

Inhaltsverzeichnis

1 Einleitung.....	1
2 Automatisiertes Testen (BDD).....	2
3 Automated CSS Regression Testing.....	5
3.1 Warum sind visuelle Tests notwendig?.....	5
3.2 Wie funktioniert Automated CSS Regression Testing?.....	7
4 Drupal und Automated CSS Regression Testing.....	9
4.1 Auswahl der Komponenten.....	9
4.2 Aufsetzen der Testumgebung.....	9
4.3 Durchführung CSS Regression Testing (Per manuellem Terminal-Befehl)	12
4.4 Durchführung Automated CSS Regression Testing.....	17
4.5 Problembehandlung „Mehrere aufeinander folgende, automatisierte Tests“	20
5 Fazit.....	21

Abbildungsverzeichnis

Abbildung 1

Screenshot PHPStorm base.feature der Testumgebung

Abbildung 2

Screenshot PHPStorm Ordnerstruktur der Testumgebung

Abbildung 3

<http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/germany>

23.06.2017 13:00 Uhr

Stand Mai 2017 StatCounter

Abbildung 4

<https://de.statista.com/statistik/daten/studie/217457/umfrage/anteil-mobiler-endgeraete-an-allen-seitenaufrufen-weltweit/>

23.06.2017 um 13:17 Uhr

Stand Januar 2017 Statista

Abbildung 5

<https://css-tricks.com/visual-regression-testing-with-phantomcss/>

23.06.2017 15:14 Uhr

Stand 17. November 2015 CSS-Tricks

Abbildung 6

Screenshot PHPStorm behat.yml der Testumgebung

Abbildung 7

Screenshot PHPStorm Javascript-Datei der Testumgebung (Abschnitt 1)

Abbildung 8

Screenshot PHPStorm Javascript-Datei der Testumgebung (Abschnitt 2)

Abbildung 9

Screenshot Terminal (Terminator) nach erster Ausführung des Testes

Abbildung 10

Screenshot Terminal (Terminator) bei erfolgreicher Ausführung des Testes

Abbildung 11

Default Screenshot Webseite der Testumgebung

Abbildung 12

Change Screenshot Webseite der Testumgebung

Abbildung 13

Screenshot Terminal (Terminator) bei fehlgeschlagener Ausführung des Testes

Abbildung 14

Diff Screenshot Webseite der Testumgebung

Abbildung 15

Screenshot PhpStorm base.feature der Testumgebung

Abbildung 16

Screenshot PhpStorm FeatureContext.php der Testumgebung (Abschnitt 1)

Abbildung 17

Screenshot PhpStorm base.feature der Testumgebung (Abschnitt 2)

Abbildung 18

Screenshot Terminal (Terminator) bei erfolgreicher Ausführung des Testes

Abbildung 19

Screenshot Terminal (Terminator) bei fehlgeschlagener Ausführung des Testes

1 Einleitung

Hypertext Markup Language (HTML) und Cascading Style Sheets (CSS) gehören zu den Kernsprachen des World Wide Web (www). Während sich HTML mit der Strukturierung von digitalen Dokumenten (z.B. Text, Bilder, Hyperlinks) beschäftigt, dient CSS ausschließlich zur Darstellung der Inhalte. Mit diesen Sprachen wurde der Grundstein für das World Wide Web gelegt und die Tür zu endlosen, neuen Möglichkeiten geöffnet.

Wie zuvor erwähnt, dient CSS der Darstellung von Inhalten. Im alltäglichen Tagesgeschäft von vielen deutschen Entwicklerfirmen stellt der Kunde täglich neue Anforderungen an seine Webseite. Diese müssen von den Entwicklern der Firma geplant, umgesetzt und gegengeprüft werden. Häufig werden optische Änderungen angefragt. Bei umfangreichen Projekten ist oftmals eine große Menge Code vorhanden. Ist jener nicht ordnungsgemäß strukturiert, oder wird vom Entwickler ein Fehler gemacht, kann es zu ungewollten Seiteneffekten auf der Webseite kommen. Z.B. wird das Layout einer Unterseite durch einen zu generischen Selektor verändert, ohne dass dies gewollt ist. Dies fällt dem zuständigen Entwickler/Kunden nicht auf und wird auf die Live-Webseite deployed. Allerdings fällt das Problem Nutzern auf und der Kunde leidet unter eventuellen Rufschäden. Dieses Problem soll durch automatisiertes Testen abgefangen werden.

Bisher sind lediglich Behavior Driven Development basierte Tests in die Firma erdfisch¹ integriert worden. Um dem Kunden mehr Sicherheit zu bieten, sollte Automated CSS Regression Testing (ACRT) in Betracht gezogen und als empfehlenswerte Methode verifiziert werden.

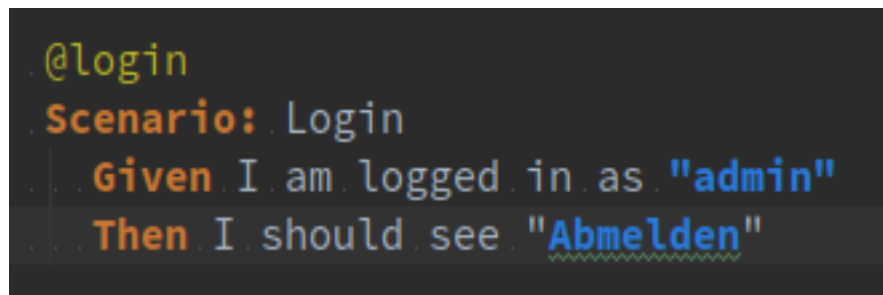
Diese Arbeit beschäftigt sich mit der Leitfrage:

Ist Automated CSS Regression Testing sinnvoll und sollte es als weitere Verifizierung bei erdfisch eingeführt werden?

1 <https://erdfisch.de/>

2 Automatisiertes Testen (BDD)

Behavior Driven Development (BDD) ist eine Technik der agilen Softwareentwicklung und soll als Verifizierung der Korrektheit der Funktionalität fungieren. Im Folgenden wird mit Hilfe von Behat² in Verbindung mit einem Continuous-Integration-Runner³ (CI-Runner) getestet.

A screenshot of a code editor showing BDD code. The code is written in a dark-themed IDE. It starts with a feature tag '@login' in green. Below it is a scenario 'Scenario: Login' in orange. Then, a 'Given' statement 'Given I am logged in as "admin"' is shown, with 'admin' in blue. Finally, a 'Then' statement 'Then I should see "Abmelden"' is shown, with 'Abmelden' in blue and underlined with a wavy line.

```
@login
Scenario: Login
    Given I am logged in as "admin"
    Then I should see "Abmelden"
```

Abbildung 1: Beispiel für BDD Code (IDE = PHPStorm)

Wie in Abbildung eins zu sehen ist, werden die Anforderungen die der Code erfüllen soll im base.feature mit Wenn-Dann-Sätzen definiert. Dies ist die Grundsyntax von Gherkin⁴ (Abzweigung von Cucumber⁵ Codebasis). Zunächst wird der "tag" des Tests angegeben, dieser wird verwendet um z.B. spezifische BDD Tests durchlaufen zu lassen, an Stelle eines kompletten Durchlauf aller Tests. In der nächsten Zeile wird das Szenario beschrieben welches der BDD Test kontrollieren soll. Dies dient dem Verständnis und der Auffindbarkeit des Szenario in den Logs. Darunter werden die Bedingungen für einen erfolgreichen Durchlauf des Tests definiert. Der Kontext der Bedingungen kann per PHP definiert werden. Die Syntax der Wenn-Dann-Sätze soll das Verständnis vereinfachen und auch "Nicht-IT"-Menschen dazu bewegen mit behat zu arbeiten.

