

Abschlussprojekt 2016



Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Entwicklung eines Tools für die Durchführung visueller Regressionstests

Auszubildender

Jan Dahlke

Weidenstraße 22

25469 Halstenbek

Ausbildungsbetrieb

t8y.com GmbH

Bernhard Nocht Straße 113

20359 Hamburg



1. Inhaltsverzeichnis

1. Inhaltsverzeichnis	2
2. Erklärung	2
2.1. Versicherung an Eides statt	3
2.2. Softwaredownload.....	3
3. Einleitung.....	4
3.1. Was sind visuelle Regressionstests?	4
3.2. Der innerbetriebliche Anwendungsfall	4
3.3. Das Projekt „visuelle Regressionstests Tool“	6
4. Überblick über die verwendeten Technologien.....	6
4.1. Basis und Basismodule	6
4.1.1. Node	6
4.1.2. npm	6
4.1.3. Phantomjs	6
4.1.4. Graphicsmagick.....	7
4.2. Die Node Pakete im Überblick	7
4.3. Die konkrete Anwendung der Technologien.....	8
4.4. Ein paar Worte zur Architektur	9
4.5. Der Befehlssatz des VRTT	9
5. Das Programm - VRTT	10
5.1. Die Projektstruktur.....	10
5.2. Anwendung am Beispielprojekt	11
5.3. Ausgewählte Quellcodebeispiele.....	12
5.4. Erforderliche Anpassungen des gm Moduls	14
6. Theoretische Überlegungen zu visuellen Regressionstests	16
7. Die einfache Nutzwertanalyse	17
8. Ausblick und Fazit	18
9. Abbildungsverzeichnis.....	20
10. Quellenverzeichnis	22
10.1. Bücher:	22
10.2. Onlineressourcen	22
10.3. Weiterhin genutzt	23

2. Erklärung

2.1. Versicherung an Eides statt

Hier erkläre ich, Jan Dahlke, an Eides statt, dass ich die vorliegende Projektarbeit selbstständig programmiert und verfasst habe. Ich erkläre, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Jan Dahlke, Hamburg 30.05.2016

2.2. Softwaredownload

Für die, von mir programmierte, Arbeit habe ich ein Transferkonto angelegt.

Dieses ist unter dem nachfolgenden Link mit Eingabe von Nutzernamen und Passwort bis zum 29.06.2016 erreichbar. Die Datei muss entpackt werden und die weiterführenden Installationsanweisungen befinden sich in der Datei Readme.md.

Link:

<https://ham-transfer.de.ddb.com/ticket.html?id=a7nnymcpz8tzy1nqnk>

User: jdahlke

Passwort: kBsX7URh

3. Einleitung

Diese Arbeit ist die Dokumentation des Abschlussprojekts, das im Rahmen der IHK Prüfung von mir, dem Prüfling Jan Dahlke, im Unternehmen t8y.com GmbH umgesetzt wurde. Die Dokumentation gewährt dabei einen Einblick in das Vorhaben und die Ausführung, ein Programm zu entwickeln, das visuelle Regressionstests in einem automatisierten Verfahren umsetzt. Sie erörtert die dabei benutzten Technologien, zeigt deren Anwendung auf ein Testprojekt. In der Dokumentation werden, anhand konkreter Quellcodebeispiele, exemplarisch wichtige Funktionen des Programms diskutiert und Probleme aufgezeigt, die während des Entwicklungsprozesses auftraten. Darüber hinaus versucht die Arbeit, die Grenzen des Testverfahrens zu beschreiben und gibt konkrete Ausblicke auf denkbare Erweiterungen des Programms. Abschließend wird der Versuch unternommen, mittels einer einfachen Nutzwertanalyse Aussagen über die Ergiebigkeit (Rentabilität) der Anwendung zu treffen.

3.1. Was sind visuelle Regressionstests?

Unter einem Regressionstest versteht man die Wiederholung von Testfällen, um sicherzustellen, dass Modifikationen in bereits getesteten Teilen der Software keine neuen Fehler (Regressionen) verursachen. Solche Modifikationen entstehen regelmäßig durch Pflege, Änderung und Korrektur von Software.

„Der Regressionstest ist ein erneuter Test eines bereits getesteten Programms nach dessen Modifikation mit dem Ziel nachzuweisen, dass durch die vorgenommenen Änderungen keine neuen Defekte eingebaut oder durch bisher maskierte Fehlerzustände freigelegt wurden.“¹

Visuelle Regressionstests zielen nicht mehr auf den Programmcode als Testgegenstand ab, sondern vielmehr auf das Endergebnis. Von einer Website, die den Status „Freigabe“ erreicht hat, werden Bildschirmfotos (Screenshots) angefertigt, die als Grundlage der Regressionstests dienen. Dieser erste Satz von Bildschirmfotos, der im Folgenden Baseline genannt wird, wird nach einer Änderung des Quellcodes des zu testenden Programms mit einem zweiten Satz Screenshots verglichen. Dieser zweite Satz wird Testline genannt. Es werden, automatisiert, die Unterschiede zwischen den beiden Bildern farblich in einem Differenzbild hervorgehoben und gespeichert. Der Entwickler erhält so direkt nach dem Durchführen des Tests eine Rückmeldung darüber, ob Unterschiede zwischen Base- und Testline-Bild aufgetreten sind und kann auf diese Weise Fehler schneller eingrenzen.

3.2. Der innerbetriebliche Anwendungsfall

Von einem Kunden meines Unternehmens erhielten wir den Auftrag, die Version eines Content Management Systems (kurz CMS), auf dem die von uns entwickelte Website² bisher

¹ Spillner/Linz: „Basiswissen Softwaretest“, Heidelberg 2012, S.77 (im Folgenden mit Spiller/Linz: Basiswissen abgekürzt).

² Aus Gründen der Geheimhaltung darf ich in dieser Arbeit keine Namen nennen oder Bildschirmfotos des Projekts zeigen. Es handelt sich bei dem Kunden um einen großen Automobilhersteller. Die Plattform, die es im Vorlauf des Projekts zu ändern galt, stellt die Multimediasysteme des Kunden vor und ordnet sie Fahrzeugen zu. Es ist eine Art Kompatibilitätsseite.

basierte, zu aktualisieren. Bei dem CMS handelt es sich um das Magnolia CMS³ das auf der Programmiersprache Java basiert.

