Validierungsbericht
Team 2

SEP WS 2021/22

Betreuer:

Prof. Dr. Christian Bachmaier

Projektphase	**Leiter**
Pflichtenheft	Johann Schicho
Entwurf	Stefanie Gürster
Feinspezifikation	Johannes Garstenauer
Implementierung	Thomas Kirz
Validierung	Sebastian Vogt

21. Januar 2021

Inhaltsverzeichnis

1 Einleitung

Johannes Garstenauer

In dieser letzen Projektphase wurde die Anwendung anhand des Pflichtenhefts validiert. Es wurde sichergestellt, dass alle Funktionen korrekt und wie vereinbart implementiert wurden. Dafür wurde die Anwendung anhand der bereits im Pflichtenheft spezifizierten *Integrationtests* getestet und durch einen Lastentest die Performance getestet, sowie die *Test Coverage* festgestellt. Zusätzlich testen die *Security Tests* die Sicherheit der Anwendung gegen Angriffe.

2 Integrationstests

Stefanie Gürster, Johann Schicho

Wir verwenden das *Selenium Framework* um mehrere Nutzerabläufe im Webbrowser zu automatisieren. Dabei übernimmt ein Programm den Browser und navigiert anhand der *ID* der Webelemente. Dabei wird dann gegen die angezeigten Meldungen getestet, um den korrekten Ablauf sicherzustellen.

2.1 Probleme zu Beginn

Johann Schicho

- **Verwendung von *IDs* in JSF-Komponenten.** Bereits während der Implementierung wurden überall, nach Definition der Feinspezifikation, IDs in den *Facelets* verwendet.
  Allerdings wurde übersehen, dass auch *Forms* und *Composite Components* immer eigene IDs benötigen, da ansonsten JSF diese dynamisch generiert. Die dynamisch generierten IDs können sich bei jedem *Render Vorgang* wieder ändern und erlauben damit also leider keine Identifikation.
  Es wurden an allen Stellen, an denen die IDs vergessen wurden, diese noch eingefügt.
- **Testsuites mit JUnit5.** Testsuite erlauben eine Zusammenfassung von mehreren Tests in ein Gesamtpaket. Das ist eine wichtige Anforderung für unsere *Integration Tests*, da diese von einander abhängig sind, also eine Reihenfolge festgelegt werden muss.
  JUnit5 besaß bis zu Release 5.8[1] am 12. September 2021 keine Möglichkeiten für Test Suites. Bisher waren immer Abhängigkeiten zu Junit4 nötig, um diese Funktionalität in JUnit5 zu verwenden. Unsere Testsuite benutzt die neue *@Suite Annotation* von JUnit5. Anfangs gab es damit Startschwierigkeiten, da das Feature sehr neu ist und die offizielle Dokumentation spärlich ist.

2.2 Testfälle

Stefanie Gürster

[1] `https://junit.org/junit5/docs/5.8.0/release-notes/`

es war nicht klar in PIT auf was getestet wurde ...

als Blackbox? ist das nicht ein @BeforeSuite ?

Die Tests sind aufeinander aufbauend und werden aufsteigend nach ihrer Testnummer durchgeführt.

Test	Beschreibung	Änderungen	Bugfixes	Zuständigkeit
/T001/	Setup. Erstellen der initialen Nutzerdaten.	-	-	Stefanie Gürster
/T010/	Erstellen eines neuen Forums.	Zur Deadline ist nun auch eine Uhrzeit angegeben.	-	Stefanie Gürster
/T020/	Navigation zu einem Forum über Suche.	-	-	Johann Schicho
/T030/	Einreichungs-seite eines bestimmten Forums.	-	-	Johann Schicho
/T040/	Hochladen eines Papers.	-	-	Johann Schicho
/T045/	Invalide Daten bei Einreichung.	Co-Autoren werden zunächst unabhängig in deine Liste eingefügt. Dabei wird bereits geprüft, ob die Daten valide sind. Erst über den Submitbutton werden die Co-Autoren tatsächlich hinzugefügt. ✓	-	Johann Schicho
/T050/	Submission einreichen.	-	Die Transaktion eine Submission einzureichen war zuvor abhängig, ob die jeweiligen E-Mails gesendet werden konnten. War dies nicht der Fall, so wurde die Transaktion abgebrochen.	Johann Schicho
/T060/	Logout.	Der Logout-Button ist nun direkt erreichbar.	-	Johann Schicho

