Konzeption und Implementierung automatisierter Last- und Performancetests im Rahmen des Testens von non-functional Requirements

Marlis Teufel (B.Sc) und Professor Dr.-Ing. Frank Herrmann
Labor für Informationstechnik und Produktionslogistik (LIP)
Ostbayerische Technische Hochschule Regensburg
E-Mail: marlis.teufel@st.oth-regensburg.de und frank.herrmann@oth-regensburg.de

ABSTRACT

Nicht-funktionale Anforderungen an Software jeglicher Art nehmen aktuell kontinuierlich zu, weswegen gerade ihre Perfomance- und Belastungsqualitäten sich gleichermaßen den gestiegenen Erwartungen anpassen müssen. Diese Arbeit befasst sich mit den Möglichkeiten, die Leistungsfähigkeit von Systemen in diesem Sinne zu überprüfen, was sich im untersuchten Beispiel einer Web-App als dringend nötig herausgestellt hat: Weder Performance noch Belastbarkeit der Applikation werden den zuvor definierten Testzielen gerecht, wodurch sogleich ein empirisch belegtes Argument für die verstärkte Einführung solcher Tests noch vor dem Release-Zeitpunkt gegeben wird. Denn nur in Form von belastbarem und validem Datenmaterial lässt sich entweder eine reibungslose Funktionsfähigkeit eines Systems oder aber der Bedarf einer Überarbeitung belegen. Der vorliegende Artikel plädiert dafür, diesen zusätzlichen, aber überschaubaren kosten- und zeittechnischen Aufwand in Kauf zu nehmen, da der Nutzen der Testverfahrens umso größer ist.

SCHLÜSSELWÖRTER

Non-functional Requirements, Lasttest, Performancetest, Open-Source-Tool, Apache JMeter

Einleitung

In der heutigen Zeit ist das Smartphone unser permanenter Wegbegleiter. Die von ihm gebotene Vielfältigkeit, für jede alltägliche Situation eine App zur Verfügung zu haben, ist kaum noch aus dem Alltag wegzudenken.
Software soll vielfältige Funktionalität bieten, mit zunehmender Komplexität wird die Dringlichkeit nach leistungsstarker Performance immer größer.
Das Testen von nicht-funktionalen Anforderungen an eine Software erhöht dessen Qualität und somit auch die Kundenzufriedenheit. Ein breites Angebot an Testing-Tools ermöglicht eine Vielfalt an Testmöglichkeiten.

Ausgangssituation

Diese Arbeit wird bei der Firma intive GmbH realisiert, welche mobile Applikationen konzeptioniert, designt und entwickelt.
Auch Qualitätssicherung spielt bei intive GmbH eine sehr große Rolle. Bislang lag der Fokus vor allem auf funktionalem Testen. Nun soll auch der Bereich des Testens von non-functional Requirements verstärkt erforscht und vor allem mehr Wissen über Performance- und Lasttests aufgebaut werden.
Gleichwohl bietet die intive GmbH ihren Kunden auch die Möglichkeit an, das Test-Management für deren Applikationen zu übernehmen. Somit sind nicht nur für die Firma selbst die Ergebnisse der Performance- und Lasttests von Wichtigkeit: Der Kunde hat ebenfalls hohes Interesse daran, zu wissen, wie „stabil“, ergo wie leistungs- und belastungsfähig, sein Produkt ist.
Für diese Arbeit wird eine Webapplikation getestet, die als Rechnungs- und Zahlungsservice dient. Es ist möglich, Konten anzulegen und diesen Zahlungsmöglichkeiten zuzuweisen. Das System ist mit Zahlungsdienstleistern, sogenannten PSP, verbunden, welche Zahlungsverfahren, z.B. Kauf auf Rechnung, Kauf mit Kreditkarte oder PayPal bereitstellen.
Bislang dient die subjektive Wahrnehmung als Indikator, ob die Performance und Belastbarkeit des Produkts den Kundenerwartungen entspricht. Es wurde die Vermutung angestellt, dass die Belastbarkeit und Performance der Webanwendung nicht im Akzeptanzbereich liegt. Es werden daher messbare Werte benötigt, um eine Bewertung vornehmen zu können.
Hierfür gibt es eine Reihe von verschiedenen Tools, mit denen man Performance- und Lasttests umsetzen kann. Neben den kommerziellen Test-Tools, die oft mit hohen Kosten verbunden sind, gibt es auch eine Auswahl von Open-Source-Werkzeugen, die für Entwickler und Unternehmen frei zur Verfügung stehen.
Die nachfolgende Abbildung zeigt die Darstellung der Arbeitsschritte, die vorgenommen werden, um die Performance und Belastbarkeit einer Webapplikation testen zu können.

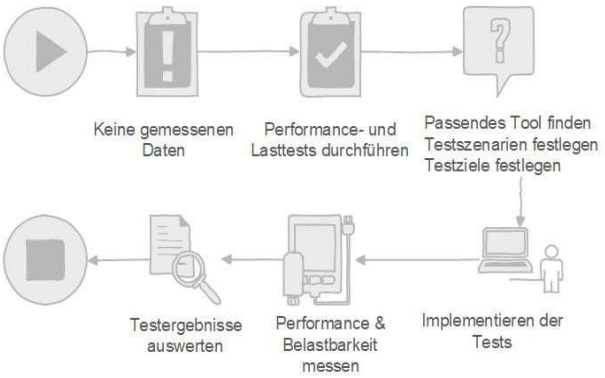

Abbildung 1 Workflow (Eigene Darstellung)

Grundlagen

In diesem Abschnitt werden die begrifflichen und methodischen Grundlagen für das konkrete Projekt gelegt.

Softwarequalität

Softwarequalität ist eine Begrifflichkeit, die nicht immer eindeutig expliziert werden kann, da Entwickler und Projektleiter des Öfteren unterschiedliche Auffassungen darüber besitzen, wie dies zu tun sei. Softwarequalität wird von der ISO/IEC 25000 (Software-Engineering – Qualitätskriterien und Bewertung von Softwareprodukten) wie folgt festgelegt [Fran14]:

„Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen."