Die Webseite bestand aus einer Vielzahl von einzelnen Webpages (ca. 2500), die einzeln über URLs erreichbar sind. Da das Projekt im Laufe der Zeit vielfache, zum Teil nicht optimale Anpassungen und Funktionalitätserweiterungen erfuhr, nahm die Performanz der Webseite nach und nach stark ab. Zudem wurde die Content-Pflege aufgrund gewachsener Pflegeprozesse sehr aufwändig, da derselbe Content an mehreren Stellen gepflegt werden musste. Da der Kunde bei zukünftigen Pflegeaufgaben Ressourcen sparen wollte, war der Auftrag, den mein Unternehmen umsetzen sollte, nur auf das Backend bezogen. Das bedeutete, dass die Darstellung der Seite nach dem Upgrade auf die neueste Magnolia Version exakt der Darstellung der Seite entsprechen sollte. Es galt, aufgrund fehlenden Budgets die Frontenprogrammierung nicht neu umzusetzen und im Anschluß wieder mit dem neuen Backend zusammenzuführen.

Dieser Auftrag stellt gewissermaßen ein Musterbeispiel für einen visuellen Regressionstest dar. Die Baseline, also Screenshots von der Darstellung des alten Systems, konnten als ein geprüfter und abgenommener Stand gelten, gegen den die Testline, also die Screenshots von der Darstellung des neuen Systems, gehalten werden konnte. Wichen diese voneinander ab, musste der Entwickler nachbessern.

Der Workflow meines Unternehmens beinhaltet zwar eine eigene Quality Assurance Abteilung, das selbstentwickelte Programm soll dem Entwickler aber zusätzlich die Möglichkeit geben, seine Ergebnisse direkt zu überprüfen, nachdem er Veränderungen am Code vorgenommen hat, also noch bevor das Projekt an die QA übergeben wird⁴. Selbstverständlich kann jedoch auch die QA das Tool nutzen, um die eigenen Testmethoden zu unterstützen.

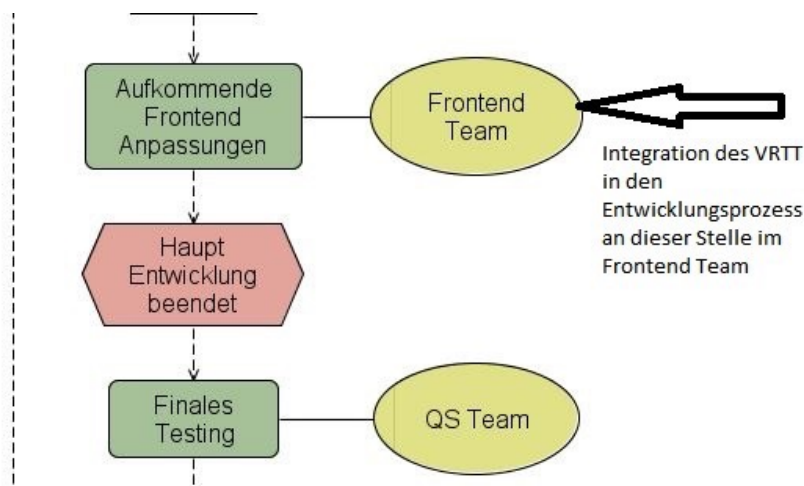


Abbildung 1 - mögliche Integration in den Workflow - Quelle A.Hems Mitarbeiter t8y.com - Stand 25.05.2016.

³ <https://www.magnolia-cms.com/> , Stand 30.05.2016.

⁴ Vergleiche dazu die Darstellung des kompletten Workflows. Diese befindet sich im Abbildungsverzeichnis, Abbildung 13.

3.3. Das Projekt „visuelle Regressionstests Tool⁵“

Mein Programm soll die folgenden Grundfunktionalitäten gewährleisten:

- Es kann Screenshots einer gegebenen URL-Liste erzeugen.
- Es kann Base- und Testline einander zuordnen.
- Es kann die Bilder miteinander auf Abweichungen vergleichen.
- Es gibt dem Entwickler/Tester Auskunft, welche Bilder voneinander abweichen und speichert diese zur Ansicht.
- Es ist einfach in der Anwendung und über das Command Line Interface zu bedienen.

4. Überblick über die verwendeten Technologien

Im Folgenden werde ich die Technologien und Pakete erläutern, die ich benutzt habe, um mein Programm anzufertigen.

4.1. Basis und Basismodule

4.1.1. Node

Die Basis der Anwendung ist node.js⁶. Node ist eine serverseitige Plattform zum Betrieb von Netzwerkanwendungen. Insbesondere lassen sich Webserver damit realisieren. Node basiert auf der Javascript-Laufzeitumgebung „V8“, die ursprünglich für Google Chrome entwickelt wurde. Das Besondere an Node ist dessen Modularität. Es enthält einige Module, die direkt in das Binärpaket kompiliert wurden; zum Beispiel Module für asynchronen Zugriff auf das Netzwerk, Dateisystem, Puffer und Zeitgeber, um nur einige Features zu nennen.

4.1.2. npm

Zusätzlich können über den Node Paket Manager⁷ (npm) und das Command Line Interface (CLI⁸) zusätzliche Module integriert werden, die spezielle Aufgaben erfüllen. Der npm sorgt, unter Berücksichtigung von Abhängigkeiten, für Installation, Aktualisierung und Kompilation von Binärpaketen.

4.1.3. Phantomjs

Phantomjs⁹ ist ein Headless Browser, also ein Webbrowser ohne ein graphisches Benutzer Interface¹⁰ und eignet sich sehr gut dazu, mit Webseiten zu interagieren und das Ergebnis

⁵ Im Folgenden mit VRTT abgekürzt.

⁶ <https://nodejs.org/en/>, Stand 23.05.2016. (im Folgenden stets mit Node abgekürzt).

⁷ <https://www.npmjs.com/>, Stand 23.05.2016.

⁸ https://en.wikipedia.org/wiki/Command-line_interface, Stand 27.05.2016.

⁹ <http://phantomjs.org/>, Stand 23.05.2016.

¹⁰ https://en.wikipedia.org/wiki/Headless_browser, (Selbstübersetzung des Verfassers), Stand 23.05.2016.

der Interaktion an andere Programme weiterzuleiten. Das Einsatzgebiet von Headless Browsern ist vielfältig. Wesentlich ist ihnen, dass sie HTML genauso rendern und interpretieren wie ein normaler Browser, dass sie CSS und Javascript Anweisungen ausführen können und zudem die Möglichkeit bieten, User-Interaktionen mit Elementen der Webseite zu skripten.

4.1.4. Graphicsmagick

Graphicsmagick¹¹ ist ein freies Softwarepaket aus dem Jahr 2002 zur Erstellung und Bearbeitung von Rastergrafiken. Es kann über 100 der üblichen Bildformate lesen, verändern und schreiben. Das Tool bietet die Möglichkeit, einen pixelgenauen Vergleich zwischen zwei Bildern vorzunehmen und die Unterschiede in einem Differenzbild zu markieren.

4.2. Die Node Pakete im Überblick

Oben wurde schon kurz erläutert, dass Node Pakete, je nach Bedarf, einfach hinzuzufügen sind. Hier wird kurz jedes einzelne genutzte Paket im Zusammenwirken für das Programm erläutert.

bash-color, Version 0.0.4

Dieses Paket ermöglicht es, den Output im Terminal farblich zu gestalten. Das Paket selbst bietet keine Kernfunktion und ist trägt zur Verbesserung der Darstellung bei.

figlet, Version 1.1.2

Dieses Paket kommt in der Datei `help.js` zum Einsatz und ermöglicht es, Text in ASCII Code in der Konsole darzustellen. Das Paket selbst bietet keine Kernfunktion und ist lediglich eine Verbesserung der Darstellung.

gm, Version 1.22.0

Dieses Paket kommt in der selbstgeschriebenen Datei `compare.js` zum Einsatz. Indem es dort importiert wird, können über das Objekt `gm` die Funktionen der Graphicsmagick Library in vollem Umfang in der Nodeumgebung genutzt werden.

line-by-line, Version 0.1.4

Dieses Paket kommt in der Datei `screens.js` zum Einsatz. Es gestattet, über das importierte Objekt `Textfiles` einzulesen. Das Modul arbeitet Event-basiert¹², da es auf Ereignisse lauscht und, erst wenn diese geschehen (abgefangen werden), die Funktionskörper ausgeführt werden.

phantom, Version 2.1.1

Dieses Paket kommt in der Datei `screens.js` zum Einsatz. Es ist nicht zu verwechseln mit dem Phantom Binary, das dem System global zur Verfügung stehen muss. Normalerweise müssten Befehle, die der Headless Browser ausführen soll, im CLI mit dem Keyword `phantomjs` beginnen. Wenn man also ein Script im Phantom Kontext ausführen wollte, wäre die Syntax

¹¹ <http://www.graphicsmagick.org/>, Stand 23.05.2016.

¹² https://en.wikipedia.org/wiki/Event-driven_programming, Stand 27.05.2016.

```
phantomjs <DeinScript>
```

erforderlich. Das Phantom Paket umschließt Phantomjs und ermöglicht es, phantomjs Prozesse im Kontext von Node auszuführen.

walk 2.3.9

Dieses Paket wird in der Datei walker.js verwendet. Es gestattet einen vereinfachten Zugriff auf das Filesystem, indem es Event-basiert auf Ereignisse wie *file.ready()* lauscht. Solange also ein File behandelt wird, bleibt der Funktionskörper in diesem Scope. Es wäre hier auch möglich gewesen, mit dem nativen Node Modul *fs* (Filesystem) zu arbeiten, allerdings hätte man die Logik selbst entwickeln müssen.

4.3. Die konkrete Anwendung der Technologien

Ich nutze für mein Projekt die aktuelle Node Version¹³, die ECMA-Script 6 Syntax Features unterstützt. Da das Programm auf einem Microsoft Windows System entwickelt wurde, war ich, wie sich während der Entwicklung herausstellen sollte, zu dieser Technologie genötigt. Die LTS, also die latest stable Version, von Node (4.4.4) konnte ich nicht nutzen, da hier die Module zusätzliche Abhängigkeiten ineinander schachteln. So kommt es mitunter zu sehr langen Pfadangaben, die von einem NTFS System, das eine maximale Länge von 255 Zeichen vorgibt, nicht mehr lesbar sind¹⁴. Die Version 6.2.0. von Node organisiert die Abhängigkeiten von Modulen auf einer gleichen Hierarchieebene und verhindert so die Problematik.

Mittels npm erstelle ich eine Datei namens package.json, die Angaben über alle Pakete enthält, die das Programm VRTT benötigt. Nachdem ein anderer Entwickler das Projekt bei Github auscheckt und in das Verzeichnis wechselt, kann er mittels des Befehls

```
npm install
```

den Download aller Pakete, sowie den Installationsprozess starten.

Mit phantomjs simuliere ich (vorerst¹⁵) keine Nutzerinteraktion mit der Seite, sondern nutze es für einen Seitenaufruf der angegebenen URLs, um dann mit der *render* Methode einen Screenshot zu machen. Zu einem späteren Zeitpunkt ist es denkbar, mittels phantomjs, Zustände der Testseite zu verändern, bzw. Userinteraktionen mit der Seite zu simulieren, damit beispielsweise *hover* Zustände von HTML Elementen erfasst werden und an die Bildvergleichsfunktionen übergeben werden können.

Das *gm*¹⁶ (Graphicsmagick) Modul für Node stellt die Kernfunktionalität meiner Anwendung bereit und wurde von Aaron Heckmann unter einer MIT Lizenz¹⁷ bei Github eingestellt. Hauptsächlich nutzt das Programm VRTT die *compare* Funktion und es mussten sogar

¹³ Node Version 6.2.0, Stand 23.05.2016.

¹⁴ <https://en.wikipedia.org/wiki/Filename> - hier NTFS -, Stand 23.05.2016.

¹⁵ Vergleiche dazu das Kapitel 5 - theoretische Überlegungen.

¹⁶ <https://github.com/aheckmann/gm>, Stand 23.05.2016.

¹⁷ <https://de.wikipedia.org/wiki/MIT-Lizenz>, Stand 30.05.2016.

einige Anpassungen an dem Modul vorgenommen werden, auf die ich im weiteren Verlauf noch vertiefend eingehen werde¹⁸.

4.4. Ein paar Worte zur Architektur

Damit das Programm VRTT läuft, müssen auf dem System wenigstens das Programm Graphicsmagick sowie eine Node Umgebung installiert sein. Sobald das Projekt aus Github ausgecheckt ist, kann der Entwickler es mit dem npm, also auch dieser ist eine Voraussetzung, lokal bei sich zum Laufen bringen. Der Nodepaketmanager übernimmt dann die Installation und die Kontrolle der Abhängigkeiten der relevanten Pakete.

Die Architektur entspricht am ehesten dem Prinzip „Pipes and Filter“¹⁹. Das System ist also dazu in der Lage, Datenströme in mehreren unabhängigen Einheiten zu bearbeiten. Über die Eingabe der Parameter wird im CLI der Datenfluss angestoßen. Je nachdem, welche Eingabe der Nutzer getroffen hat, wird in der *start.js* die Funktion aufgerufen, die hinter dem Befehl steht. Diese aufgerufene Funktion ist allerdings nur als Wrapper zu verstehen, da sie mehrere Funktionen nutzt, um die ausgewählte Funktionalität umzusetzen. So wird beispielsweise für das Erzeugen der Screenshots nicht nur die Instanz von Phantomjs benötigt, sondern auch eine Logik, nach der die Screenshots auf dem Dateisystem abgespeichert werden, und eine Logik die Eingabe der URLs zu verarbeiten.

Da ich mit meinem Projekt ein Prototyping betrieben habe, ist die Wahl der Systemarchitektur noch nicht final entschieden. Mein Entwicklungsprozess war erfahrungsgesteuert, da es mir zunächst nur um den Nachweis der Praktikabilität der Idee ging. Es gelang mir, meine Idee mit den Prinzipien von Event-basierten Ereignissen, Callbacks, ECMAScript²⁰ Imports und Promises²¹ in einer Art Event-Chain²² umzusetzen. Das Hindernis dieser Programmierung ist, dass man sehr schnell den Überblick über den Ausgang der verschiedenen Events verlieren kann. Wenn also erwogen würde, dass Programm VRTT über einen bloßen Prototyp hinaus zu entwickeln, mit dem Ziel, ihn produktiv einzusetzen, sollte man vor der Entwicklung die Entscheidung zu einer Architektur treffen. Dies würde es fremden Entwicklern erleichtern, das Programm, das eben noch der Struktur für das Testprojekt entspricht, zu erweitern und für Sonderfälle anzupassen.

4.5. Der Befehlssatz des VRTT

Zum derzeitigen Stand der Entwicklung kann der Nutzer des Programms VRTT die folgenden Befehle aufrufen, hinter denen die jeweils beschriebene Funktionalität steckt.

Führende Zeile	Befehl	Funktionalität
----------------	--------	----------------

¹⁸ Vergleiche dazu weiterführend Kapitel 4.4. - Anpassungen.

¹⁹ https://de.wikipedia.org/wiki/Pipes_und_Filter, Stand 30.05.2016.

²⁰ <http://es6-features.org> und vertiefend dazu <http://es6-features.org/#ValueExportImport>, Stand 30.05.2016.

²¹ [https://de.wikipedia.org/wiki/Future_\(Programmierung\)](https://de.wikipedia.org/wiki/Future_(Programmierung)), Stand 30.05.2016.

²² https://en.wikipedia.org/wiki/Event_chain_methodology, Stand 30.05.2016.

node start	createBaseline	Erzeugt Screenshots auf Grundlage des Files <i>baseline.txt</i> , das im Projektverzeichnis liegen muss, und speichert diese im Ordner <i>/baseline</i> .
node start	createTestline	Erzeugt Screenshots auf Grundlage des Files <i>testline.txt</i> , das im Projektverzeichnis liegen muss, und speichert diese im Ordner <i>/testline</i> .
node start	fullReport	Gibt im Terminal den rechnerischen Bericht über den Bildvergleich von Test- und Baselinebild aus.
node start	compareImage	Prüft zunächst auf rechnerische Abweichungen zwischen den Bildern und wenn diese besteht, wird ein Differenzbild im Ordner <i>/results</i> abgelegt.
node start	help	Der Nutzer kann sich einen Überblick über den Befehlssatz verschaffen.

Dabei muss der Entwickler lediglich das Terminal CLI geöffnet haben und im Verzeichnis des Projekts sein

5. Das Programm – VRTT

5.1. Die Projektstruktur

Im Folgenden werde ich kurz die Projektstruktur erläutern.

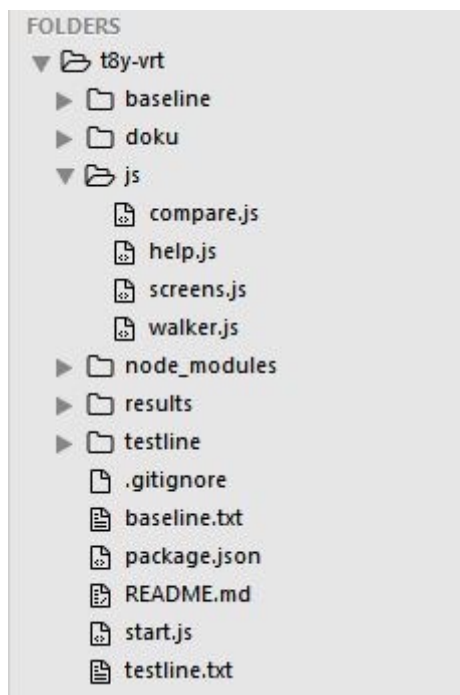
Abbildung 2 - Projektstruktur - selbst erstellt 24.05.2016.

Im Ordner */baseline* werden die Screenshots gespeichert, die zu Beginn aufgenommen wurden. Diese dienen als Grundlage des Tests.

Der Ordner */testline* enthält alle Screenshots, die von dem neuen Zustand der Seite gemacht wurden. Sie sind das Vergleichsmuster, das auf Abweichungen überprüft wird.

Treten Abweichungen auf, landen die Screenshots mit den farblich markierten Bereichen im Ordner */results*.

Der Ordner */js* speichert alle Javascript Dateien, die ich selbst geschrieben habe und die das Programm benötigt. Die Datei *start.js* ist der „Einstiegspunkt“ des Programms. Über diesen werden alle anderen Funktionalitäten, die implementiert wurden, erreicht. Die Datei



compare.js enthält die Vergleichsfunktion; *help.js* die Angaben zur Hilfe; das File *screen.js* ist wird benötigt, damit die Screenshots gerendert werden und die Datei *walker.js* organisiert Zugriff auf die Speicherordner. Zu einem späteren Zeitpunkt wurde diesem Ordner noch eine Datei namens *gm_compare.js* hinzugefügt²³. Alle Dateien exportieren Funktionen, die dann im Einstiegspunkt importiert werden.

Der build Prozess legt alle Module und auch abhängige Module, die in der *package.json* definiert sind, in den Ordner */node_modules* ab.

Die Dateien *baseline.txt* und *testline.txt* liegen direkt im Root-Verzeichnis. In ihnen ist die URL Liste hinterlegt die es für die jeweilige line festzuhalten (zu fotografieren) gilt. Im Fall des Beispielprojekts entsprechen diese einander. Für das Projekt, das mein Unternehmen für den Kunden umsetzen sollte, war das nicht der Fall, da mit der neuen Implementation Unterseiten über andere URLs erreichbar gewesen sind.

Das mappen²⁴, also Gegenüberstellen der URLs von alter und neuer Seite ist zum jetzigen Stand des Projekts VRTT noch unumgänglich und enthält derzeit eine potentielle Fehlerquelle.

Die Datei *.gitignore* gibt an, welche Dateien und Unterverzeichnisse beim Pushen nach Github nicht übertragen werden sollen. Normalerweise ist der Ordner *node_modules* in diesem File enthalten. Aber die Ordner, die die erstellten Bilder enthalten, sind angegeben, da es überflüssig wäre, konkrete Testergebnisse in das Repository einzuchecken.

5.2. Anwendung am Beispielprojekt

Da ich, wie schon eingangs erwähnt, keine Screenshots des ursprünglichen Projekts zeigen darf, habe ich ein Beispielprojekt²⁵ lokal mit XAMPP²⁶ aufgesetzt. Die Unterseiten des Beispielprojekts sind über 5 URLs erreichbar und ich füge diese meiner *baseline.txt* hinzu.

In meinem Terminal wechsele ich in das Projektverzeichnis *t8y-vrt* und nachdem die *baseline.txt* in den root kopiert wurde, kann ich mit der durch Eingabe des Befehls `node start createBaseline` den Prozess anstoßen, der die Baseline Screenshots erstellt.

```

193 .sidebar h2
194 { padding: 5px 0 0 10px;
195   font: normal 140% Arial, Helvetica, sans-serif;
196   height: 30px;
197   text-shadow: 0px -1px 0px #000;
198   color: #fff;
199   background: #0043A8;
200   background: -moz-linear-gradient(#43A9FF, #0043A8);
201   background: -o-linear-gradient(#43A9FF, #0043A8);
202   background: -webkit-linear-gradient(#43A9FF, #0043A8);
203   /*border-radius: 15px 15px 15px 15px;
204   -moz-border-radius: 15px 15px 15px 15px;
205   -webkit-border: 15px 15px 15px 15px;
206   -webkit-box-shadow: rgba(0, 0, 0, 0.5) 0px 0px 5px;
207   -moz-box-shadow: rgba(0, 0, 0, 0.5) 0px 0px 5px;
208   box-shadow: rgba(0, 0, 0, 0.5) 0px 0px 5px;"}

```

Diese befinden sich danach im Ordner */baseline*, und um zu simulieren, dass eine Veränderung am Quellcode der zu testenden Seite stattgefunden hat, kommentiere ich aus dem *style.css* File der Webseite die abgerundeten Ecken aus.

²³ Die Begründung zu dieser Entscheidung findet sich in Kapitel 4.4 - Anpassungen.

²⁴ <https://de.wikipedia.org/wiki/Mapping>, Stand 28.05.2016.

²⁵ <http://www.opendesigns.org/design/blue-skies/>, Stand 25.05.2016.

²⁶ <https://www.apachefriends.org/de/index.html>, Stand 30.05.2016.

Nun starte ich mit dem Befehl `node start createTestline` die Erstellung der Testline. Diese werden im Ordner `/testline` gespeichert.

Durch `node start compareImage` startet der Bildvergleichsprozess, über dessen Ergebnisse der Entwickler im Terminal informiert wird. Im Ordner `/results` liegen nun alle Bilder, die eine Differenz aufweisen.

Nachfolgend ist ein Ausschnitt aus den Ergebnissen zu sehen. Die Unterschiede in der Darstellung der Buttons werden im Resultatbild gelblich hervorgehoben²⁷.

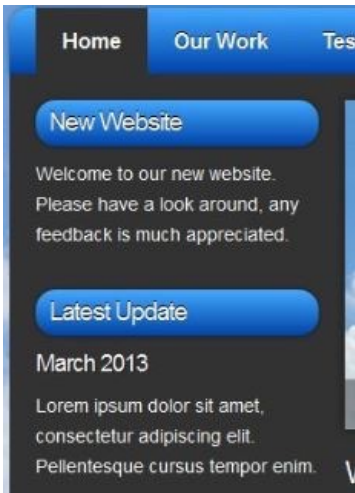
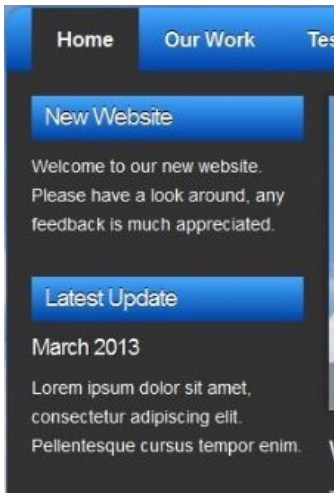

Baseline	Testline	Resultat
		

Abbildung 4 - direkter Bildvergleich - Tabelle und Screenshots selbst erstellt - Stand 27.05.2016.

Mit `node start fullReport` erhält der Entwickler im Terminal CLI die rechnerische Abweichung der Bilder, ohne das Erzeugen des Differenzbildes anzustoßen, und kann sich so einen ersten Überblick verschaffen.

Um eine Übersicht über die Möglichkeiten von VRTT zu erhalten, habe ich eine Hilfe hinzugefügt, die mittels `node start help` erreichbar ist.

5.3. Ausgewählte Quellcodebeispiele

Im Folgenden werde ich eine Funktion, die ich für das Programm geschrieben habe, komplett erläutern.

Im Ordner `js` befindet sich die Datei `compare.js`. Diese Datei importiert das `gm` Modul mit folgender Syntax und stellt es in der benannten Variable bereit.

```
var gm = require('gm');
```

Mit diesem Objekt kann ich meine Vergleichsfunktion `createDifferImg` ausführen.

²⁷ Die kompletten Bilder sind im Abbildungsverzeichnis zu betrachten - Abbildungen 10-12.

```

35 function createDifferImg(baseImg, newImg, count){
36     var resultPathMade = '' + resPath + resName + count + resEnding;
37     var options = {
38         file: resultPathMade,
39         highlightColor: 'yellow'
40         //metric: 'psnr' //needs to be arg to pass, rewrite gm's compare.js
41     }
42     gm.compare(baseImg, newImg, function(err, isEqual, equality, raw){
43         if(err){
44             throw err;
45         }
46         // !=0 instead trying to approach the float(->discuss in theory)
47         if(equality > 0.01){
48             console.log('from differImg');
49             gm.compare(baseImg, newImg, options, function(err){
50                 if(err){
51                     throw err;
52                 }
53                 console.log(color.red('es trat eine Abweichung auf zwischen '+baseImg+' und '+newImg));
54                 console.log(color.yellow('Das Differenzbild liegt in: ./results'));
55             });
56         }
57         else{
58             console.log(color.green('es gab keine Abweichung zwischen '+baseImg+' und '+newImg));
59         }
60     })
61 };
62
63
64 exports.createDifferImg = createDifferImg;

```

Abbildung 5 - function createDifferImg - selbst erstellt 25.05.2016.

Der Funktion werden drei Argumente übergeben; das Baseline-Bild, das Testline-Bild und eine Hilfsvariable namens *count*. Die Variable *resultPathMade* wird zusammengesetzt²⁸ aus dem Zielpfad, Dateiname, dem übergebenen *count* und einer Dateiendung. Der Entwickler kann über diese Variablen den Output des Vergleichsprozesses bestimmen. Derzeit würde für *count* = 0 der Dateiname *diffImg0.jpg* lauten und das File wird in den Ordner */results* gespeichert.

In den Zeilen 42-45 benutzt man nun das *gm* Objekt dazu, die *compare* Methode des *gm* Node-Moduls aufzurufen. Dieser übergebe ich Base- und Testline-bild und eine Callbackfunktion²⁹, die wiederum mehrere Argumente enthält. Diese sind *err*, damit ich Fehler abfangen kann, *isEqual*, das einen Booleanwert zurückgibt, ob die Bilder identisch sind, *equality*, das die mittlere quadratische Abweichung (MSE – Mean square Error³⁰) angibt und *raw*, das den MSE einzeln nach Farbkanal RGB aufschlüsselt und ausgeben kann.

Derzeit nutze ich, beginnend im Quellcode Zeile 46-60, den Wert *equality* aus dem Callback der vorangegangenen Funktion, um festzustellen, ob es eine rechnerische Abweichung zwischen den beiden Bildern gibt³¹. Trifft dieser Fall zu, dann rufe ich die Vergleichsmethode *compare* des *gm* Objekts auf und übergebe zusätzlich noch die Options aus Zeile 37-41. Dadurch, dass die *compare* Methode über die Options mehr Argumente erhält, als den übergebenen Pfadnamen, wird ein Differenzbild erzeugt.

²⁸ Vgl. Quellcode Zeile 36.

²⁹ https://de.wikipedia.org/wiki/R%C3%BCckruf_funktion, Stand 25.05.2016.

³⁰ https://de.wikipedia.org/wiki/Mittlere_quadratische_Abweichung, Stand 25.05.2016.

³¹ Dieses Verfahren werde ich in einem späteren Kapitel noch genauer beleuchten, da sich die Frage stellt, wie aussagekräftig dieser Wert ist. Die Alternative, die darin besteht, nicht zu versuchen, sich einem Fehlerwert anzunähern, sondern vielmehr die Aussage zu negieren, in der Pseudocode Form `wenn equality isNot = 0;`, wird schon im Kommentar angedacht, ist aber aus Zeitgründen nicht weiter umgesetzt.

Allgemeiner formuliert bedeutet das: Wird eine rechnerische Abweichung durch das Bilden des mean-square-errors festgestellt, dann wird ein Differenzbild erzeugt und entsprechend abgelegt.

Neben den Argumenten `Base-`, `Testlinebild` und `Options` nutze ich erneut eine Callbackfunktion mit dem Argument `err`, um eventuell auftretende Fehler abzufangen, wie in Zeile 50-52 zu sehen.

Im *equality* else Fall, also dem Fall, dass der MSE bei genau 0 liegt, gebe ich dem Entwickler über die Konsole eine Benachrichtigung darüber, dass beim Vergleich keine Abweichungen aufgetreten sind und verzichte auf das Generieren eines Differenzbildes.

In Zeile 64 exportiere ich die Funktion an die Umgebung und kann sie dann im File *start.js*, also dem Einstiegspunkt nutzen, wenn man in dieser:

1. das File *compare.js* mit `var compare = require('./compare.js')` importiert.
2. die Funktion *createDifferImg* des Objekts aus 1. aufgerufen wird und die korrekten Argumente übergeben werden. Also:

```
compare.compareImgFullReport(path.join(val.base),  
path.join(val.test));
```

5.4. Erforderliche Anpassungen des gm Moduls

Im Laufe der Entwicklung stieß ich zunehmend an die Grenzen der Node Implementierung von *gm*, so dass es zu einem bestimmten Zeitpunkt unumgänglich war, den Quellcode anzupassen, damit er meinen Anforderungen entsprach.

Die meisten Probleme hatte ich mit der Datei *compare.js*³² aus Heckmanns *gm* Implementation. Zunächst stellte sich heraus, dass das Umstellen der Farbe, mit der die Unterschiede im Differenzbild kenntlich gemacht werden, nicht möglich ist.³³ Es ist also notwendig gewesen, eine if-Abfrage auszukommentieren, die diesen Fehler verursacht hatte.

Die Anpassung eines *node_modules* ist ein kritischer Vorgang. Durch die Angabe der Abhängigkeiten in der *package.json* wird definiert, welche Module für den Build benötigt werden. Jeder Entwickler, der das Projekt erstmalig bei sich lauffähig macht, wird den Fehler über dieses Node Modul wieder importieren, wenn er ein `npm install` ausführt. Zuvor war ich versucht, die *node_module* bei Github einzuchecken, um das Problem zu umgehen. Indem ich die Änderungen mit in das Repository committed hätte, wäre der Fehler dort auskommentiert gewesen. Allerdings hätte ich so die Codebase deutlich und unnötig erweitert.

Eine weitere Option, das Problem zu umgehen, wäre, das Repository auf Github zu forken, den Fork anzupassen und in der *package.json* auf diese zu zeigen. Auch das schien mir den erhöhten Aufwand nicht zu rechtfertigen, zumal ich so Anpassungen am Original Repo immer wieder auf meinen Fork übertragen müsste, um aktuell zu bleiben.

³² Ich bitte zu beachten, dass es sich trotz der Namensgleichheit der Datei *compare.js* um zwei unterschiedliche Dinge handelt. Einmal nutze ich meine selbst geschriebene Datei, beziehe mich mit dieser Fußnote aber auf das File *compare.js* im Ordner *node_modules/gm/lib/*.

³³ Vgl. dazu <https://github.com/aheckmann/gm/issues/480>, Stand 25.05.2016.

Ich wählte daher den folgenden Weg:

Ich kopierte zunächst den kompletten Inhalt der Datei *compare.js* in eine andere Datei namens *gm_compare.js* und legte diese in mein Verzeichnis */js*. Dort nahm ich die notwendigen Anpassungen vor. In meiner selbstgeschriebenen *compare.js* überschreibe ich die *gm.compare* Methode, indem ich meine geänderte an dieser Stelle importiere. Syntaktisch gestaltet sich das wie folgt:

```
var gm = require('gm');  
gm.compare = require('./gm_compare.js')(gm.prototype);
```

Nun kann ich das compare File nach Belieben verändern, anpassen und erweitern.

Einen Ausblick auf eine denkbare und sinnvolle Erweiterung möchte ich an dieser Stelle noch kurz in Ansätzen ausführen.

Momentan arbeitet A. Heckmanns Implementation zum Feststellen der Unterschiede zwischen Bildern mit der Metrik MSE, und das per Default. Es ist also in der Node Umgebung nicht möglich, eine andere Metrik zu wählen.

```
29 var args = ['-metric', 'mse', orig, compareTo] // mean square error - mittlere quadratische abweichung
```

Abbildung 6 - Default Metrics - selbst erstellt 25.05.2016.

GraphicsMagik bietet allerdings die Möglichkeit, Bilder auch mit anderen Metriken zu untersuchen, etwa der Peak-Signal-to-Noise-Ratio (PSNR³⁴) oder dem mean-absolute-error (MAE³⁵). An dieser Stelle sollte man es möglich machen, Bilder auch mit diesen Metriken zu vergleichen, um im Ergebnis genauere Aussagen über die Art der Unterschiede treffen zu können.

Programmiertechnisch habe ich das in meinem File *gm_compare.js* so gelöst, dass ich ein neues Feld in *options* erzeuge, dem diese als Value zu *metric* übergeben werden können.

```
35 if (typeof options === 'object') {  
36   if(options.metric){  
37     args[1] = options.metric;  
38   }  
39 }
```

Abbildung 7 - metric in options erzeugt - selbst erstellt 25.05.2016.

³⁴ https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio, Stand 25.05.2016.

³⁵ https://en.wikipedia.org/wiki/Mean_absolute_error, Stand 25.05.2016.

6. Theoretische Überlegungen zu visuellen Regressionstests

Während der Anwendung des Programms auf der Website unseres Kunden, dem Vorher-Nachher-Vergleich, stellte sich heraus, dass VRTT nur unter bestimmten Bedingungen funktionieren kann.

Die alte Webprogrammierung des Kundenprojekts bildete noch jeden erreichbaren Zustand der Seite unter einer URL ab. Anfangs versteckte DOM-Elemente, wie Container, die sich ausklappten, waren über Parameter, die der URL übergeben worden, erreichbar³⁶. Mit dem Umbau der Seite wurde dieses Verfahren zugunsten moderner Webtechnologien aufgegeben. Moderne Webseiten verändern mit Javascript die Eigenschaften der DOM-Elemente zur Laufzeit. Nach Interaktion mit dem (beispielhaften) Tabcontainer ändert sich die URL nicht mehr, der Container zeigt seinen Inhalt an. Da das von außen, über die URL nicht mehr abbildbar ist, kann auch das Verfahren das hinter VRTT steht jedoch nicht mehr greifen. Da visuelle Regressionstests gegen das fertig gerenderte HTML prüfen, aber nicht mehr alle Darstellungen über URLs erreichbar sind, scheitert das Verfahren an dieser Stelle. Wollte man es dennoch umsetzen, wäre es denkbar, sich an dem Page Object Model³⁷ zu orientieren, die Userinteraktionen für die einzelnen Elemente mit dem Headless Browser simuliert zu skripten und dann die Zustände mit einem Screenshot zu erfassen. Allerdings stiege damit der Aufwand des Testverfahrens um ein Vielfaches.

Ein weiteres Problem bereitete die rechnerische Prüfung auf Unterschiede zwischen den Bildern, die von der compare Methode als MSE, mittlere quadratische Abweichung, dargestellt werden. Dazu eine kleine Rechnung:

Auf einer fiktiven Webseite mit insgesamt 100 Pixeln sind 10 Pixel abweichend. Die Abweichung ist, relativ betrachtet, sehr groß, während 10 Pixel Abweichung bei einer 1.000.000 Pixel großen Webseite marginal ist. Zu Beginn war ich noch versucht, einen zulässigen MSE so einzugrenzen, dass ich keinen falsch-positiven³⁸ Fehler bei der Messung erzeuge, aber gleichzeitig zuverlässig aussagen kann, dass das eine Bild vom anderen abweicht³⁹. Ich habe dazu zwei Möglichkeiten gefunden:

1. Eine Hochrechnung des Fehlers mit der Bildgröße um einen absoluten Fehler zu erhalten, also in der Form $(\text{Bild[Breite*Höhe]}) * \text{MSE} = \text{absoluter Fehler}$. An diesem Wert könnte ich Erfahrungen sammeln und versuchen ihn so einzugrenzen, dass ich Unterschiede zwischen Bildern, aber keine falsch-positiven erhalte.
2. Ich negiere die Aussage, indem ich behaupte, dass die Bilder stets dann unterschiedlich sind, wenn ihr MSE nicht gleich 0 ist. Dieses Verfahren birgt jedoch ein erhöhtes falsch-positiv Risiko, da auch eine ein-Pixelabweichung einen Fehler wirft.

³⁶ Die Parametrisierung des GET in kann in der Form `#tab/open/e-remote` oder `#tab/open/app-connect` der URL angehängt werden. Die so erzeugten Zustände der Seite sind dann selbstverständlich über die komplette URL zu erreichen.

³⁷ <http://martinfowler.com/bliki/PageObject.html>, Stand 27.05.2016.

³⁸ https://de.wikipedia.org/wiki/Fehler_1._Art, Stand 27.05.2016.

³⁹ Vergleiche dazu meine Beschreibung in Kapitel 4.3. - Quellcodebeispiele.

7. Die einfache Nutzwertanalyse

Im Folgenden wird der Versuch unternommen, eine Vergleichbarkeit zwischen 1. dem Service eines professionalisierten Anbieters für Bildvergleiche Applitools⁴⁰, 2. dem „Eyeball“-Test⁴¹, dem momentan in meinem Unternehmen am häufigsten eingesetzten Testverfahren, und 3. dem Programm VRTT zu schaffen. Dazu werde ich zunächst Bewertungskriterien festlegen, die mir aus dem Entwicklungs- und Testingalltag heraus sinnvoll erscheinen. Die Kriterien werden von mir gewichtet und dann Punkte auf einer Skala von 1-5 (1 = mangelhaft, 5 = sehr gut) vergeben, die mit der Gewichtung verrechnet werden, um Aussagen treffen zu können, ob die Entwicklung rentabel ist.

Applitools ist ein Dienstleister, der visuelle Regressionstests anbietet. Pro Monat kann man für 99 US\$ bis zu 1000 Bildvergleiche vornehmen. Der Anbieter arbeitet beim Bildvergleich mit Ebenen. Der Nutzer kann Bereiche definieren, die abweichend voneinander sein können, ohne dass der Test bricht. Denkbar sind hier Bannerwerbungen oder Sliderelemente, die ständig rotierenden Content darstellen. Weiterhin kann Applitools auch einen Tests auf mehrere Browser anwenden und die Darstellung der Webseite auf diesen vergleichen, ohne dabei wegen abweichender Schriftrender-Laufweiten Tests nicht zu bestehen.

Der interne Prozess bei t8y.com zum Testing von Webseiten ist bisher noch wenig automatisiert. Die fertig programmierte Webseite wird mit den Layouts verglichen. Teilweise sogar pixelgenau mit einem sogenannten Bildschirmlineal⁴². Das Verfahren ist sehr langwierig, da der Tester die Seiten im Browser händisch aufrufen muss. Außerdem (es liegt wohl in der Natur des Menschen, bei langwieriger gleichbleibender Arbeit gedanklich abzuschweifen) ist die Fehlerquote relativ hoch, da manuelles Testing ein langwieriger und konzentrationsfordernder Prozess ist. Trotzdem ist dieser Prozess für eine Übergabe an den Kunden unumgänglich.

Da ich das Programm VRTT in seiner Funktion und Wirkweise schon in vorangegangenen Kapiteln erläutert habe, nenne ich es hier nur.

Die Kriterien

- Der „Zeitliche Aufwand“ trifft Aussagen darüber, wie lange es dauert einen Testdurchlauf auszuführen.
- „Anzahl Test“ soll abbilden, wie viele Tests durchführbar sind.
- „Bedienung / Umsetzungsaufwand“ stellt dar, wie aufwändig die Umsetzung des Tests ist.
- „Direkte Kosten“ soll zeigen, welche Kosten durch die Umsetzung des Tests entstehen.

⁴⁰ <https://applitools.com/> Stand 28.05.2016.

⁴¹ <http://www.urbandictionary.com/define.php?term=eyeball%20test>, Stand 28.05.2016.

⁴² Exemplarisch <https://www.clarosoftware.com/screenruler>, Stand 28.05.2016.

- „Präzision“ soll Auskunft über die zu erwartende Genauigkeit des Tests geben.

Kriterien	Gewichtung	Applitoools		T8y.com QA		VRTT	
		Nutzen	gewichtet	Nutzen	gewichtet	Nutzen	gewichtet
Zeitlicher Aufwand	30%	4	1,2	1	0,3	5	1,5
Anzahl Tests	25%	2	0,5	4	0,8	4	1,0
Bedienung / Umsetzungsaufwand	10%	4	0,4	2	0,2	4	0,4
Direkte Kosten	15%	1	0,15	3	0,45	5	0,75
Präzision	20%	5	1,0	3	0,6	4	0,8
Ergebnis	100 %	3,25		2,35		4,45	

Abbildung 8 - einfache Nutzwertanalyse Matrix - selbst erstellt 28.05.2016.

Nach dieser Analyse spricht vieles dafür, die initialen Entwicklungskosten für ein solches Tool (VRTT) in Kauf zu nehmen, um in Zukunft schnell eine große Anzahl von Tests umsetzen zu können. Wenn man eine Seite unter qualitativen Gesichtspunkten testen möchte, ist der Einsatz von Applitoools in Erwägung zu ziehen. Die relativ hohen direkten Kosten, verbunden mit nur einer limitierten Anzahl an Tests, erfordern aber, den Einsatz gut abzuwägen.

In der Summe würde ich mich dafür aussprechen, die Tools nicht als miteinander konkurrierend zu betrachten, sondern sie vielmehr nebeneinander als Werkzeuge in den Testingprozess zu integrieren. Vor allem die Entwickler können von VRTT profitieren, da sie eine automatisierte Möglichkeit haben, ihre Arbeit vor der Übergabe an die QA-Abteilung selbst zu prüfen und auf diese Weise einen „fehlerfreieren“ Stand erreichen zu können.

8. Ausblick und Fazit

Die naheliegende Erweiterung des Programms ist den Befehlssatz der Testmethoden zu vergrößern. Vorstellbar ist hier zum Beispiel, die Highlight Farbe der Unterschiede oder auch die Metrik, mit der die Unterschiede gemessen werden, mit in den CLI-Aufruf zu übergeben. Node `start compareImage green mae` würde den Bildvergleich auf Grundlage der Metrik MAE anstoßen. Die Farbe, mit der die Abweichungen der Bilder voneinander gekennzeichnet würden, wäre grün.

Die momentane Syntax, um die Funktionen des VRTT aufzurufen, ist derzeit noch umständlich. Es muss stets `node <Dateiname> <Funktionsname>` als führendes Wort geschrieben werden. Sinnvoll wäre es, an dieser Stelle mit der „Shebang“⁴³ zu arbeiten, was allerdings nur im UNIX System ohne Workaround möglich ist. Eine typische Shebang-Zeile für Node sieht so aus: `#!/usr/bin/env node`. