches Testen ist eine Methode zur Bewertung der Durchführbarkeit eines Softwareprogramms durch Eingabe und Prüfung der Ausgabe. Die dynamische Methode erfordert, dass der Code kompiliert und ausgeführt wird. Dynamische Tests werden in zwei Kategorien unterteilt: Whitebox- und Blackboxtestverfahren.

Whitebox-Testverfahren

Whitebox-Testen ist eine Softwaretestmethode, bei der dem Tester die interne Struktur/das Design bekannt ist. Das Hauptziel vom Whitebox-Testen ist die Überprüfung der Korrektheit der Software Anweisungen, Codepfade, Bedingungen, Schleifen und Datenflüsse. Dieses Ziel wird oft als logische Abdeckung bezeichnet (vgl. [29], S.107).

Blackbox-Testverfahren

Blackbox-Testen ist eine Testmethode, bei der die interne Struktur/der Code/das Design dem Tester nicht bekannt ist. Das Hauptziel dieses Tests ist es, die Funktionalität des zu testenden Systems zu überprüfen, und diese Art von Tests erfordert die Ausführung der kompletten Testsuite (vgl. [29], S.112).

Die Blackbox-Methode ist diejenige, die in dieser Arbeit entwickelt wird. Sie selbst ist in mehrere Arten unterteilt, auf die im nächsten Kapitel eingegangen wird.

2.6. Verschiedene Testarten

Bei der Blackbox-Methode gibt es zwei Hauptarten von Testen: funktionale Tests und nicht-funktionale Tests.

2.6.1. Funktionale Tests

Funktionale Tests werden durchgeführt, um zu überprüfen, ob alle entwickelten Funktionen mit den funktionalen Spezifikationen übereinstimmen. Dies wird durch die Ausführung der funktionalen Testfälle erreicht. In der Funktionstestphase wird das System getestet, indem Eingaben gemacht, Ausgaben überprüft und die tatsächlichen Ergebnisse mit den erwarteten Ergebnissen verglichen werden. Bei diesen Tests werden Benutzeroberfläche, APIs, Datenbank, Sicherheit, Client/Server-Kommunikation und andere Funktionen der zu testenden Anwendung überprüft. Die Tests können entweder manuell oder durch Automatisierung durchgeführt werden.

Unit-Tests

James Whittaker erklärt, dass Unit-Test einzelne Softwarekomponenten oder eine Sammlung von Komponenten testet (vgl. [38], S.70-79). Unit-Testing ist die erste Stufe des Softwaretests, bei der einzelne Komponenten eines Softwarepakets getestet werden, während der Rest des Systems ignoriert wird. Auch Modul- oder Komponententest genannt, wird Unit-Test während der Entwicklung einer Anwendung durchgeführt, um zu prüfen, ob die einzelnen Einheiten oder Module einer Anwendung ordnungsgemäß funktionieren.

Das Ziel von Unit-Tests ist es, Probleme in einem frühen Stadium des Entwicklungszyklus zu finden. Dadurch werden die Testkosten gesenkt (die Kosten für das frühzeitige Auffinden eines Fehlers sind wesentlich geringer als die Kosten für das spätere Auffinden). Sie reduzieren Fehler bei der Änderung bestehender Funktionen, so dass sie leicht gefunden und behoben werden können. Dies vereinfacht den Debugging-Prozess erheblich.

Integration-Tests

Es ist eine Erweiterung des Unit-Tests. Beim Integrationstest wird die Konnektivität oder der Datentransfer zwischen den einzelnen getesteten Modulen (unit tested modules) getestet. Nach Leung und White sind Integrationstests die Tests, die durchgeführt werden, wenn alle einzelnen Module zu einem funktionierenden Programm kombiniert werden (vgl. [18], S.290). Das Testen erfolgt auf Modulebene und nicht auf Anweisungsebene wie beim Unit-Test. Beim Integrationstest liegt der Schwerpunkt auf den Interaktionen zwischen den Modulen und ihren Schnittstellen.

Mit Integrationstests wird überprüft, ob das funktionale und nicht-funktionale Verhalten der Schnittstellen dem Softwaredesign und -spezifikationen entspricht. Dies führt dazu, dass Fehler daran gehindert werden, in höhere Teststufen zu gelangen.

System-Tests

Systemtests sind eine Stufe der Softwaretests über den Integrationstests. Laut Briand et al. werden sie an einer vollständigen und voll integrierten Anwendung durchgeführt, um die Übereinstimmung des Systems mit den spezifizierten Anforderungen zu bewerten (vgl. [7], S.10). Bei dieser Art von funktionalen Tests validieren die Tester das vollständige und integrierte Softwarepaket. Damit wird sichergestellt, dass es den Anforderungen entspricht. Bei Bedarf können die Tester Feedback zur Funktionalität und Leistung der App oder Website geben, ohne vorher zu wissen, wie sie programmiert wurde. Dies hilft den Teams bei der Entwicklung von Testfällen, die in Zukunft verwendet werden sollen. Systemtests werden auch als End-to-End-Tests bezeichnet.

Automatisierte Benutzeroberfläche-Tests (UI Automation Testing)

Die erste Interaktion zwischen einem Benutzer und einer Software findet über eine grafische Benutzeroberfläche (GUI acronym) statt. Beim UI-Testing wird überprüft, ob die Endbenutzeroberfläche korrekt funktioniert. Bei der Durchführung von UI-Tests wird geprüft, ob jedes Stückchen Logik, jede UI-Funktion oder jeder Aktionsablauf wie erwartet funktioniert. Hier konzentrieren sich die Tester auf die Validierung jedes Klicks auf eine Schaltfläche, der Dateneingabe, der Navigation, der Berechnung von Werten und anderer Funktionalitäten, die für die Benutzerinteraktion verwendet werden.

UI Automation Testing ist eine Technik, bei der diese Testprozesse mithilfe eines Automatisierungstools durchgeführt werden. Anstatt dass sich die Tester durch die Anwendung klicken, um die Daten- und Aktionsflüsse visuell zu überprüfen, werden für jeden Testfall Testskripte geschrieben. Anschließend wird eine Reihe von Schritten hinzugefügt, die bei der Überprüfung der Daten zu befolgen sind. Der Prozess der UI-Automatisierungstests vereinfacht die Erstellung von UI-Tests, die Ausführung der Tests und die Anzeige der Ergebnisse (vgl.[23]). Es ermöglicht Testern, die Interaktion einer Anwendung mit den Endnutzern zu simulieren und zu testen. Weitere Vorteile sind die Automatisierung aller Testaktivitäten für die zu testende Anwendung und die Integration von Benutzerschnittstellentests in den Entwicklungsprozess.

Automatisiertes Testen ist eine der Funktionen, die im Mittelpunkt dieser Arbeit stehen. Es ist daher wichtig, dieses Konzept zu verstehen, um weiterzukommen.

2.6.2. Nichtfunktionale Tests

Nichtfunktionales Testen ist das Testen einer Softwareanwendung oder eines Systems auf seine nichtfunktionalen Anforderungen. Es dient dazu, die Bereitschaft eines Systems in Bezug auf nichtfunktionale Parameter zu testen, die bei funktionalen Tests nicht berücksichtigt werden. Nichtfunktionale Tests beziehen sich auf verschiedene Aspekte der Software wie Leistung, Belastung, Stress, Skalierbarkeit, Sicherheit, Kompatibilität. Der Schwerpunkt liegt auf der Verbesserung der Benutzererfahrung.

Es gibt viele Arten von Nichtfunktionalen Tests, von denen einige in den folgenden Abschnitten behandelt werden:

Performancetests

Nach Vokolos et al. sind Performancetests die Gesamtheit aller Aktivitäten, die an der Bewertung der Leistung einer Software in Produktion beteiligt sind ([36], S. 80). Diese Leistung wird aus der Sicht des Benutzers bewertet und in der Regel in Form von Durchsatz, Reaktionszeit auf Stimuli oder einer Kombination aus beiden bewertet. Performancetests können auch verwendet werden, um den Grad der Verfügbarkeit bzw. Stabilität der Software zu bewerten. Für viele Anwendungen in der Telekommunikation oder im medizinischen Bereich ist es beispielsweise von entscheidender Bedeutung, dass das System immer verfügbar ist.

Performancetests sind entscheidend, um festzustellen, ob eine Anwendung die Leistungsanforderungen erfüllt (z.B. muss das System bis zu 1 000 gleichzeitige Benutzer verwalten können). Sie helfen auch dabei, Engpässe (Bottleneck) in einer Anwendung zu lokalisieren. Sie werden für den Vergleich von zwei oder mehr Systemen verwendet, um das leistungsfähigste zu identifizieren (z. B. jExam 2009 und jExam New). Es ermöglicht auch eine effektive Messung der Stabilität in Bezug auf den Datenverkehr. Performancetests unterteilen sich in weitere Unterkategorien:

Lasttests: Sie sind eine Art von Performancetests, bei denen die Anwendung auf ihre Leistung bei normaler und Spitzenbelastung getestet wird. Die Leistung einer Anwendung wird im Hinblick auf ihre Reaktion auf Benutzeranfragen und ihre Fähigkeit, innerhalb einer akzeptierten Toleranz bei unterschiedlichen Benutzerlasten konsistent zu reagieren, überprüft.

Stresstests: Sie werden eingesetzt, um Wege zu finden, das System zu brechen. Der Test liefert auch den Bereich der maximalen Belastung, die das System aushalten kann. In der Regel wird beim Stresstest ein schrittweiser Ansatz verfolgt, bei dem die Belastung schrittweise erhöht wird. Der Test wird mit einer Last begonnen, für die die Anwendung bereits getestet wurde. Dann wird die Last langsam erhöht, um das System zu belasten. Der Punkt, an dem wir feststellen, dass die Server nicht mehr auf die Anfragen reagieren, wird als Sollbruchstelle betrachtet.

Dies sind die einzigen beiden Unterkategorien von Performancetests, die in dieser Arbeit verwendet werden.

Penetrationstests

Ein Penetrationstest ist ein Versuch, die Sicherheit einer IT-Infrastruktur zu bewerten, indem auf sichere Weise (in einer gesicherten Umgebung und unter sicheren Umständen) versucht wird, Schwachstellen auszunutzen. Diese Schwachstellen können in Betriebssystemen, Diensten und Anwendungsfehlern, unsachgemäßen Konfigurationen oder riskantem Verhalten der Endbenutzer bestehen. Solche Bewertungen sind auch nützlich, um die Wirksamkeit von Verteidigungsmechanismen und die Einhaltung von Sicherheitsrichtlinien durch die Endbenutzer zu überprüfen.

Penetrationstests werden in der Regel mit manuellen oder automatisierten Technologien durchgeführt, um systematisch Server, Endpunkte, Webanwendungen, drahtlose Netzwerke, Netzwerkgeräte, mobile Geräte und andere potenzielle Angriffspunkte zu überprüfen. Sobald die Schwachstellen in einem bestimmten System erfolgreich ausgenutzt wurden, können die Tester versuchen, das kompromittierte System für weitere Angriffe auf andere interne Ressourcen zu nutzen. Insbesondere indem sie versuchen, schrittweise höhere Sicherheitsstufen und einen tieferen Zugang zu elektronischen Ressourcen und Informationen über die Ausweitung von Berechtigungen zu erreichen.

Laut Brad Arkin et al. sind Penetrationstests die am häufigsten und am weitesten verbreiteten Best Practices im Bereich der Softwaresicherheit, was zum Teil daran liegt, dass es sich um eine attraktive Aktivität am Ende des Lebenszyklus handelt (vgl. [4], S.84). Sobald eine Anwendung fertiggestellt ist, sollen die Tester sie im Rahmen der Endabnahme Penetrationstests unterziehen. Der Hauptzweck von Penetrationstests besteht also darin, Schwachstellen in einem System zu identifizieren, um sie zu schließen. Dies ermöglicht es den Testern, die Sicherheit der Anwendung zu erhöhen und die Sicherheitsstrategie zu verbessern.

So wie funktionale Tests zeigen, wie gut eine Anwendung funktioniert, so sind auch nicht-funktionale Tests für die Benutzererfahrung von großer Bedeutung.

Funktionale Tests zeigen, wie gut eine Anwendung funktioniert. Aber auch nicht-funktionale Tests sind sehr wichtig, da sie helfen, bestimmte Kriterien der Nutzererfahrung zu messen, um diese zu verbessern.

2.7. Verschiedene Schritte zur Erstellung einer Testsuite

In der heutigen wettbewerbsorientierten und schnelllebigen Welt ist das Internet zu einem integralen Bestandteil des Lebens geworden. Heutzutage treffen die meisten ihre Entscheidungen, indem sie im Internet nach Informationen suchen. Das Hosten einer Website ist daher nicht mehr optional, sondern für alle Arten von Unternehmen obligatorisch. Es ist der erste Schritt, um auf dem Markt relevant zu werden und zu bleiben. Es reicht nicht mehr aus, nur eine Website zu haben. Ein Unternehmen muss eine Website entwickeln, die informativ, zugänglich und benutzerfreundlich ist. Um all diese Qualitäten zu erhalten, sollte die Website getestet werden, und dieser Prozess des Testens einer Website ist als Web-Testing bekannt. Um diese Aufgabe zu bewältigen, durchlaufen die Testerteams mehrere Schritte:

Erstellung eines Testplan: Ein Testplan ist ein Dokument, das den Umfang, die Vorgehensweise und den Zeitplan der geplanten Testaktivitäten festlegt (vgl. [29], S. 79). Das Hauptziel eines Testplans ist die Erstellung einer Dokumentation, die beschreibt,

wie der Tester überprüft, ob das System wie vorgesehen funktioniert. Das Dokument sollte beschreiben, was getestet werden muss, wie es getestet werden soll und wer dafür verantwortlich ist. Er ist die Grundlage für jede Testarbeit (vgl. [29], S. 79). Der Testplan enthält in der Regel die folgenden Informationen:

1. Das Gesamtziel des Testaufwands.
2. Einen detaillierten Überblick über die Art und Weise, wie die Tests durchgeführt werden.
3. Die zu prüfenden Funktionen, Anwendungen oder Komponenten.
4. Detaillierte Zeit- und Ressourcenzuweisungspläne für Tester und Entwickler während aller Testphasen.

Erstellung einer Teststrategie : Die Teststrategie beim Softwaretesten ist definiert als eine Reihe von Leitprinzipien, die das Testdesign bestimmen und regeln, wie der Softwaretestprozess durchgeführt wird. Das Ziel der Teststrategie ist es, einen systematischen Ansatz für den Softwaretestprozess zu liefern, um die Qualität, Rückverfolgbarkeit, Zuverlässigkeit und bessere Planung zu gewährleisten. Die Teststrategie wird sehr oft mit dem Testplan verwechselt. Es gibt jedoch einen bemerkenswerten Unterschied zwischen den beiden. Der Testplan konzentriert sich auf die Organisation und das Management der Ressourcen sowie die Definition der Ziele, während die Teststrategie sich auf die Umsetzung und die Testtechniken konzentriert. Beide helfen bei der Planung, aber auf unterschiedlicher Ebene.

Testentwurf und implementierung : Der Testentwurf ist ein Prozess, der beschreibt, "wie" die Tests durchgeführt werden sollen und das Schreiben von Testsuiten für das Testen einer Software. Er umfasst Prozesse zur Identifizierung von Testfällen durch Aufzählung von Schritten der definierten Testbedingungen. Die in der Teststrategie oder dem Testplan definierten Testtechniken werden für die Aufzählung der Schritte verwendet (vgl. [29], S. 46).

Je nach den Zielen, die im Testplan festgelegt wurden, konzentrieren sich die Entwickler auf bestimmte Arten von Tests, z. B. funktionale Tests, Performancetests oder Sicherheitstests. Für diese Arbeit wurde vorab ein Testplan erstellt. Aus diesem Grund wird der Schwerpunkt auf der Implementierung von Tests und der Einrichtung der Testinfrastruktur liegen.

3. Problemanalyse

Nachdem die für das Verständnis notwendigen Grundlagen geklärt wurden, wird in diesem Kapitel eine Analyse der jExam-Anwendung vorgenommen. Ziel dieses Abschnitts ist es, die Anwendung zu untersuchen und sich das nötige Wissen anzueignen, um eine vollautomatische Testsuite einrichten zu können.

3.1. Verwandte Arbeiten

In der Literatur gibt es mehrere Artikel, die Ansätze zur Automatisierung des Testens von Webanwendungen vorstellen (vgl. [26] und [37]). Die meisten dieser Ansätze folgen den Schritten zum Schreiben von UI-Tests, die in Abschnitt 2.7 beschrieben wurde. Für die Erstellung eines Test Automation Framework mit Selenium wählten Fei Wang et al. (vgl. [37]) zunächst die verschiedenen Webbrowser aus, die sie für die Entwicklung der Tests verwenden wollten. Anschließend erstellten sie eine Testumgebung. Diese Umgebung sollte mit generierten Testdaten erstellt werden. Dann folgte die Phase des Designs von Testfällen, die sie in ein Set von Aktionen übersetzen werden. Diese Aktionen werden dann in verschiedene Befehle/Skripte übersetzt, je nach dem, welches Tool zum Testen verwendet wird (in diesem Fall Selenium). Danach werden die Aktionen von dem verwendeten Testwerkzeug ausgeführt und ein Testbericht erstellt.

Die letzte Phase besteht aus dem Testen des Test-Frameworks und seiner Infrastruktur. Die Leistung in Bezug auf Wiederverwendbarkeit, Erweiterbarkeit und Kompatibilität muss bewertet werden. Um diese Tests durchzuführen, empfehlen Fei Wang et al. die Verwendung unterschiedlicher Testdaten bei der Vorbereitung der Testfelder und die Beobachtung der erzielten Ergebnisse. Die Ergebnisse der Evaluierung des webbasierten Testautomatisierungs-Frameworks haben gezeigt, dass das Framework den Testern ein Werkzeug gibt, mit dem sie bequem verschiedene Browser für den Test von Webanwendungen konfigurieren, eine große Menge an Daten für den Leistungstest vorbereiten können, indem sie die Datenressourcen direkt konfigurieren, einen Testtyp einfach konfigurieren, indem sie einen Testtyp modifizieren, und zwischen verschiedenen Tests umschalten können, ohne irgendwelche Elemente im Zusammenhang mit den Testfällen zu verändern (vgl. [37]).

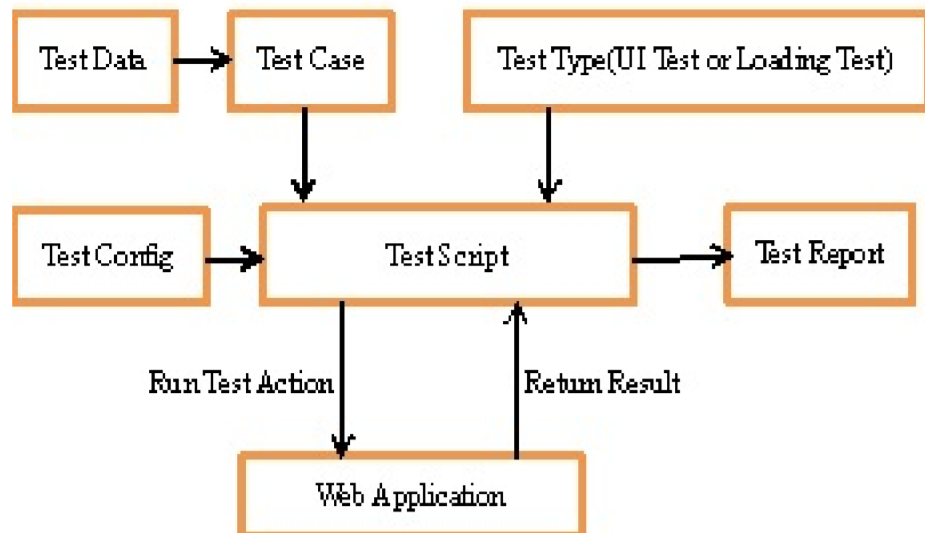


Abbildung 3.1. Funktionsweise eines WebAutomationTesting-Frameworks (vgl. [37])

3.2. Beschreibung von jExam

jExam ist eine in der Programmiersprache Java entwickelte Software, die seit dem Jahr 2000 an der Fakultät Informatik der Technischen Universität Dresden für die Anmeldung zu Lehrveranstaltungen und Prüfungen sowie für die Mitteilung von Prüfungsergebnissen eingesetzt wird. Die Entwicklung der Weboberfläche in der jetzigen Form basiert auf Technologien, welche auf das Jahr 2009 und Frühere zurückgehen. Seit 2009 haben sich die Programmiersprachen und Technologien jedoch stetig weiterentwickelt. Allein bei Java hat sich die Entwicklung von Java SE 6, das 2009 verwendet wurde, zu Java SE 11, das heute verwendet wird, fortgesetzt. Die Anwendung muss auch mit neuen Versionen von Programmiersprachen und Frameworks aktualisiert werden, und das aus mehreren Gründen:

Schwachstellen verringern: Das ist der Hauptgrund für ein Software-Update. Wenn eine Plattform dem Internet ausgesetzt ist, sind die Datenbanken, in denen alle Details der Nutzer gespeichert sind, zunehmend Sicherheitsbedrohungen ausgesetzt. Böswillige Personen, die über immer bessere Werkzeuge verfügen, finden Schwachstellen in der Software. Durch Software-Updates werden einige dieser Sicherheitslücken überdeckt, sodass sie nicht ausgenutzt werden können (vgl [13], S.82).

Behebung von Fehlern und Abstürzen: Ausfälle, Probleme, Fehler werden behoben, wenn ein Unternehmen eine aktualisierte Version eines Programms erstellt. Jede entwickelte Software hat inhärente Fehler oder Verbesserungsmöglichkeiten. Wenn Hersteller Sicherheitslücken oder Bugs entdecken, kleinere Verbesserungen an Programmen vornehmen oder Kompatibilitätsprobleme beheben, veröffentlichen sie Updates. Durch die Aktualisierung wird sichergestellt, dass die neueste und stabilste Version weniger Fehler aufweist (vgl [13], S.82).

Gewährleistung der Kompatibilität mit anderen aktualisierten Technologien: Anwendungen funktionieren heute nicht mehr völlig unabhängig voneinander, sondern

kommunizieren miteinander. Daher ist es wichtig, dass eine Plattform auf dem neuesten Stand ist, um kompatibel zu sein und die Vorteile anderer Bibliotheken, Frameworks oder Tools nutzen zu können.

Gewinn an Leistung und Funktionalität: Die Aktualisierung einer Programmiersprache oder einer Software ist oft mit einem Leistungszuwachs verbunden.

Aus diesen Gründen wird derzeit eine neue Version von jExam (jExam New) mit neueren Technologien entwickelt. Um herauszufinden, ob die neue Plattform funktional und effizienter als die alte Version (jExam 2009) ist, muss eine Testinfrastruktur eingerichtet werden. Dies erfordert nicht nur das Testen der beiden Plattformen, sondern auch einen Vergleich zwischen ihnen.

jExam ist eine kritische Plattform für Studierende der TU Dresden. Sie speichert viele sensible persönliche Informationen über die Nutzer. Außerdem muss sie immer online und schnell verfügbar sein, um eine gute Softwarequalität zu gewährleisten. Aufgrund der vielen Vorteile, die das Testen einer Plattform mit sich bringt (siehe Kapitel 2), sollte die Plattform mit einer Reihe von Anforderungen getestet werden. Das ist das Ziel des nächsten Abschnitts.