Durch diese Definition wird klar, dass sich Softwarequalität anhand verschiedener Kriterien ermitteln lässt und es auf das Zusammenspiel eben dieser einzelnen Bewertungskategorien ankommt.
Laut der Norm wird in sechs Qualitätsmerkmale unterschieden, eines davon ist die Effizienz. Sie beschreibt, wie sich das Leistungsniveau des Anwendungsprogramms zu den eingesetzten Betriebsmitteln unter zuvor festgelegten Bedingungen verhält. Ein Teilmerkmal hiervon wäre das Zeitverhalten, die sogenannte Performanz [Fran14].

Nicht-funktionale Anforderungen

Um die Softwarequalität zu steigern, ist es wichtig nicht nur die Funktionalität eines Programms zu testen, sondern auch die nicht-funktionalen, sogenannten technischen Anforderungen.
Diese betreffen in der Regel mehrere oder alle funktionalen Anforderungen, und sie beeinflussen sich außerdem oft auch gegenseitig. Funktionale Anforderungen beantworten immer die Frage, *was* ein Softwareprodukt tun soll, nicht-funktionale Anforderungen stellen dagegen fest, *wie* ein Programm arbeitet. [BaLi11]

Zu den meistgenannten nicht-funktionalen Anforderungen gehören nach Balzert [BaLi11]:

Leistung (performance)
Wie schnell reagiert das System auf z.B. einen Button-Klick? Hängt das System bei zu vielen Zugriffen? Wird es im Laufe der Zeit langsamer?

Zuverlässigkeit (reliability)
Werden die Daten zwischen den Komponenten richtig übermittelt? Läuft die Software auf verschiedener Hardware fehlerfrei?

Benutzbarkeit, Gebrauchstauglichkeit (usability)
Wie benutzerfreundlich ist die Oberfläche dieser Software? Sind Vorkenntnisse notwendig um diese Software zu nutzen?

Sicherheit (security)
Sind Daten vor Verlust gesichert? Und gleichermaßen von Manipulation? Konkreter gefragt: Wird nicht-autorisierte Datenmanipulation verhindert?

Wartbarkeit (maintainability)
Ist es möglich am System Änderungen oder Verbesserungen vorzunehmen?

Da es den Rahmen dieser Arbeit sprengen würde, auf jede der nicht-funktionalen Anforderungen einzugehen, liegt der Fokus auf dem Leistungsaspekt, was heißt, dass zum einen die Performance, zum anderen die Belastbarkeit getestet werden.

Performance- und Lasttests

Die Performance gewinnt immer mehr an Bedeutung, da die Endnutzer in der Regel nur sehr kurze Antwortzeiten erwarten, laut [Walt08] gilt ein Wert von fünf Sekunden als Orientierungsmaß.
In der Literatur werden Performance- und Lasttests oft als Synonyme verwendet. Die Grundidee dahinter verfolgt das Ziel, ermitteln zu können, ob ein System, wenn es unter großer Belastung zur Anwendung kommt, selbst dann noch immer reibungslos funktioniert.
Eine Software wird durch eine große Zahl gleichzeitiger Anfragen oder auch komplexer Datenbankabfragen an ihre Kapazitätsgrenzen gebracht, was in der Folge möglich machen soll, Performance-Probleme zu lokalisieren und zu beheben. [Pilo12]
Der Performance-Test im Speziellen geht der Frage nach, wie viel Ressourcen (Zeit und Speicherplatz) des getesteten Systems unter normalen Bedingungen verbraucht werden. Dagegen soll der Lasttest überprüfen, wie sich das Gesamtsystem bei steigender Systemlast verhält. Dadurch sollen Schwachstellen – sogenannte *Bottlenecks* – im System feststellbar gemacht werden. [Fran14]
Somit sind Performance- und Lasttests ein wichtiges Instrument, um die Stabilität und Funktionsfähigkeit

einer Software sicherzustellen. Damit stellen sie, um es noch einmal etwas anders zu formulieren, ein nützliches und sinnvolles Hilfsmittel dar, um einerseits einen fundierten Eindruck bezüglich der Leistungsfähigkeit eines Anwendungssystems zu erhalten, andererseits um sichtbar zu machen, ob in Hinblick darauf Verbesserungen möglich – wenn nicht gar nötig – sind. Durch diese nicht-funktionale Testmethode soll garantiert werden, dass auch bei hoher Kundenlast die Server im Stande sind, den anfallenden Datenverkehr in einem akzeptablen Maße verarbeiten zu können [Hoff08].

Testautomatisierung

Da es beim Performance- und Last-Testen darum geht, eine Vielzahl von Benutzern über einen längeren Zeitraum auf eine Software zugreifen und Aktionen ausführen zu lassen, liegt es nahe, die Testdurchführung automatisiert durchzuführen [Fran14].
Performance- und Lasttests lassen sich mit verschiedensten Lasttestwerkzeugen testen. Durch das Automatisieren der Tests kann überprüft werden, ob sich das Verhalten der Anwendung nach wiederholter Ausführung verändert oder die Belastbarkeit sinkt, da aussagekräftige Parameter wie Antwortzeiten, übertragene Datenmengen und verbrauchte Ressourcen aufgezeichnet werden [Fran14]. So kann herausgefunden werden, ob temporäre Daten gespeichert werden und es dadurch ebenfalls zu einer Überlastung kommen kann.
Nun gibt es verschiedene kommerzielle sowie Open-Source-Produkte um Performance- sowie Lasttests durchzuführen. Des Weiteren soll sich auf die kostenlosen Varianten beschränken, da mittlerweile genügend Open-Source-Test-Tools auf dem Markt verfügbar sind, die den Anforderungen entsprechen.