2 <http://behat.org/en/latest/>

3 <https://about.gitlab.com/features/gitlab-ci-cd/>

4 <https://github.com/cucumber/cucumber/wiki/Gherkin>

5 <https://github.com/cucumber/cucumber>

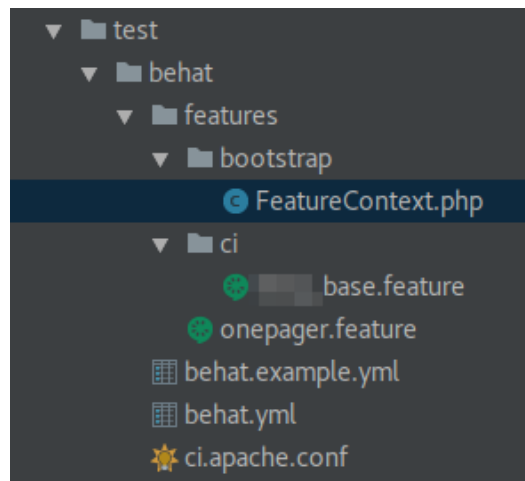


Abbildung 2: Beispiel für BDD Ordnerstruktur (IDE = PHPStorm)

In Abbildung zwei ist eine beispielhafte Ordnerstruktur für eine BDD Umgebung in Drupal abgebildet. Im “ci“-Ordner werden die Test mit der Syntax aus Abbildung eins definiert. Der Kontext der Bedingungen ist wiederum im “FeatureContext.php“ im “bootstrap“-Ordner festgehalten. Die „behat.yml“-Datei ist eine Konfiguration die die Default-Konfiguration für behat überschreibt. Des weiteren ist “ci.apache.conf“ die Konfigurationsdatei für den Apache-Server, welchen der CI-Runner startet. Dieser ermöglicht in Verbindung mit Behat eine vollautomatische, virtuelle Umgebung die überprüft, ob das Setup der Webseite den Bedingungen der definierten Tests entspricht.

Da PHP⁶ eine umfangreiche Programmiersprache ist und als Basis für das zu testende System gilt, können im Kontext fast alle Use Cases abgedeckt werden, die mit existierenden Objekten auf der Webseite zu tun haben. z.B. Prüfen ob ein Element nach dem rendern vorhanden ist, bzw. wie dessen Inhalt lauten soll. Jedoch gibt es Use Cases, die nicht mit einem Check auf Existenz eines Objektes zu lösen sind. Z.B. wird eine optische Änderung im Layout auf der Startseite vom Kunden veranlasst. Entwickler X benutzt einen allgemeinen CSS-Selektor um diese Änderung durchzuführen, bedenkt dabei aber nicht, dass dieser Selektor auch für andere Elemente einer oder mehrerer Unterseiten

6 <http://php.net/manual/de/intro-what-is.php>

verwendet wird. Da der prüfende Entwickler Y dies nicht bemerkt, wird diese Änderung auf die stage gebracht. Der Kunde bemerkt den Fehler ebenfalls nicht, was dazu führt, dass die ungewollte/ungesehene Änderung auf die Live-Webseite deployed wird.

Durch Fehler im Bereich CSS kann es zu Anzeigefehlern und Unbenutzbarkeit der Website kommen. Dies gilt es zu vermeiden.

3 Automated CSS Regression Testing

3.1 Warum sind visuelle Tests notwendig?

Wie in Abbildung drei⁷ zu sehen ist, nehmen mobile Endgeräte 32,45% des Marktes (in Deutschland) ein. Auch Tablets werden zu 6,76% verwendet. Trotzdem, dass Desktop-Geräte (Tower-PC, Laptops, etc.) mit 60,79% den Markt dominieren, wird in Abbildung vier⁸ sichtbar, dass in Europa mit mobilen Endgeräten 28,14% der Webseiten aufgerufen werden. Da Tablets ebenfalls eine andere Bildschirmauflösung als Desktop-Geräte haben, werden insgesamt 35,02% der Webseiten über kleinere Displays aufgerufen.

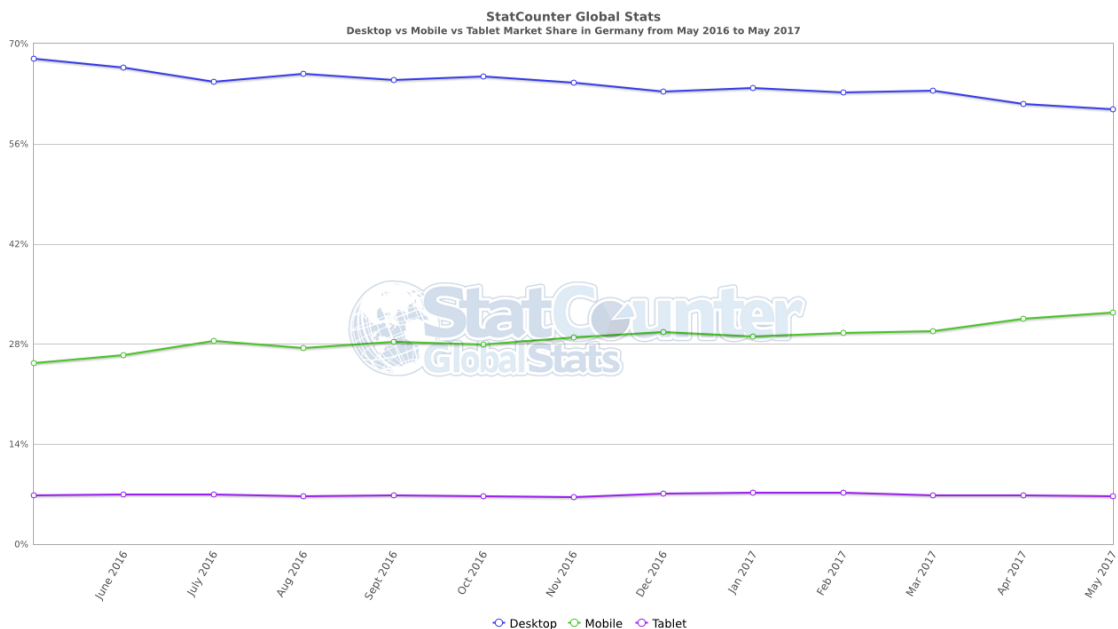


Abbildung 3: Desktop vs. Mobile vs. Tablet Market Share in Germany from May 2016 to May 2017

⁷ <http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/germany>

⁸ <https://de.statista.com/statistik/daten/studie/217457/umfrage/anteil-mobiler-endgeraete-an-allen-seitenaufrufen-weltweit/>