3.3. Anforderungen an die jExam-Testsuite

Wie bereits in der Einleitung erwähnt, läuft die jExam 2009-Plattform derzeit ohne automatische Tests. Es besteht ein dringender Bedarf, eine Infrastruktur zum Testen der Plattform aufzubauen. Nicht nur die grundlegenden Funktionen der Anwendung müssen getestet werden, sondern auch die Performance und die Sicherheit. jExam New, das sich in der Entwicklung befindet, muss jedoch am Ende des Entwicklungszyklus gleich getestet werden. Die zwei Plattformen werden mit unterschiedlichen Technologien entwickelt und können daher nicht einheitlich getestet werden (mit Unit-Tests). Für die Entwicklung der Testinfrastruktur wird die Blackbox-Methode verwendet. Das bedeutet, dass die Technologien, die für die Entwicklung des alten und neuen jExam verwendet wurden, keine Rolle spielen beim Testen (siehe Kapitel 2). Die Frage ist, ob die Tests für die neue Plattform neu geschrieben werden müssen, sobald sie verfügbar ist.

3.3.1. jExam UI-Tests

Diese Methode ermöglicht es, eine Plattform sowie ihre Funktionen anhand ihrer grafischen Benutzeroberfläche (UI oder Frontend) zu testen. Mit Hilfe moderner Werkzeuge und Frameworks ist es möglich, die Kontrolle über einen Webbrowser vollautomatisch zu übernehmen. Dies ermöglicht es, das Verhalten eines Benutzers auf einer Weboberfläche zu simulieren. Es wurde bereits erwähnt, dass jExam 2009 und jExam New die gleiche grafische Benutzeroberfläche haben. Es gibt jedoch einige bemerkenswerte Unterschiede im Frontend-Code der beiden Plattformen, die bei der Entwicklung von Tests Probleme verursachen könnten (siehe Abbildung 3.2). Trotzdem gibt es eine Möglichkeit, die Tests nur einmal und für alle zwei Plattformen zu schreiben. Auf diese Weise sparen die Tester Zeit und können herausfinden, ob jExam New genauso gut funktioniert wie sein Vorgänger. Zu den Funktionen, die getestet werden müssen,

gehören :

1. Login
2. Registrierung
3. Abruf der Noten in der Übersicht
4. Einschreibung in Prüfungen
5. Einschreibung in Lehrveranstaltungen
6. Einschreibung in Seminargruppen

```

▼<nav> (flex)
  ▶<a id="nav-logo" class="jE-logo" href="/web/">...</a> (flex)
  ▶<button class="hamburger-btn" onclick="closeNav(this)" aria-label="Open navigation">...
  </button>
  ▼<div class="responsive-wrapper active">
    ▶<div class="flex-wrapper">...</div>
    ▼<div class="flex-wrapper">
      <!-- logged in-->
      ▼<div class="nav-entry animate-fade-in logout" style="opacity: 1; display: block;">
        ...
        <a class="btn fill-primary" href="/web/spring/logout">Sign Out</a> == $0
        </div>
        ▶<div class="nav-entry animate-fade-in settings-button" style="opacity: 1; display:
        block;">...</div>
        </div>
      </div>
    </div>
  </nav>

```

jExam_2009 Sign Out Button

unterschiedlich von

```

▼<nav> (flex)
  ▶<a class="jE-logo" href="/">...</a> (flex)
  ▼<div class="responsive-wrapper active">
    ▶<div class="flex-wrapper">...</div>
    ▼<div class="flex-wrapper">
      <!-- logged in-->
      ▼<div class="nav-entry animate-fade-in logout" style="opacity: 1; display: block;">
        ▼<form id="command" class="no-margin" action="/logout" method="POST">
          ...
          <button class="btn fill-primary" type="submit"> Sign Out </button> == $0
          <input type="hidden" name="_csrf" value="6e447cfc-da7c-40af-b7b4-4e327f82df0f">
          </div>...</div>
        </form>
      </div>
      ▶<div class="nav-entry animate-fade-in settings-button" style="opacity: 1; display:
      block;">...</div>
      </div>
    </div>
  ▶<button class="hamburger-btn" onclick="closeNav(this)" aria-label="Open navigation">...
  </button>
</nav>

```

jExam_New Sign Out Button

Abbildung 3.2. Frontend Quelltext von jExam 2009 und jExam new

3.3.2. jExam Sicherheitstests

Keine Software ist frei von Fehlern und Bugs. Dennoch muss eine qualitativ hochwertige Anwendung ihren Nutzern Datenschutz garantieren. Das gilt auch für jExam. Das

Sicherheits- oder Penetrationstesten ist eine schwierige Aufgabe. Das Penetrationstesten ist eine hauptsächlich manuelle Aktivität.

Experten-Penetrationstester sind ein wesentlicher Bestandteil von Penetrationstests. Zunächst verwenden sie Schwachstellenscanner. Es handelt sich um automatisierte Tools, die eine Umgebung untersuchen und nach Abschluss der Untersuchung einen Bericht über die gefundenen Schwachstellen erstellen. Diese Scanner listen die Schwachstellen häufig mithilfe von CVE auf, die Informationen über bekannte Schwachstellen liefern. Da Scanner Tausende von Schwachstellen entdecken können, kann es sein, dass es genügend schwerwiegende Schwachstellen gibt, die eine zusätzliche Priorisierung erforderlich machen.

Für komplexe Tests, bei denen tief in verschiedene Systeme und Anwendungen eingedrungen werden muss oder Übungen mit mehreren Angriffsketten durchgeführt werden müssen, wird immer eine erfahrenere Person oder ein erfahrenes Team benötigt. Um ein realistisches Angriffsszenario zu testen, ist ein Team erforderlich, das ausgeklügelte Strategien und Lösungen verwendet, die den Techniken der Bedrohungsakteure ähneln.

Aufgrund des Ressourcenmangels im jExam-Team muss der Prozess automatisiert werden. Daher sollte die Sicherheit nur oberflächlich mit einem Schwachstellenscanner getestet werden. Auf diese Weise ist es möglich, hohe Vulnerabilitäten herauszufinden, wenn sie vorhanden sind. Schwachstellen-Scans können also auch beim Vergleich der beiden Versionen von jExam helfen. Es wird in Erfahrung gebracht werden können, ob die neue Version mehr oder weniger Schwachstellen hat als die alte. Dadurch können mögliche Sicherheitslücken behoben werden.

3.3.3. jExam Performancetests

“ 2 Sekunden ist die Schwelle für die Akzeptanz einer E-Commerce-Website. Bei Google streben wir eine Zeit unter einer halben Sekunde an.” Maile Ohye

Das obige Zitat stammt aus einem Video, das 2010 von Google Webmasters veröffentlicht wurde (vgl. [21]). Wenn eine Seite länger als 2 Sekunden zum Laden braucht, verliert sie Plätze in ihrer Google SEO-Position. Es sollte sichergestellt werden, dass eine Webanwendung dieses Kriterium erfüllt, daher ist es wichtig, die Leistung zu testen. Die Performance einer Webanwendung spielt eine entscheidende Rolle für die Benutzererfahrung. jExam hat Dienste, die immer in Echtzeit verfügbar sein müssen. Mithilfe von Performancetests kann die Anwendung unter bestimmten Bedingungen getestet werden, um sie zu bewerten.

Die einzelnen Schritte des Leistungstests sind von Unternehmen zu Unternehmen und von Anwendung zu Anwendung unterschiedlich. Sie hängen davon ab, welche Leistungsindikatoren das Unternehmen für besonders wichtig hält. Die allgemeinen Ziele von Leistungstests sind jedoch weitgehend identisch, so dass es einen bestimmten Arbeitsablauf gibt, dem die meisten Testpläne folgen. Daher ist es wichtig, die Einschränkungen, Ziele und Schwellenwerte festzulegen, die den Erfolg des Tests belegen. Die wichtigsten Kriterien, die für die Tests von jExam verwendet werden, sind:

1. Beim Navigieren auf der Anwendung sollte die Webseite auch in weniger als 03 Sekunden geladen werden, unabhängig davon, welche Anfrage ein Benutzer ausführt (Load Testing).
2. Die Seiten von jExam müssen in weniger als 3 Sekunden laden, wenn 200 Benutzer gleichzeitig eine Anfrage stellen (Stress Testing).

Die beiden Versionen sollten daher getestet und verglichen werden, um eine mögliche Leistungssteigerung zu untersuchen. Dies wird es auch ermöglichen, nach einer Änderung zu beobachten, ob es einen Leistungsrückgang gegeben oder zum Bottlenecks geführt hat.

3.3.4. Zusatzanforderungen

Um das Schreiben und die Pflege von Tests so zeitsparend wie möglich zu machen, müssen diese Tests außerdem folgende Anforderungen erfüllen :

Vollautomatisierung: Die Tests sollten vollständig automatisiert sein, um die Iteration von sich wiederholenden Aufgaben zu ermöglichen und Zeit zu sparen.

Wiederverwendung der Testsuite: Die Testsuite muss für beide Versionen von jExam verwendbar sein.

Hohe Wartbarkeit: Die Testinfrastruktur und das Schreiben der Tests müssen leicht zu warten und einfach zu verwenden sein.

Leicht zu installieren: Die Testinfrastruktur muss einfach zu installieren sein und muss unter Windows, MAC und Linux laufen können.

3.4. Zusammenfassung der Analyse

Vor der Implementierung der Testinfrastruktur wurden in diesem Kapitel Designentscheidungen getroffen, um den Rahmen der Entwicklung zu begrenzen. Die Ziele und Bedingungen der Entwicklung wurden festgelegt. Das nächste Kapitel wird sich auf die Einrichtung der Testumgebung und die Testentwicklung an sich konzentrieren. Die Wahl der verschiedenen Technologien wird ebenfalls im nächsten Kapitel diskutiert.

4. Implementierung der automatisierten Testinfrastruktur

Das Endziel dieser Arbeit ist die Einrichtung einer automatisierten Testinfrastruktur, um die Webanwendung jExam und ihre zukünftige Version, die sich noch in der Entwicklung befindet, zu testen. Dabei sollen nicht nur die wichtigsten Funktionen getestet werden, sondern auch die Sicherheit und die Performance. Dieses Kapitel konzentriert sich auf die Einrichtung dieser Testinfrastruktur sowie auf das Design und die Entwicklung der Tests. Außerdem werden die verschiedenen verwendeten Technologien vorgestellt und ihre Verwendung begründet.

4.1. Vorstellung der Testansatzes

Die jExam-Webanwendung muss auf drei verschiedenen Ebenen getestet werden: Funktionalitäten, Performance und Sicherheit. Es ist jedoch nicht möglich, diese drei Ebenen in einer einzigen Testsuite zu kombinieren. Dies erfordert die Einrichtung einer Infrastruktur, die die Erstellung und Ausführung der Tests verwaltet. Die Anwendung, die verwendet wird, um die verschiedenen Dienste zu trennen, ist Docker [10] (wird in den folgenden Kapiteln behandelt). Docker wird nicht nur für die Trennung der Testebenen verwendet. Sie ermöglicht es auch, beide Versionen von jExam zu deployen, sodass sie effizient und lokal getestet werden können. Über diese Infrastruktur wird es möglich sein, Tests auszuführen und am Ende ihrer Ausführung Berichte und Metriken zu erhalten. Über diese Infrastruktur wird es möglich sein, Tests auszuführen und am Ende ihrer Ausführung Berichte und Metriken zu erhalten. Diese ermöglichen, die Leistung der beiden Plattformen zu beobachten und zu vergleichen. In den nächsten Kapiteln werden alle diese Konzepte im Detail behandelt.

4.2. Entwicklung von Sicherheitstests

4.2.1. OWASP

OWASP ist das Akronym für **Open Web Application Security Project** und beschreibt eine offene Organisation, deren Hauptziel ist, die Sicherheit von Anwendungen, Diensten und Software zu verbessern. Sie wurde am 1. Dezember 2001 gegründet und am 21. April 2004 als gemeinnützige Organisation offiziell anerkannt. Sie ermöglicht Organisationen, Unternehmen oder Einzelpersonen, sichere Anwendungen zu entwickeln und zu warten. Die OWASP hat eine Reihe von Sicherheitstools und -richtlinien entwickelt. Dazu gehören die OWASP Top 10 und der Zed Attack Proxy (ZAP). Ein wichtiger Grundsatz der OWASP ist, dass jeder, der sich für die Sicherheit von Webanwendungen interessiert, weltweit kostenlos über das nötige Wissen und die Werkzeuge verfügen kann (vgl. [22]). In diesem Zusammenhang hat sie die OWASP TOP 10 erstellt, die eine Liste der zehn häufigsten Sicherheitslücken im Internet darstellt. Ihr Hauptzweck ist die Schulung all jener, die mit der Entwicklung sicherer Anwendungen zu tun haben. Sie sollen über die üblichsten Sicherheitslücken informiert werden und dadurch diese vermeiden. Im Folgenden werden die zehn Angriffe der OWASP TOP 10 auf der Basis der OWASP TOP 10 2017 Release Candidate 2 (vgl. [34]) beschrieben.

Tabelle 4.1. OWASP TOP 10 2017

OWASP TOP10 2017

- 1- Injektion (Injection)
 - 2- Fehler in der Authentifizierung (Broken Authentication)
 - 3- Verlust der Vertraulichkeit von Daten (Sensitive Data Exposure)
 - 4- XML External Entities (XML)
 - 5- Fehler in der Zugriffskontrolle (Broken Access Control)
 - 6- Sicherheitsrelevante Fehlkonfiguration (Security Misconfiguration)
 - 7- Cross-Site-Scripting (XSS)
 - 8- Unsichere Deserialisierung (Insecure Deserialization)
 - 9- Nutzung von Komponenten mit bekannten Schwachstellen (Using Components with Known Vulnerabilities)
 - 10- Unzureichendes Logging und Monitoring (Insufficient Logging and Monitoring)
-

Injektion

Eine Injektion ist eine Schwachstelle, die einem Angreifer ermöglicht, bösartigen Code durch eine Anwendung auf ein anderes System zu schleusen. Dabei können sowohl Backend-Systeme als auch andere Clients, die mit der angreifbaren Anwendung verbunden sind, gefährdet werden. Zu den Auswirkungen dieser Angriffe gehören:

1. Einem Angreifer erlauben, Betriebssystemaufrufe auf einem Zielrechner auszuführen.
2. Einem Angreifer ermöglichen, Backend-Datenspeicher zu kompromittieren.
3. Einem Angreifer ermöglichen, die Sitzungen anderer Benutzer zu kompromittieren oder umzuleiten.

4. Einem Angreifer erlauben, Aktionen im Namen anderer Benutzer oder Dienste zu erzwingen.

Viele Webanwendungen hängen von Betriebssystemfunktionen, externen Programmen und der Verarbeitung von Datenabfragen ab, die von Benutzern eingereicht werden. Wenn eine Webanwendung Informationen aus einer HTTP-Anfrage als Teil einer externen Anfrage weitergibt, sollte diese Anwendung eine Möglichkeit zur Überprüfung und Validierung der Nachricht einrichten. Andernfalls kann ein Angreifer spezielle (Meta-) Zeichen, bösartige Befehle/Codes oder Befehlsmodifikatoren in die Nachricht einfügen. Diese Angriffe sind zwar nicht schwer auszuführen, aber es gibt immer mehr Tools, die nach diesen Fehlern suchen. Ein Angreifer kann diese Techniken nutzen, um den Inhalt der Datenbank zu erhalten, zu beschädigen oder zu zerstören, Backend-Systeme zu kompromittieren oder andere Benutzer anzugreifen. Erfolgreiche Injektionsangriffe können ein System vollständig gefährden oder zerstören. Es ist wichtig, auf diese Arten von Angriffen zu testen und sich dagegen zu schützen.

Als Beispiel kann man die SQL-Injektion nennen. Diese ist eine besonders weit verbreitete und gefährliche Form der Injektion. Um einen SQL-Injektion-Fehler auszunutzen, muss ein Angreifer einen Parameter finden, den die Webanwendung an eine Datenbankinteraktion weitergibt. Ein Angreifer kann dann bösartige SQL-Befehle in den Inhalt des Parameters einbetten. Das bringt die Webanwendung dazu, eine bösartige Abfrage an die Datenbank weiterzuleiten. SQL-Abfragen könnten durch Hinzufügen zusätzlicher "Einschränkungen" zu einer Anweisung (z. B. OR 1=1) geändert werden, um Zugriff auf nicht autorisierte Daten zu erhalten oder diese zu ändern.

Fehler in der Authentifizierung

Wenn die Authentifizierungsfunktionen einer Anwendung nicht korrekt implementiert sind, können Angreifer Passwörter oder Sitzungs-IDs kompromittieren oder andere Implementierungsfehler ausnutzen, indem sie die Anmeldedaten anderer Benutzer verwenden. Diese Schwachstelle wird als "Fehler in der Authentifizierung" (Broken authentication in Englisch) bezeichnet. Es wird in der Regel durch schlecht implementierte Authentifizierungs- und Sitzungsverwaltungsfunktionen verursacht. In diesem Fall zielen Angriffe darauf ab, die Kontrolle über ein oder mehrere Benutzerkonten zu erlangen, indem der Angreifer die gleichen Privilegien erhält wie der angegriffene Benutzer. Es wird vom "Fehler in der Authentifizierung" besprochen, wenn Angreifer in der Lage sind, Passwörter, Schlüssel oder Sitzungs-Tokens, Benutzerkontoinformationen und andere Details zu kompromittieren, um Benutzeridentitäten zu übernehmen. Zu den üblichen Risikofaktoren gehören:

1. Vorhersehbare Anmeldekennungen
2. Benutzerauthentifizierungskennungen, die nicht geschützt sind, wenn sie gespeichert werden.
3. Sitzungs-IDs, die in der URL offengelegt werden (z. B. durch URL-Rewriting).
4. Sitzungs-IDs, die anfällig für Session-Fixing-Angriffe sind.
5. Sitzungswert, der nach dem Abmelden nicht unterbrochen oder ungültig gemacht wird.

6. Sitzungskennungen, die nach einer erfolgreichen Anmeldung nicht erneuert werden.
7. Passwörter, Sitzungskennungen und andere Identifikationsinformationen, die über unverschlüsselte Verbindungen gesendet werden.

Verlust der Vertraulichkeit von Daten

Der Verlust der Vertraulichkeit von Daten tritt auf, wenn eine Organisation unwissentlich sensible Daten offenlegt oder wenn ein Sicherheitsvorfall dazu führt, dass sensible Daten versehentlich oder rechtswidrig zerstört, verloren, verändert, unbefugt offengelegt oder unbefugt darauf zugegriffen wird. Eine solche Datenexposition kann das Ergebnis eines unzureichenden Schutzes einer Datenbank, einer Fehlkonfiguration bei der Erstellung neuer Instanzen von Datenspeichern oder einer unsachgemäßen Nutzung von Datensystemen sein.

Klassische Beispiele dafür sind in Klartext gespeicherte Daten, wie Passwörter oder Kreditkartendaten, fehlendes HTTPS auf authentifizierten Webseiten, und Hash-Passwörter, die ohne zugefügtes "Salz" erstellt wurden. Salz bezeichnet eine zufällige Zeichenfolge, die den Klardaten vor der Verschlüsselung hinzugefügt werden. Dadurch kann anhand identischer Hashwerte nicht darauf geschlossen werden, dass es sich um dieselben Daten handelt.

XML External Entities (XML)

XML External Entity Injection (auch bekannt als XEE) ist eine Sicherheitslücke, die einem Angreifer ermöglicht, in die Verarbeitung von XML-Daten durch eine Anwendung einzugreifen. Dieser Angriff erfolgt, wenn die XML-Eingabe, die einen Verweis auf eine externe Entität enthält, von einem schwach konfigurierten XML-Parser verarbeitet wird. Dieser Angriff kann zur Offenlegung vertraulicher Daten, Denial of Service, serverseitiger Anfragefälschung, Portanalyse aus der Sicht des Computers, auf dem sich der Parser befindet, und anderen Auswirkungen auf das System führen. Um XEE Injections zu verhindern, sollen folgende Punkte beachtet werden:

1. Einfachere Datenformate wie JSON zur Datenübertragung verwenden.
2. Das Verbot in allen XML-Parsern, XML-Entitäten und DTDs verändern. DTD steht für Document Type Definition und bezeichnet einen XML Dokument, in welchem XML-Entities definiert werden. Auf dieser Weise wird vermieden, dass ein Benutzer eigenen Entities erstellt.
3. Deaktivierung des Hochladens von XML-Dateien

Fehler in der Zugriffskontrolle

Fehler in der Zugriffskontrolle ist eine Sicherheitslücke, die es Angreifern ermöglicht, Berechtigungsgarantien zu umgehen und Tätigkeiten auszuführen, als wären sie privilegierte Benutzer.

4. Implementierung der automatisierten Testinfrastruktur

Die Zugriffskontrolle sorgt dafür, dass Benutzer nicht außerhalb der ihnen zugewiesenen Befugnisse handeln können. Fehler führen in der Regel zur unbefugten Offenlegung von Informationen, zur Änderung oder Zerstörung aller Daten oder zur Ausführung einer Geschäftsfunktion außerhalb der Grenzen des Benutzers. Häufige Schwachstellen bei der Zugriffskontrolle sind:

1. Umgehung von Zugriffskontrollprüfungen durch Änderung der URL, des internen Anwendungsstatus oder der HTML-Seite oder einfach durch Verwendung eines benutzerdefinierten API-Angriffstools.
2. Ermöglichung der Änderung des Primärschlüssels in den Datensatz eines anderen Benutzers, wodurch die Anzeige oder Bearbeitung des Kontos eines anderen Benutzers ermöglicht wird.
3. Ausweitung der Rechte. Als Benutzer handeln, ohne eingeloggt zu sein, oder als Administrator handeln, wenn man als Benutzer eingeloggt ist.
4. Manipulation von Metadaten, wie z. B. die Wiedergabe oder Manipulation eines JSON Web Token (JWT)-Zugangskontrolltokens oder eines Cookies oder eines versteckten Feldes, das manipuliert wurde, um die Privilegien zu erhöhen, oder der Missbrauch der JWT-Ungültigkeitserklärung.
5. CORS-Fehlkonfiguration ermöglicht nicht autorisierten API-Zugriff.
6. Erzwingen des Navigierens auf authentifizierten Seiten als nicht authentifizierter Benutzer oder auf privilegierten Seiten als Standardbenutzer. Zugriff auf API mit fehlenden Zugriffskontrollen für POST, PUT und DELETE.

Sicherheitsrelevante Fehlkonfiguration

Sicherheitsrelevante Fehlkonfigurationen sind Sicherheitskontrollen, die ungenau konfiguriert oder unsicher sind. Dadurch werden Ihre Systeme und Daten gefährdet. Grundsätzlich kann jede schlecht dokumentierte Konfigurationsänderung, Standardeinstellung oder ein technisches Problem bei einer beliebigen Komponente zu einer Fehlkonfiguration führen.

Eine Fehlkonfiguration kann aus einer Vielzahl von Gründen auftreten. Moderne Netzwerkinfrastrukturen sind äußerst komplex und zeichnen sich durch ständige Veränderungen aus. Organisationen können leicht entscheidende Sicherheitseinstellungen übersehen, insbesondere bei neuen Netzwerkgeräten, die Standardkonfigurationen beibehalten können. Wenn sichere Konfigurationen für Zugangspunkte eingerichtet werden, müssen Konfigurationen mit Sicherheitskontrollen häufig überprüft werden, um unvermeidliche Konfigurationsabweichungen zu erkennen. Systeme ändern sich, neue Geräte werden in das Netzwerk eingeführt, Patches werden eingespielt - all dies trägt zu fehlerhaften Konfigurationen bei.

Cross-Site-Scripting (XSS)

Cross-Site-Scripting (XSS)-Angriffe sind eine Art von Injektion, bei der bösartige Skripte in Websites eingeschleust werden (vgl. [16]). Cross-Site Scripting (XSS)-Angriffe treten

auf, wenn :

1. Daten gelangen über eine nicht vertrauenswürdige Quelle, meist eine Webanfrage, in eine Webanwendung.
2. Die Daten sind in einem dynamischen Inhalt enthalten, der an einen Webnutzer gesendet wird, ohne auf bösartige Inhalte überprüft zu werden.

Der Browser des Benutzers hat keine Möglichkeit zu erkennen, dass das Skript nicht vertrauenswürdig ist, und führt es aus. Da er glaubt, dass das Skript aus einer vertrauenswürdigen Quelle stammt, kann das bösartige Skript auf alle Cookies, Sitzungstoken oder andere sensible Informationen zugreifen, die vom Browser gespeichert und mit dieser Website verwendet werden. Diese Skripte können sogar den Inhalt der HTML-Seite umschreiben.

Der bösartige Inhalt, der an den Webbrowser gesendet wird, hat häufig die Form eines JavaScript-Segments, kann aber auch HTML, Flash oder jede andere Art von Code enthalten, die der Browser ausführen kann. Die Vielfalt der XSS-basierten Angriffe ist nahezu unbegrenzt, aber sie umfassen in der Regel :

1. Die Übermittlung privater Daten wie Cookies oder andere Sitzungsinformationen an den Angreifer.
2. Die Umleitung des Opfers auf Webinhalte, die vom Angreifer kontrolliert werden.
3. die Ausführung anderer bösartiger Operationen auf dem Rechner des Benutzers unter der verwundbaren Website.

Unsichere Deserialisierung

Der Begriff Serialisierung stammt aus der objektorientierten Programmierung und bezeichnet die Umwandlung eines programmierten Objekts (z. B. ein Java-, Python-Objekt) in eine Folge von Bytes. Die Serialisierung sorgt dafür, dass erstellte Objekte nicht nur zur Laufzeit, sondern auch danach existieren. Es ist daher eine Methode, die für die Persistenz von Objekten verwendet wird. Sie wird auch verwendet, um Objekte über ein Netzwerk zu übertragen (vgl. [19]). Die Deserialisierung ist der umgekehrte Prozess der Serialisierung und erzeugt wieder ein Objekt aus einem Bytestream.

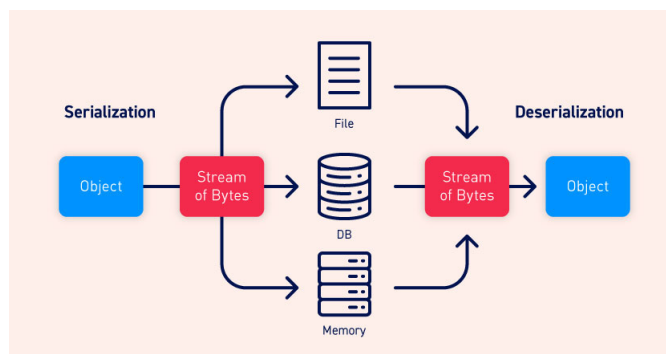


Abbildung 4.1. Deserialisierung Diagram (vgl. [24])

Eine unsichere Deserialisierung liegt vor, wenn vom Benutzer kontrollierbare Daten von einer Website deserialisiert werden. Dies ermöglicht es einem Angreifer potenziell, die serialisierten Objekte zu manipulieren, um schädliche Daten in den Anwendungscode einzuschleusen. Es ist sogar möglich, ein serialisiertes Objekt durch ein Objekt einer völlig anderen Klasse zu ersetzen. Alarmierenderweise werden Objekte auf der Website verfügbaren Klasse deserialisiert und instanziiert, unabhängig von der erwarteten Klasse. Aus diesem Grund wird die unsichere Deserialisierung manchmal auch als "Objektinjektions"-Schwachstelle bezeichnet (vgl. [24]).

Ein Objekt einer unerwarteten Klasse kann eine Ausnahme auslösen. Zu diesem Zeitpunkt kann der Schaden jedoch bereits eingetreten sein. Viele Angriffe, die auf Deserialisierung basieren, werden durchgeführt, bevor die Deserialisierung abgeschlossen ist. Das bedeutet, dass der Prozess der Deserialisierung selbst einen Angriff auslösen kann, auch wenn die eigentliche Funktionalität der Website nicht direkt mit dem bösartigen Objekt interagiert. Aus diesem Grund können auch Webseiten, deren Logik auf stark typisierten Sprachen basiert, für diese Techniken anfällig sein (vgl. [24]).

Nutzung von Komponenten mit bekannten Schwachstellen

Bekannte Schwachstellen sind Sicherheitslücken, die entweder vom Entwickler/Verkäufer der verwendeten Produkte, vom Benutzer/Entwickler oder vom Hacker/Intruder/Angreifer identifiziert wurden. Um bekannte Sicherheitslücken auszunutzen, identifizieren Hacker eine schwache Systemkomponente, indem sie das System mithilfe automatisierter Tools analysieren (häufiger, da diese Hacking-Tools online verfügbar sind) oder die Komponenten manuell analysieren (seltener, da dies fortgeschrittenere Fähigkeiten erfordert).

Fast jede Anwendung weist mindestens einige Schwachstellen auf, die auf Schwächen in den Komponenten oder Bibliotheken zurückzuführen sind, von denen die Anwendung abhängt (vgl. [1]). Manchmal sind die Schwachstellen absichtlich eingebaut. Anbieter sind dafür bekannt, Backdoor-Schwachstellen in ihren Systemen zu hinterlassen, um nach dem Einsatz des Systems aus der Ferne auf dieses zugreifen zu können. Die meisten Schwachstellen sind jedoch unbeabsichtigt. Sie sind auf Sicherheitslücken zurückzuführen, die dem Produktdesign inhärent sind (vgl. [1]).

Unzureichendes Logging und Monitoring

Protokolle (Logs) geben Einblick in die Aktivitäten einer Organisation. Die erstellten Protokolle und Prüfpfade ermöglichen einer Organisation die Fehlerbehebung, die Verfolgung von Ereignissen, die Erkennung von Zwischenfällen und die Einhaltung gesetzlicher Vorschriften. Unzureichende Protokollierung und Überwachung bedeutet, dass sicherheitskritische Informationen nicht protokolliert werden oder dass das richtige Protokollformat, der richtige Kontext, die richtige Speicherung, die richtige Sicherheit und eine rechtzeitige Reaktion fehlen, um einen Vorfall oder eine Sicherheitsverletzung zu erkennen.

Laut dem IBM-Bericht über Datenschutzverletzungen aus dem Jahr 2020 dauert es durchschnittlich 280 Tage, bis eine Datenschutzverletzung entdeckt und eingedämmt wird (vgl. [27], S. 11). Protokolle sind ein wichtiger Bestandteil der Reaktion auf Vorfälle.

Ein Unternehmen kann von einer Sicherheitsverletzung überrascht werden, die unentdeckt bleiben kann und irreparable rechtliche, finanzielle und regulatorische Folgen haben kann. Eine ordnungsgemäße Protokollverwaltung sorgt für eine schnellere Erkennung und Eindämmung von Sicherheitsverletzungen und spart dem Unternehmen Zeit, Geld und Ansehen.

4.2.2. ZAP Proxy

Wie in Unterabschnitt 3.3.2 erwähnt, müssen beide Versionen von jExam auf größere Sicherheitslücken gescannt werden. Schwachstellen-Scanner für Webanwendungen sind automatisierte Tools, die Webanwendungen - normalerweise von außen - auf Sicherheitslücken wie Cross-Site-Scripting, SQL Injektion, Command Injektion, Path Traversal und unsichere Serverkonfiguration untersuchen (vgl. [43]). Diese Kategorie von Tools wird häufig als Dynamic Application Security Testing (DAST) Tools bezeichnet. Es gibt eine große Anzahl kommerzieller und Open-Source-Tools dieser Art, und alle diese Tools haben ihre eigenen Stärken und Schwächen. Die Analyse der verschiedenen Tools zum Aufspüren von Schwachstellen wird in dieser Arbeit nicht behandelt. Es gibt jedoch das OWASP Benchmark-Projekt (vgl. [41]), das die Effektivität aller Arten von Tools zum Aufspüren von Schwachstellen, einschließlich dast, wissenschaftlich misst. Das Tool zum Scannen von Schwachstellen, das in dieser Arbeit verwendet wird, ist ZAP Proxy.

ZAP steht für Zed Attack Proxy und bezeichnet eine Open-Source-Sicherheitssoftware, die in der Programmiersprache Java geschrieben und 2010 veröffentlicht wurde. Sie wird verwendet, um Webanwendungen auf Schwachstellen zu scannen. Es wurde als kleines Projekt vom Open Web Application Security Project (OWASP) gestartet und ist heute das aktivste Projekt, das von Tausenden von Menschen auf der ganzen Welt betreut wird. Es ist der am häufigsten verwendete Webanwendungsscanner (vgl. [44]). ZAP ist für Linux, Windows und Mac in 29 Sprachen verfügbar. Zed Attack Proxy wird verwendet, um Schwachstellen auf beliebigen Webservern zu erkennen. Zu den wichtigsten Schwachstellen, die von zap erkannt werden können, gehören :

1. SQL injektion (Injection)
2. Fehler in der Authentifizierung (Broken Authentication)
3. Verlust der Vertraulichkeit von Daten (Sensitive data exposure)
4. Fehler in der Zugriffskontrolle (Broken Access control)
5. Sicherheitsrelevante Fehlkonfiguration (Security misconfiguration)
6. Cross Site Scripting (XSS)
7. Unsichere Deserialisierung (Insecure Deserialization)
8. Nutzung von Komponenten mit bekannten Schwachstellen (Components with known vulnerabilities)
9. Fehlende Sicherheitsheader (Missing security headers)

Was Zap zum meistgenutzten Werkzeug für Sicherheitsprüfungen macht, ist in erster Linie die Tatsache, dass es nicht nur für die Verwendung durch erfahrene

4. Implementierung der automatisierten Testinfrastruktur

Penetrationstester, sondern auch für Anfänger auf diesem Gebiet konzipiert wurde. ZAP ist ein kostenloses Open-Source-Tool, das einfach einzurichten und zu verwenden ist. Da es von einer breiten Community verwendet wird, gibt es online im ZAP-Blog und in anderen Artikeln eine Menge Hilfe, die bei der Einrichtung und Verwendung des Tools hilft. ZAP kann in einem Docker-Container ausgeführt werden. Außerdem ist die Funktionalität skalierbar mit vielen verschiedenen Erweiterungen, die auf GitHub veröffentlicht wurden.

ZAP ist ein so genannter "Man-in-the-middle-Proxy". Er wird zwischen den Browser und die Webanwendung geschaltet. Während der Tester durch alle Funktionen der Website navigiert, erfasst er alle Aktionen. Anschließend greift er die Website mit bekannten Techniken an, um Sicherheitslücken zu finden. ZAP ist ein Werkzeug, das menschliche Intelligenz benötigt, um richtig eingesetzt zu werden. Penetrationstester verwenden es zunächst, um automatisch nach Schwachstellen in einer Webanwendung zu scannen. Sobald sie eine Schwachstelle gefunden haben, die sie ausnutzen können, verwenden sie ZAP, um die Anwendung anzugreifen. Wie bereits in Unterabschnitt 3.3.2 erwähnt, ist die einzige Funktion, die jExam verwenden wird, der Scanner. Es geht darum, die Anwendung zu scannen, um größere Sicherheitslücken zu finden. Wenn die Anwendung solche Lücken enthält, ist es für die Entwickler unerlässlich, zu handeln und herauszufinden, wie man sie beheben kann. Auf diese Weise ist es möglich, ZAP vollautomatisch zu verwenden.

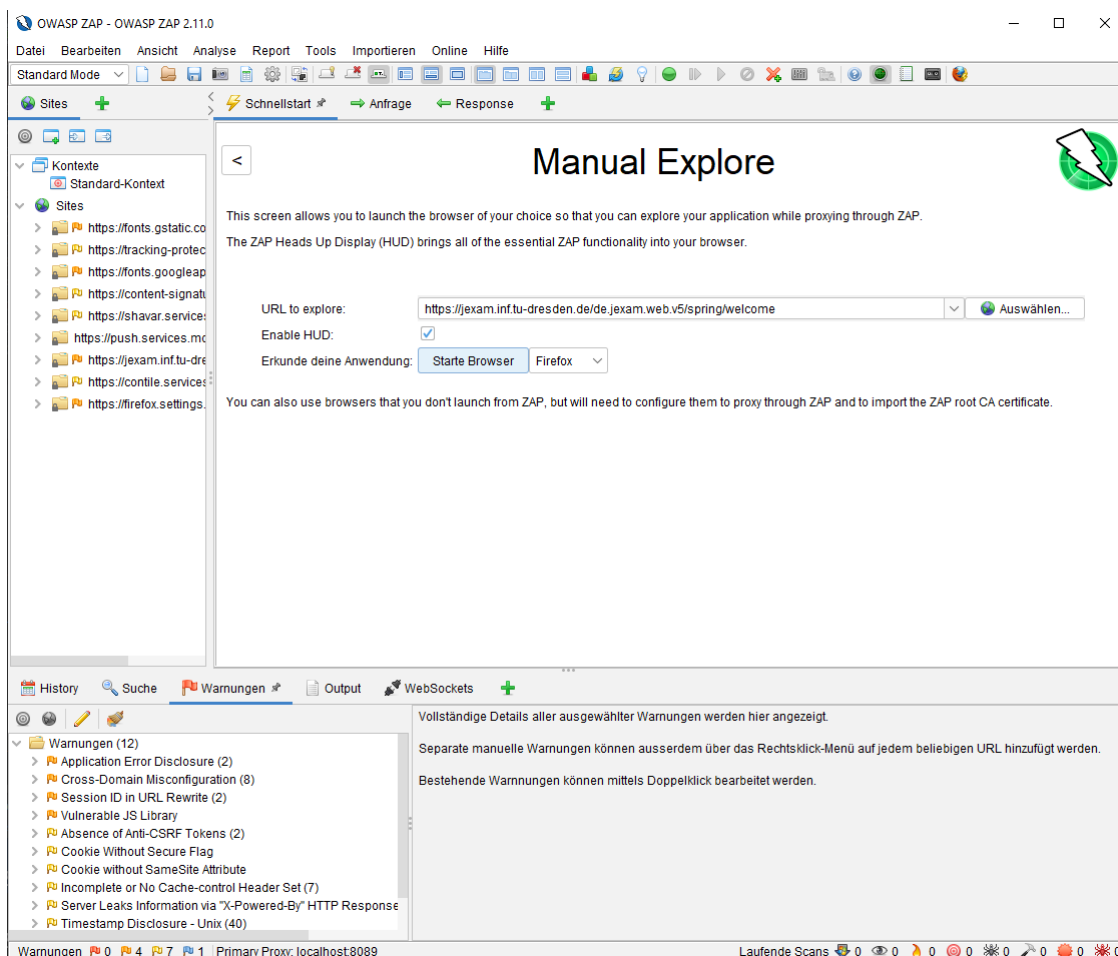


Abbildung 4.2. Grafische Benutzeroberfläche von Zap

4.2.3. Tests und Ergebnisse

Die Testinfrastruktur von jExam wurde mit Docker unter Verwendung von docker-compose entwickelt. Dies bietet die Möglichkeit, die Testdienste in verschiedene Container aufzuteilen (dieses Konzept wird in den nächsten Kapiteln ausführlich behandelt). Zu den Testdiensten gehört auch ein Container, der speziell für Sicherheitstests vorgesehen ist und automatisch bestimmte Befehle ausführt, um eine Version von jExam zu scannen. Zunächst ist es notwendig, einige Begriffe zu erklären, die im Folgenden verwendet werden.

ZAP Spider

Der Spider ist ein Werkzeug, das dazu dient, automatisch neue Ressourcen (URLs) auf einer bestimmten Seite zu entdecken. Er beginnt mit einer Liste von zu besuchenden URLs, den sogenannten Seeds, die davon abhängen, wie der Spider gestartet wird. Der Spider besucht dann diese URLs, identifiziert alle Hyperlinks auf der Seite und fügt sie der Liste der zu besuchenden URLs hinzu, und der Prozess wird rekursiv fortgesetzt, solange neue Ressourcen gefunden werden (vgl. [45]). Dieser Prozess kann Stunden dauern, je nachdem, wie viele Seiten und Links die Website hat. Aus diesem Grund wird er oft mit einer Zeitbegrenzung durchgeführt.

AJAX Spider

Der AJAX Spider ist ein Add-on für einen Crawler namens Crawljax. Das Add-on richtet einen lokalen Proxy in ZAP ein, um mit Crawljax zu kommunizieren. Mit dem AJAX Spider ist es möglich Webanwendungen, die AJAX benutzen, in weit größerer Tiefe crawlen (Prozess der automatischen Entdeckung neuer Ressourcen in einer Anwendung) als mit dem nativen Spider. AJAX ist eine Reihe von Webentwicklungstechniken, die verschiedene Webtechnologien auf der Client-Seite verwenden, um asynchrone Webanwendungen zu erstellen. Mit AJAX können Webanwendungen asynchron (im Hintergrund) Daten von einem Server senden und abrufen, ohne die Anzeige und das Verhalten der bestehenden Seite zu beeinträchtigen. Diese Computerarchitektur ermöglicht den Aufbau von Webanwendungen und dynamischen, interaktiven Webseiten. AJAX spider wird beim Testen von Webanwendungen empfohlen, die AJAX nutzen. Für eine vollständige Abdeckung einer Webanwendung (z. B. um HTML-Kommentare abzudecken) soll auch den nativen Spider verwendet werden (vgl. [40]).

Passive Scanning

Passive Scanning ist eine Methode zur Erkennung von Schwachstellen, die auf Informationen basiert, die aus Netzwerkdaten gesammelt werden. Beim Sammeln dieser Informationen gibt es keine direkte Interaktion mit der Zielanwendung (deshalb passiv). Beim passiven Scannen wird der gesamte Datenverkehr zwischen dem Browser und der Website gelesen und aufgezeichnet, d. h. die POSTs/GETs und ihre Antworten. ZAP analysiert diese Daten und sucht in seiner Angriffsbibliothek nach bekannten Problemen. Passives Scannen führt keine Angriffe aus und wird daher als harmlos

4. Implementierung der automatisierten Testinfrastruktur

angesehen. Beim passiven Scannen können viele mögliche Fehler und Schwachstellen in einer Webanwendung entdeckt werden. Zu den bekanntesten gehören :

1. Cross Domain Script Inclusion
2. Cross Domain Misconfiguration
3. X-Debug-Token Information Leak
4. Username Hash Found
5. Insecure Authentication
6. Information Disclosure: Suspicious Comments
7. Information Disclosure: Referrer
8. Information Disclosure: In URL
9. Information Disclosure: Debug Errors
10. CSRF Countermeasures

Diese Sicherheitslücken werden in dieser Arbeit nicht näher erläutert. Informationen zu diesen Sicherheitslücken sind jedoch auf der Website von Zap Proxy zu finden (vgl. [42]).

Active Scanning

Active Scanning ist eine Methode zur Erkennung von Schwachstellen, bei der versucht wird, potenzielle Schwachstellen mithilfe bekannter Angriffe auf ausgewählte Ziele zu finden. Im Gegensatz zum passiven Scanning ist es nicht risikolos. Es kann potenziell zu schwerwiegenden Problemen auf einem Webserver führen. Aus diesem Grund wird Testern empfohlen, es nur für ihre eigenen Anwendungen zu verwenden. Active Scanning ermöglicht es, mehrere große Sicherheitslücken zu entdecken, die in einer Anwendung vorhanden sein können. Zu den bekanntesten gehören :

1. SQL Injection
2. Directory Browsing
3. CRLF Injection
4. Cross Site Scripting (persistent und Reflected)
5. Command Injection
6. .htaccess Information Leak

Es gibt auch viele andere, die in dieser Arbeit nicht behandelt werden. Informationen zu diesen Sicherheitslücken sind jedoch auf der Website von Zap Proxy zu finden (vgl. [39]).

Auf der jExam-Plattform wurden zwei Skripte für ihre Ausführung implementiert. Es

handelt sich dabei um die Skripte ZAP Baseline und ZAP Full scan. Da es zwei Versionen von jExam gibt, muss bei der Ausführung der Skripte entschieden werden, welche der beiden Plattformen getestet werden soll.

ZAP Baseline

Dieses Skript führt zuerst einen ZAP-Spider und dann ein Passiv Scanning anhand der Daten aus dem Spider aus. Dies ist ein schneller und gründlicher Prozess, der es ermöglicht, schnell zu erkennen, ob es ernsthafte Schwachstellen gibt, die leicht ausgenutzt werden können. Schließlich erstellt das Skript einen Bericht (siehe Abbildung 4.3), der von den Testern sorgfältig geprüft werden muss. Das Skript führt keinen echten "Angriff" durch und läuft nur für eine relativ kurze Zeit (höchstens ein paar Minuten).

Quelltext 4.1 ZAP Baseline Ausführungsbefehl

```
command: [ "./wait-for-it.sh", "web:8080", "bash" , "-c",
"zap-baseline.py -t http://web:8080 -r owaspReport.html" ]

# Web:8080: URL der Neuen Version von jExam, die sich im
Container namens Web befindet.
# ./wait-for-it.sh: Script, das einen Healt-Check
ausführt, um herauszufinden, ob die parametrisierte
Anwendung (In diesem Fall Web:8080) bereits gestartet ist.
# owaspReport.html: Webseite, die am Ende
der Testdurchführung generiert wird.
```

ZAP Fullscan

Dieses Skript führt einen ZAP Spider für eine unbestimmte Zeit aus. Je mehr Ressourcen und Hyperlinks die Anwendung besitzt, desto länger dauert dieser Prozess. Danach wird ein Ajax Spider ausgeführt, der ebenfalls für eine unbestimmte Zeit ausgeführt wird. Danach wird ein Full Active Scan ausgeführt und schließlich ein Bericht erstellt (siehe Abbildung 4.3), auf den der Tester zugreifen kann. Das Skript führt echte "Angriffe" aus und kann potenziell über einen längeren Zeitraum (mehrere Stunden) laufen. Das Skript ist jedoch viel effizienter und findet mehr Sicherheitslücken als ZAP Baseline.

Quelltext 4.2 ZAP Fullscan Ausführungsbefehl

```
command: [ "./wait-for-it.sh", "web:8080", "bash" , "-c",
"zap-full-scan.py -d -j -m 1 -t http://web:8080 -r owaspReport.html" ]
```

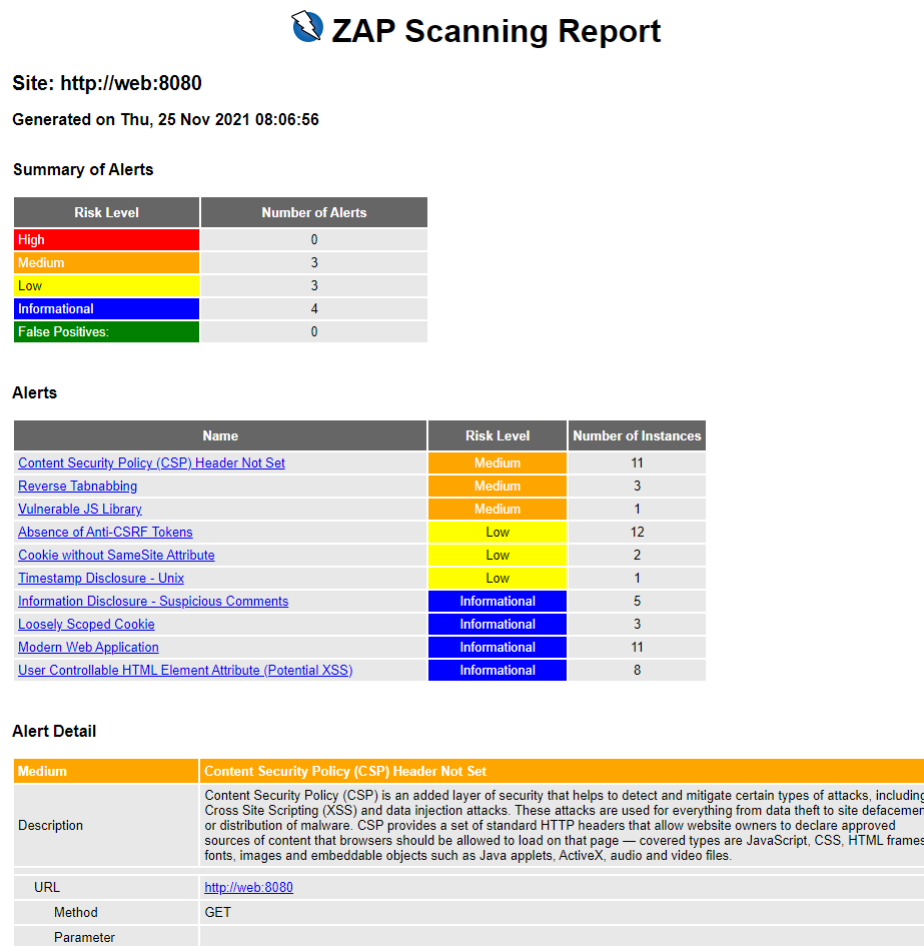


Abbildung 4.3. Bericht nach der Ausführung einer Zap Baseline

Die Sicherheit einer Anwendung zu testen ist eine schwierige Aufgabe, aber Werkzeuge wie Zap geben vielen Menschen die Möglichkeit, sich in diesem Bereich leicht einzuarbeiten. Mit Hilfe der Scanner ist es möglich, Sicherheitslücken zu finden, die den Entwicklern ermöglichen, sie zu beheben. Dadurch wird eine gute Softwarequalität für die Zukunft gewährleistet.