Anforderung Tool-Unterstützung

In diesem Abschnitt sollen verschiedene Tools, die für Last- und Performancetests in Frage kommen, untersucht und evaluiert werden. Die Evaluierung beschränkt sich auf Open-Source-Produkte, da der Quellcode öffentlich zugängig und individuelle Anpassungen möglich sind. Dies macht das System für den Nutzer flexibel. Da verschiedene Anwender ihre Lösungen mit ein bringen, ist die Weiterentwicklung dieser Tools sehr kundenorientiert.
Die Auswahl basiert auf der Auflistung der Top 20 Lasttestwerkzeuge 2017 der Homepage Easy QA, die als Grundlage verwendet wurde. Diese beschränkt sich nicht nur auf Open-Source-Produkte, sondern evaluiert die Top 20 aller Performance- und Lasttesttools die 2017 am Markt zu finden sind. Die Open-Source-Tools werden dennoch separiert von den kostenpflichtigen Tools dargestellt. Demnach zählen zu den besten Open-Source Load-Testing-Tools 2017 [Easy17]:

Apache JMeter
Das am häufigsten verwendete kostenfreie Werkzeug für Lasttests. Ursprünglich wurde es zum Testen von Webanwendungen und FTP-Anwendungen entwickelt, mittlerweile ist kann es beispielsweise auch für Funktionstest und Datenbank-Server-Tests verwendet werden [Easy17]. Eine Besonderheit des Tools ist die breite Protokollunterstützung, so können nicht nur die typischen Webprotokolle wie http, SOAP, FTP, SMTP, POP3, IMAP verwendet werden, sondern auch JAVA EE-Standardprotokolle, dazu zählen JDBC, LADP und JMS [Apac17].
JMeter simuliert das Senden von Anfragen einer vielfachen Anzahl von Benutzern, es unterstützt somit das sogenannte *Multithreading* und gibt die Performance-Ergebnisse zurück. Um JMeter besser zu verstehen wird kurz der Arbeitsprozess laut Easy QA [Easy17] erklärt:

Schritt 1: Erzeugen einer Anfrage an den Ziel-Server

Schritt 2: Reaktion des Servers

Schritt 3: Speichern aller Rückmeldungen

Schritt 4: Sammeln und Berechnen statistischer Informationen

Schritt 5: Erstellen eines Testreports

The Grinder
The Grinder ist ein gängiges Java-basiertes Framework für Lasttests, welches plattformübergreifend ist; das heißt, sofern eine JVM eingerichtet ist, läuft es von überall [Easy17].
Grinder stellt die sogenannte Grinder-Konsole zur Verfügung, diese steuert verschiedene Grinder-Agenten und überwacht die Ergebnisse in Echtzeit. Die Konsole kann als grundlegende interaktive Entwicklungsumgebung (IDE) zum Bearbeiten oder Entwickeln der Testfälle verwendet werden. Grinder-Agenten sind Lastgeneratoren ohne graphische Oberfläche, welche eine Reihe von Arbeiten ausführen um Last zu erzeugen. [AsFi13]

Gatling
Gatling ist ein leistungsstarkes, auf Scala basiertes Tool für Belastungstests. Seine Besonderheit ist, dass es zwei ausführbare Programme zur Verfügung stellt, eines für die Testfallaufnahme und eines für die Testfallausführung [Easy17].
Es ist eine rein Kommandozeilen-orientierte Lösung zum Testen von Last und Performance. Durch die eigene domänenspezifische Sprache (DSL) sind die Szenarien für jedermann lesbar. Technische Basis für das Werkzeug ist die Programmiersprache Scala, daher ist Gatling eher für Programmierer als für reine Testingenieure geeignet. Um das Lasttesttool starten zu können, wird als Ablaufumgebung ein JDK 8 benötigt. Testberichte, denen die Auswertungen der Ergebnisse entnommen werden können, werden im HTML-Format ausgegeben. [Gatl17]
Die Struktur von Gatling lässt sich in vier Teile aufgliedern [Easy17|:

Die *HTTP-Protokollkonfiguration* hilft dabei, die Basis-URL zu definieren, mit der die Tests durchführen werden.
Die *Header-Definition* stellt die Header für die Anfrage bereit, die an den Server gesendet wird.
Die *Szenario-Definition* gibt eine Gruppe von Aktionen an, die ausgeführt werden sollen, um eine Benutzerinteraktion mit der Anwendung zu simulieren.
Die *Simulation-Definition* soll die Anzahl der Benutzer angeben, die gleichzeitig das Last-Szenario für einen bestimmten Zeitraum ausführen.

Locust
Dies ist ein code-driven, verteiltes Lasttestwerkzeug. Der häufigste Anwendungsbereich ist das Testen von Websites hinsichtlich Belastbarkeit [Easy17].

ApacheBench
wird als das einfachste Werkzeug für Performance- und Lasttests gehandelt. Ursprünglich wurde es als Kommandozeilen-Programm für das Testen des Apache HTTP Servers entwickelt [Easy17].

Taurus
die von BlazeMeter entwickelte kostenlose Multi-Tool-Testplattform, die im Grunde eine Abstraktionsschicht über JMeter, The Grinder oder Gatling ist. Hauptziel ist eine reibungslose Testautomatisierung [Easy17].

Siege
wird speziell für den Belastungstest des HTTP- und HTTPS-Protokolls ausgelegt. Es kann auch als Webserver-Benchmarking-Tool verwendet werden [Easy17].

Bewertung der Tool-Unterstützung

Bei der Bewertung wurde sich auf drei ausgewählte Tools fokussiert, hierzu zählen Apache JMeter, The Grinder und Gatling. Gründe hierfür sind die hohe Bandbreite an Funktionalitäten, welche diese Tools bieten sowie der Aspekt, dass diese Produkte auch kontinuierlich von der Open-Source-Community weiterentwickelt werden.
Die Testfälle im Tool The Grinder werden in Jython geschrieben., dies ist eine Migration der Programmiersprache Phyton in die JVM. Bei der Ausgabe der Ergebnisse ist man sehr beschränkt, da dies nur über die Konsole möglich ist. Da The Grinder mit Testskripten arbeitet, ist eine gewisse Einarbeitungszeit erforderlich.
JMeter wird als „Alleskönner“ unter den Open-Source-Tools gehandelt, da es eine große Bandbreite an zu testenden Protokollen vorlegt, so unterstützt es nicht nur die typischen Webprotokolle, sondern auch Java-EE-Standardprotokolle. Würden der zu testenden Webapplikation noch Features hinzugefügt werden, die von anderen Protokollen unterstützt werden, ist man mit dem Lasttestwerkzeug JMeter auf der sicheren Seite.
Des Weiteren ist zu sagen, dass Apache JMeter wohl das etablierteste Open-Source-Tool ist, das es momentan auf dem Markt gibt. Alleine die Vielzahl an Literatur ermöglicht einen schnellen Kompetenzaufbau. Die hohe Anzahl an Plug-Ins, die für JMeter angeboten werden, sprechen ebenfalls für die Auswahl dieses Werkzeugs. So sind verschiedene Zusatzfunktionen zur graphischen Darstellung der ausgewerteten Parameter verfügbar. [Pokh17] Daher wird in für diese Arbeit empfohlen, JMeter als Lasttestwerkzeug zu benutzen.
Gatling ist ein noch sehr „junges“ Testwerkzeug und vermag vor allem durch seine zeitgemäße Architektur, welche auf Scala, Akka und Netty basiert, zu überzeugen. Es ist möglich asynchron und nicht-blockierend zu arbeiten. Ein weiterer Pluspunkt ist die Darstellung der Testberichte, diese wirkt modern und übersichtlich. Jedoch ist der Umfang der unterstützenden Protokolle sehr überschaubar, d.h. es muss vorher geprüft werden, ob Gatling die jeweils verwendeten Protokolle auch unterstützt.
Letzten Endes ist die Auswahl des Lasttestwerkzeugs jedoch eine subjektive Entscheidung und hängt auch von den Kompetenzen des Benutzers ab. Gatling spricht sicherlich eher Java- und Scala-Programmierer an, JMeter hingegen kann nach kurzer Einarbeitungszeit auch von Testern, die über wenig beziehungsweise keine Expertise auf dem Gebiet des Programmierens verfügen, verwendet werden.

Anwendung

In diesem Kapitel wird zunächst die zu testende Software beschrieben. Danach werden alle Komponenten erklärt, die es braucht, um einen Performance- bzw. Lasttest mit dem ausgewählten Open-Source-Tool durchzuführen. Anschließend soll die Implementierung und Ausführung der Tests erläutert werden.

Testobjekt

Bei der zu testenden Software handelt es sich um eine Web-Applikation. Diese wird als internes Rechnungserstellungs- und Zahlungssystem vom Kunden genutzt. Sogenannte *Operations Manager* haben einen autorisierten Zugriff auf diese Webapplikation. In der Hauptnavigation findet man *Billing,* von hier aus ist es möglich zu der Unternavigation *Accounts* und *Invoices* zu gelangen.
Unter Accounts findet man eine Aufführung aller bisweilen angelegten Kunden-Accounts. Es ist auch möglich verschiedene Filter anzuwenden um die Suche nach einer Kundengruppe oder einem einzelnen Kunden zu erleichtern. Es ist möglich Kundenkonten anzulegen, entweder ein persönliches Konto, das von einer Privatperson genutzt wird oder ein Firmenkonto für Geschäftskunden. Dabei spielen festgelegte Pflichtfelder eine wichtige Rolle, werden diese nicht ausgefüllt, ist die Erstellung eines Kundenkontos nicht machbar. Bei einem Firmenkunden-Account ist der Firmenname ein „Muss“, während bei privaten Konten der Vor- und Nachname auszugeben sind. Um sich den Ablauf besser vorstellen zu können, wird die Erstellung eines Kunden-Accounts nun in einem Sequenzdiagramm dargestellt.

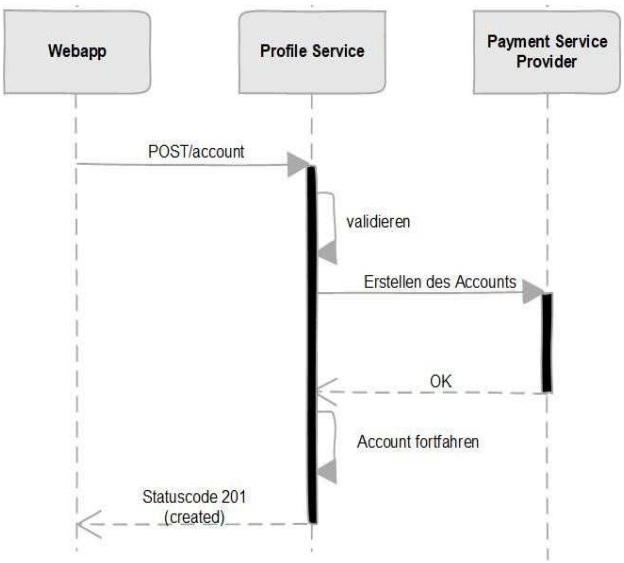

Abbildung 2: Sequenzdiagramm Erstellen eines Kunden-Accounts (eigene Darstellung)

Ein Kunden-Account kann verschiedene Status annehmen. Ein Kunde hat den Status „aktiv“ wenn mindestens eine Zahlungsmethode hinterlegt wurde. Den Status „unvollständig“ besitzt ein Kunde, wenn noch keine Zahlungsmethoden hinterlegt wurden.
Für jedes Kundenkonto können ein oder mehrere Profile angelegt werden. Ein Profil weist im Wesentlichen einen Rechnungskontakt, zusätzlich gegebenenfalls einen Versandkontakt, auf. Der Rechnungskontakt enthält die rechtsverbindliche Adresse des Kunden für das angegebene Profil, einschließlich des Namen und ggf. der Firma und eine optionale E-Mail-Adresse. Diese Daten sind ein wesentlicher Bestandteil jeder Rechnung, die ein Kreditgeber seinen Kunden ausstellt. Der Versandkontakt ist notwendig, sobald die Rechnung eines Kunden nicht an die Adresse des Rechnungskontakts, sondern an eine andere Adresse geschickt werden soll. Hat eine Firma beispielsweise ihre Buchhaltung an eine Drittfirma übergeben, wäre dies der Fall
Ein Account braucht immer ein voreingestelltes *default profile*, dieses ist zunächst immer automatisch das erstangelegte Profil. Es ist möglich, Zahlungsmethoden einem Profil zuzuweisen, es gibt hier fünf standardisierte Zahlungsmöglichkeiten: Lastschrift, Debitkarte, Kreditkarte, Kauf auf Rechnung, Vorauszahlung.
Den Debit- und Kreditkarten können *Authorization holds* hinzugefügt werden, so wird das Konto vorab mit einem festgelegten Betrag belastet, um die Kreditwürdigkeit zu überprüfen und dann bei der Bezahlung mit dem Rechnungsbetrag verrechnet. Auch bei den Zahlungsmethoden muss eine Zahlungsmethode immer auf *default* gesetzt sein. Die Webapplikation ist mit einem Payment Service Provider verbunden, über den die Zahlungen abgewickelt werden.
Die zweite Kategorie ist mit *Invoice* benannt. Hier findet man eine Übersicht über alle angelegten Rechnungen, auch hier sind Filteroptionen eingebaut, um die Suche nach einer bestimmten Rechnung oder einer Rechnungsgruppe zu erleichtern.
Um diesen Vorgang besser zu verstehen, wird dieser anhand eines Sequenzdiagramms dargestellt.

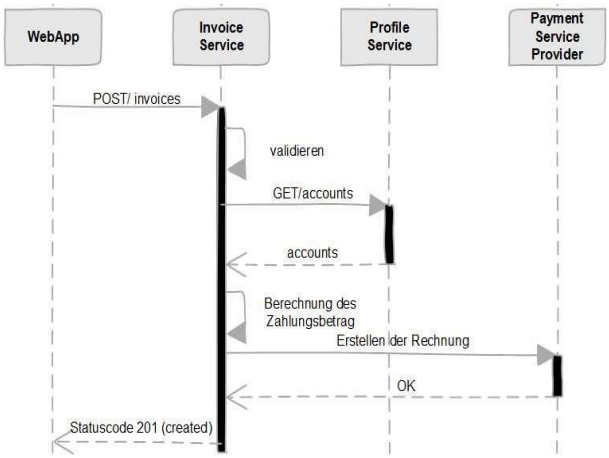

Abbildung 3: Sequenzdiagramm Erstellen einer Rechnung (eigene Darstellung)

Es besteht die Möglichkeit mehrere Rechnungspositionen anzulegen, Umsatzsteuer sowie Beträge festzulegen und das Fälligkeitsdatum der Zahlung zu wählen. Des Weiteren muss eine der Zahlungsmethoden ausgewählt werden, welche wiederum selbst über einen Status verfügen. Nur Zahlungsmethoden mit dem Status *aktiv*, können in Rechnungen benutzt werden. Hat die Zahlungsmethode den Status *ungültig*, da beispielsweise die Kreditkarte verfallen ist oder den Status *gelöscht* hat, ist sie nicht verfügbar. Eine Rechnung ist eindeutig identifizierbar durch die ihr zugewiesene Rechnungsnummer. Die Rechnungen werden in Typen unterschieden, eine neu angelegte Rechnung ist vorerst vom Typ *reguläre Rechnung:* da die Möglichkeit besteht, eine Rechnung ganz oder teilweise zu stornieren, kann der Typ auf *Stornierung* oder *Teilstornierung* abgeändert werden. Rechnungen haben wie Accounts einen Status, welcher wie folgt beschrieben werden kann:

- *offen*, eine Rechnung wurde erfolgreich erstellt
- *storniert*, es wurde eine Teil- oder Stornierung vorgenommen
- *bezahlt*, die Zahlungsabwicklung lief erfolgreich seitens des PSP
- *Fehler*, es tritt ein Fehler bei der Zahlung auf

Testszenarien

Die Aktionen bilden die möglichen Vorgänge ab, die ein Benutzer mit dem System ausführen kann. Damit kann die Anwendung des Systems praxisnah simuliert werden. [Fran14]. Vor allem Szenarien, die oft wiederholt werden, sollten getestet werden, da hier die Wahrscheinlichkeit eines Absturzes des Systems am größten ist. Es wurden für die zu testende Webanwendung drei Testszenarien ausgewählt.

Szenario 1: Einloggen eines Operations Managers in das System und Erstellen eines neuen Accounts: Dieses Szenario, so wird empfohlen, sollte vor dem Release getestet werden, da in der Eingangsphase verstärkt Kundenkonten angelegt werden. So kann einem unerwarteten Zusammenbruch vorgebeugt werden.
Da die Webapplikation des Weiteren für die Zahlungsabwicklung und Rechnungserstellung verwendet wird, sollten Szenarien gewählt werden, die oft ausgeführt und somit wiederholt werden.
Szenario 2: Einloggen eines Operations Managers in das System und Hinzufügen einer Zahlungsmethode zu einem Account.
Szenario 3: Einloggen eines Operations Manager in das System, Auswählen eines Kundenkontos und Erstellen einer Rechnung.

Testziele

Nun werden die Ziele des Performance- und Lasttestens festgelegt. Diese Ziele sollen die Grundlage für die spätere Auswertung des Erfolgs der Testergebnisse sein.

Stabilität
Das System soll nur bei gewollter Überlastung abstürzen, ansonsten wird erwartet, dass das Programm stabil läuft und keine unvorhersehbaren Zusammenbrüche erfolgen. Das heißt, das System soll nur bei einem Wert, der größer als die Hochlast ist, abstürzen. Die Hochlast wurde auf den Wert 50 Threads pro Sekunde festgelegt.
Erst bei einem gleichzeitigen Aufruf von mehr als 50 Threads pro Sekunde - wobei in diesem Fall jeder Threadaufruf das Anlegen eines Accounts, einer Rechnung oder das Hinzufügen einer Zahlungsmethode darstellt - darf das System zusammen brechen.