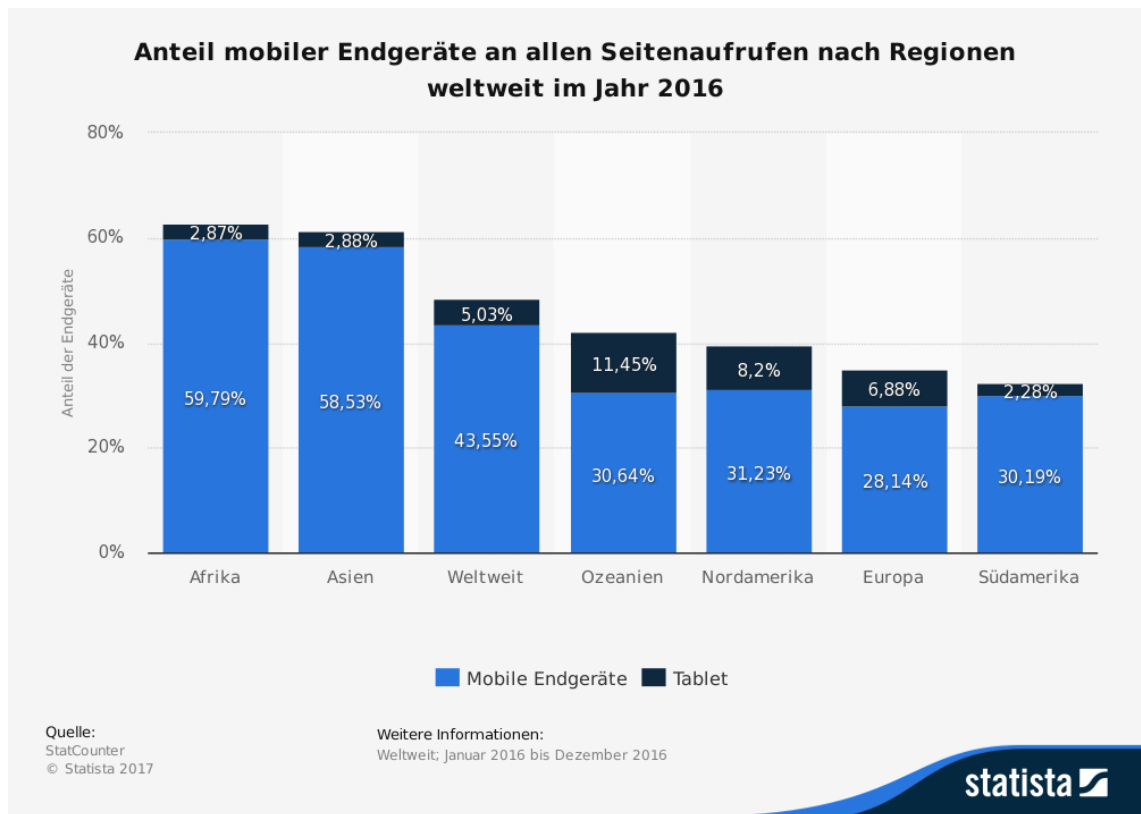


Abbildung 4: Anteil mobiler Endgeräte an allen Seitenaufrufen nach Regionen weltweit im Jahr 2016

Firmen müssen dem Kunden also “responsive Websites“ zur Verfügung stellen, wenn sie am Markt konkurrieren möchten. Viele Kunden, die sich für ihre Webseite an IT-Firmen wenden, haben die Notwendigkeit von Responsive Webdesign bereits erkannt.

Responsive Webseiten können durch eine umfangreiche Auswahl an CSS-Preprozessoren, Frameworks und Templates leicht umgesetzt werden. Allerdings ist das Entwickeln und Testen von Webseiten aufwendiger geworden durch die “responsiveness“. Durch den Mehraufwand sind Tests finanziell tragbar geworden. Deshalb ist es zu empfehlen, diese auch einzuführen.

3.2 Wie funktioniert Automated CSS Regression Testing?

Automated CSS Regression Testing (ACRT) ist ein automatisierter Test der überprüft, ob Änderungen im CSS zu ungewollten, visuellen Problemen führen. Momentan ist die Testmethode noch nicht zur Gänze ausgereift, dies sollte sich jedoch innerhalb der nächsten Jahre ändern.

Zunächst werden automatisiert Screenshots jeder Seite & Unterseite vor dem ersten Durchlauf des Tests aufgenommen. Diese Screenshots gelten als "Default" oder "Baseline", sprich sie werden als Standard verwendet. Bei jedem Durchlauf des Tests werden neue Screenshots jeder Seite & Unterseite aufgezeichnet und mit dem Baseline-Wert verglichen. Wie in Abbildung fünf zu sehen ist, legt die jeweilig verwendete Software die beiden Screenshots (Baseline und Change) übereinander und vergleicht diese auf Unterschiede. Diese Unterschiede werden mit Farbe hervorgehoben (Diff).

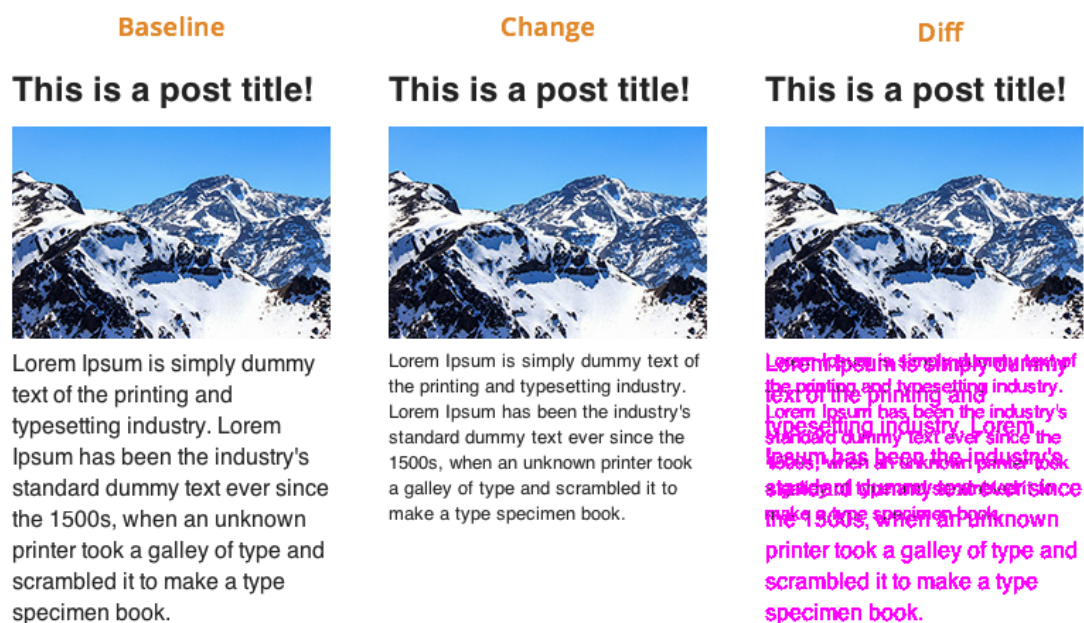


Abbildung 5: CSS Regression Testing Prozess

Falls Unterschiede auftauchen sollten, schlägt der Test fehl. Die Unterschiede werden ausgegeben und müssen lediglich ausgewertet werden. Diese Methodik spart sowohl dem Kunden als auch der IT-Firma Ressourcen ein.

4 Drupal und Automated CSS Regression Testing

4.1 Auswahl der Komponenten

Als Grundlage für das ACRT wurde PhantomCSS⁹ gewählt und eine bereits bestehende Drupal-Webseite.

PhantomCSS ist ein CasperJS¹⁰ Modul, welches auf PhantomJS 2¹¹(oder SlimerJS¹²) und Resemble.js¹³ basiert. CasperJS dient als headless¹⁴ Grundlage für PhantomCSS, d.h. es kann unabhängig von grafischen Oberflächen getestet werden. Resemble.js wird zum vergleichen der - beim testen erstellten - Screenshots benutzt. Dieser Vergleich erfolgt auf Basis der Suche nach rgb Pixel Unterschieden. PhantomCSS generiert daraus die Bilder Differenzen (Diffs).

Tests dieser Art funktionieren jedoch nur, wenn die Seite vorhersehbar ist. Es ist zwar möglich User-Interface (UI) Komponenten auszublenden oder nur gewisse Komponenten zu testen, jedoch ist es empfehlenswert, wenn die Webseite dabei statisch ist.

4.2 Aufsetzen der Testumgebung

Die Testumgebung wurde als neuer Branch, innerhalb eines bereits bestehenden Drupal-Projektes der Firma erdfisch, erzeugt. Aus Verschwiegenheitsgründen und Rücksicht auf die Kunden, wird der Name dieses Projektes in dieser Arbeit nicht erwähnt und in Screenshots verpixelt. Dies gilt ebenfalls für sensible Daten wie z.B. Pfade (Paths).

Zwar wird der Code nicht zugänglich gemacht, jedoch kann die Code-

9 <https://github.com/Huddle/PhantomCSS>

10 <https://github.com/casperjs/casperjs>

11 <https://github.com/ariya/phantomjs/>

12 <https://slimerjs.org/>

13 <http://huddle.github.io/Resemble.js/>

14 https://en.wikipedia.org/wiki/Headless_software

Grundlage, auf der die Testumgebung basiert, unter folgendem Link heruntergeladen werden: <https://github.com/Huddle/PhantomCSS>. Des Weiteren werden Screenshots eingefügt, um die Funktionsweise nachvollziehen zu können.

Als Integrierte Entwicklungsumgebung (IDE = *integrated development environment*) wurde PHPStorm in Verbindung mit Git (Versionierungssoftware) verwendet. Als Entwicklungsdatenbank (Repository) wurde ein erdfisch-interner Gitlab¹⁵ Server verwendet.

Um die Tests automatisiert ausführen zu können, wurde Behat (als Framework) in Verbindung mit einem Selenium-Standalone-Server¹⁶ (automatisiert Browser, d.h. dessen Aktionen) in die Testumgebung integriert. Diese Tests laufen über den zuvor erwähnten CI-Runner.

15 <https://about.gitlab.com/>

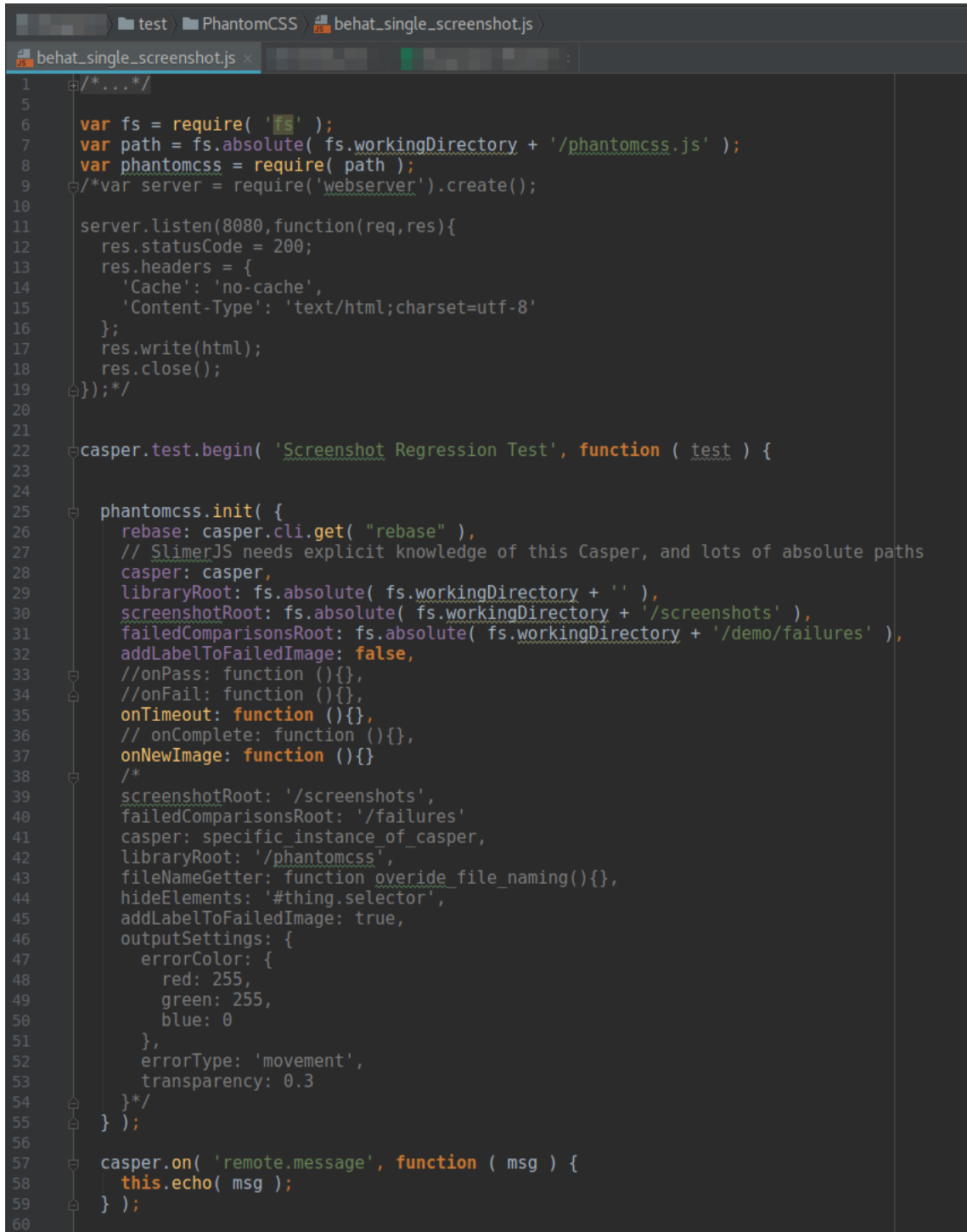
16 <http://www.seleniumhq.org/>

```
1 default:
2   calls:
3     error_reporting: 32759
4   suites:
5     default:
6       contexts:
7         - FeatureContext:
8           screenshot_path:
9         - Drupal\DrupalExtension\Context\DrupalContext
10        - Drupal\DrupalExtension\Context\MinkContext
11        - Drupal\DrupalExtension\Context\BatchContext
12   formatters:
13     pretty:
14       verbose: true
15       paths: false
16       snippets: true
17     html:
18       output_path:
19   extensions:
20     emuse\BehatHTMLFormatter\BehatHTMLFormatterExtension:
21       name: html
22       renderer: Behat2
23       print_args: true
24       print_outp: true
25       loop_break: true
26     Behat\MinkExtension:
27       browser_name: "chrome"
28       selenium2:
29         wd_host: "http://localhost:4444/wd/hub"
30       goutte: ~
31       base_url:
32     Drupal\DrupalExtension:
33       api_driver: "drupal"
34       blackbox: ~
35       drush:
36         root:
37       drupal:
38         drupal_root:
39
```

Abbildung 6: behat.yml des Projektes

Die Konfiguration der Behat-Selenium-Verbindung erfolgt in der behat.yml wie in Abbildung sechs zu sehen ist. Diese Konfiguration liegt innerhalb der Drupal Installation, und dient als Schnittstelle zwischen Drupal und dem aufgesetzten Testkonstrukt.

4.3 Durchführung CSS Regression Testing (Per manuellem Terminal-Befehl)



```
1  /* ... */
5
6  var fs = require( 'fs' );
7  var path = fs.absolute( fs.workingDirectory + '/phantomcss.js' );
8  var phantomcss = require( path );
9  /*var server = require('webserver').create();
10
11  server.listen(8080,function(req,res){
12    res.statusCode = 200;
13    res.headers = {
14      'Cache': 'no-cache',
15      'Content-Type': 'text/html;charset=utf-8'
16    };
17    res.write(html);
18    res.close();
19  });*/
20
21
22  casper.test.begin( 'Screenshot Regression Test', function ( test ) {
23
24
25    phantomcss.init( {
26      rebase: casper.cli.get( "rebase" ),
27      // SlimerJS needs explicit knowledge of this Casper, and lots of absolute paths
28      casper: casper,
29      libraryRoot: fs.absolute( fs.workingDirectory + ' ' ),
30      screenshotRoot: fs.absolute( fs.workingDirectory + '/screenshots' ),
31      failedComparisonsRoot: fs.absolute( fs.workingDirectory + '/demo/failures' ),
32      addLabelToFailedImage: false,
33      //onPass: function (){},
34      //onFail: function (){},
35      onTimeout: function (){},
36      // onComplete: function (){},
37      onNewImage: function (){}
38      /*
39      screenshotRoot: '/screenshots',
40      failedComparisonsRoot: '/failures'
41      casper: specific instance of casper,
42      libraryRoot: '/phantomcss',
43      fileNameGetter: function override_file_naming(){},
44      hideElements: '#thing.selector',
45      addLabelToFailedImage: true,
46      outputSettings: {
47        errorColor: {
48          red: 255,
49          green: 255,
50          blue: 0
51        },
52        errorType: 'movement',
53        transparency: 0.3
54      }
55    } );
56
57    casper.on( 'remote.message', function ( msg ) {
58      this.echo( msg );
59    } );
60
```

Abbildung 7: Javascript Datei der Testumgebung (Abschnitt 1)


```
test > PhantomCSS > behat_single_screenshot.js >
behat_single_screenshot.js x
47     errorColor: {
48         red: 255,
49         green: 255,
50         blue: 0
51     },
52     errorType: 'movement',
53     transparency: 0.3
54 } */
55 } );
56
57 casper.on( 'remote.message', function ( msg ) {
58     this.echo( msg );
59 } );
60
61 casper.on( 'error', function ( err ) {
62     this.die( "PhantomJS has errored: " + err );
63 } );
64
65 casper.on( 'resource.error', function ( err ) {
66     casper.log( 'Resource load error: ' + err, 'warning' );
67 } );
68 /*
69     The test scenario
70 */
71 var mainUrl = casper.cli.get("url");
72 var pageName = casper.cli.get("pagename");
73 var element = casper.cli.get("element");
74
75 casper.start(mainUrl + '/' + pageName);
76
77 casper.viewport( 1024, 768 );
78
79 casper.then( function () {
80     phantomcss.screenshot( element, pageName.replace(/\\//g, '-') + '***' + element );
81 } );
82
83 casper.then( function now_check_the_screenshots() {
84     // compare screenshots
85     phantomcss.compareAll();
86 } );
87
88 /*
89     Casper runs tests
90 */
91 casper.run( function () {
92     console.log( '\nTHE END.' );
93     casper.test.done();
94     console.log( '[' + phantomcss.getExitStatus() + ']' ) // pass or fail?
95 } );
96
97 } );
98
```

Abbildung 8: Javascript Datei der Testumgebung (Abschnitt 2)

In Abbildung sieben und acht ist die Javascript Datei der Testumgebung zu sehen. Hier wird mit Hilfe von Casper der Test initialisiert, nachdem dieser im Terminal angestoßen wurde (siehe Abbildung neun). Das eigentliche Szenario wird ab Zeile 71 definiert. Zunächst werden hier die drei Parameter abgefangen, die beim Initialbefehl (siehe Abbildung neun) mitgegeben wurden und in

Variablen gespeichert, um sie in den folgenden Funktionen verwenden zu können. Darauf folgend wird die Start URL aus den Parametern gebildet und die Größe des Viewport festgelegt.

Die erste Funktion erstellt einen Screenshot und benennt diese nach einem Muster um, mit Hilfe einer Regular Expressions. Die darauf folgende Funktion stößt den Vergleich der Screenshots an. Die letzte Funktion gibt die Meldung aus die in Abbildung zehn in den beiden letzten Zeilen zu sehen ist. „THE END.“ wird per console.log übergeben und zwischen den [] Klammern wird entweder einer 0 oder 1 ausgegeben. 0 bedeutet, dass der Test erfolgreich war; 1, dass dieser fehlgeschlagen ist. In Abbildung neun ist „undefined“ als Status gegeben, dies ist der Fall, wenn ein Test zum ersten Mal aufgeführt wird. Da hier kein Test angestoßen wird, wird nichts zurückgegeben (Kein return).

```
+ PhantomCSS git:(feature/TestBehatCss) x casperjs test behat single screenshot.js --url='http://[REDACTED].local' --pagename='home' --element='.row-6'
Test file: behat single screenshot.js
# Screenshot Regression Test
PASS Screenshot Regression Test (NaN test)

Must be your first time?
Some screenshots have been generated in the directory [REDACTED]/test/PhantomCSS/screenshots
This is your 'baseline', check the images manually. If they're wrong, delete the images.
The next time you run these tests, new screenshots will be taken. These screenshots will be compared to the original.
If they are different, PhantomCSS will report a failure.

THE END.
[[undefined]]
NaN Looks like you didn't run any tests.
+ PhantomCSS git:(feature/TestBehatCss) x
```

Abbildung 9: Terminal Ausgabe nach erster Ausführung des Testes

```
+ PhantomCSS git:(feature/TestBehatCss) x casperjs test behat single screenshot.js --url='http://[REDACTED].local' --pagename='home' --element='.row-6'
Test file: behat single screenshot.js
# Screenshot Regression Test
PASS Screenshot Regression Test (NaN test)

PASS Should look the same [REDACTED]/test/PhantomCSS/screenshots/home**.row-6_0.png
PhantomCSS found 1 tests, None of them failed. Which is good right?
If you want to make them fail, change some CSS.

THE END.
[[0]]
PASS 1 test completed in 0.001s. 0 failed, 0 pending, 0 skipped.
+ PhantomCSS git:(feature/TestBehatCss) x
```

Abbildung 10: Terminal Ausgabe bei erfolgreicher Ausführung des Testes



Abbildung 11: Default Screenshot nach erster Ausführung des Testes

Abbildung elf ist der Screenshot, der bei erstmaliger Ausführung des Testes aufgenommen wurde. Dieser wird als Default gesetzt, kann jedoch durch löschen und erneuter Ausführung des Testes ersetzt werden (siehe Abbildung neun). Er dient als Grundlage an dem (siehe Abbildung zehn & 13) getestet wird, ob Veränderungen vorliegen.