4.3. Entwicklung von Performancetests

Performancetests sind eine nicht-funktionale Art von Tests, die durchgeführt werden, um die Leistung einer Anwendung zu bestimmen. Die Tests werden anhand von Metriken wie Geschwindigkeit, Stabilität und Skalierbarkeit durchgeführt. Die Leistung einer Anwendung zu testen ist eine recht umfangreiche und für jede Anwendung spezifische Aufgabe. Tester müssen die wichtigsten Leistungsindikatoren der Anwendung bestimmen und eine Reihe von Tests entwerfen. Der Zugang zu diesem Bereich des Testens wurde jedoch durch das Aufkommen von automatischen Werkzeugen für Anfänger stark vereinfacht. Es gibt mehrere kostenpflichtige und kostenlose Tools, die alle ihre eigenen Vor- und Nachteile haben. In diesem Kapitel werden keine Vergleiche zwischen diesen Tools angestellt, aber es gibt Ressourcen, die diesen Vergleich detailliert durchgeführt haben (vgl. [14]). Zu den bekanntesten Performancetests Werkzeugen

gehören WebLoad, LoadNinja , LoadView und Apache JMeter. Apache JMeter ist das Werkzeug, das zum Testen von jExam verwendet wird.

4.3.1. Apache JMeter

Nach der offiziellen Dokumentation ist Apache JMeter eine reine Open-Source-Software, eine 100 % reine Java-Anwendung, die von Stefano Mazzocchi von der Apache Software Foundation entwickelt wurde, um das Funktionsverhalten zu testen und die Leistung zu messen (vgl [12]). JMeter kann verwendet werden, um die Leistung von Webanwendungen oder einer Vielzahl von Diensten zu analysieren und zu messen. Diese Software wurde ursprünglich zum Testen von Webanwendungen oder FTP-Anwendungen verwendet. Heutzutage wird es für funktionale Tests, Datenbankserver-Tests. Apache JMeter kann verwendet werden, um die Leistung statischer und dynamischer Ressourcen sowie dynamischer Webanwendungen zu testen. Es kann verwendet werden, um eine starke Belastung eines Servers, einer Gruppe von Servern, eines Netzwerks oder eines Objekts zu simulieren, um dessen Stärke zu testen oder um die Gesamtleistung unter verschiedenen Lasttypen zu analysieren. Die wichtigsten Vorteile von JMeter sind :

1. **Open-Source-Lizenz:** JMeter ist völlig kostenlos und erlaubt Entwicklern die Verwendung des Quellcodes für die Entwicklung.
2. **Freundliche GUI:** JMeter ist extrem einfach zu bedienen und es dauert nicht lange, sich damit vertraut zu machen.
3. **Plattformunabhängig:** JMeter ist eine 100% reine Java-Desktop-Anwendung. Daher kann es auf mehreren Plattformen laufen.
4. **Vollständiges Multithreading-Framework:** JMeter ermöglicht die gleichzeitige und simultane Abtastung verschiedener Funktionen durch eine separate Thread-Gruppe.
5. **Visualisierung des Testergebnisses:** Das Testergebnis kann in verschiedenen Formaten wie Diagramm, Tabelle, Baum und Protokolldatei angezeigt werden.
6. **Skalierbarkeit:** JMeter unterstützt auch Visualisierungs-Plugins, mit denen Tests erweitert werden können.
7. **Mehrere Teststrategien:** JMeter unterstützt viele Teststrategien wie Lasttests, verteilte Tests und funktionale Tests.
8. **Simulation:** JMeter kann mehrere Benutzer mit gleichzeitigen Threads simulieren und eine hohe Last auf die zu testende Webanwendung erzeugen.
9. **Unterstützung mehrerer Protokolle:** JMeter unterstützt nicht nur das Testen von Webanwendungen, sondern bewertet auch die Leistung von Datenbankservern. Alle Basisprotokolle wie HTTP, JDBC, LDAP, SOAP, JMS und FTP werden von JMeter unterstützt.
10. **Aufzeichnung und Wiedergabe:** JMeter ermöglicht das Aufzeichnen von Benutzeraktivität im Browser und hilft dabei sie in einer Webanwendung zu simulieren.
11. **Skript-Test:** JMeter kann mit Bean Shell für automatisierte Tests integriert werden.

Im Allgemeinen hat JMeter ein vereinfachtes Funktionsprinzip. Es simuliert eine Gruppe von Benutzern, die Anfragen an einen Zielservers senden, und gibt Statistiken zurück, die die Leistung/Funktionalität des Zielservers/der Zielanwendung in Tabellen und Diagrammen zeigen (siehe Abbildung 4.4).

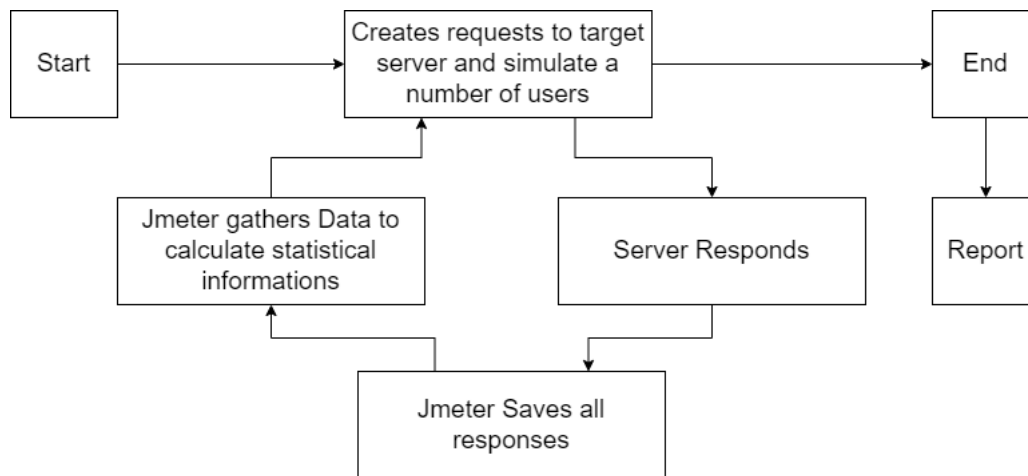


Abbildung 4.4. Funktionsweise von JMeter

JMeter wird normalerweise über seine grafische Benutzeroberfläche bedient. Es ist jedoch möglich, Tests mit der grafischen Benutzeroberfläche zu erstellen und sie in einem Docker-Container auf einer Anwendung auszuführen. Dieses Prinzip wurde zum Testen der beiden Versionen von jExam verwendet und wird im nächsten Abschnitt vorgestellt.

4.3.2. Implementierung der Performancetests

Die Performancetests, die für beide Versionen von jExam geschrieben wurden, basieren im Wesentlichen auf den Kriterien in Unterabschnitt 3.3.3. Die Tests, die in diesem Kapitel durchgeführt wurden, sind nur eine Demonstration dessen, was mit JMeter möglich ist. Diese Tests könnten im Laufe der Zeit weiterentwickelt und verbessert werden, wenn die Kriterien für die Bewertung der Plattformen spezifischer werden. Sie werden daher als Grundlage für die Entwicklung zukünftiger Tests dienen. JMeter wird in diesem Fall zum Testen einer Webanwendung verwendet. Deshalb wird HTTP als Kommunikationsprotokoll verwendet. Zunächst ist es notwendig, einige Begriffe zu erklären, die im Folgenden verwendet werden:

Thread-Gruppe : Sie ist eine Gruppe von Threads, die das gleiche Szenario ausführen. Sie ist das Basiselement für jeden JMeter-Testplan. Es stehen mehrere Thread-Gruppen zur Verfügung, die konfiguriert werden können, um zu simulieren, wie die Benutzer mit der Anwendung interagieren, wie die Last aufrechterhalten wird und über welchen Zeitraum. Jeder Thread führt den Testplan in seiner Gesamtheit und völlig unabhängig von anderen Test-Threads aus. Mehrere Threads werden verwendet, um gleichzeitige Verbindungen zu Ihrer Serveranwendung zu simulieren.

Assertions : Sie sind die Komponente eines Tests, mit der ein Benutzer bestätigen kann, dass die von JMeter erhaltene Antwort den erwarteten Kriterien entspricht. Sie stellen sicher, dass der Benutzer sich einer inkonsistenten oder unerwarteten Antwort

der Zielerwartung bewusst ist. Die Assertions von JMeter sind ein mächtiges Werkzeug, aber ihre Wirksamkeit hängt sowohl von den ausgewählten Assertionskriterien als auch von der Genauigkeit der vorherigen Anfrage ab. Die Assertion validiert, dass die Antwort der Anwendung wie erwartet empfangen wird, aber diese Antwort beruht in der Regel auf der korrekten Formulierung einer vorherigen Anfrage.

Resultlisteners : Ein Listener ist eine Komponente, die die Ergebnisse der Proben anzeigt. Die Ergebnisse können in einer Baumstruktur, in Tabellen oder Diagrammen angezeigt oder einfach in eine Protokolldatei geschrieben werden.

Für die Integration von Performancetests wurde eine Testsuite entwickelt, die zwei Tests enthält, die in zwei Thread-Gruppen aufgeteilt sind. Der erste Test (erste Thread-Gruppe) dient dazu, das Navigieren eines Benutzers auf den Plattformen zu simulieren und so die Antwortzeiten des Servers zu überwachen. Auf diese Weise ist es möglich, die beiden Versionen anhand verschiedener Kriterien zu vergleichen, wie z.B. der schnellsten Antwortzeit. Diese Thread-Gruppe besteht aus einem einzigen Benutzer und ist in vier Phasen unterteilt:

Login-Phase: Zunächst führt der Test eine GET-Anfrage an die Login-Seite der Anwendung durch. Danach führt er auf derselben Seite eine POST-Anfrage durch, um sich zu registrieren und das Recht zu erhalten, auf gesicherte Seiten zuzugreifen. Außerdem wurden zwei weitere Assertions hinzugefügt: die Duration Assertion, um herauszufinden, ob die Seite in weniger als drei Sekunden empfangen wird, und die Response Assertion, die sicherstellt, dass der Test einen korrekten HTTP-Response-Code (200) erhält.

Home-Phase: In diesem Teil führt der Test eine GET-Anfrage an die Home Page der Anwendung aus. Wenn die Authentifizierung erfolgreich war, sollte der Test eine HTTP-Response von 200 zurückgeben. Um dies sicherzustellen, wurden die Assertion Duration (Für die drei Sekunden) und Response hinzugefügt.

Veranstaltungsphase: In diesem Teil führt der Test eine GET-Anfrage an die Veranstaltungsseite aus. Wenn die Authentifizierung erfolgreich war, sollte der Test eine HTTP-Response 200 zurückgeben. Um dies sicherzustellen, wurden auch die Assertion Duration und Response hinzugefügt.

Logout: In dieser Phase führt der Test eine HTTP-POST-Anfrage aus, um sich abzumelden, und wie die anderen Phasen verfügt auch diese Phase über die Assertions Duration und Response.

Der zweite Test ist viel einfacher. Die Thread-Gruppe besteht aus 200 Benutzern, die in Zeitintervallen von 10 Millisekunden eine GET-Anfrage an die Home-Seite der Anwendung ausführen. Auf diese Weise können die Antwortzeiten des Servers bei einer gleichzeitigen Überlastung durch Anfragen beobachtet werden. Assertion Duration und Response wurden ebenfalls hinzugefügt.

Nachdem der Tester die Skripte für beide Versionen von jExam geschrieben und angepasst hat, kann er sie kompilieren und so .jmx-Dateien erzeugen. Diese Dateien werden dann in Docker Container kopiert, die für die automatische Ausführung der Tests zuständig sind. Nach der Ausführung der Tests werden Berichte für jExam 2009 und jExam New generiert. Der Tester kann sich diese Berichte ansehen und sie mit den Zielkriterien vergleichen.

Quelltext 4.3 JMeter Ausführungsbefehl

```
command: [ "./wait-for-it.sh", "web:8080/",  
"/bin/sh" , "-c",  
"jmeter -f -n -t jExam_New.jmx -l results.csv  
-e -o reports/html" ]  
  
# jExam_New.jmx : jExam_new Generierter JMeter-Test  
# reports/html : Ordner, der den HTML-Bericht enthalten wird
```

4.4. Entwicklung von UI-Tests

Die Entwicklung von UI-Tests ist der zentrale Punkt dieser Arbeit. Im Gegensatz zu den Performance- und Sicherheitstests, die nicht funktional sind, besteht das Hauptziel dieser UI-Tests darin festzustellen, ob die beiden Versionen von jExam alle Funktionalitäten besitzen, die zukünftige Benutzer erwarten und die von den Entwicklern beabsichtigt sind. Dieses Kapitel konzentriert sich auf die Entwicklung der UI-Tests, die verschiedenen Tools und die Design Patterns, die verwendet wurden. Zunächst werden die verschiedenen Werkzeuge vorgestellt, die bei der Entwicklung verwendet wurden. Anschließend werden die Design Patterns vorgestellt, die dabei helfen, den Code robust und leicht wartbar zu machen. In Teil drei wird der Schwerpunkt auf die Implementierung von Tests gelegt, und im letzten Teil werden die erzielten Ergebnisse und zusätzliche Funktionen vorgestellt.

4.4.1. Verwendete Werkzeuge

Selenium Webdriver

Selenium ist ein Open-Source-Projekt für eine Reihe von Tools und Bibliotheken zur Unterstützung der Webbrowser-Automatisierung (vgl. [6]). Selenium bietet Werkzeuge zur Erstellung funktionaler Tests, ohne dass eine Testskriptsprache erlernt werden muss. Außerdem bietet es eine testspezifische Sprache (Selenese), mit der Tests in einer Reihe beliebiger Programmiersprachen geschrieben werden können, darunter JavaScript (Node.js), C, Groovy, Java, Perl, PHP, Python, Ruby und Scala. Die Tests können dann mit den meisten modernen Webbrowsern ausgeführt werden. Selenium läuft auf Windows, Linux und macOS. Selenium besteht aus mehreren Komponenten, von denen jede eine bestimmte Rolle bei der Entwicklung der Testautomatisierung von Webanwendungen übernimmt. Zu diesen Komponenten gehören: Selenium IDE, Selenium Client API, Selenium Remote Control, Selenium Grid und schließlich der Selenium WebDriver, der in dieser Arbeit verwendet wird.

Das Kernelement von Selenium ist der Selenium WebDriver, eine Schnittstelle zum Schreiben von Anweisungen, die in verschiedenen Browsern austauschbar sind. Selenium WebDriver nimmt Befehle entgegen (die in Selenese oder über eine Client-API gesendet werden) und sendet sie an einen Browser. Dies wird durch einen browser-spezifischen Browsertreiber implementiert, der Befehle an einen Browser sendet und Ergebnisse abrufen. Die meisten Browsertreiber starten tatsächlich eine Browseranwendung (z. B. Firefox, Google Chrome, Internet Explorer, Safari oder Microsoft

Edge) und greifen darauf zu. Selenium WebDriver benötigt keinen speziellen Server, um Tests auszuführen. Stattdessen startet der WebDriver direkt eine Browserinstanz und steuert sie. Wo immer möglich, verwendet WebDriver native Funktionen auf Betriebssystemebene und nicht browserbasierte JavaScript-Befehle zur Steuerung des Browsers. Dadurch werden Probleme mit subtilen Unterschieden zwischen nativen und JavaScript-Befehlen, einschließlich Sicherheitseinschränkungen, vermieden (vgl. [33]). Selenium WebDriver ist vollständig implementiert und wird in JavaScript (Node.js), Python, Ruby, Java, Kotlin und C unterstützt.

Selenium ist derzeit das von Testern am meisten geliebte Framework für UI-Testing (vgl. [6], S. 08-09). Es hat viele Vorteile, wie z.B. die Übertragbarkeit auf alle Systeme, die einfache Integration mit andere Technologien, eine große und dynamische Entwicklergemeinschaft sowie die Dokumentation und die Fülle an verfügbaren Ressourcen. Im Rahmen dieser Arbeit wird der Selenium Web Driver mit der Programmiersprache Java verwendet.

TestNG

TestNG ist ein Open-Source-Testautomatisierungs-Framework für Java. Es wird nach dem Vorbild von JUnit und NUnit entwickelt. Einige fortgeschrittene und nützliche Funktionen, die TestNG bietet, machen es zu einem robusteren Framework im Vergleich zu seinen Mitbewerbern (vgl. [30]). Das NG in TestNG steht für "Next Generation". Es wird immer häufiger von Entwicklern und Testern bei der Erstellung von Testfällen verwendet, da es einfach ist, mehrere Annotationen, Gruppierungen, Abhängigkeiten, Priorisierungen und Parametrierungsfunktionen zu verwenden. Durch die Beseitigung der meisten Einschränkungen des älteren Frameworks gibt TestNG dem Entwickler die Möglichkeit, flexiblere und leistungsfähigere Tests zu schreiben.

Der Hauptgrund, warum TestNG für die Entwicklung der Tests in dieser Arbeit ausgewählt wurde, ist in erster Linie die Tatsache, dass es im Vergleich zu seinem Konkurrenten JUnit mehr nützliche Funktionen bietet (vgl. [30]):

1. Annotationen von TestNG sind im Vergleich zu JUnit einfacher zu verstehen
2. TestNG erfordert im Gegensatz zu JUnit keine obligatorische Deklaration von @BeforeClass und @AfterClass
3. Die Funktion der Parametrisierung, die TestNG bietet, ist bequemer und einfacher durch den Datenprovider zu nutzen.
4. Funktionen wie Priorisierung und Gruppierung von Tests, die von TestNG bereitgestellt werden, machen es im Vergleich zu JUnit realistischer und leichter anpassbar.
5. TestNG bietet im Vergleich zu JUnit in mehrfacher Hinsicht die Möglichkeit der parallelen Testausführung.

Neben diesen Vorteilen gegenüber JUnit lässt sich TestNG leicht in das Selenium-Framework integrieren. Dies macht es zu einem der am häufigsten verwendeten Werkzeuge für das Schreiben von Selenium-Testskripten (vgl. [6], S.13).

AssertJ

AssertJ ist eine Java Bibliothek zur Vereinfachung des Schreibens von Assert-Anweisungen in Tests. Sie verbessert auch die Lesbarkeit von Assert-Anweisungen. Sie hat eine fließende Schnittstelle für Assertions, die die Code-Vervollständigung leicht macht und beim Schreiben hilft. AssertJ bietet einen umfangreichen Katalog von Assertions, hilfreiche Fehlermeldungen, verbessert die Lesbarkeit von Testcode und ist so konzipiert, dass es sehr einfach zu verwenden ist. AssertJ ist ein Werkzeug, das sich leicht in TestNG und Selenium integrieren lässt und das Schreiben von Tests erheblich vereinfacht. Es verfügt über eine explizite Dokumentation und mehrere online verfügbare Ressourcen.

Extent Reports

Extent Report ist eine Bibliothek, die verwendet wird, um einen individuellen detaillierten Bericht zu erstellen. Sie kann in TestNG, und JUNIT integriert werden. Dieser Bericht kann in JAVA gebaut werden und liefert eine detaillierte Zusammenfassung jedes Testfalls und jedes Testschritts in grafischer Form. Diese Berichte sind HTML-Dokumente, die die Ergebnisse in Form von Tortendiagrammen darstellen. Sie ermöglichen auch die Erstellung von benutzerdefinierten Protokollen, Snapshots und anderen benutzerdefinierten Details. Sobald ein automatisiertes Testskript erfolgreich ausgeführt wurde, müssen die Tester einen Testausführungsbericht erstellen. Dieses Tool ist sehr nützlich für Entwickler und Tester. Die von extent erstellten Berichte geben Aufschluss darüber, ob die Ausführung eines Tests erfolgreich war oder ob der Test an sich falsch implementiert wurde.

4.4.2. Entwurfsmuster

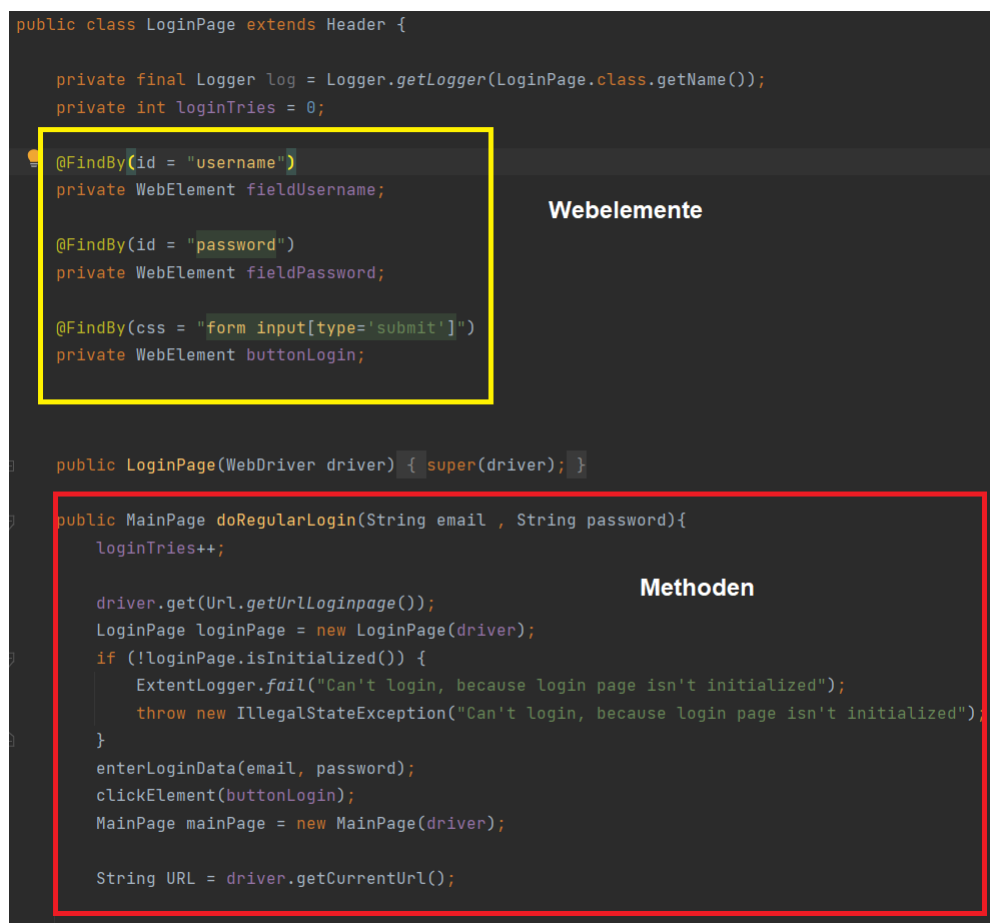
Zu den Voraussetzungen, die von der Testinfrastruktur erwartet werden, gehört die hohe Wartbarkeit der Testsuite. Dieses Ziel könnte ohne die Verwendung von Entwurfsmustern nicht erreicht werden. Entwurfsmuster können den Entwicklungsprozess beschleunigen, indem sie getestete und bewährte Entwicklungsparadigmen bereitstellen. Ein effektiver Softwareentwurf erfordert die Berücksichtigung von Problemen, die möglicherweise erst später bei der Implementierung sichtbar werden. Die Wiederverwendung von Entwurfsmustern hilft, subtile Probleme zu vermeiden, die zu großen Problemen führen können, und verbessert die Lesbarkeit des Codes für Programmierer und Architekten, die mit den Mustern vertraut sind. Für die Entwicklung von Tests mit Selenium gibt es zwei Entwurfsmuster, die sehr beliebt sind und von Testern verwendet werden : **Page Object** und **Factory** (vgl. [25]).

Page Object Model

Page Object Model ist ein in Selenium verwendetes Entwurfsmuster, bei dem ein Objektrepository zur Speicherung von Webelementen erstellt wird. Es wird eine Java-Klasse erstellt, die jeder Webseite entspricht (siehe Abbildung 4.6). Diese Seiten bestehen aus Webelementen und den entsprechenden Methoden, die auf diese Elemente

einwirken (siehe Abbildung 4.5). Alle Webseitenelemente befinden sich in einer Java-Klasse, indem sie durch ihre Locators identifiziert werden. Darüber hinaus werden für die verschiedenen Seiten der Webseite mehrere Java-Klassen erstellt. Diese Java-Klassen dienen als Repository, in dem die verschiedenen Elemente gespeichert werden, mit denen Testfällen interagieren können. Die Verwendung des Page Object Model hat viele Vorteile:

1. **Erleichtert die Wartung des Codes** : Da die Testklassen von den Klassen getrennt sind, die die Webelemente und die Operationen auf ihnen enthalten, ist die Aktualisierung des Codes sehr einfach, wenn ein Webelement aktualisiert oder ein neues hinzugefügt wird.
2. **Erleichtert die Lesbarkeit des Codes** : Der Benutzer kann das Projekt und die Testskripte aufgrund der feinen Trennung zwischen den Testklassen und den verschiedenen Webseiten leicht durchlesen.
3. **Wiederverwendbarkeit des Codes** : Wenn mehrere Testskripte dieselben Webelemente verwenden, müssen nicht in jedem Testskript Code zur Behandlung des Webelements geschrieben werden. Die Unterbringung in einer separaten Seitenklasse macht es wiederverwendbar, indem es von jedem Testskript aus zugänglich gemacht wird.



```

public class LoginPage extends Header {

    private final Logger log = Logger.getLogger(LoginPage.class.getName());
    private int loginTries = 0;

    @FindBy(id = "username")
    private WebElement fieldUsername;

    @FindBy(id = "password")
    private WebElement fieldPassword;

    @FindBy(css = "form input[type='submit']")
    private WebElement buttonLogin;

    public LoginPage(WebDriver driver) { super(driver); }

    public MainPage doRegularLogin(String email, String password){
        loginTries++;

        driver.get(Url.getUrlLoginPage());
        LoginPage loginPage = new LoginPage(driver);
        if (!loginPage.isInitialized()) {
            ExtentLogger.fail("Can't login, because login page isn't initialized");
            throw new IllegalStateException("Can't login, because login page isn't initialized");
        }
        enterLoginData(email, password);
        clickElement(buttonLogin);
        MainPage mainPage = new MainPage(driver);

        String URL = driver.getCurrentUrl();
    }
}

```

Abbildung 4.5. Darstellung der jExam LoginPage mit dem Page Object Model

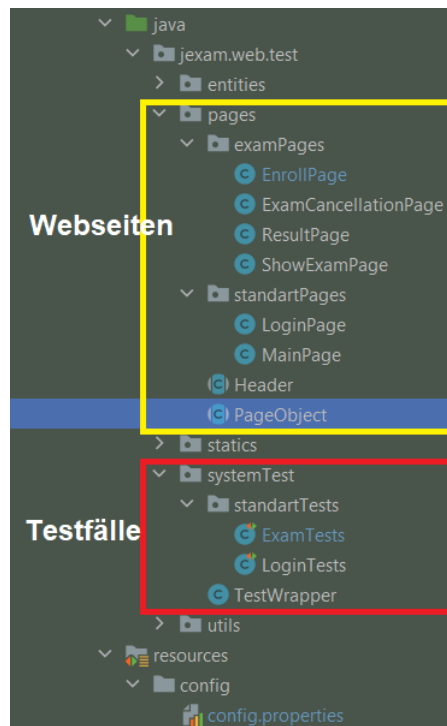


Abbildung 4.6. Projektstruktur von jExam Page Object Model

Factory

Page Factory ist eine Klasse, die von Selenium WebDriver bereitgestellt wird, um das Page Object Model zu implementieren. Das Page Object Repository wird mit Hilfe des Page Factory-Konzepts von den Testmethoden getrennt. Page Factory bietet Annotationen, um Elemente zu initialisieren und sie anschaulich und lesbar zu machen. Zu den Vorteilen seiner Verwendung gehören:

1. **Sauberer Code:** Das definierte Webelement wird von den Methoden getrennt, um eine Webseite in einem Page Object sauber und aufgeräumt zu gestalten.
2. **Lesbar und beschreibend:** Ein Webelement wird als Variable (bekannt als Object Field) deklariert, und die Field-Annotation (@FindBy siehe Abbildung 4.5) wird verwendet, um den Namen, den Typ und die Position des Elements zu beschreiben. Auf diese Weise können definierte Webelement anhand seiner Annotationen wie Name, Typ usw. leicht identifiziert werden.
3. **Einfache Wartbarkeit:** Das definierte Webelement kann ohne Neudefinition überall in der Page Object Klasse und den Unterklassen verwendet werden. Das bedeutet, dass ein bestimmtes Webelement mehrmals verwendet werden kann, aber nur an einer Stelle definiert ist.

Im nächsten Teil werden die Implementierung der Tests und die verwendeten Methoden genauer beschrieben.

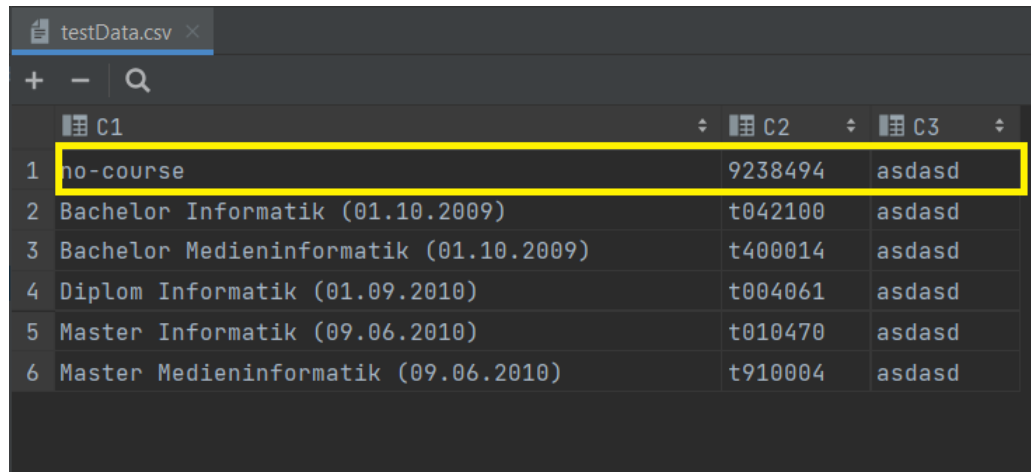
4.4.3. Implementierung der Testsuite

Das Schreiben eines UI-Tests mit Selenium ist ein Prozess, der je nach der zu testenden Funktionalität mehr oder weniger lange dauert. Je komplexer die Funktionalität ist, desto komplexer ist auch das Schreiben des Tests. In diesem Kapitel geht es darum, den Prozess der Testimplementierung zu beschreiben. Dazu wird ein Test verwendet, der als Beispiel für die Beschreibung des Implementierungsprozesses dienen soll : **Der Registrierungstest**. Dieser Test wurde ausgewählt, weil er fast alle Funktionen abdeckt, die bei der Erstellung der anderen Tests verwendet wurden. Das Schreiben von Tests ist in fünf Phasen unterteilt:

Phase 1: Initialisierung der Daten

Vor der Durchführung von Tests ist es wichtig, die zu testende Plattform mit Daten vorzubereiten. Dies ermöglicht es, die Ausgangssituation der Plattform, die man testen möchte, zu reproduzieren. Wenn beispielsweise die Verbindung eines Benutzers mit einer Plattform getestet werden soll, ist es wichtig, dass die Daten dieses Benutzers bereits in der Datenbank vorhanden sind.

Im aktuellen Fall wird dies von einem Docker-Container namens Initializer (wird im Kapitel über Docker ausführlich behandelt) durchgeführt, dessen Zweck es ist, ein Skript auszuführen, das Daten in den JBoss-Server von jExam einspeist. Im Falle des Registrierungstests wird eine gültige, aber nicht auf der Plattform registrierte Matrikelnummer generiert (siehe Abbildung 4.7).



	C1	C2	C3
1	no-course	9238494	asdasd
2	Bachelor Informatik (01.10.2009)	t042100	asdasd
3	Bachelor Medieninformatik (01.10.2009)	t400014	asdasd
4	Diplom Informatik (01.09.2010)	t004061	asdasd
5	Master Informatik (09.06.2010)	t010470	asdasd
6	Master Medieninformatik (09.06.2010)	t910004	asdasd

Abbildung 4.7. Vom Initializer erzeugte CSV-Daten

Phase 2: Erstellung eines Testszenarios

In dieser Phase muss der Tester ein Ausführungsszenario für seinen Test erstellen. Es geht darum, die zu testende Funktion manuell auf der Plattform auszuführen und die Ausführungsschritte zu dokumentieren, die in den nächsten Schritten automatisch wiederholt werden. Dieser Prozess könnte auch in schriftlicher Form erfolgen (normalerweise bei langen Szenarien). Diese Phase dient dazu, den Test im Detail zu planen

4. Implementierung der automatisierten Testinfrastruktur

und so die spätere Umsetzung zu erleichtern. Im Falle des Registrierungstests ist das Szenario wie folgt festgelegt:

1. Gehen Sie auf die Login-Seite und füllen Sie das Formular mit der nicht registrierten Matrikelnummer aus.
2. Klicken Sie auf den Login-Button und Sie werden zur Registrierungsseite weitergeleitet.
3. Füllen Sie das Registrierungsformular aus und klicken Sie auf Registrieren.
4. Weiterleitung zur Seite Privacy Policy und Zustimmung zu den allgemeinen Nutzungsbedingungen (prüfen Sie, ob der TestUser automatisch eingeloggt wird).
5. Logout und Login erneut mit den Zugangsdaten und der Matrikelnummer, die Sie bei der Registrierung verwendet haben.
6. Wenn alles richtig funktioniert, war die Registrierung erfolgreich.

Phase 3: Erstellung von Objekt-Seiten

Wie bereits im Unterabschnitt 4.4.2 erwähnt, wird das Object Page Entwurfsmuster für die Erstellung von Tests verwendet. Vor der Implementierung des Szenarios muss der Tester zunächst die Objekte und Seiten erstellen, die für die Ausführung des Szenarios erforderlich sind. Im Rahmen des Registrierungstests müssen folgende Seiten erstellt werden: LoginPage, RegistrationPage, ContractPage (Privacy Policy) und die MainPage. Wenn die RegistrationPage im Detail betrachtet wird, kann man feststellen, dass diese Java-Klasse (wie alle anderen Object-Seiten) einige besondere Merkmale aufweist.

Zunächst erbt sie von der Klasse Header (die den auf allen Seiten vorhandenen Header repräsentiert). Der Header selbst stammt von PageObject ab, das die Grundstruktur einer Seite repräsentiert (siehe Abbildung 4.8). Die Klassen Header und PageObject enthalten Webelemente und Methoden, die normalerweise auf allen Seiten der Anwendung zugänglich sind. Dadurch kann das Design Pattern Page Object optimal genutzt werden.

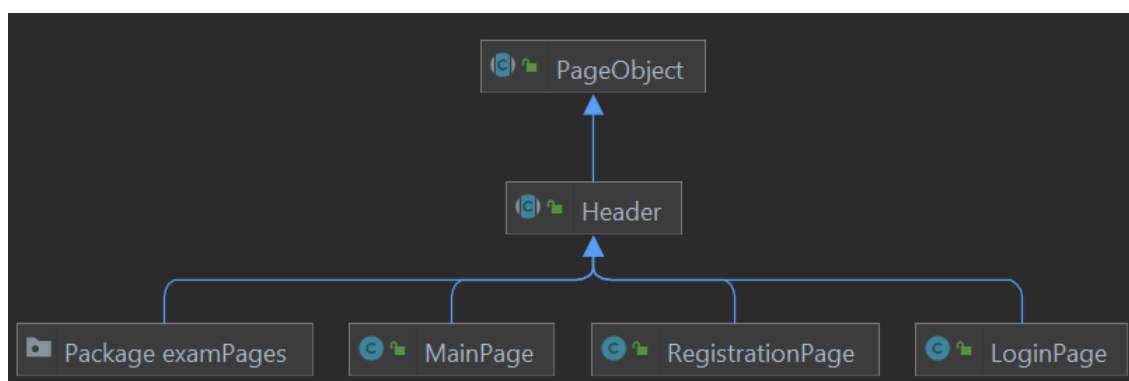


Abbildung 4.8. UML Diagramm für die Java Page-Objekte

Zweitens enthält die Registrationpage eigene Webelemente. Zum Beispiel die verschiedenen Inputs des auszufüllenden Formulare und die anzuklickenden Bestäti-

gungsbuttons (siehe Quellcode 4.4). Darüber hinaus verfügt die Seite über die Methode `registerUser`, die ein Formularobjekt (`RegistrationForm`) als Parameter hat und alle Eingaben des Formulars ausfüllt und die `ContractPage` zurückgibt.

Quelltext 4.4 RegistrationPage Quellcode

```
// RegistrationPage.java

public class RegistrationPage extends Header {

    @FindBy(id = "cfirstname")
    private WebElement fieldFirstName;

    @FindBy(id = "csurname")
    private WebElement fieldName;

    @FindBy(id = "cmatrikel")
    private WebElement matrikel;

    public RegistrationPage(WebDriver driver) {
        super(driver);
    }

    @Override
    public boolean isInitialized() {
        WebElement h1_title;
        try{
            h1_title =
                driver.findElement(By.xpath("//H1[text()='Registrierung']"));
        } catch (NoSuchElementException e ){
            ExtentLogger.info("Main Title was not found");
            return false;
        }
        return h1_title.isDisplayed();
    }

    public ContractPage registerUser(RegistrationForm registrationForm){

        sendKeysToElement(fieldFirstName, registrationForm.firstName);
        sendKeysToElement(fieldName, registrationForm.lastName);

        Select fieldGender = new Select(driver.findElement(By.id("cgender")));
        fieldGender.selectByValue(registrationForm.gender);

        Select fieldCountry = new
            Select(driver.findElement(By.id("ccountry")));
        fieldCountry.selectByValue(registrationForm.country);

        Select fieldLanguage = new
            Select(driver.findElement(By.id("clanguage")));
        fieldLanguage.selectByValue(registrationForm.defaultLanguage);

        matrikel.clear();
        sendKeysToElement(matrikel, registrationForm.matrikel);

        Select fieldFaculty = new
            Select(driver.findElement(By.id("ou-combo")));
```

4. Implementierung der automatisierten Testinfrastruktur

```
fieldFaculty.selectByValue(registrationForm.faculty);

Select fieldDegree = new
    Select(driver.findElement(By.id("degree-combo")));
fieldDegree.selectByValue(registrationForm.degree);

Select fieldExamReg = new
    Select(driver.findElement(By.id("er-combo")));
fieldExamReg.selectByValue(registrationForm.examRegulation);

WebElement addFaculty = driver.findElement(By.id("addEr"));

clickElement(addFaculty);

WebElement validationBtn =
    driver.findElement(By.xpath("(//INPUT[@class='btn
        fill-primary'])[2]"));
clickElement(validationBtn);

return new ContractPage(driver);
}
}
```

Phase 4: Erstellen von Verbindungen zwischen den Seiten

Wenn auf einen Link geklickt oder einfach ein Formular bestätigt wird, wird man normalerweise auf eine andere Seite weitergeleitet. Diese Phase befasst sich im Wesentlichen mit der Umsetzung dieser Aufgabe. Im Rahmen der Implementierung des Registrierungstests geht es darum, die Seiten zwischen zwei Aktionen miteinander zu verbinden. Zum Beispiel sollte die Registrierungsfunktion nach einem Klick auf den Button Login die RegisterPage zurückgeben (siehe Abbildung 4.9). Dies ist in der Tat eine Modellierung der Web-Navigation in Java. Diese Arbeit muss durchgeführt werden und macht in der letzten Phase Sinn.

```
public RegistrationPage startRegistration(String matrikel, String password){
    driver.get(Url.getUrlLoginPage());
    LoginPage loginPage = new LoginPage(driver);
    if (!loginPage.isInitialized()) {
        ExtentLogger.fail("Can't login, because login page isn't initialized");
        throw new IllegalStateException("Can't login, because login page isn't initialized");
    }
    enterLoginData(matrikel, password);
    clickElement(buttonLogin);

    return new RegistrationPage(driver);
}
```

Abbildung 4.9. StartRegistration Funktion

Phase 5: Implementierung der tests nach dem szenario

Diese Phase konzentriert sich auf die Umsetzung des in Phase 2 beschriebenen Szenarios. Es handelt sich um das Schreiben des Tests an sich. In dieser Phase wird

der Tester die Aktionen eines Nutzers nacheinander ausführen.

Im Fall des Registrierungstests geht es zunächst darum, sich mit einer Matrikelnummer zu registrieren, die nicht in der Datenbank verzeichnet ist. Dann wird das Registrierungsformular mit den zuvor festgelegten Daten ausgefüllt. Im nächsten Schritt werden die Allgemeinen Geschäftsbedingungen akzeptiert und der Testuser dann einloggt. Um zu überprüfen, ob die Registrierung erfolgreich war, muss der Tester einen Login-Test mit den Daten simulieren, die er bei der Registrierung verwendet hat (siehe Quellcode 4.5). Zwischen all diesen Schritten werden Verifikationsprozesse durchgeführt (z. B. ob eine Seite korrekt initialisiert wurde).

Es ist wichtig, Logs und ExtentLogger zu verwenden, um jeden Schritt der Tests zu dokumentieren. Dadurch kann bei der Erstellung des Ausführungsberichts nachvollzogen werden, welche Schritte während des Tests durchgeführt wurden. Außerdem können diese Logs zeigen, ob die Tests abgeschlossen wurden oder ob während der Ausführung ein Fehler aufgetreten ist.

Quelltext 4.5 Registration Test Quellcode

```
// Registration Test Quellcode

@Test
@TestDescription("New User Registration")
public void doRegistrationTest(){
    ExtentReport.createTest("Test User Registration");
    ExtentLogger.info(" If you dont see \"Test Passed\" at the end, the
        Test didn't passed !");

    TestUser user = TestUser.REGISTRATION_USER;
    RegistrationPage registrationPage = startRegistration(user.username,
        user.password);
    Assert.assertTrue(registrationPage.isInitialized(), "Registrationpage
        was not correctly initialized");
    ExtentLogger.info("Registration Page is correctly initialized !");

    RegistrationForm form = new RegistrationForm(user.username+"@",
        user.username+"@", "f",
        "DE", user.username, "30010", "Bachelor Informatik", "1533186");
    ContractPage contractPage = registrationPage.registerUser(form);
    Assert.assertTrue(contractPage.isInitialized(), "Contractpage should
        be initialized !");
    ExtentLogger.info("Contract Page is initialized");

    MainPage mainPage = contractPage.acceptPrivacyPolicy();
    ExtentLogger.info("Privacy Policy was accepted");

    ExtentLogger.info("Try logout and login !");
    LoginPage loginPage = mainPage.selectLogoutLink();
    Assert.assertTrue(loginPage.isInitialized() , "Expected to land on
        login page, after log in, but landed on another page");
    ExtentLogger.info("Logout was successful, got on login page");
    log.info("Logout was successful, got on login page");

    // New Login with New Identifier
    mainPage = regularLogin(user.username , user.password);
    Assert.assertTrue(mainPage.isInitialized() , "Expected to land on
```

```
        main page, after log in, but landed on another page");
ExtentLogger.info("Login was successful, got on main page");
log.info("Login was successful, got on main page");

loginPage = mainPage.selectLogoutLink();
Assert.assertTrue(loginPage.isInitialized() , "Expected to land on
    login page, after log in, but landed on another page");
ExtentLogger.info("Logout was successful, got on login page");
log.info("Logout was successful, got on login page");

ExtentLogger.info("Registration was successful");
log.info("Registration was successful");
ExtentLogger.pass("Test passed");
}
```

Es gibt noch viele weitere Funktionen und Klassen, die in dieser Arbeit nicht beschrieben werden können. Die Details des Schreibens und aller Funktionen werden jedoch in der Dokumentation der Testinfrastruktur ausführlich erläutert.

4.4.4. Erreichte Ergebnisse

Wie bereits im vorherigen Kapitel erwähnt, werden viele Funktionen der Testinfrastruktur in dieser Arbeit nicht näher erläutert. Es ist jedoch sinnvoll, einen Überblick über die bisherigen Möglichkeiten und Ziele des UI-Tests zu geben.

Einrichtung einer UI-Testumgebung

Eine grundlegende Testumgebung wurde erstellt, um die Implementierung zukünftiger UI-Tests zu ermöglichen. In dieser Testumgebung wurden die Funktionen Login, Registrierung, Abruf der Noten und Einschreibung in Prüfungen getestet. Diese Tests funktionieren für jExam 2009 (nur der Login-Test funktioniert auch für jExam New). Da sich jExam New noch in der Entwicklung befindet, verfügt die getestete Version noch nicht über alle Funktionen, die getestet werden sollen. Die Tests wurden so geschrieben, dass sie auf beiden Plattformen funktionieren. Es ist jedoch möglich, dass in Zukunft noch Anpassungen vorgenommen werden müssen (dieses Problem wird im Auswertungsteil dieser Arbeit ausführlich behandelt). Das ursprüngliche Ziel, eine Testsuite für jExam New zu erstellen und gleichzeitig jExam 2009 auf einmal zu testen, wurde erreicht. Die UI-Testumgebung kann nun verwendet und verbessert werden, um die anderen zahlreichen Funktionen von jExam zu testen.

Testdurchführungsbericht

Nach der Ausführung der UI-Tests generiert die Testumgebung eine HTML-Seite, die einen detaillierten Bericht über die Ausführung der Tests liefert (siehe Unterunterabschnitt 4.4.1). Dies ermöglicht es Ihnen, die Ausführung der Tests zu überwachen und mögliche Fehler zu erkennen.

The screenshot displays the Extent Report HTML page. The top header shows the report title 'Report of Tests execution for Jexam [LOCAL_OLDWEB]' and the date 'Dez. 11, 2021 05:06:06 PM'. The left sidebar lists several test cases, all marked as 'Pass'. The main content area shows the details for the test 'Test login with wrong data', which was executed on 12.11.2021 at 17:06:11. The test duration was 00:00:06:347. The test ID is #test-id=1. The details table shows the status of the test as 'Info' and the timestamp as 17:06:11. The details include a warning message: 'If you dont see "Test Passed" at the end, the Test didn't passed !'. The details also show the steps of the test: 'Navigate to login page, enter data and click login', 'Try to click on: input', 'Clicked on:', 'Main Title was not found', 'Try to click on: input', 'Clicked on:', and 'Main Title was not found'.

STATUS	TIMESTAMP	DETAILS
Info	17:06:11	⚠️ If you dont see "Test Passed" at the end, the Test didn't passed !
Info	17:06:11	Navigate to login page, enter data and click login
Info	17:06:12	Try to click on: input
Info	17:06:12	Clicked on:
Info	17:06:13	Main Title was not found
Info	17:06:13	Try to click on: input
Info	17:06:13	Clicked on:
Info	17:06:15	Main Title was not found

Abbildung 4.10. Extent Report HTML Page

Zusätzliche Funktionalitäten

Die UI-Testumgebung verfügt über mehrere Ausführungsmodi. Sie ermöglicht es, die Tests lokal und remote in einem Docker-Container auszuführen und den Prozess der Testausführung vollständig zu automatisieren.

Die lokale Ausführung wird in der Regel bei der Implementierung der Tests verwendet. Sie ermöglicht es, schnell zu testen, ob die Tests korrekt funktionieren, ohne dass sie in einem Container ausgeführt werden müssen. Dies führt zu einer erheblichen Zeitersparnis bei der Implementierung. Der Local Run Mode bietet auch die Möglichkeit, Tests im Headless-Modus auszuführen. Headless bedeutet, dass die Selenium-Tests mit einem Headless-Browser ausgeführt werden. Er funktioniert wie ein typischer Browser, jedoch ohne Benutzeroberfläche, welcher sich für automatisierte Tests hervorragend eignet. Der lokale Run Mode verwendet einen WebDriverManager. Dies ist sehr nützlich, da es das Problem der Versionierung von Webdrivern löst. Der Tester muss nicht mehr einen Webdriver auf seinen Computer herunterladen, sondern kann den WebDriverManager verwenden, um auf einen bereits konfigurierten und sofort einsatzbereiten Driver zuzugreifen.

Der Remote-Modus ermöglicht die Ausführung von Tests in Docker-Containern mit Hilfe des Selenium-Hubs (dieses Konzept wird im Kapitel über die Docker-Infrastruktur behandelt). Dies ist die Methode, die für eine vollautomatische Ausführung von UI-Tests verwendet wird. Der Remote-Modus ist so konfiguriert, dass er die aktuellste Version des RemoteWebDrivers verwendet (in einem Docker-Container enthalten und mit dem Selenium-Hub verbunden).

Die beiden Testmodi ermöglichen die Durchführung von Tests unter Verwendung der Browser Chrome und Firefox.

4.5. Einrichtung der Docker-Infrastruktur

Zu den Herausforderungen, die von der Testinfrastruktur erwartet werden, gehören Konzepte wie die vollständige Automatisierung, die lokale Installation und die Übertragbarkeit der Testwerkzeuge auf verschiedene Umgebungen. Um diese Erwartungen zu erfüllen, wurden Werkzeuge wie Docker entwickelt. In diesem Kapitel wird zunächst eine kurze Beschreibung von Docker gegeben. Anschließend wird gezeigt, wie Docker bei der Entwicklung der Testinfrastruktur von jExam eingesetzt wurde, und schließlich werden die Ergebnisse der Entwicklung vorgestellt.

4.5.1. Docker

Bei der Entwicklung einer Anwendung gibt es einige Probleme, die sehr oft während der Entwicklung oder sogar während der Produktion auftreten. Diese Probleme werden oft durch die folgenden Aussage formuliert:

1. Das Skript funktionierte gestern aber heute nicht mehr.
2. Das Skript funktioniert auf diesem Rechner und nicht auf dem Rechner eines Kollegen/Kunden.
3. Das Skript funktioniert nur mit Java 5 und nicht mit Java 8.
4. Die Anwendung muss in allen aktuellen Browser-Versionen von Chrome (80,81,82,...,89) und Firefox (78,79,80,...,84) getestet werden. Müssen die alle installiert werden ?

Docker ist eine Open-Source-Containerisierungsplattform. Sie ermöglicht es Entwicklern, Anwendungen in Container (standardisierte ausführbare Komponenten) zu verpacken. Sie kombinieren den Quellcode der Anwendung mit den Betriebssystembibliotheken und Abhängigkeiten, die für die Ausführung dieses Codes in jeder Umgebung erforderlich sind. Container vereinfachen die Bereitstellung verteilter Anwendungen und erfreuen sich zunehmender Beliebtheit (vgl. [10]).

Docker macht das Erstellen, Bereitstellen und Verwalten von Containern leichter, einfacher und sicherer (vgl. [11]). Docker ist im Wesentlichen ein Toolkit, das es Entwicklern ermöglicht, Container mit einfachen Befehlen und arbeitssparender Automatisierung über eine einzige API zu erstellen, bereitzustellen, auszuführen, zu aktualisieren und zu beenden (vgl. [11]). Zu den Vorteilen der Verwendung von Docker gehören :

Portabilität: Ein Container erstellt ein ausführbares Softwarepaket, das vom Host-Betriebssystem abstrahiert (nicht an dieses gebunden oder von diesem abhängig ist) und daher portabel und in der Lage ist, auf jeder Plattform oder Cloud einheitlich und konsistent zu laufen.

Agilität: Docker für die Ausführung von Containern hat den Industriestandard für Container mit einfachen Entwicklertools und einem universellen Paketierungsansatz geschaffen, der sowohl unter Linux- als auch unter Windows-Betriebssystemen funktioniert. Das Container-Ökosystem hat sich auf Engines verlagert, die von der Open Container Initiative (OCI) verwaltet werden. Softwareentwickler können weiterhin agile oder DevOps-Tools und -Prozesse für eine schnelle Anwendungs-

entwicklung und -verbesserung verwenden.

Geschwindigkeit: Container werden oft als leichtgewichtig bezeichnet, was bedeutet, dass sie den Betriebssystemkern des Rechners gemeinsam nutzen und nicht mit diesem zusätzlichen Overhead belastet werden. Dies steigert nicht nur die Effizienz des Servers, sondern senkt auch die Server- und Lizenzkosten und verkürzt die Startzeiten, da das Betriebssystem nicht gebootet werden muss.

Isolierung von Fehlern: Jede containerisierte Anwendung ist isoliert und arbeitet unabhängig von den anderen. Der Ausfall eines Containers hat keine Auswirkungen auf den weiteren Betrieb der anderen Container. Entwicklungsteams können alle technischen Probleme innerhalb eines Containers identifizieren und beheben, ohne dass es zu Ausfallzeiten in anderen Containern kommt. Außerdem kann die Container-Engine alle Sicherheitsisolationstechniken des Betriebssystems nutzen, um Fehler innerhalb der Container zu isolieren.

Effizienz: Software, die in containerisierten Umgebungen ausgeführt wird, teilt sich den Betriebssystemkern des Rechners, und Anwendungsschichten innerhalb eines Containers können von mehreren Containern gemeinsam genutzt werden. Daher haben Container von Natur aus eine geringere Kapazität als eine VM und benötigen weniger Anlaufzeit, so dass weitaus mehr Container auf der gleichen Rechenkapazität wie eine einzelne VM ausgeführt werden können. Dies führt zu einer höheren Server-Effizienz und senkt die Server- und Lizenzierungskosten.

Sicherheit: Die Isolierung von Anwendungen in Form von Containern verhindert von Natur aus, dass bösartiger Code in andere Container oder das Hostsystem eindringt. Darüber hinaus können Sicherheitsberechtigungen definiert werden, um unerwünschte Komponenten automatisch am Eintritt in Container zu hindern oder die Kommunikation mit unnötigen Ressourcen einzuschränken.

Docker bietet also die Möglichkeit, ein Werkzeug zu bauen, das auf allen Arten von Systemen unabhängig von den lokalen Konfigurationen der einzelnen Benutzer funktioniert. Es spielt also keine Rolle, welche Java-Version oder welches Betriebssystem ein Benutzer hat (vgl. [11]). Anstatt eine Vielzahl von Abhängigkeiten herunterzuladen und zu installieren, kann man auch einfach einen Container mit allen notwendigen Abhängigkeiten erstellen und ihn auf jedem System verwenden, das Docker unterstützt. Die Verwendung von Docker wird all diese Vorteile für die Entwicklung der Testinfrastruktur nutzen. Das zusätzliche Werkzeug zur Orchestrierung der verschiedenen Container ist jedoch Docker-compose.

Wenn Sie eine Anwendung aus Prozessen in mehreren Containern erstellen, die sich alle auf demselben Host befinden, können Sie Docker-Compose verwenden, um die Anwendungsarchitektur zu verwalten. Docker Compose erstellt eine YAML-Datei, die die Dienste angibt, die in der Anwendung enthalten sein sollen, und kann die Container mit einem einzigen Befehl bereitstellen und ausführen. Mithilfe von Docker Compose ist es möglich, persistente Volumes für den Speicher zu definieren, Basisknoten anzugeben und die Abhängigkeiten der Dienste zu dokumentieren und zu konfigurieren.

Im nächsten Kapitel wird beschrieben, wie Docker und Docker-Compose für die Einrichtung der Testinfrastruktur von jExam verwendet wurden.

4.5.2. Docker jExam Infrastructur

Um eine stabile Testinfrastruktur zu schaffen und eine bessere Verwaltung der Container zu ermöglichen, wurde Docker-Compose verwendet, um die verschiedenen Testdienste von jExam zu trennen und zu gruppieren. So kann der Tester nur die Dienste ausführen, die er braucht und die er testen will. Wenn alle Container gleichzeitig gestartet würden, könnte dies sehr viele Ressourcen auf dem Host-Computer beanspruchen und das gesamte System verlangsamen. Da einige Dienste voneinander abhängig sind, wurde ein Docker-Netzwerk geschaffen, das die Kommunikation zwischen den ausgeführten Diensten ermöglicht. Dies ermöglicht ein geschlossenes lokales Netzwerk, das von außen nicht einsehbar ist (siehe Abbildung 4.11).

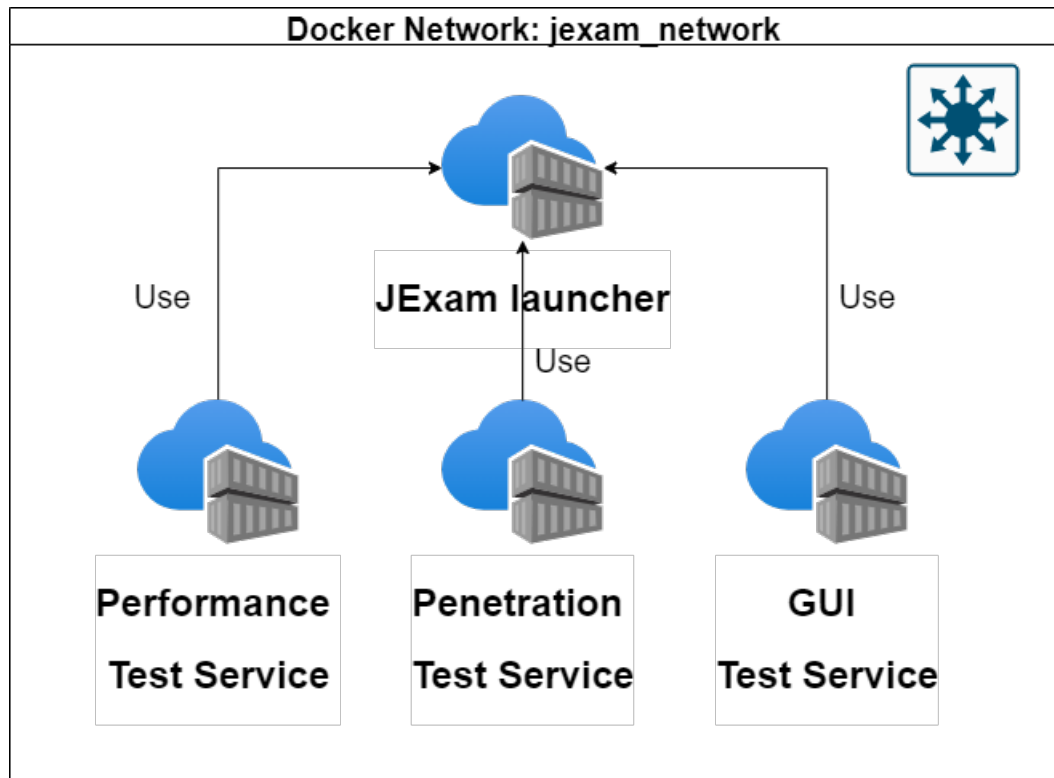


Abbildung 4.11. jExam Docker Network Darstellung

jExam Launcher

jExam Launcher ist der Hauptdienst der Docker Infrastruktur. Er ist für die Ausführung und Bereitstellung von jExam New und jExam 2009 zuständig. Der Dienst besteht aus fünf Containern, darunter :

JBoss: Dieser Container enthält das Skript zum Bereitstellen des JBoss-Servers, den jExam New und jExam 2009 benötigen, um funktionieren zu können. Dieser Server dient auch als Datenbank für die beiden Apps. Er legt daher einige seiner Ports offen, um für externe Verbindungen zugänglich und offen zu sein.

Initializer: Dieser Container enthält ein Java Skript, das automatisch (oder auf Wunsch des Testers auch manuell) ausgeführt wird und dazu dient, Testdaten in

den JBoss-Server zu injizieren. Es initialisiert die Daten, die für die Ausführung der Tests erforderlich sind.

OldWeb: Dieser Container ermöglicht das Deployment von jExam 2009. Er enthält einen Tomcat-Webserver, der den Code von jExam 2009 ausführt und sich dann mit dem Container verbindet, der den JBoss-Server enthält. Dieser Container legt ebenfalls seine Ports offen, damit er von den Tests erreicht werden kann, wenn diese im lokalen Modus ausgeführt werden.

Webservice und Web: Diese beiden Container ermöglichen die Ausführung und Bereitstellung von jExam New. Der Webservice enthält eine API, die vor allem mit dem JBoss-Server kommuniziert. Der Web-Container enthält das Frontend der Anwendung. Web verbindet sich mit dem Webservice, wenn er gestartet wird, und stellt Ports zur Verfügung, die ihn zugänglich machen, wenn die Tests im lokalen Modus gestartet werden.

jExam launcher

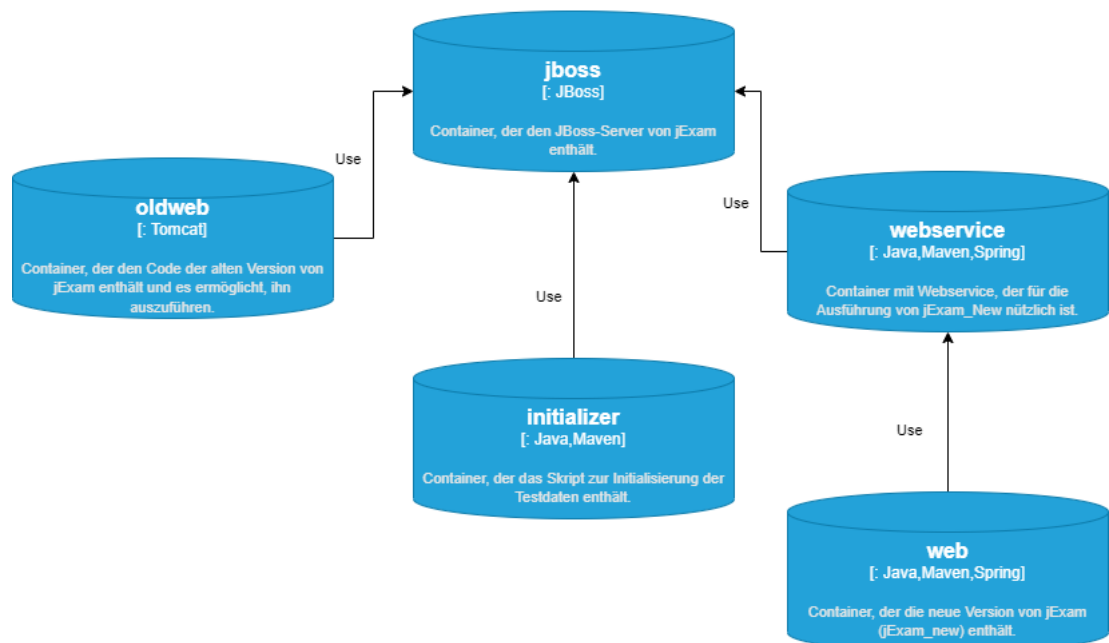


Abbildung 4.12. jExam Launcher Testservice

UI-Testservice

Das UI-Testservice ist die Dienststelle, die für die Durchführung von UI-Tests verantwortlich ist. Es besteht im Wesentlichen aus vier Containern:

Selenium-Hub, Chrome- und Firefox-RemoteWebDriver: Der Selenium Hub ist ein Server, der die Zugriffsanfragen vom WebDriver-Client entgegennimmt und die JSON-Testbefehle an die entfernten (remote) Driver weiterleitet. Er nimmt die Befehle vom Client entgegen und führt sie parallel auf den verschiedenen remote Driver aus. Dies ermöglicht die Verwendung der RemoteWebDriver Chrome und Firefox, die direkt mit dem Selenium-Hub verbunden sind. Der Tester kann

also wählen, welchen Browser er für die Ausführung seiner Tests verwenden möchte. Es ist jedoch wichtig, darauf hinzuweisen, dass die Container, die die RemoteWebDriver enthalten, so konfiguriert sind, dass sie mit der neuesten Version des Browsers ausgeführt werden, die bei jedem Docker Image Build automatisch heruntergeladen wird.

UI-Test: Dieser Container enthält die UI-Tests und ist dafür zuständig, diese automatisch mithilfe eines Maven-Skripts auszuführen. Wenn die Tests im Remote-Modus konfiguriert sind, verbindet sich der Container mit dem Selenium-Hub, um Zugriff auf den verfügbaren RemoteWebDriver zu erhalten.

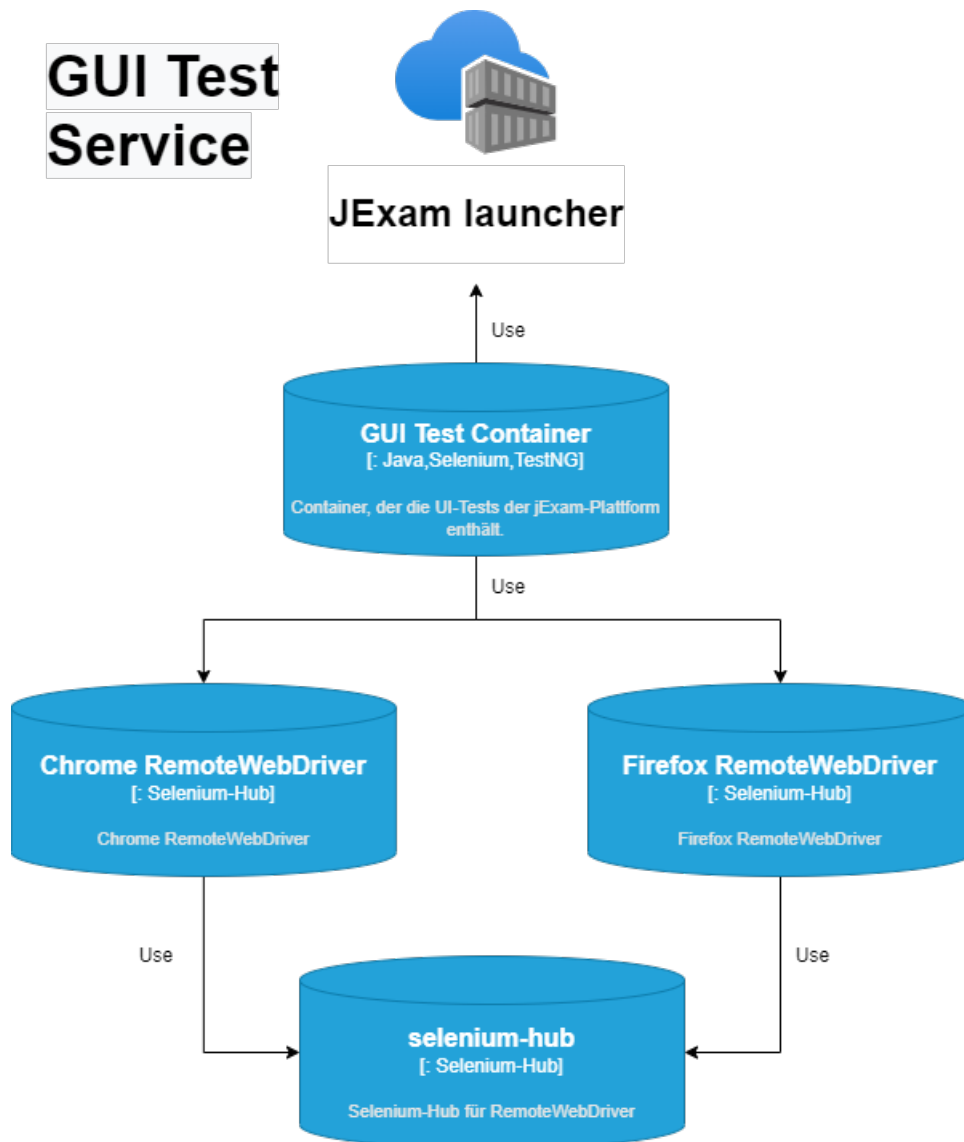


Abbildung 4.13. jExam UI Testservice

Penetration Testservice

Das Penetration Testservice ist der Dienst, der für die Ausführung von Penetrations- und Sicherheitstestskripten verantwortlich ist. Dieser Dienst besteht aus einem einzigen Container:

Owasp: Dieser Container besteht aus einem stabilen Abbild von Zaproxy und enthält die verschiedenen Skripte, die auf jExam New und jExam 2009 ausgeführt werden: zap-baseline und zap-fullscan.

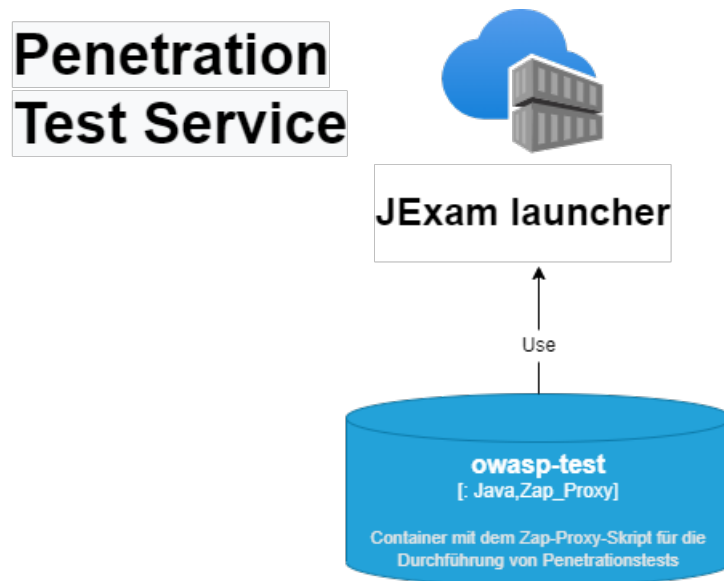


Abbildung 4.14. jExam Penetration Testservice

Performance Testservice

Der Performance Testservice ist der Dienst, dessen Rolle die Durchführung von Performancetests ist. Er besteht im Wesentlichen aus zwei Containern:

JMeter-old und JMeter-New: Diese beiden Container enthalten jeweils die JMeter-Skripte für jExam 2009 und jExam New. Die JMeter-Skripte wurden für beide Plattformen angepasst, da sie mehrere unterschiedliche Endpunkte haben.