Antwortzeitverhalten
Die vorgegebene Antwortzeit wird dauerhaft eingehalten und nicht überschritten. Dies bedeutet, das System soll bei wiederholter Durchführung eines festgelegten Testfalls eine solche im 90. Perzentil von unter 1.000 Millisekunden erreichen.

Laufzeitverhalten
Das Programm darf nicht die Reaktionszeit überschreiten und es soll ausgeschlossen werden, dass zu viel Arbeitsspeicher verbraucht wird. Das System soll bei andauender Belastung von drei Minuten durch Anfragen in Normallastbereich die Antwortzeit von unter 1.000 Millisekunden erreichen.

Entwicklungsumgebung

RubyMine ist eine IDE für reine Ruby-Implementierung sowie Webanwendungen, die auf Ruby und Rails basieren. Für dieses Projekt wird die Version 2017.2.4 benutzt. Die Entwicklungsumgebung bringt eine Vielfalt an attraktiven Vorteilen mit sich. So wird schnelleres Arbeiten mit einem Smart Editor versprochen. Features wie die sprachspezifische Syntax- und Fehlerhervorhebung, die Codeformatierung, die Codevervollständigung und die schnelle Dokumentation machen das Arbeiten mit dieser IDE sehr angenehm. Es verfügt über ein intelligentes Suchsystem, mit dem zu einer beliebigen Klasse, Datei oder einem Symbol gesprungen werden kann. [Jetb17]

Biblothek ruby-jmeter

Die integrierte Bibliothek, *ruby-jmeter* ermöglicht das Schreiben von JMeter-Testplänen unter der Verwendung der Ruby Sprache. Es ist also nicht zwingend notwendig, die Benutzeroberfläche von JMeter zu verwenden. Dafür stellt Ruby sogenannte RubyGems zur Verfügung, dies ist die Paketverwaltung von Ruby und macht es dem Nutzer möglich, neue Programmbibliotheken zu installieren, verwalten und auch wieder zu entfernen. [Koop17]
Die nachfolgenden Methoden beschreibt Tim Koopmans wie folgt auf GitHub [Koop17].
Ein JMeter-Plan kann ausgeführt werden, indem die *run*-Methode des Tests aufgerufen wird. Die Methode startet JMeter *headless*, das heißt ohne den Aufruf der graphischen Benutzeroberfläche und führt den Testplan aus. Es ist möglich einige Parameter festzulegen, wie den Dateipfad bzw. Dateinamen für die JMX-Datei und den Protokollpfad bzw. Protokollnamen zur Ausgabe von JMeter-Protokollen.
Es steht die Thread-Methode zur Verfügung, diese kann verwendet werden um eine Gruppe von Benutzern zu definieren. Ein Thread ist eine Ausführungseinheit, die in einem Prozess läuft; es ist möglich gleichzeitig mehrere dieser Einheiten agieren zu lassen. Da sich alle Threads des jeweiligen Prozesses dessen Adressraum teilen, können sie gleichermaßen auf Prozessdaten zugreifen. [Mand13]
Die zur Verfügung gestellte Methode ermöglicht das sogenannte Multithreading, das heißt mehrere Threads – wobei in diesem Fall ein Thread einen Benutzer simuliert - können parallel abgearbeitet werden.

Jenkins

Da an einem Programmcode oftmals viele Entwickler über mehrere Standorte verteilt arbeiten und der einzelne Programmierer nicht garantieren kann, dass seine Änderungen Auswirkungen an anderer Stelle des Programmes haben, wird kontinuierliche Integration von neugeschriebenen Codes immer wichtiger. Jenkins ist ein eigenständiger, kostenloser Automatisierungsserver, der alle Arten von Aufgaben im Zusammenhang mit dem Erstellen, Testen und Bereitstellen von Software automatisieren kann, womit die kontinuierliche Integration von Software durch Computer steuerbar wird. Es soll dem Entwickler ermöglichen, frühzeitig Störungen und Fehler im Code zu erkennen und diesen somit schneller Abhilfe leisten zu können. Der Server kann über native Systempakete oder Docker installiert werden oder sogar eigenständig von jedem Rechner mit installierter JRE ausgeführt werden [Behr11].

Somit können die implementierten Performance- und Lasttests automatisch durch Anlegen eines neuen Jobs in Jenkins abgespielt werden.

Performance Plug-In für Jenkins

Mit einem Zusatzprogramm für Jenkins ist die Ausgabe der Testergebnisse der Performance- und Lasttests möglich. Die Leistungstests können als Build-Schritt für einen Jenkins-Job ausgeführt oder Berichte aus bereits vorhandenen Testergebnisdateien erstellt werden [Gith17].
Jenkins kann grafische Diagramme mit dem Trendbericht über Leistung und Robustheit erstellen. Außerdem enthält es eine Funktion, welche das Festlegen des endgültigen Build-Status als gut, instabil oder fehlgeschlagen, basierend auf dem gemeldeten Fehlerprozentsatz, ermöglicht. Das Plug-In unterstützt das Berichtsformat Apache JMeter XML und CSV-Format [Gith17].
Um die neueste Plug-In-Version verwenden zu können, muss es heruntergeladen, kompiliert und installiert werden. Als Voraussetzung muss auf dem verwendeten Computer Git, Maven und Java installiert sein [Gith17].

Der Performance Breakdown im Speziellen stellt graphisch die Entwicklung der Antwortzeit nach wiederholter Durchführung eines Tests dar. Aus der Tabelle sind verschiedene Metriken ablesbar, die unter anderem eine differenzierte Darstellung der Antwortzeiten erlauben:

- *Durchschnittliche Antwortzei*t: Summe aller Antwortzeiten geteilt durch die Anzahl der Antwortzeiten
- *Median*: Mittlerer Zahlenwert aller Antwortzeiten
- *90 Prozent Marke*: Unter diesem Wert liegen 90 Prozent aller Antwortzeiten
- *Minimum*: kürzeste Antwortzeit
- *Maximum*: längste Antwortzeit

Erstellung und Durchführung der Testfälle

Im nächsten Schritt werden die Testfälle anhand der festgelegten Testszenarien erstellt. Es werden in der Entwicklungsumgebung RubyMine diese Testfälle implementiert und mit Hilfe des Automatisierungsservers Jenkins ausgeführt.
Exemplarisch wird jeweils ein Last- und ein Performancetest vorgestellt.