/T080/	Hinzufügen eines Gutachters.	-	War die Koautorrelation in der Datenbank leer, konnte nicht auf die Einreichung zugegriffen werden.	Stefanie Gürster
/T085/	Gutachten annehmen.	Ein Gutachten wird nun über einen Button auf der jeweiligen Einreichungsseite angenommen oder abgelehnt.	-	Stefanie Gürster
/T090/	Gutachten hochladen.	-	-	Stefanie Gürster
/T100/	Gutachten freischalten.	-	-	Johann Schicho
/T110/	Anonymer Nutzer.	-	-	Johann Schicho
/T120/	Registrierung.	-	-	Johann Schicho
/T130/	Verifizierung	-	-	Johann Schicho
/T140/	Neuer Nutzer ist bereits Co-Autor.	-	-	Johann Schicho
/T150/	Download eines Gutachtens.	Selenium kann keine Downloads durchführen. Es wird nur überprüft, ob ein Download möglich ist.	-	Johann Schicho
/T160/	Löschen des eigenen Accounts.	-	Nach dem Löschen des Accounts wurde die Session nicht beendet.	Johann Schicho
/T170/	Überprüfe kaskadierendes Löschen.	Neu. Die Überprüfung wurde aus /T160/ in einen eigenen Test heraus getrennt.	-	Johann Schicho
/T200/	Fehlerhafter Zugriff.	-	-	Stefanie Gürster
/T210/	Illegaler Zugriff.	Hier wird ebenso auf eine 404-Fehlerseite weitergeleitet, um so wenig wie möglich Informationen nach außen zu geben.	-	Stefanie Gürster

/T1000/	Reset. Löschen der erstellten Daten.	-	-	Stefanie Gürster

3 Stresstest

Sebastian Vogt

3.1 Versuchsaufbau

3.1.1 Beteiligte Geräte

- Der FIM Rechner bueno als Referenzsystem für den Datenbankserver
- Der FIM Rechner ds9 als Referenzsystem für den Tomcat-Server.
- Der FIM Rechner galactica als System für die Latenzmessungen.
- Die drei Entwicklerlaptops *Lenovo IdeaPad Flex 5 14IIL05* (kurz **Lenovo5**), *Lenovo IdeaPad C340-14IML* (kurz **LenovoC**) und *Acer Aspire A515-54G* (kurz **Acer**), wie im Pflichtenheft spezifiziert. Diese haben über das Bayern-WLAN an der Universität Passau und VPN auf den Webserver zugegriffen.

3.1.2 Messung der Antwortzeiten

Wir haben uns dafür entschieden, die Messung der Antwortzeiten auf einem Rechner in der FIM zu machen. Dies hat folgende Vorteile:

- Die Stressoren und der Messungsrechner sind nicht im selben Netwerk, d.h. die von den Stressoren generierte Auslastung des Netzwerks wird vom Messungsrechner nicht mitgemessen
- Da der Messungsrechner im selben Netzwerk ist wie der Webserver, verfälscht die Internetverbindung des Messungsrechners nicht das Ergebnis.
- Da der Messungsrechner und die Stressoren verschieden sind, verfälscht die Auslastung der Stressoren selbst durch die Belastungstests nicht das Messergebnis.

Natürlich hat diese Entscheidung einen Nachteil:
Die im Pflichtenheft geforderte Antwortzeit von maximal 3 Sekunden bezieht sich auf den Endnutzer, aber in den Messdaten fehlt die Propagationszeit zwischen dem Endnutzer und dem Universitätsnetzwerk. Es ist wohl eine sinnvolle Annahme, dass eine Verdreifachung der Messwerte diesen Effekt im Mittel ausgleicht. Also müssen wir im folgenden Überprüfen, dass unsere Messwerte unter einer Sekunde liegen.