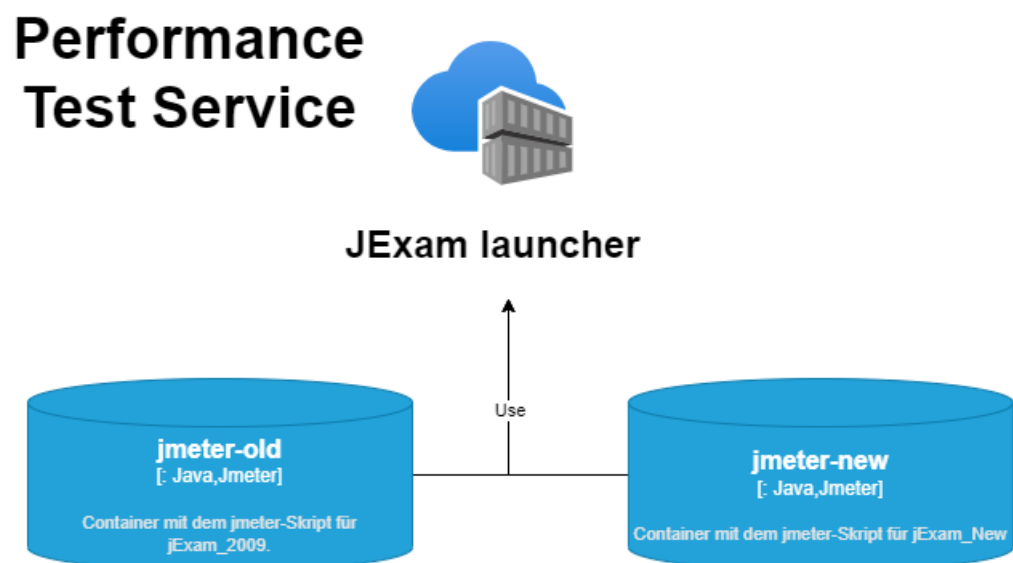


Abbildung 4.15. jExam Performance Testservice

4.5.3. Erreichte Ergebnisse

Durch die Verwendung von Docker wurde eine Infrastruktur aufgebaut, die es ermöglicht, beide Versionen von jExam automatisch zu testen. Diese Infrastruktur bietet die Möglichkeit, Sicherheits-, Performance- und funktionale Tests durchzuführen. Dank dieser Infrastruktur muss der Tester nichts mehr lokal auf seinem Computer konfigurieren. Sobald die Infrastruktur installiert ist, muss er nur noch wissen, wie die verschiedenen Docker-Compose-Befehle funktionieren, um die Dienste zu starten, die er verwenden möchte. Details zum Code und zur Nutzung der Infrastruktur sind in der Dokumentation beschrieben.

Die Infrastruktur weist jedoch noch einige Unzulänglichkeiten auf, die behoben werden sollten. Im nächsten Kapitel werden diese Unzulänglichkeiten und die möglichen Fehler, die bei der Durchführung von Tests auftreten können, ausführlich beschrieben. Zweitens werden mögliche Lösungen und Perspektiven für die Verbesserung der Infrastruktur und der Testwerkzeuge diskutiert.

5. Evaluation

Das Ziel dieser Arbeit war es, eine Testinfrastruktur für die jExam-Plattform zu erstellen. Diese Infrastruktur sollte den Testern als Basis dienen und den Prozess der Testentwicklung erleichtern. Dieses Kapitel konzentriert sich auf die Bewertung, ob diese Ziele erreicht wurden, und geht auf die Einschränkungen ein, während potenzielle Verbesserungen vorgeschlagen werden.

5.1. Erreichte Ziele

In einem ersten Schritt sollten die Funktionen von jExam 2009 getestet werden. Danach sollte herausgefunden werden, ob es möglich ist, die Tests von jExam 2009 zum Testen von jExam New wiederzuverwenden und sogar eine Testsuite für jExam New bereitzustellen, bevor die Entwicklung abgeschlossen ist. Dies wurde durch die Verwendung des Selenium-Tools ermöglicht, da jExam 2009 und jExam New praktisch die gleiche grafische Benutzeroberfläche haben. Die in Unterabschnitt 3.3.1 beschriebenen Funktionen wurden für jExam 2009 getestet und nur die Login-Funktion für jExam New (weil die anderen Funktionen noch nicht entwickelt wurden). Der gleiche Login-Test funktioniert also für beide Plattformen. Das Ziel für UI-Tests wurde erreicht.

Zweitens wurde die Frage gestellt, inwieweit es möglich ist, Sicherheitstests zu integrieren. Diese Frage wurde in Abschnitt 4.2 behandelt und es zeigte sich, dass es tatsächlich möglich ist, Sicherheitstests zu integrieren. Zu diesem Zweck wurden Schwachstellen-Scanner eingebaut, die beide Versionen von jExam auf ausnutzbare Sicherheitslücken scannen.

Drittens ging es um die Frage, ob Bottleneck-Performance in beiden Versionen von jExam festgestellt werden konnte. Dies war ein schwer zu lösendes Problem, da sich jExam New noch in der Entwicklung befindet. Das JMeter-Tool wurde jedoch in die Testinfrastruktur integriert und mit Skripten versehen, um beide Versionen zu testen. Bei der Analyse der von diesen Skripten erstellten Berichte wurden kürzere Antwortzeiten für jExam New festgestellt, was auf eine mögliche Verbesserung hindeuten könnte. Die verwendeten JMeter-Skripte sind jedoch recht einfach und eignen sich nicht zum genauen Testen von Anwendungen. Sie sollten daher verbessert werden.

Die Testinfrastruktur wurde so konzipiert, dass sie auf allen Systemen tragbar, leicht installierbar, vollautomatisch und für die Durchführung von Tests konfigurierbar ist. Daher kann man sagen, dass die Testinfrastruktur die in Abschnitt 3.3 beschriebenen Anforderungen erfüllt.

5.2. Begrenzung und mögliche Lösungen

Es gibt jedoch noch einige Probleme, die gelöst werden sollten, um die Testinfrastruktur erheblich zu verbessern. Dieser Abschnitt konzentriert sich vor allem auf diese Themen.

5.2.1. Probleme beim UI-Testing

Die Entwicklung von UI-Tests erforderte, UI-Tests für jExam New bereitzustellen und gleichzeitig jExam 2009 zu testen. Die beiden Anwendungen haben eine fast identische grafische Benutzeroberfläche, aber es gibt deutliche Unterschiede im HTML-Code (siehe Abbildung 3.2). Selenium stützt sich beim Auffinden von Elementen hauptsächlich auf diesen HTML-Code. Dies könnte für die Entwicklung von Tests problematisch werden. Tests, die für jExam 2009 funktionieren, funktionieren nicht automatisch für jExam New. Sie müssen daher angepasst werden.

Dieses Problem kann leicht gelöst werden, wenn die Entwickler einen Selenium-Entwicklungsansatz verwenden. Dieser Ansatz besteht darin, dass man entwickelt, während man weiß, dass man Selenium-Tests zum Testen schreiben wird. Entwickler, die diesen Ansatz verwenden, fügen ihren anklickbaren Elementen (Hyperlinks, Buttons, Formularelementen) HTML-Attribute (id, name, value) hinzu, die einen einfachen Zugriff auf ein HTML-WebElement ermöglichen. Dies erleichtert die Entwicklung von Tests mit Selenium erheblich. Die Entwickler sollten sich auch bemühen, die gleiche Frontend-Struktur in beiden Anwendungen beizubehalten. Auf diese Weise müssen die Tester einen Test nur einmal schreiben und er kann auf beiden Plattformen funktionieren.

Unter den Erwartungen an UI-Tests gibt es einen bestimmten Test, der mit Selenium nicht durchführbar ist. Es handelt sich dabei um den Abruf der Noten als PDF. Selenium ist begrenzt und hat keine Möglichkeit, diese Aufgabe zu realisieren.

5.2.2. Probleme bei der Initialisierung von Daten

Die Initialisierung der Daten auf der Plattform erfolgt mithilfe eines Java Skripts, das die Daten in den JBoss-Server injiziert. Dies bringt jedoch einige Probleme mit sich, die den Testzyklus verlangsamen. Beim Testen einer Anwendung muss der Tester sehr oft zu einer Ausgangssituation zurückkehren, um seine Tests erneut auszuführen, was mit dem Initializer nicht möglich ist. Es ist daher technisch nicht möglich, denselben Benutzer mehrmals für bestimmte Tests zu verwenden, da die Daten nicht mehr neu sind und beschädigt wurden. Aus diesem Grund muss der Tester nach jedem Test eine Datei testData.csv (Abbildung 4.7) erstellen und diese Benutzer verwenden. Dies verkompliziert den Testprozess. Die Tatsache, dass man nicht zur Ausgangssituation zurückkehren kann (die Daten nicht zurücksetzen kann), verhindert, dass man die volle Kontrolle über das Schreiben der Tests hat. Das Testen von bestimmten Fällen

ist noch schwieriger zu reproduzieren. Die fehlende Kontrolle über das Erzeugen und Löschen von Daten nimmt dem Tester die Möglichkeit, die Anwendung vollständig zu testen.

Es ist wichtig zu beachten, dass der Initializer, der für die Testinfrastruktur verwendet wird, noch in der ersten Version vorliegt und noch wenig entwickelt ist. Es könnte vermieden werden, dass bei jedem Start neue Benutzer generiert werden, wenn es die Möglichkeit gäbe, die Testdaten zurückzusetzen. Eine Lösung könnte auch darin bestehen, eine lokale Testdatenbank zu erstellen, auf die von einem Docker-Container aus zugegriffen werden kann.

5.2.3. Leistungsprobleme

Die Testinfrastruktur ist nicht frei von Fehlern. Sie stammen nicht direkt aus der Infrastruktur, sondern aus den verschiedenen Anwendungen, aus denen sie besteht (jExam 2009, jExam New und vor allem JBoss). Bei der Durchführung von Tests kann es zu Inkonsistenzen und Fehlern auf der Ebene des JBoss-Servers kommen. Manchmal werden die Daten nicht korrekt angezeigt. Dies führt dazu, dass der JBoss-Server in einem Docker-Container nicht stabil ist.

Das zweite Problem ist, dass die Testinfrastruktur schwerfällig ist und sehr viele Ressourcen auf dem Host-Computer beansprucht. Um dieses Problem zu lösen, sollte die Infrastruktur durch eine bessere Nutzung von Docker-Compose oder sogar durch eine mögliche Integration mit dem Kubernetes-Tool optimiert werden.

6. Zusammenfassung und Ausblick

6.1. Zusammenfassung

Das Ziel dieser Arbeit ist die Entwicklung einer WebTestsuite für die jExam-Plattform. Dies wurde in die Entwicklung einer Testinfrastruktur übersetzt. Diese Testinfrastruktur sollte nicht nur nicht-funktionale Tests wie Performance- und Sicherheitstests, sondern auch funktionale Tests beinhalten. Darüber hinaus sollte die Infrastruktur vollautomatisch, leicht wartbar und einfach zu installieren sein. Da sich die neue Version von jExam noch in der Entwicklung befindet, sollten die Tests der Infrastruktur bereits für die neue Version vorbereitet sein, während sie noch auf der alten Version laufen.

Um dieses Ziel zu erreichen, musste zunächst die Grundlage für die Arbeit gelegt werden, indem die wichtigsten Begriffe und Konzepte definiert wurden, die zum Verständnis der Arbeit beitragen können. Danach folgte eine Analysephase, in der das zu lösende Problem analysiert und die Ziele definiert wurden, die bei der Entwicklung erreicht werden sollten. Nach dieser Phase folgte die Implementierungsphase, in der die Testinfrastruktur im Detail beschrieben und die verschiedenen Werkzeuge, die zu ihrer Erstellung verwendet wurden, vorgestellt wurden. Schließlich folgte die Auswertungsphase (Evaluation), in der es konkret analysiert wurde, ob die Ziele der Infrastruktur erreicht wurden.

Mit Hilfe von Docker, Maven und anderen Werkzeugen wurde eine Infrastruktur aufgebaut, die beide Versionen von jExam mit dem ZAP-Proxy-Tool auf Sicherheitslücken untersuchen kann. Diese Infrastruktur ermöglicht es auch, die Leistung der beiden Versionen auf verschiedenen Ebenen mit dem integrierten JMeter-Tool zu testen. Funktionale Tests wurden mit Selenium, einem der beliebtesten und von Entwicklern am häufigsten verwendeten Testwerkzeuge, durchgeführt. Die Infrastruktur ermöglicht die automatische Ausführung von Tests, sobald sie geschrieben und in die Plattform integriert wurden. Zu den Herausforderungen dieser Arbeit gehört die Erstellung einer Dokumentation. Diese Aufgabe wurde ebenfalls erfüllt. Die Dokumentation ist im Anhang dieser Arbeit zu finden. Die ursprünglichen Ziele, die in der Einleitung zu dieser Arbeit definiert wurden, wurden erreicht.

6.2. Ausblick

Die Testinfrastruktur in ihrem derzeitigen Zustand dient lediglich als Basis für die Entwicklung von jExam-Tests. Die Sicherheits- und Performancetests, die integriert wurden, sind ziemlich minimalistisch und sollten weiter verbessert werden. Dasselbe gilt für die UI-Tests, die derzeit nur einige Funktionen der Plattform testen. Für eine bessere Abdeckung ist es jedoch wichtig, mehr Tests zu schreiben und alle Funktionen abzudecken. Eine so sensible Plattform wie jExam sollte in allen Details getestet werden.

Es gibt jedoch noch einige Probleme, die unabhängig von der Testinfrastruktur an sich sind und die gelöst werden sollten, um die Testinfrastruktur erheblich zu verbessern. Ein Beispiel hierfür ist die Abhängigkeit der Testinfrastruktur vom JBoss-Server, die häufig zu Dateninkonsistenzproblemen führt, da dieser Server Verbindungen zu entfernten Datenbanken herstellt. Dieses Problem und mögliche Lösungen wurden in Kapitel 5 behandelt.

Die jExam-Testinfrastruktur ist relativ jung und könnte noch erheblich verbessert werden. Nicht nur die bestehenden Tests können weiter verbessert werden, sondern es besteht auch die Möglichkeit, neue Testarten zu integrieren. Dazu gehören Stress-tests, Volumetests, Usabilitytests und andere Arten von funktionalen Tests, die sich von UI-Tests unterscheiden. Die Möglichkeiten sind sehr breit gefächert und Docker könnte an einem bestimmten Punkt auch eingeschränkt werden. Dies würde eine Tür für den Einsatz von Tools wie Kubernetes öffnen, das ein Open-Source-System zur Automatisierung der Bereitstellung, Skalierung und Verwaltung von containerisierten Anwendungen ist (vgl. [5]).

Literatur

- [1] A9:2017-Using Components with Known Vulnerabilities. https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities. 2017.
- [2] Rafa E. Al-Qutaish Alain Abran. „ISO 9126: Analysis of Quality Models and Measures“. In: (2010).
- [3] A Anand. „Importance of Software Testing in the Process of Software Development“. In: *International Journal for Scientific Research and Development* 12.6 ().
- [4] B. Arkin, S. Stender und G. McGraw. „Software penetration testing“. In: *IEEE Security Privacy* 3.1 (2005), S. 84–87.
- [5] The Kubernetes Authors. *Kubernetes*. <https://kubernetes.io/>. 2021.
- [6] Francisco Gortázar Boni García Micael Gallego und Mario Munoz-Organero. „A Survey of the Selenium Ecosystem“. In: (30/07/2020).
- [7] Lionel Briand und Yvan Labiche. „A UML-based approach to system testing“. In: *Software and systems modeling* 1.1 (2002), S. 10–42.
- [8] Mark Buenen und Govind Muthukrishnan. „World Quality Report“. In: https://www.capgemini.com/wp-content/uploads/2018/01/world-quality-report-seured-2017-181.pdf?utm_source=pardot&utm_medium=email&utm_content=none_none_none_report_none&utm_campaign=optimize_wqr (2017-18).
- [9] B. Vasundhara Devi. „A Review of Testing Techniques and Principles in Software Quality Assurance Testing“. In: *International Journal of Scientific Engineering Research, Volume 6, Issue 8, August-2015* (2015).
- [10] Docker. <https://www.docker.com/>.
- [11] IBM Cloud Education. *Containerization*. <https://www.ibm.com/cloud/learn/containerization>. 2021.
- [12] Apache Software Foundation. *JMeter Documentation*. <https://jmeter.apache.org/>. 2021.
- [13] Gerard J. Holzmann. „The Logic of Bugs“. In: *SIGSOFT Softw. Eng. Notes* 27.6 (Nov. 2002), S. 81–87.
- [14] QA InfoTech. *Best 9 Performance Testing Tools Trending In 2021*. <https://qainfotech.com/best-9-performance-testing-tools/>. 2021.

- [15] Gerti Kappel u. a. *Web Engineering : Die Disziplin zur systematischen Entwicklung von Web-Anwendungen*.
- [16] KirstenS. OWASP. *About the OWASP Fondation*. 2020: <https://owasp.org/www-community/attacks/xss/>. 2017.
- [17] Pavan Kumar und Khasim Syed. „Software testing goals, principles, and limitations“. In: 12 (2010).
- [18] H.K.N. Leung und L. White. „A study of integration testing and software regression at the integration level“. In: *Proceedings. Conference on Software Maintenance 1990*. 1990, S. 290–301.
- [19] Microsoft. *Serialisierung (C)*. <https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/concepts/serialization/>.. 2020.
- [20] S. Murugesan. „Attitude towards testing: a key contributor to software quality“. In: *Proceedings of 1994 1st International Conference on Software Testing, Reliability and Quality Assurance (STRQA'94)*. 1994, S. 111–115.
- [21] Maile Ohye. „Every second does count“. In: https://youtu.be/0pMfx_Zie2g?t=567 (2010).
- [22] OWASP. *About the OWASP Fondation*. 2020: <https://owasp.org/about/>.
- [23] Perfecto. „Switching to UI Automation: Everything You Need to Know“. In: <https://www.perfecto.io/blog/switching-ui-automation-everything-you-need-know#whatisuiautomationtesting> (2020).
- [24] Portswigger *Insecure deserialization 2021*. <https://portswigger.net/web-security/deserialization>. 2021.
- [25] Chaitanya Pujari. *Page Object Model and Page Factory in Selenium*. <https://www.browserstack.com/guide/page-object-model-in-selenium>. 2021.
- [26] Rosnisa Abdull Razak und Fairul Rizal Fahrurazi. „Agile testing with Selenium“. In: *2011 Malaysian Conference in Software Engineering*. 2011, S. 217–219.
- [27] IBM Security. *Cost of a Data Breach Report*. 2020.
- [28] Monika Sharma und Rigzin Angmo. „Web based automation testing and tools“. In: *International Journal of Computer Science and Information Technologies* 5.1 (2014), S. 908–912.
- [29] Alan Shultz. „Software Testing: Testing Across the Entire Software Development Lifecycle“. In: *Software Quality Professional* 14.1 (2011), S. 50.
- [30] Sadhvi Singh. *Learn about TestNG Framework – How to Automate using Selenium*. <https://www.browserstack.com/guide/testng-framework-with-selenium-automation>. 2019.
- [31] Sanjay Kumar Singh und Amarjeet Singh. *Software testing*. Vandana Publications, 2012.
- [32] Statista. „Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025“. In: <https://www.statista.com/statistics/871513/worldwide-data-created/> (2021).
- [33] Simon Stewart. „The Architecture of Open Source Applications : Selenium Web-Driver“. In: *aosabook.org* (2016).
- [34] Andrew van der Stock. „OWASP Top 10 2017 The Ten Most Critical Web Application Security Risks“. In: (2017).

- [35] Kazuyuki Takeuch. „AIRCRAFT ACCIDENT INVESTIGATION REPORT“. In: https://reports.aviation-safety.net/1994/19940426-0_A306-B-1816.pdf (1996).
- [36] Filippou I Vokolos und Elaine J Weyuker. „Performance testing of software systems“. In: *Proceedings of the 1st International Workshop on Software and Performance*. 1998, S. 80–87.
- [37] Fei Wang und Wencai Du. „A Test Automation Framework Based on WEB“. In: *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*. 2012, S. 683–687.
- [38] J.A. Whittaker. „What is Software Testing? And Why Is It So Hard?“ In: *IEEE Software*, vol. 17, no. 1 (2000).
- [39] OWASP WWW-Community. *Active Scan Rules*. <https://www.zaproxy.org/docs/desktop/addons/active-scan-rules/>. 2021.
- [40] OWASP WWW-Community. *Ajax Spider*. <https://www.zaproxy.org/docs/desktop/addons/ajax-spider/options/>. 2021.
- [41] OWASP WWW-Community. *OWASP Benchmark*. <https://owasp.org/www-project-benchmark/>. 2021.
- [42] OWASP WWW-Community. *Passive Scan Rules*. <https://www.zaproxy.org/docs/desktop/addons/passive-scan-rules/>. 2021.
- [43] OWASP WWW-Community. *Vulnerability Scanning Tools*. https://owasp.org/www-community/Vulnerability_Scanning_Tools. 2021.
- [44] OWASP WWW-Community. *ZAP Official Website*. <https://www.zaproxy.org/>. 2021.
- [45] OWASP WWW-Community. *ZAP Spider*. <https://www.zaproxy.org/docs/desktop/start/features/spider/>. 2021.

A. Appendix

A.1. Dokumentation der jExam Testinfrastruktur

A.1.1. Über die Testinfrastruktur

Das Ziel dieses Projekts ist die Entwicklung einer vollautomatischen automatisierte Testinfrastruktur zum Testen der jExam-Plattform zu schaffen. In ihrem derzeitigen Zustand umfasst die Test Testinfrastruktur Leistungstests, Sicherheitstests Sicherheits-tests und funktionale UI-Tests. Der Projektcode ist eine Grundlage für die Entwicklung von Tests für die der jExam-Plattform und muss in den nächsten Monaten und Jahren weiterentwickelt Monaten und Jahren weiterentwickelt und gepflegt werden.

A.1.2. Verwendete Technologien

Zum Erstellen der Testinfrastruktur:

Docker (Stellen Sie sicher, dass Sie Docker auf Ihrem Gerät installiert haben)

Für UI-Tests:

1. Selenium
2. TestNG
3. AssertJ - fluent assertions java library
4. Extent Reports

Für Leistungstests:

JMeter

Für Sicherheitstests:

ZAP Proxy

Um die Skripte ändern zu können oder Fehler zu verstehen, die in Zukunft auftreten könnten, ist es für den Tester unerlässlich, Grundkenntnisse über die oben genannten Technologien zu haben.

A.1.3. Einrichtung des Projekts

Klonen Sie das Repository mit :

Quelltext A.1 Clone repository

```
git clone ssh://ipra38.inf.tu-dresden.de/jexam/jexam-webtest.git
```

Die allgemeine Struktur der Docker-Infrastruktur sieht folgendermaßen aus:

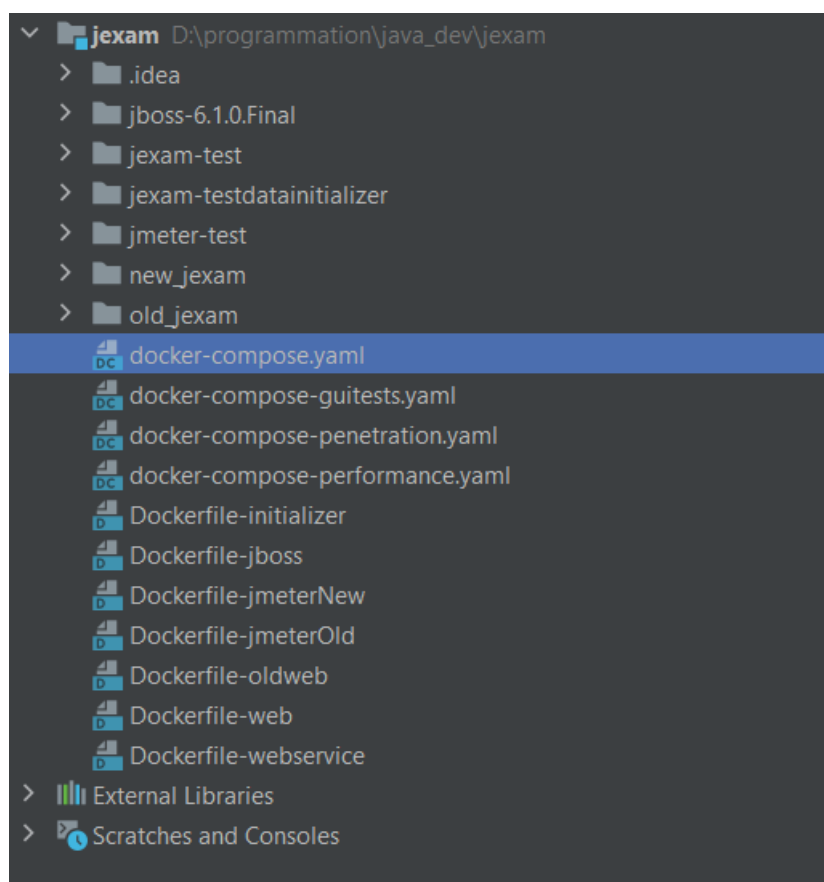


Abbildung A.1. Globale Dateistruktur der jExam-Testinfrastruktur

1. **jboss-6.1.0.Final** : Dieser Ordner enthält den Quellcode zum Starten des jboss-Servers, der von der jExam-Infrastruktur verwendet wird.
2. **jexam-test**: Dieser Ordner ist das Selenium-Projekt, das die UI-Tests enthält.
3. **jexam-testdatainitializer**: Dieser Ordner enthält das Java-Skript zum Einspeisen der Daten in den jboss-Server.
4. **JMeter-test**: Dieser Ordner enthält das JMeter-Skript, das für Leistungstests nützlich ist.