Performancetest: Erstellen eines Kunden-Accounts

Testfall	PT_CreateUser
Testobjekt	Erstellen eines Accounts
Voraussetzung	Autorisierter Zugriff auf die Homepage. Der eingeloggte Operations Manager besitzt Schreibrechte auf die Homepage, d.h. er ist befugt, ein neues Kundenkonto anzulegen
Testdaten	x = 30, y = 180
Testbeschreibung	x Operations Manager erstellen pro Sekunde gleichzeitig jeweils einen Kunden-Account. Dies wird über y Sekunden lang wiederholt. Bei der Erstellung sind nur die Pflichtfelder auszufüllen. Diese sind der Typ des Accounts sowie die Kundenummer.
Soll-Ergebnis	90 Prozent der Antwortzeiten unter 1.000 Millisekunden über drei Minuten hinweg.
Ist-Ergebnis	90% Linie: 4.153 Millisekunden
Bestanden	Nicht bestanden
Kommentar	*Das Testergebnis ist viermal höher als das zu erwartende Ergebnis.*
Tester	Marlis Teufel
Datum/ Uhrzeit	15.01.2018, 13:36

Lasttests: Erstellen eines Kunden-Accounts

Testfall	LT_CreateUser
Testobjekt	Erstellen eines Accounts
Voraussetzung	Autorisierter Zugriff auf die Homepage. Der eingeloggte Operations Manager besitzt Schreibrechte auf die Homepage, d.h. er ist befugt, ein neues Kundenkonto anzulegen.
Testdaten	x = 10, y = 180, z = 30
Testbeschreibung	Es erstellen x Operations Manager pro Sekunde gleichzeitig jeweils einen Kunden-Account. Dies wird über y Sekunden lang wiederholt. Es wird alle z Sekunden die Anzahl der Threads um 10 Stück erhöht bis eine Überlast von 51 Threads erreicht ist. Bei der Erstellung sind nur die Pflichtfelder auszufüllen. Diese sind der Typ sowie die Kundenummer des Accounts.
Soll-Ergebnis	Erst bei einer Überlast von 51 Threads pro Sekunde wird das System signifikant langsamer und bricht zusammen. Die durchschnittliche Antwortzeiten liegt bei unter 1.000 Millisekunden.
Ist-Ergebnis	Average: 2585 Millisekunden
Bestanden	Nicht bestanden

Kommentar	System bricht nicht zusammen, vereinzelt 500 – HTTP Statuscode. Die Antwortzeit ist fast 1,5-fach höher als erwartet.
Tester	Marlis Teufel
Datum/ Uhrzeit	15.01.2018, 14:17 Uhr

Ergebnisanalyse

Bei dem ersten Testfall *PT_CreateUser* wurde eine gleichmäßige, dauerhafte Last von 30 Threadaufrufen pro Sekunde auf das System über drei Minuten hinweg ausgeübt.
Bei der Betrachtung der Abbildung 4 fällt sofort ins Auge, dass die Antwortzeit während des Testablaufs geringer wird. Dies kann bedeuten, dass das System sich auf die Last eingestellt hat und diese nun besser regulieren kann. Natürlich muss auch mit einbezogen werden, dass die Last nach drei Minuten langsam abnimmt, so schließen alle zehn Sekunden fünf Threads ihren Job ab. Jedoch sind signifikante Ausreißer durchgehend vorhanden, hier liegt die Antwortzeit bei bis zu 10.000 Millisekunden. Es wurde eine Antwortzeit von höchstens 1.000 Millisekunden bis zur 90% Linie erwartet. Das Ist-Ergebnis zeigt jedoch einen weit höheren Wert. So liegt die Antwortzeit bei der 90% Linie schon bei 4.153 Millisekunden. Anders ausgedrückt bedeutet dies, dass die Reaktionsspanne viermal so lange wie erwartet ist.
Auch die durchschnittliche Antwortzeit liegt bei fast 2.000 Millisekunden. Der Testfall ist somit nicht bestanden und es wird dringend angeraten an dem Performance-Problem zu arbeiten.
Vereinzelt tritt außerdem der Fehler 500 - Internal Server Error auf, somit wird nicht jeder Kunden-Account auch wirklich angelegt.

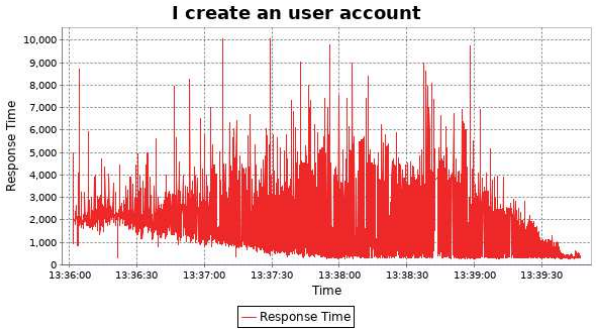

Abbildung 4: Darstellung der Antwortzeiten des Performancetests(Jenkins)

Betrachtet man nun den Lasttest *LT_CreateUser* (Abbildung 5) zu diesem Szenario, stellt man fest, dass die Ergebnisse noch defizitärer sind. In diesem Testfall werden alle 30 Sekunden zehn Threads hinzugefügt. Somit steigt die Anzahl an Threads, die pro Sekunde abgearbeitet werden müssen, gemächlich an. Die Anzahl an Ausreißer, gegenüber dem vorher durchgeführten Performance-Test, ist gestiegen. Die durchschnittliche Antwortzeit beträgt 2.585 Millisekunden, das heißt sie ist 1,5-fach höher als erwartet. Auch hier kommt es zum Fehlercode 500, obschon sehr selten (0.79%). Der Wert der 90% Linie liegt hier mittlerweile bei 5.809 Millisekunden. Maximal ist mit einer Antwortzeit von 10.353 Millisekunden zu rechnen. Positiv bewertet wird die Tatsache, dass das System nicht wie erwartet abstürzt, sondern nur sehr vereinzelte Fehler auftreten.

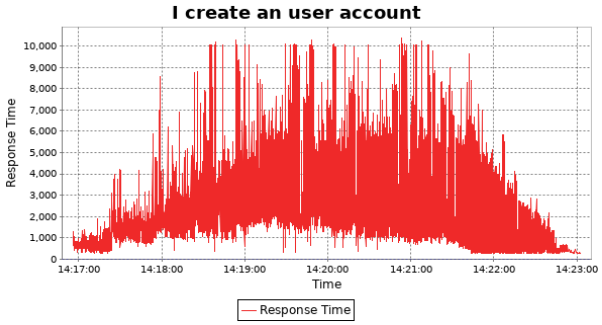

Abbildung 5: Darstellung der Antwortzeiten des Lasttests (Jenkins)

Bewertung des Nutzens

Durch die Einführung von Performance- und Lasttests wurde die Dringlichkeit nach Verbesserungen festgestellt. Nun kann durch frühzeitige Erkennung die Antwortzeit von Anfragen verkürzt werden. Ohne die Durchführung dieser Test wäre es nicht möglich eine notwendige Performanceverbesserung anzustoßen, da keine Messwerte als Belegung des Leistungsdefizits vorhanden waren. So können Änderungen an der Software vorgenommen, und erneut Tests durchgeführt werden. Dies ist solange wiederholbar bis das Testergebnis als zufriedenstellend festgehalten wird.
Es gibt verschiedene Möglichkeiten Leistung und Performance einer Software zu messen, somit ist es nicht zwingend notwendig das die Person über solide Programmierkenntnisse verfügt, da beispielsweise Apache JMeter auch über eine graphische Oberfläche bedienbar ist.
Durch die große Auswahl an zuverlässigen Open-Source Produkten ist das Preis-Leistungs-Verhältnis sehr gut. Die Einführung solcher Tests wird somit als sehr sinnvoll und nützlich betrachtet.
Das resultierende Proof of Concept kann nun auch auf Webanwendungen anderer Projekte adaptiert werden.

Schluss

Im Verlauf der Arbeit wurde die Notwendigkeit der Einführung von Performance- und Lasttest deutlich. Funktionales testen von Software hat sich längst auf dem Markt etabliert und wird in den Arbeitsprozess längst mit einbezogen. Jedoch das Testen von nicht funktionalen Anforderungen wird oft vernachlässigt oder gar weg gelassen. Jedoch muss Software auch diesen Anforderungen gerecht werden um langfristig gesehen am Markt wettbewerbsfähig zu bleiben. Stürzt eine Mobile App oder Webanwendung oft ab oder hat lange

Antwortzeiten, wird der Kunde unzufrieden und wechselt, wenn möglich, zu einer Alternative.
Im Fall der beschriebenen Webapplikation können nun Änderungen vorgenommen werden, um die Leistung und Belastbarkeit zu verbesseren, damit die Qualität des Produkts sowie die Kundenzufriedenheit gesteigert werden.

Literaturverzeichnis

[Apac17] Apache Software Foundation: Apache JMeter, 2017, http://JMeter.apache.org/. Abruf am 2017-11-06.

[AsFi13] Aston, P.; Fitzgerald, C.: The Grinder, a Java Load Testing Framework, 2013, http://grinder.sourceforge.net. Abruf am 2017-11-06.

[BaLi11] Balzert, Helmut; Liggesmeyer, Peter: Lehrbuch der Softwaretechnik. 2: Entwurf, Implementierung, Installation und Betrieb. Spektrum Akad.Verl., Heidelberg, 2011.

[Behr11] Behrendt, Mario: Jenkins - kurz & gut. O'Reilly, Beijing, 2011

[Easy17] EasyQA: Top 20 tools for load testing in 2017, 2017. https://geteasyqa.com/blog/best-tools-load-testing/. Abruf am 2017-12-08.

[Fran14] Franz, Klaus: Handbuch zum Testen von Web- und Mobile-Apps: Testverfahren, Werkzeuge, Praxistipps. Springer Vieweg, Berlin Heidelberg, 2014.

[Gatl17] Gatling Corp: Gatling Load Testing, 2017, https://gatling.io/performancetesting/. Abruf am 2017-11-18.

[Gith17] GitHub Pages GitHub, Inc.: Running Performance Tests, 2017. http://jenkinsci.github.io/performance-plugin/. Abruf am 2017-11-24.

[Hoff08] Hoffmann, Dirk W.: Software-Qualität. Springer, Berlin, 2008.

[Jetb17] JetBrains s.r.o: WHY RUBYMINE, 2017, https://www.jetbrains.com/ruby/?fromMenu. Abruf am 2017-11-20.

[Koop17] Koopmans, Tim: flood-io/ruby-jmeter, 2017, https://github.com/flood-io/ruby-jmeter. Abruf am 2017-11-25.

[Mand13] Mandl, Peter: Grundkurs Betriebssysteme: Architekturen, Betriebsmittelverwaltung, Synchronisation, Prozesskommunikation. Springer Vieweg, Wiesbaden, 2013.

[Pilo12] Pilorget, Lionel: Testen von Informationssystemen: integriertes und prozessorientiertes Testen. Vieweg + Teubner, Wiesbaden, 2012.

[Pokh17] Pokhilko, Andrey: Custom Plugins for Apache JMeter, 2017, https://jmeter-plugins.org. Abruf am 2017-11-11.

[Walt08] Walter, Thomas: Kompendium der Web-Programmierung: dynamische Web-Sites. Springer, Berlin, 2012.