3.1.3 Software

Für die Lasttests wurden zwei gängige Nutzerflüsse auf LasEs mit Selenium automatisiert:

- Nutzerfluss 1: Hier registriert sich ein neuer Nutzer im System, navigiert eine Zeit lang im System und löscht dann sein Konto wieder.
- Nutzerfluss 2: Hier wird ein neues wissenschaftliches Forum erstellt, eine neue Einreichung erstellt und eine Revision angefordert und eingereicht.

In der Browserautomatisierung wird bei jedem HTTP-Request die Antwortzeit des Servers gemessen und zusammen mit Kontextinformation über die ausgeführte Interaktion abgespeichert. Diese Testsuite wurde auf den Stressoren und dem Messungsrechner ausgeführt.

Dabei gab es zwei Konfigurationen, bei denen die Testsuite je folgendermaßen ausgeführt wurde:

Konfiguration 1: 30 Stressoren

- **Lenovo5**: 7 parallele Ausführungen von Nutzerfluss 1 und gleichzeitig 8 parallele Ausführungen von Nutzerfluss 2
- **LenovoC**: 7 parallele Ausführungen von Nutzerfluss 1 und gleichzeitig 8 parallele Ausführungen von Nutzerfluss 2

Konfiguration 2: 50 Stressoren

- **Lenovo5**: 7 parallele Ausführungen von Nutzerfluss 1 und gleichzeitig 8 parallele Ausführungen von Nutzerfluss 2.
- **LenovoC**: 10 parallele Ausführungen von Nutzerfluss 1 und gleichzeitig 10 parallele Ausführungen von Nutzerfluss 2.
- **Acer**: 7 parallele Ausführungen von Nutzerfluss 1 und gleichzeitig 8 parallele Ausführungen von Nutzerfluss 2.

Konfigurationsunabhängig

Bei beiden Konfigurationen wurde auf den restlichen Rechnern je folgendes ausgeführt:

- Der Messungsrechner führt jeden Nutzerfluss einmal parallel aus und speichert danach die Antwortzeitmessungen in einer CSV Datei.
- Application Server und Datenbankserver führen das LasEs System auf Tomcat und die PostgreSQL Datenbank aus.

gutes Setup bei vielen Stressoren

+ wieviele DB Connections werden benutzt: das wird sich wahrscheinlich auch nochmal ändern...

3.2 Ergebnisse

Bei beiden Konfigurartionen wurden für eine Reihe verschiedener Nutzerinteraktionen Messwerte aufgezeichnet. Meistens existiert pro Nutzerinteraktion ein Messwert. Wenn zwei oder drei Messwerte existieren, wurde im folgenden Balkendiagramm jeweils der größte Messwert aufgenommen. Alle Messwerte werden in Millisekunden angegeben. In Abbildung 1 findet sich das Balkendiagramm für Konfiguration 1, in Abbildung 2 findet sich das Balkendiagramm für Konfiguration 2.