5. **new_jexam** und **old_jexam**: Diese Ordner enthalten den Quellcode der neuen bzw. alten Version der jExam-Webanwendung.

A.1.4. jExam-Anwendungen starten

Bevor ein Skript ausgeführt werden kann, muss zunächst Docker gestartet werden. Wenn dies geschehen ist, muss der Tester ein Docker-Netzwerk mit dem Befehl erstellen:

Quelltext A.2 Docker Network Einrichtung

```
docker network create jexam_network
```

Sobald das Netzwerk erstellt ist, soll der Tester den folgenden Befehl ausführen:

Quelltext A.3 Befehl zum Starten der jExam-Plattformen

```
docker-compose up
```

```
/* oder wenn Änderungen in einem der folgenden Ordner vorgenommen wurden  
   (jboss-6.1.0.Final, old_jexam, new_jexam, jexam-testdatainitializer): */
```

```
docker-compose up --build
```

Dies ermöglicht die Ausführung von jExam 2009 und jExam New, die über die folgenden Links aufgerufen werden können:

jExam 2009 : <http://localhost:8085/web/>

jExam New : <http://localhost:8080/>

Nach der Ausführung des Skripts erzeugt der Initialisierer eine csv-Datei, die unter `jexam-test/initializer-csv/testData.csv` zugänglich ist und alle Testdaten enthält, die in den JBoss-Server eingespeist wurden.

A.1.5. Durchführung von Performancetests

Die Durchführung der Performancetests erfordert die Ausführung des folgenden Befehls:

Quelltext A.4 Befehl zur Durchführung der Performancetests

```
docker-compose -f docker-compose-performance.yaml up
```

```
/* oder wenn Änderungen in jmeter-test Ordner vorgenommen wurden: */
```

```
docker-compose -f docker-compose-performance.yaml up --build
```

Im Ordner `jmeter-test/tests` befinden sich die Dateien `jExam_new.jmx` und `jExam_old.jmx`, die jeweils die JMeter-Tests für die Plattformen jExam New und jExam 2009 enthalten. Nach der Ausführung der Tests wird ein Bericht erstellt, der vom Tester in:

jExam 2009 : jexam-test/jmeter-newjexam-test-output

jExam New : jexam-test/jmeter-oldjexam-test-output

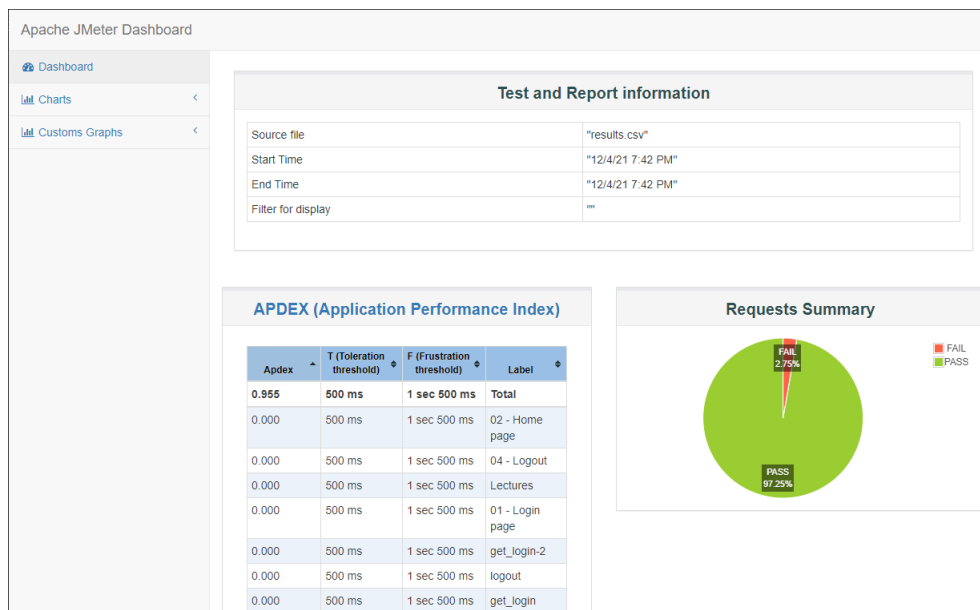


Abbildung A.2. Beispiel für einen JMeter-Bericht

A.1.6. Durchführung von Sicherheitstests

Die Durchführung der Sicherheitstests erfordert die Ausführung des folgenden Befehls:

Quelltext A.5 Befehl zur Durchführung der Sicherheitstests

```
docker-compose -f docker-compose-penetration.yaml up
```

/ oder wenn nderungen in docker-compose-penetration.yaml Datei vorgenommen wurden: */*

```
docker-compose -f docker-compose-penetration.yaml up --build
```

Es gibt zwei mögliche ausführbare Skripte für Sicherheitstests: ZAP-Baseline und ZAP-Fullscann. Der Tester muss im Voraus auswählen, welches Skript er für welche Anwendung ausführen möchte. Zu diesem Zweck muss er die Datei **docker-compose-penetration.yaml** ändern:

Quelltext A.6 docker-compose-penetration.yaml

```
command: [ "./wait-for-it.sh", "web:8080", "bash" , "-c", "zap-baseline.py
-t http://web:8080 -r owaspReport.html" ]
# web:8080 for jExam New and oldweb:8080 for jExam 2009
# "zap-baseine.py -t" for ZAP-Baseline scan or
# "zap-full-scan.py -d -j -m 1 -t" for ZAP-Fullscan
```

Der von der Testausführung erstellte Bericht ist im Ordner jexam-test/owasp-test-output verfügbar.

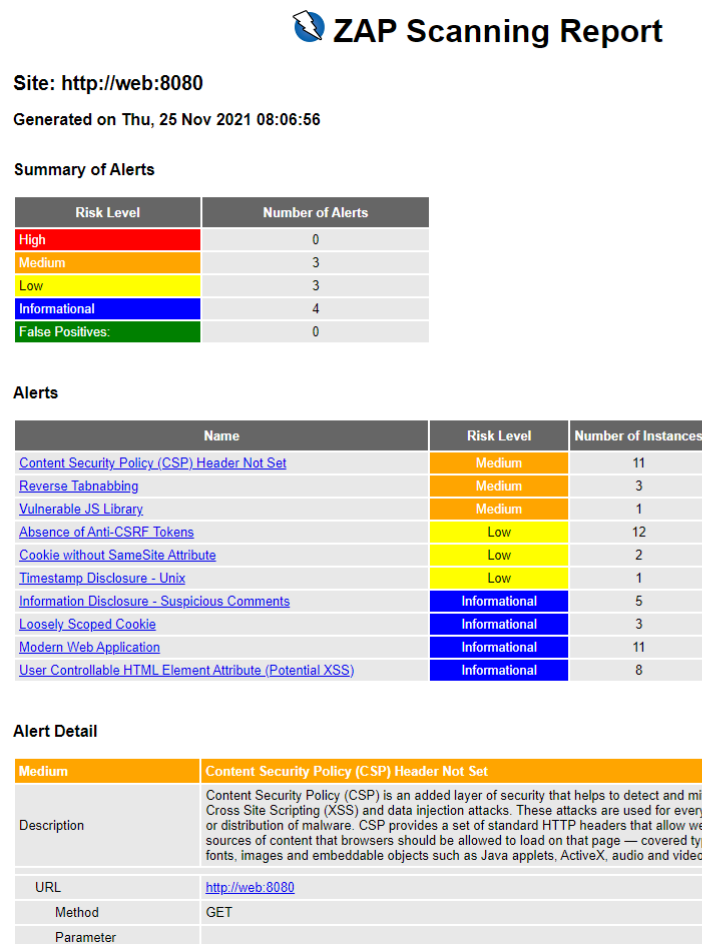


Abbildung A.3. Beispiel für einen ZAP Proxy-Bericht

A.1.7. Durchführung von Seleniumtests

Vor der Ausführung der Tests muss der Tester unbedingt die vom jexam-testdatainitializer bereitgestellten Testdaten einrichten. Der Tester muss die Daten aus der Datei jexam-test/initializer-csv/testData.csv in die Java-Klasse jexam-test/src/test/java/jexam/web/test/entities/TestUser kopieren.

```
public enum TestUser {

    TEST_USER_RANDOM( username: "random", password: "random", course: "" ),
    REGISTRATION_USER( username: "9238494", password: "asdasd", course: "no-course" ),
    //REGISTRATION_USER("1288989","asdasd","no-course"),
    TEST_USER_0( username: "s7099086", password: "asdasd", course: "Bachelor Medieninformatik (2009)" ),
    BA_INF( username: "t923868", password: "asdasd", course: "" ),
    BA_MEDINF( username: "t613982", password: "asdasd", course: "" ),
    DIPL_INF( username: "t271976", password: "asdasd", course: "" ),
    DIPL_INF2( username: "t236893", password: "asdasd", course: "" ),
    MA_INF( username: "t246893", password: "asdasd", course: "" ),
    MA_MINF( username: "t696832", password: "asdasd", course: "" );

}
```

Abbildung A.4. TestUser Java Class

Ausführungsmodus

Es gibt zwei Ausführungsmodi für UI-Tests, nämlich den **lokalen** Modus und den **Remote** Modus.

Der **lokale** Modus ist der Modus, den der Tester bei der Entwicklung der Tests verwendet. Der Hauptvorteil dieses Modus ist, dass er eine schnelle Testausführung ermöglicht und keinen Remote-Webdriver erfordert.

Um die Tests im lokalen Modus auszuführen, muss der Tester in die Datei `jexam/jexam-test/src/test/resources/config/config.properties` gehen und dort einige Anpassungen vornehmen:

Quelltext A.7 config.properties

```
#Browser (chrome or firefox)
browser=chrome

#run mode (local) --> Jexam version (LOCAL_WEB for jExam New, LOCAL_OLDWEB
    for jExam 2009)
#run mode (remote) ---> Jexam version (WEB jExam New, OLDWEB for jExam 2009)
runmode=local

# Jexam version (WEB, OLDWEB, LOCAL_WEB, LOCAL_OLDWEB)
jexam_version = LOCAL_WEB

seleniumgridurl=http://selenium-hub:4444/wd/hub

#headless or not
headless=false

overridereports = yes
```

Nach der Konfiguration muss der Tester den folgenden Befehl im `jexam-test`-Ordner ausführen:

Quelltext A.8 Maven Test Command

```
mvn clean test
```

Der **remote** Modus wird hauptsächlich für die Ausführung von Tests in Docker-Containern verwendet. Um ihn zu verwenden, muss der Tester auch Anpassungen in der Datei `jexam/jexam-test/src/test/resources/config/config.properties` vornehmen. Zum Beispiel:

Quelltext A.9 config.properties

```
runmode=remote
jexam_version=WEB
```

Der Tester sollte den folgenden Befehl in der Stammdatei der Testinfrastruktur verwenden, um sie im Remote-Modus in Docker-Containern auszuführen und am Ende der Testausführung wird ein Extent-Bericht erstellt, der im Ordner `jexam-test/extent-test-output` verfügbar ist.

Quelltext A.10 docker-compose execution command

```
docker-compose -f docker-compose-guitests.yaml up --build
```

A.2. Präsentation der grafischen Oberfläche der Berichtstools

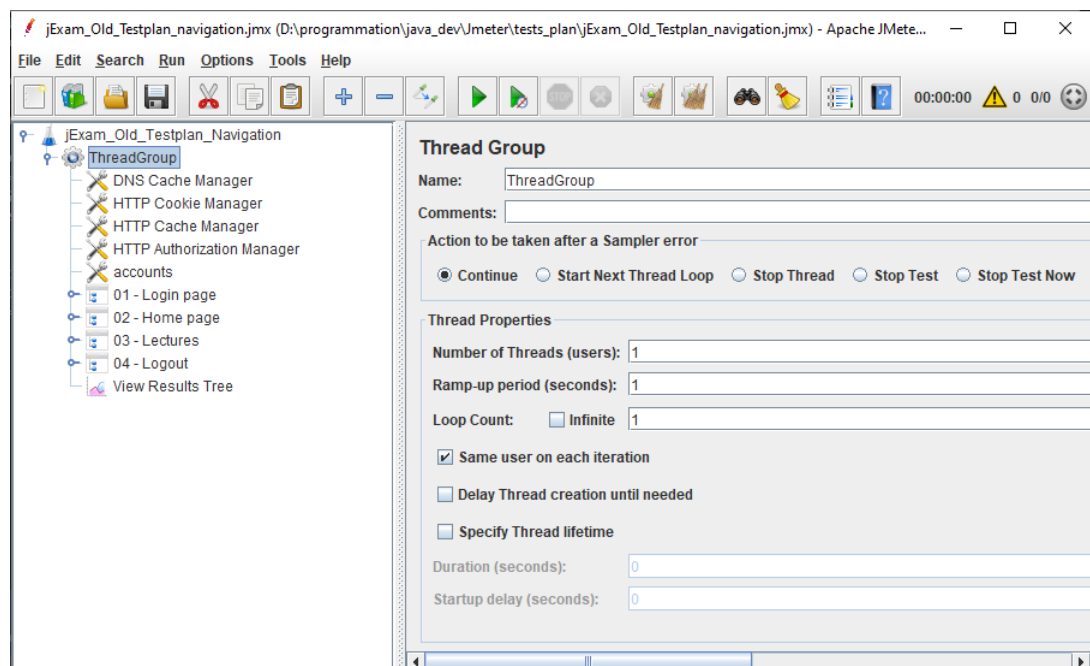


Abbildung A.5. Grafische Benutzeroberfläche von Jmeter

A.3. UML-Klassendiagramm und globale Struktur für einige Pakete der UI-Tests

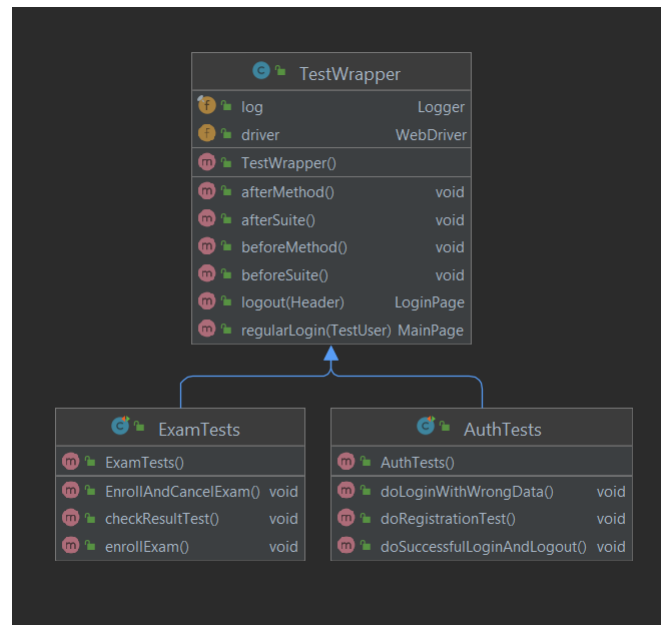


Abbildung A.6. UML-Diagramm des Pakets systemTests

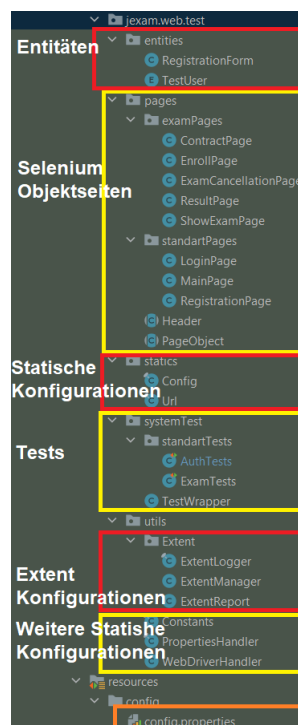


Abbildung A.7. Genereller Struktur der UI-Tests

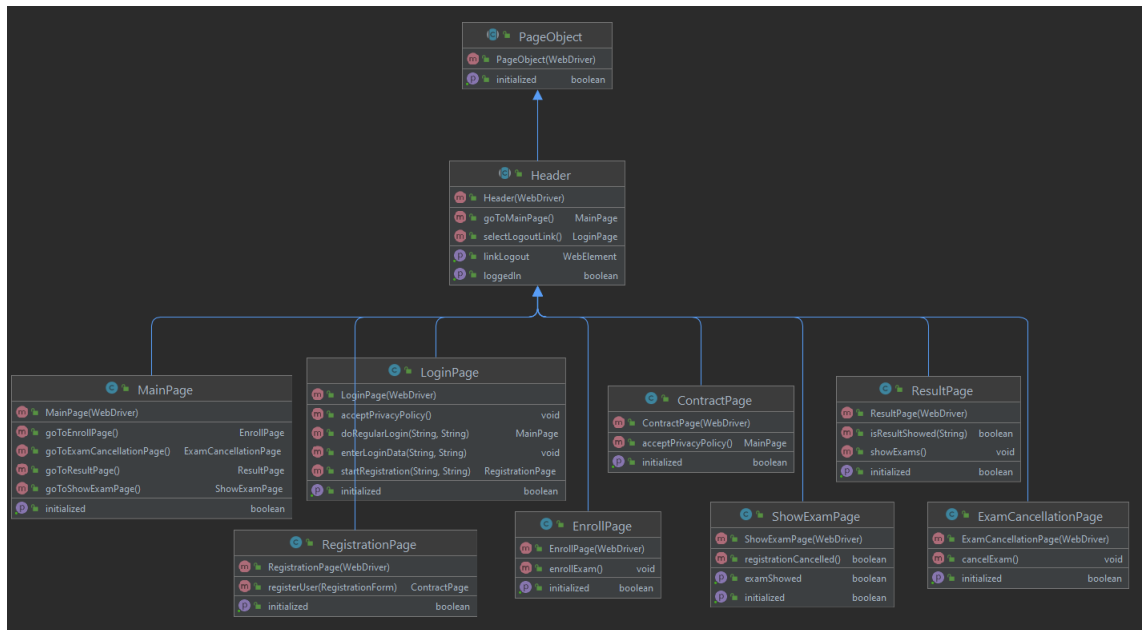


Abbildung A.8. UML-Diagramm des Pakets pages

A.4. Einige Beispiele für Code Sources, die zum Verständnis beitragen

Quelltext A.11 Docker-Compose-Datei des Launcher-Dienstes

```

# Docker Compose File
version: "3"

# docker network create jexam_network
# important command !
# docker-compose up
# docker-compose -f docker-compose-penetration.yaml up
# docker-compose -f docker-compose-performance.yaml up
# docker-compose -f docker-compose-guitests.yaml up
# Must connect on TU Dresden vpn

services:
  jboss:
    build:
      context: ./
      dockerfile: ./Dockerfile-jboss
    container_name: jboss
    ports:
      - "1090:1090"
      - "1091:1091"
      - "1099:1099"
      - "40003:40003"
      - "4446:4446"
      - "4457:4457"
      - "4712:4712"
      - "4713:4713"
  
```

A. Appendix

```
- "4714:4714"
- "9009:9009"

oldweb:
  build:
    context: ./
    dockerfile: Dockerfile-oldweb
  container_name: oldweb
  stdin_open: true
  tty: true
  depends_on:
    - jboss
  ports:
    - "8085:8080"

webservice:
  build:
    context: ./
    dockerfile: ./Dockerfile-webservice
  container_name: webservice
  stdin_open: true
  tty: true
  depends_on:
    - jboss
  ports:
    - "8081:8081"

web:
  build:
    context: ./
    dockerfile: Dockerfile-web
  container_name: web
  stdin_open: true
  tty: true
  depends_on:
    - webservice
  ports:
    - "8080:8080"

initializer:
  build:
    context: ./
    dockerfile: Dockerfile-initializer
  container_name: initializer
  stdin_open: true
  tty: true
  depends_on:
    - jboss
  volumes:
    - ./jexam-test/initializer-csv:/initializer/data/

networks:
  default:
    external:
      name: jexam_network
```

Quelltext A.12 Dockerfile Datei des jExam New Container

```
# Dockerfile-Web (For jExam_New)

FROM openjdk:17-alpine
RUN apk --update add bash && apk --no-cache add dos2unix

COPY ./new_jexam/jexam-web /webapp

WORKDIR /webapp

# JARS kopieren

COPY ./new_jexam/jexam-webservice/. /webapp
COPY ./jboss-6.1.0.Final/server/jExamV5/lib/bos.jar /webapp
COPY ./jboss-6.1.0.Final/server/jExamV5/lib/common.jar /webapp
COPY ./jboss-6.1.0.Final/server/jExamV5/lib/csapis.jar /webapp

RUN rm -rf target
RUN rm -rf de.jexam.webservice

# CHANGE BASE_URL
RUN sed -i "s/localhost/web-service/g"
    src/main/java/de/jexam/web/data/WebserviceConnector.java

RUN dos2unix mvnw

# RUN INSTALL JARS
RUN ./mvnw install:install-file -Dfile=bos.jar -DgroupId=de.jexam
    -DartifactId=bos -Dversion=1.0 -Dpackaging=jar
RUN ./mvnw install:install-file -Dfile=common.jar -DgroupId=de.jexam
    -DartifactId=common -Dversion=1.0 -Dpackaging=jar
RUN ./mvnw install:install-file -Dfile=csapis.jar -DgroupId=de.jexam
    -DartifactId=csapis -Dversion=1.0 -Dpackaging=jar

# INSTALL WEB CLASSES
RUN ./mvnw install:install-file
    -Dfile=/webapp/de.jexam.web.classes/target/classes-0.0.1-SNAPSHOT.jar
    -DgroupId=de.jexam -DartifactId=web-classes -Dversion=1.0
    -Dpackaging=jar

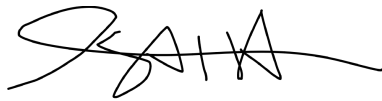
RUN ./mvnw clean compile

ENTRYPOINT sleep 40;./mvnw spring-boot:run
```

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich das vorliegende Dokument mit dem Titel *Entwicklung einer Web-Testsuite am Beispiel von jExam* selbstständig und ohne unzulässige Hilfe Dritter verfasst habe. Es wurden keine anderen als die in diesem Dokument angegebenen Hilfsmittel und Quellen benutzt. Die wörtlichen und sinngemäß übernommenen Zitate habe ich als solche kenntlich gemacht. Es waren keine weiteren Personen an der geistigen Herstellung des vorliegenden Dokumentes beteiligt. Mir ist bekannt, dass die Nichteinhaltung dieser Erklärung zum nachträglichen Entzug des Hochschulabschlusses führen kann.

Dresden, 29. Januar 2021

A handwritten signature in black ink, appearing to read 'M. Saha', with a stylized, flowing script.

Michael Vivian Mboni Saha