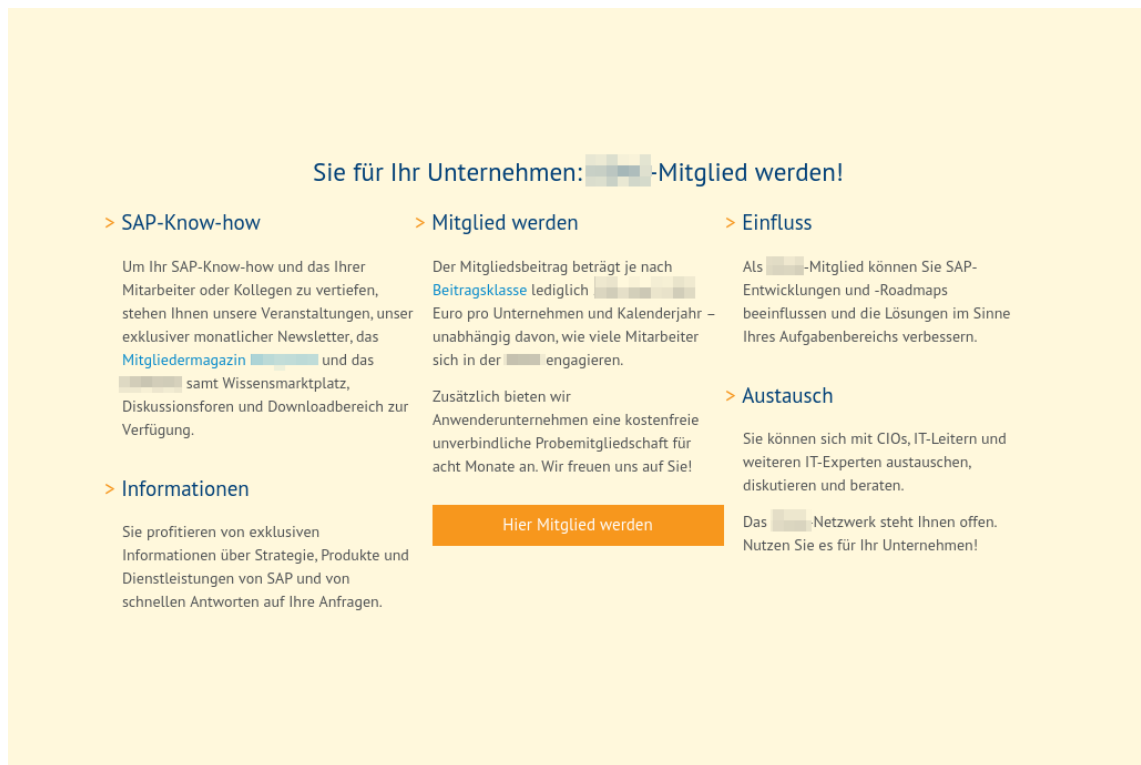


Abbildung 12: Change Screenshot nach Änderung im CSS und zweiter Ausführung des Testes

In Abbildung zwölf ist der Change-Screenshot zu sehen. Dies zeigt den getesteten Ausschnitt wie er nach der Veränderung auf der Seite aussieht. Im Test wurde der Hintergrund gelöscht und durch einen Farbton ersetzt. Nach Beendigung des Testes, wird ein Fehler ausgegeben (siehe Abbildung 13). Dieser gibt sowohl den Pfad zum Screenshot, als auch die Prozente aus, in welchem Maß sich die Screenshots von einander differenzieren.

```

* PhantomCSS git:(feature/TestBehatCSS) x casperjs test behat single screenshot.js --url='http://localhost' --pagename='home' --element='.row-6'
test file: behat_single_screenshot.js
# Screenshot Regression Test
PASS Screenshot Regression Test (NaN test)

Looks different (46.19% mismatch) /test/PhantomCSS/demo/failures/home*.row-6_0.fail.png
# type: fail
# file: behat_single_screenshot.js
# subject: false

PhantomCSS found 1 tests, 1 of them failed.

PhantomCSS has created some images that try to show the difference (in the directory /test/PhantomCSS/demo/failures). Fuchsia colored pixels indicate a difference between the new
nd old screenshots.

THE END.
[[[1]]]

```

Abbildung 13: Terminal Ausgabe bei fehlgeschlagener Ausführung des Testes



Abbildung 14: Diff Screenshot

Der fehlgeschlagene Test erzeugt des weiteren einen Diff Screenshot (siehe Abbildung 14), welcher beide Zustände der Seite vergleicht und visuell rosa markiert. Rosa ist die Standard-Farbe zum markieren von Unterschieden, da es am besten zu sehen ist, falls nur minimale Unterschiede entstehen sollten. Wie in Abbildung 14 zu sehen, ist hier der ersetzte Hintergrund markiert und hervorgehoben.

4.4 Durchführung Automated CSS Regression Testing

Der zuvor durchgeführte Test wurde manuell angestoßen. Dies ist zwar eine Möglichkeit, jedoch umständlich und schlecht in die Behat-Tests integrierbar. Aufgrund dessen wurden (wie zu Anfang erwähnt) die Komponenten Behat2 & der CI-Runner implementiert.

```

416
417 @Testbehatcss
418 Scenario: Css Screenshot Test
419     Given I am "Mitarbeiter"
420     And I go to "news"
421     Then I take a screenshot of current page "news"
422     Then I compare screenshots for regression
423
424 @Contentcheck
425 Scenario: Check element on page
426     Then Do regression test for ".row-6" on "/"
427
428

```

Abbildung 15: base.feature (Gherkin Code) der Testumgebung

Wie in Abbildung 15 zu sehen, wurden hier mit Gherkin mehrere Tests verfasst, die als Use Cases dienen sollen. Diese können für jede Seite (siehe @Testbehatcss) oder für einzelne Komponenten einer definierten Seite (siehe @Contentcheck) definiert werden. Diese Tests werden bei jedem Commit, in einem eigens dafür erzeugten Docker¹⁷-Container, angestoßen (vom CI-Runner).

```

632 /**
633  * @Then /~I take a screenshot of current page "([^"]*)"$/
634  */
635 public function takeAScreenshotOfCurrentPage($pagename) {
636     if ($this->enableDiffScreenshots) {
637         $diff = '.diff';
638     }
639     else {
640         $diff = '';
641     }
642     $image_data = $this->getSession()->getDriver()->getScreenshot();
643     file_put_contents( filename: 'test/PhantomCSS/screenshots/behat_' . $pagename . $diff . '.png', $image_data);
644 }
645
646 /**
647  * @Then I compare screenshots for regression
648  */
649 public function compareScreenshots() {
650     shell_exec( cmd: "cd test/PhantomCSS && casperjs test demo/testsuite.js");
651 }