Ergebnisse besser mehrfach ausführen und Mittelwerte nehmen

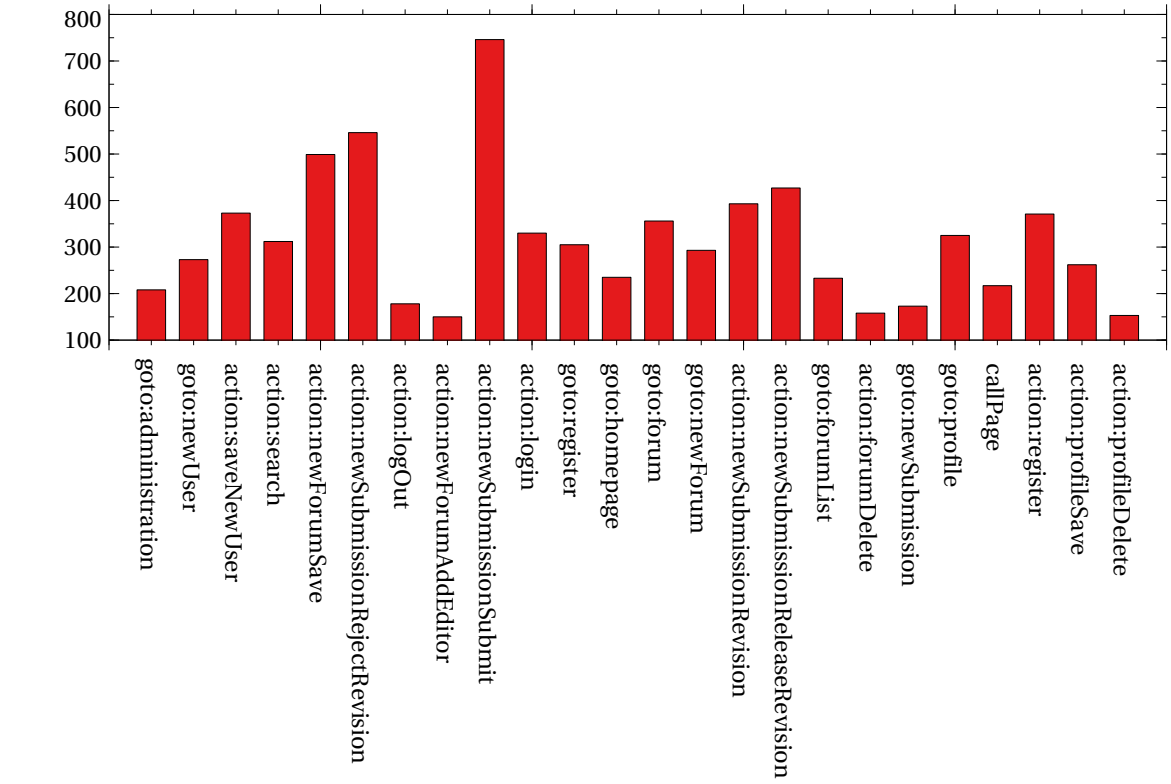

Abbildung 1: Konfiguration 1

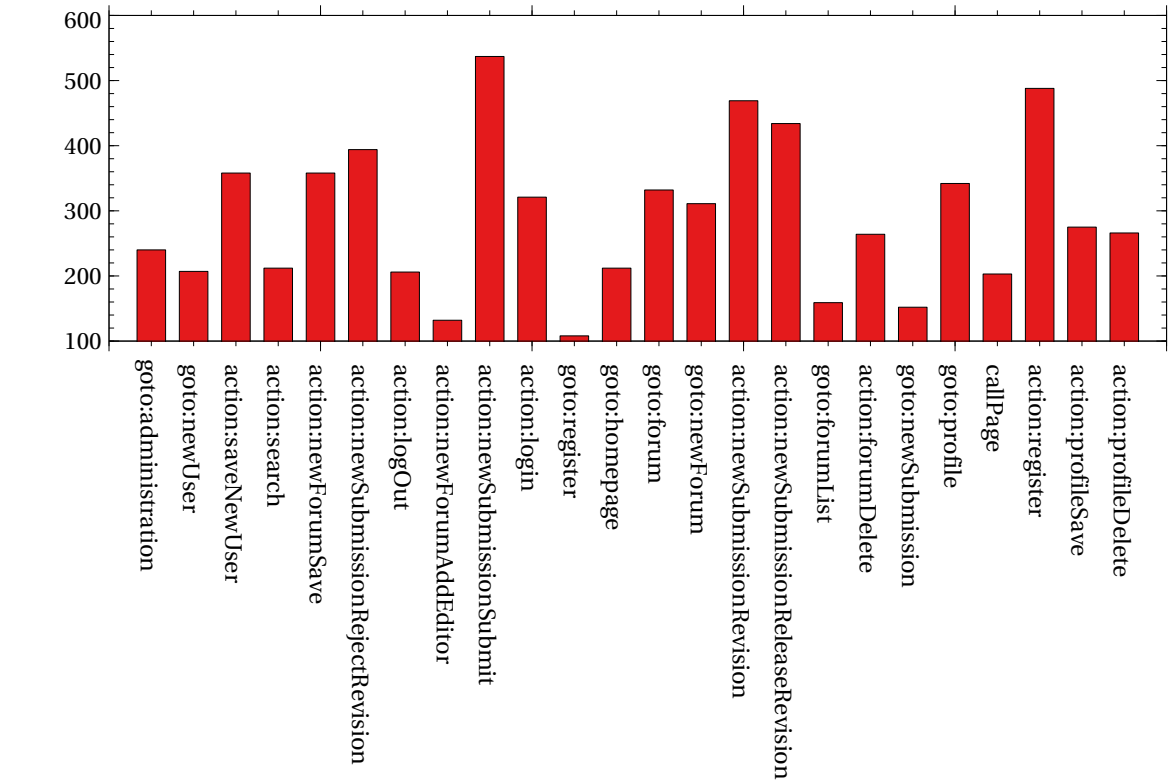

Abbildung 2: Konfiguration 2

In Abbildung 2 sieht man einen Boxplot, der die Werte der zwei Konfigurationen vergleicht. Im Boxplot zeigen die Whisker das 9/91 Perzentil. Der Kreis in der Box ist jeweils der Mittelwert. Die Kreise außerhalb sind alle Messwerte, die nicht innerhalb des Wertebereichs der Whisker liegen.

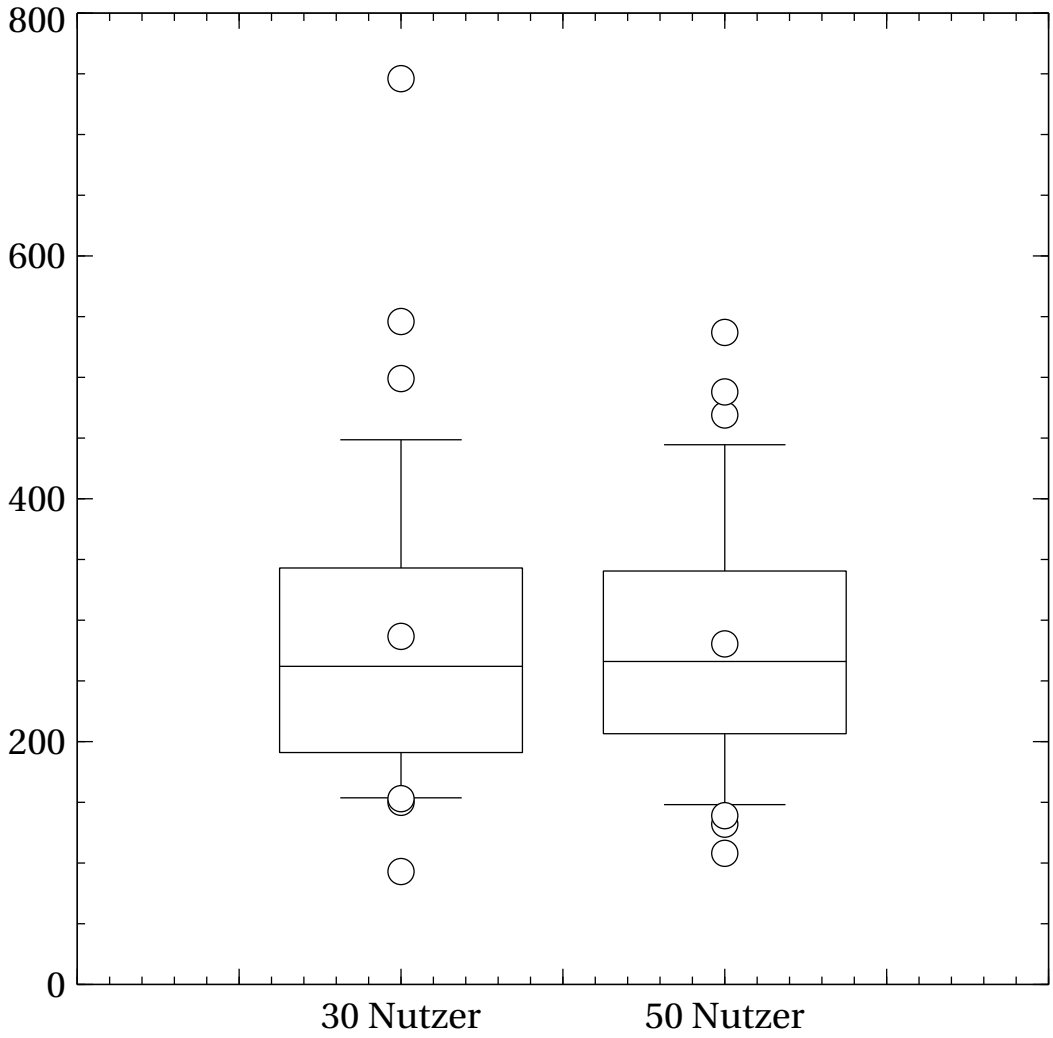

Abbildung 3: Boxplot

3.3 Interpretation

Alle Zeitmessungen befinden sich im Bereich unter einer Sekunde, insbesondere bei 50 gleichzeitigen Nutzern. Wie im Bereich Messung der Antwortzeiten begründet, erfüllt dies die Skalierbarkeitsanforderung des Pflichtenhefts.

Die langsamste Nutzerinteraktion war in beiden Konfigurationen das Einreichen einer neuen Einreichung. Dies war bereits in der Entwicklungsphase abzusehen, da die Einreichungsseite eine große Menge an Daten anzeigen und somit laden muss.

Interessanterweite ist die Ausführung bei 50 Nutzern im Median nur verschwindend

langsamer, im Durchschnitt sogar schneller als bei 30 Nutzern. Dies ist vielleicht dadurch zu erklären, dass bei 50 Nutzern die Ausführung der Stressoren auf einem der drei Laptops in einigen Threads sehr schnell zu Fehlern geführt hat und neugestartet werden musste. Dies führt uns auch schon zum letzten Punkt, und zwar den einschränkenden Bedingungen dieser Untersuchung.

3.4 Einschränkungen

Die Ausführung der Stressoren war sehr instabil. Dies ist auf zwei Gründe zurückzuführen. Einerseits ist die Ausführung der Selenium-Routinen oft plattformabhängig. Nicht nur das Betriebssystem, sondern auch verschiedene Hardwarekonfigurationen führen dazu, dass eine Operation auf einem Rechner fehlschlägt und auf einem anderen nicht. Andererseits stießen die verwendeten Laptops bei der Ausführung der Stresstest sehr schnell an ihre eigenen Belastungsgrenzen, was dazu führte, dass einzelne Threads wegen Zeitüberschreitung mit einer Selenium Exception endeten. Diese brachten dann narürlich den ganzen Ausführungsthread zum erliegen, wodurch ein simulierter Nutzer verloren ging.

Als zweiter einschränkender Faktor ist natürlich die Näherung der tatsächlichen Endnutzer Antwortzeit durch das Dreifache unserer Antwortzeit zu nennen. Im Idealfall würde die Antwortzeit durch einen seperaten Rechner, der sich in einem anderen Netz als die Stressoren oder der Webserver befindet, durchgeführt.

4 Sicherheitstests

Thomas Kirz

Zusätzlich zu den Tests aus dem Pflichtenheft wurden folgende Tests geschrieben, um verschiedene Sicherheitsaspekte des Systems zu testen. Es handelt sich auch hier um Integrationstests, die das Selenium-Framework verwenden.

Nach der Beiseitigung von Problemen, die in Kapitel 2 beschrieben wurden und nichts mit der hier zu testenden Sicherheit zu tun hatten, liefen die Tests durch. Nur der Name des Session-Cookies bzw. URL-Parameters wurde wie in der Feinspezifikation vorgegeben von `jsessionid` zu `sessionid` geändert, da diese Einstellung in der web.xml-Datei noch gefehlt hatte.

Die geprüften Bedingungen für den Erfolg der Tests sind **fett** geschrieben.

4.1 Injection

Dieser Test prüft den Schutz gegen SQL- und HTML/JavaScript-Injection.

Dazu wird ein Nutzerkonto angelegt mit dem Titel *";DROP table user;* (auch bekannt als Little Bobby Tables, siehe Abbildung 4). Wäre hier SQL-Injection möglich, könnte dieser Titel bei Ausführung des `INSERT`-Statements in der Datenbank die `user`-Tabelle löschen. **Es wird geprüft, dass die Nutzerdaten auf der Profilseite korrekt angezeigt werden.**

[2] Abbildung von XKCD, lizenziert unter Namensnennung-Nicht kommerziell 2.5 Generic (CC BY-NC 2.5)

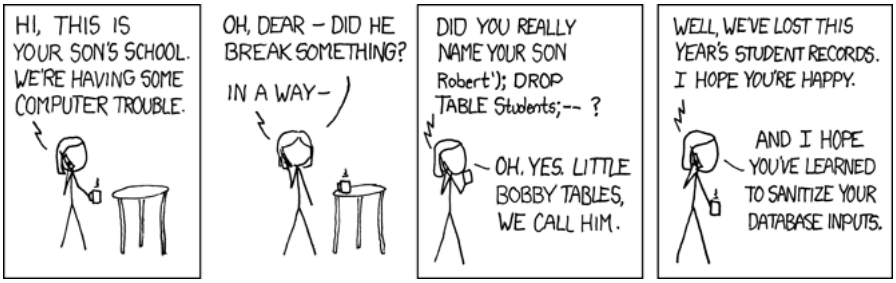

Abbildung 4: *Exploits of a Mom*[2]

Dann lädt der Nutzer eine Einreichung mit dem Titel *<script>alert('XSS');</script>* hoch. **Der Titel soll jetzt so wie eingegeben angezeigt werden anstatt dass der JavaScript-Code ausgeführt wird.**

4.2 Aufruf ohne Login

Ohne sich anzumelden, wird versucht, die Startseite (für authorisierte Nutzende) aufzurufen. **Der Nutzer soll auf die Anmeldeseite weitergeleitet werden.**

4.3 Session Fixation

Für diesen Test werden Cookies deaktiviert, damit die Session-ID in die URL geschrieben wird.

Ein Nutzer ruft die Webseite auf und bekommt eine Session-ID. Mit dieser Session meldet er sich an. **Wird die Webseite nun erneut mit der alten Session-ID aufgerufen, darf der Nutzer nicht angemeldet sein.**

4.4 Insecure Direct Object Reference

Dieser Test prüft, dass ein Nutzer nicht unberechtigt auf Inhalte zugreifen kann, indem er deren ID herausfindet.

Dazu legt ein angemeldeter Nutzer eine Einreichung an. Ein weiterer, neu registrierter Nutzer versucht jetzt, die Seite für diese Einreichung (mit richtiger ID als URL-Parameter) aufzurufen. **Die Einreichung darf nicht angezeigt werden; der Nutzer wird auf eine 404-Seite weitergeleitet.**

5 Testmetriken

Johannes Garstenauer

Im Folgenden soll die Qualität der Testsuites bestimmt werden, um ihre Aussagekraft hinsichtlich der Bestätigung der Qualität des LaSesSystem zu evaluieren. Hierzu wird die Quelltextüberdeckung der *Unittests* sowie der *Integrationstests* untersucht. Schlußendlich werden die Ergebnisse interpretiert.

5.1 Quelltextüberdeckung

Zur Messung der Quelltextüberdeckung wurde die populäre *Java Code Coverage Library* ***JaCoCo*** verwendet. In der Analyse wird ein besonderes Augenmerk auf die Werte der *Zeilen-* und *Zweigüberdeckung* gelegt, da diese am aussagekräftigsten bezüglich der Testqualität sind.

5.1.1 Unit Tests

Es existieren *Whiteboxtests* zu den meisten Klassen und Methoden des Projekts. Es besteht jedoch kein Anspruch auf Vollständigkeit. Die *Unittests* entstanden größtenteils parallel zur Entwicklung. Bestimmte Arten von Klassen wurden hierbei ausgespart:

- *Backing-Beans*: Es bestand die Erwartung, dass diese sinnvoller in den *Blackboxtests* abgedeckt werden können.
- *Klassen in den internal-Paketen*: Hier war in der Regel ein großer *Mockingaufwand* vonnöten, um erwartetes Verhalten testen zu können. Dies liegt an der großen Anzahl von *Jakarta Server Faces* spezifischen Komponenten, welche in diesen Klassen verwendet werden. Aufgrund des Wunsches, vonseiten der Entwickler, die begrenzte Entwicklungszeit wirkungsvoll einzusetzen und der oft überschaubaren *JSF* fremden Logik, welche zu überprüfen war, wurden diese Klassen beim *unittesting* oft ausgespart. Beispiele hierfür sind z.B. der *UncheckedExceptionHandler* oder *MessageResourceBundleProducer*.

Die *Quelltextüberdeckungswerte* sind vor diesem Hintergrund zu betrachten.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
de.lases.persistence.repository		56 %		36 %	473	755	1.365	3.187	95	312	3	22
de.lases.business.service		40 %		17 %	244	406	1.334	2.146	82	240	3	20
de.lases.control.backing		18 %		5 %	414	454	848	1.018	319	355	35	45
de.lases.control.internal		4 %		0 %	80	83	192	202	39	42	10	11
de.lases.control.validation		31 %		41 %	39	65	115	152	19	35	8	10
de.lases.persistence.util		20 %		0 %	14	18	92	100	11	15	2	4
de.lases.business.util		47 %		37 %	15	33	51	100	10	25	1	5
de.lases.global.transport		79 %		38 %	45	213	69	358	25	191	1	23
de.lases.persistence.exception		16 %		n/a	33	40	66	80	33	40	9	14
de.lases.business.internal		0 %		0 %	16	16	37	37	15	15	4	4
de.lases.persistence.internal		63 %		50 %	1	5	5	16	0	4	0	1
de.lases.control.conversion		0 %		n/a	3	3	5	5	3	3	1	1
de.lases.control.exception		28 %		n/a	3	4	6	8	3	4	1	2
Total	17.037 of 30.369	43 %	1.171 of 1.604	26 %	1.380	2.095	4.185	7.409	654	1.281	78	162

Abbildung 5: Übersicht über die genauen Überdeckungswerte (gegliedert nach Paketen)

Die 96 Testmethoden der Unittests erreichen eine *Zeilenüberdeckung* von **43%** und eine *Zweigüberdeckung* von **26%**.

5.1.2 Integration Tests

Die *Blackboxtests* decken große Teile der vorgesehenen Funktionalität ab und beinhalten zusätzlich Tests zu sicherheitskritischen Szenarien, wie *Skript- und Sequelinjektionen*. Es besteht wiederum kein Anspruch auf Vollständigkeit. Erwartete Schwachstellen der *Quelltextberdeckung* sind zu erwarten aufgrund von:

- Den entgegengesetzten Prinzipien von *Isolation* und *Integration* im Software Testing. Integrationstests weisen eine höhere Integration (d.h. ein höheres Clustering

der Features in den Tests) auf Kosten der Isolation auf und erreichen aufgrunddessen weniger Granularität, beispielsweise in der Zweigabdeckung. Daraus resultiert eine geringere *Quelltextüberdeckung*.

- Codeabschnitte, welche sich der defensiven Programmierung widmen, werden seltener überdeckt

Die *Quelltextüberdeckungswerte* sind vor diesem Hintergrund zu betrachten.

Die Integrationtests erreichen eine *Zeilenüberdeckung* von **45%** und eine *Zweigüberdeckung* von **34%**.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
de.lases.persistence.repository		47 %		34 %	411	545	1.155	2.130	38	123	0	10
de.lases.business.service		28 %		27 %	181	258	974	1.401	40	106	0	9
de.lases.control.backing		56 %		44 %	208	435	410	897	127	336	6	41
de.lases.control.validation		42 %		43 %	31	49	78	130	5	19	1	9
de.lases.global.transport		72 %		16 %	51	208	88	351	32	186	1	22
de.lases.persistence.util		30 %		50 %	14	18	75	100	11	15	2	4
de.lases.control.internal		65 %		57 %	36	83	68	202	8	42	1	11
de.lases.business.util		53 %		37 %	14	23	38	77	6	15	1	3
de.lases.persistence.exception		10 %		n/a	36	40	72	80	36	40	12	14
de.lases.business.internal		75 %		50 %	4	16	8	37	3	15	0	4
de.lases.persistence.internal		63 %		50 %	1	5	5	16	0	4	0	1
de.lases.control.conversion		0 %		n/a	3	3	5	5	3	3	1	1
de.lases.control.exception		28 %		n/a	3	4	6	8	3	4	1	2
Total	11.845 of 21.621	45 %	991 of 1.534	35 %	993	1.687	2.982	5.434	312	908	26	131

Abbildung 6: Übersicht über die genauen Überdeckungswerte (gegliedert nach Paketen)