```

Abbildung 16: FeatureContext.php der Testumgebung (Abschnitt 1)

```

653 /**
654  * @Then Do regression test for element on :page
655  */
656 public function cssRegressionTest($pagename, $element) {
657     global $base_url;
658     print( DIR );
659     $output = shell_exec( cmd: "cd " . DIR . "/../test/PhantomCSS && casperjs test behat_single_screenshot.js --url=\"$base_url\" --pagename=\"$pagename\" --element=\"$element\"");
660
661     $begin = strpos($output, '[[[');
662     $end = strpos($output, ']]]');
663
664     $result = substr($output, $begin + 3, $end - ($begin + 3));
665
666     if ($result == 0) {
667     }
668     else {
669         throw new Exception( message: "Regression Test Result: WARNING Visual regressions found, check file " . str_replace( search: '/', replace: '-', $pagename ) . " . " . $element );
670     }
671 }
672
673

```

Abbildung 17: FeatureContext.php der Testumgebung (Abschnitt 2)

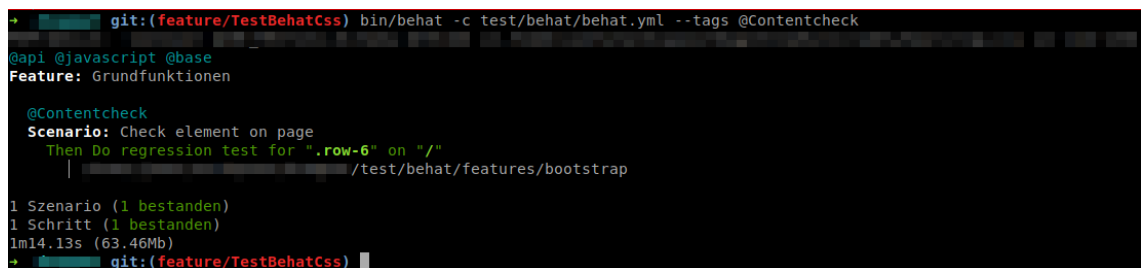
Die in Abbildung 15 verwendeten Phrasen wurden in der FeatureContext.php definiert (Abbildung 16 und 17). Hier werden die (beim Initialbefehl

17 <https://www.docker.com/>

eingeegebenen) Variablen abgefangen. Abbildung 16 enthält die beiden Funktionen die für das erste Szenario notwendig sind (@Testbehatcss).

Die erste Funktion benennt die Screenshots, je nachdem, ob eine Differenz vorliegt oder nicht und setzt die Metadaten der Bilder. Der zensierte Teil enthält die persönlich Pfaddefinition, welche bei Bedarf durch eine Pfadvariable ersetzt werden kann. Die zweite Funktion reicht einen Befehl an die Shell des Benutzers weiter, der den Vergleich der Screenshots anstößt (per casper.js).

Die Funktion in Abbildung 17 enthält den Anstoß zum Screenshot Vergleich (anderer Anstoß als der zuvor genannte), überprüft die letzten Zeilen der Ausgabe nach den [] Klammern und definiert eine Exception, falls das Ergebnis nicht 0 ist.

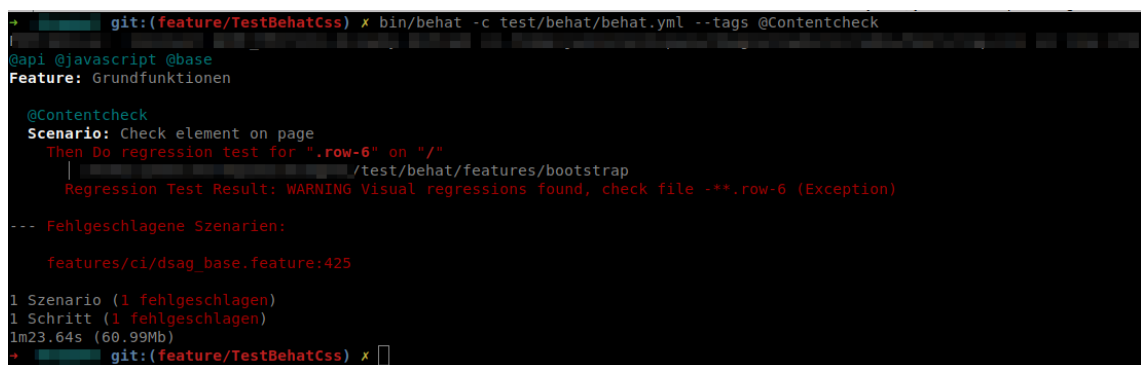


```
+ git:(feature/TestBehatCss) bin/behac -c test/behac/behac.yml --tags @Contentcheck
@api @javascript @base
Feature: Grundfunktionen

@Contentcheck
Scenario: Check element on page
  Then Do regression test for ".row-6" on "/"
    | /test/behac/features/bootstrap

1 Szenario (1 bestanden)
1 Schritt (1 bestanden)
1m14.13s (63.46Mb)
+ git:(feature/TestBehatCss)
```

Abbildung 18: Ausgabe Terminal bei erfolgreichem Test



```
+ git:(feature/TestBehatCss) x bin/behac -c test/behac/behac.yml --tags @Contentcheck
@api @javascript @base
Feature: Grundfunktionen

@Contentcheck
Scenario: Check element on page
  Then Do regression test for ".row-6" on "/"
    | /test/behac/features/bootstrap
    Regression Test Result: WARNING Visual regressions found, check file **.row-6 (Exception)

--- Fehlgeschlagene Szenarien:

    features/ci/dsag_base.feature:425

1 Szenario (1 fehlgeschlagen)
1 Schritt (1 fehlgeschlagen)
1m23.64s (60.99Mb)
+ git:(feature/TestBehatCss) x
```

Abbildung 19: Ausgabe Terminal bei fehlgeschlagenem Test

Die Ausgabe des Terminals wird in Abbildung 18 dargestellt. Dies ist die automatische Ausgabe des CI-Runners, der wiedergibt ob der Test bestanden (Abbildung 18) wurde oder nicht (zu sehen in Abbildung 19). Die entstandenen

Screenshots, sind die selben wie im manuellen Test, da beide auf dieselben Vergleichsmechanismen zurückgreifen.

4.5 Problembehandlung „Mehrere aufeinander folgende, automatisierte Tests“

Ein Problem der momentanen Testumgebung ist die Ausführung von mehreren, aufeinander folgenden Test die jeweils ein anderes Element testen.

Das Problem ist, dass ein Test ausgeführt wird und ein Base-Screenshot erstellt wird. Wird darauf ein anderer Test angestoßen, der eigentlich einen eigenen Screenshot für das Element erzeugen sollte, nimmt dieser stattdessen den zuvor aufgenommenen Screenshot und vergleicht das neue Element mit dem alten. Dadurch schlägt jeder Test fehl, der nicht genau das gleiche Element überprüft, wie der erste Test. Dieses Problem tritt nur auf, wenn die Tests über behat angestoßen werden.

Um dieses Problem zu lösen, sollte Datenmapping als Prozess eingebunden werden, der Datenrelationen identifiziert, d.h. den einzelnen Tests die entsprechenden Screenshots zuteilt. Dies kann durch eine relationelle Datenbank oder XSLT umgesetzt werden.

5 Fazit

Ist Automated CSS Regression Testing sinnvoll und sollte es als weitere Verifizierung bei erdfisch eingeführt werden?

Automated CSS Regression Testing ist zur Zeit ein unausgereifter Prozess. Bevor dieser in ein Großprojekt eingebaut wird, sollte gewartet werden, bis eine stabile und sichere Integration möglich ist. Ist dieser Zeitpunkt jedoch erreicht, sollten diese Tests integriert werden, da sie Mehraufwände wie z.B. die Behebung visueller Bugs in Hotfixes vermeiden. Des Weiteren können durch die automatisierten Tests Personentage (PT) eingespart werden. Diese sind meist zuverlässiger als, für Fehler anfällige, von Menschenhand ausgeführte Tests.

Das erstellte Testkonstrukt besteht momentan aus einer Custom-Lösung, die drei PT in Anspruch genommen hat und weitere Arbeit (wie z.B. Mapping) benötigt. Um eine stabile und sichere Integration zu gewährleisten, müssten weitere Ressourcen (PT, Entwickler) verwendet werden. Dies würde mindestens drei weitere PT in Anspruch nehmen, da noch Fallbacks implementiert und Tests durchgeführt werden müssen, um eventuelle Bugs zu beheben. Jedoch lohnt sich dieser Aufwand in Hinsicht auf die uneingeschränkte Nutzung von den ACRT Tests.

Das oben ausgeführt Testkonstrukt ist ein Proof-of-concept, der zu Bildungszwecken erstellt wurde. Zur Zeit werden umfangreiche Tools entwickelt, welche mit aufwändigen Suites und UI's ausgestattet sind, um ACRT sauber in Drupal zu integrieren (z.B. Wraith¹⁸, Galen¹⁹, Gemini²⁰).

Ja, erdfisch sollte ACRT einführen. Allerdings sollte dies auf Basis einer größeren Suite geschehen (siehe Wraith) und mit BDD verbunden werden.

18 <https://github.com/BBC-News/wraith>

19 <https://github.com/galenframework/galen>

20 <https://github.com/gemini-testing/gemini>

Literaturverzeichnis

Alexander Wunschik (17. Oktober 2017): Awesome Visual Regression Testing, online im Internet, <https://github.com/mojoaxel/awesome-regression-testing>, Abfrage vom 17.10.2017 um 13:57 Uhr

aurweb Development Team (2017): Selenium-Server-Standalone, online im Internet, <https://aur.archlinux.org/packages/selenium-server-standalone/>, Abfrage vom 30.09.2017 um 12:45 Uhr

Benutzer: blackmore (27 September 2017): Gherkin, online im Internet, <https://github.com/cucumber/cucumber/wiki/Gherkin>, Abfrage vom 02.10.2017 um 15:22 Uhr

Benutzer: Lustiger_seth (24. April 2017): Hypertext Markup Language, online im Internet, https://de.wikipedia.org/wiki/Hypertext_Markup_Language, Abfrage vom 31.05.2017 um 10:43 Uhr

Benutzer: MovGP0 (2. Mai 2017): Behavior Driven Development, online im Internet, https://de.wikipedia.org/wiki/Behavior_Driven_Development, Abfrage vom 22.06.2017 um 9:14 Uhr

Benutzer: Nick Number (12. Juli 2017): Headless software, online im Internet, https://en.wikipedia.org/wiki/Headless_software, Abfrage vom 30.09.2017 um 11:06 Uhr

Benutzer: Trustable (19. Mai 2017): Liste von Programmiersprachen, online im Internet, https://de.wikipedia.org/wiki/Liste_von_Programmiersprachen, Abfrage vom 21.06.2017 um 9:57 Uhr

Dan North (2017): Introducing BDD, online im Internet,
<https://dannorth.net/introducing-bdd/>, Abfrage vom 22.06.2017 um 9:09 Uhr

Dmitriy Dudkevich (01. September 2017): Gemini, online im Internet,
<https://github.com/gemini-testing/gemini>, Abfrage vom 17.10.2017 um 14:00 Uhr

Frank Holldorff (2017): erdfisch, online im Internet, <https://erdfisch.de/>, Abfrage vom 21.06.2017 um 10:14 Uhr

Horst Gräbner (13. März 2017): Cascading Style Sheets, online im Internet,
https://de.wikipedia.org/wiki/Cascading_Style_Sheets, Abfrage vom 31.05.2017 um 10:29 Uhr

Ivan Shubin (06. September 2017): Galen, online im Internet,
<https://github.com/galenframework/galen>, Abfrage vom 17.10.2017 um 13:55 Uhr

James Cryer (3. April 2017): PhantomCSS, online im Internet,
<https://github.com/Huddle/PhantomCSS>, Abfrage vom 31.05.2017 um 10:50 Uhr

James Cryer (2017): Resemble.js, online im Internet,
<http://huddle.github.io/Resemble.js/>, Abfrage vom 29.09.2017 um 16:47 Uhr

Jamie Mason & Maurice Svay (2017): PhantomJS, online im Internet,
<http://phantomjs.org/>, Abfrage vom 31.05.2017 um 10:44 Uhr

Jon Bellah (17. November 2017): Visual Regression Testing with PhantomCSS, online im Internet, <https://css-tricks.com/visual-regression-testing-with-phantomcss/>, Abfrage vom 23.06.2017 um 15:34 Uhr

Joseph Wynn (15. März 2017): Wraith, online im Internet,

<https://github.com/BBC-News/wraith>, Abfrage vom 17.10.2017 um 13:46 Uhr

Ken Soh (23. August 2017): CasperJS, online im Internet,
<https://github.com/casperjs/casperjs>, Abfrage vom 29.09.2017 um 16:25 Uhr

Konstantin Kudryashov (2016): behat, online im Internet,
<http://behat.org/en/latest/>, Abfrage vom 22.06.2017 um 12:09 Uhr

Laurent Jouanneau (2017): SlimerJS, online im Internet, <https://slimerjs.org/>,
Abfrage vom 29.09.2017 um 16:45 Uhr

Marit van Dijk (01. Oktober 2017): Cucumber, online im Internet,
<https://github.com/cucumber/cucumber>, Abfrage vom 2.10.2017 um 15:23 Uhr

PHP Group (2017): Was ist PHP?, online im Internet,
<http://php.net/manual/de/intro-what-is.php>, Abfrage vom 23.06.2017 um 9:30 Uhr

Sean Packham (2017): Gitlab Continuous Integration & Deployment, online im Internet,
<https://about.gitlab.com/features/gitlab-ci-cd/>, Abfrage vom 22.06.2017 um 12:00 Uhr

SELFHTML e.V. (17. März 2017): Viewport, online im Internet,
<http://wiki.selfhtml.org/wiki/Viewport>, Abfrage vom 30.09.2017 um 15:30 Uhr

Simon Stewart (27. September 2017): SeleniumHQ, online im Internet,
<http://www.seleniumhq.org/>, Abfrage vom 30.09.2017 um 12:54 Uhr

Vitaly Slobodin (25. Juni 2017): PhantomJS - Scriptable Headless WebKit,
online im Internet, <https://github.com/ariya/phantomjs/>, Abfrage vom 28.09.2017 um 8:59 Uhr

Ich versichere hiermit, dass ich diese Projektarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

.....

Ort, Datum

.....

Unterschrift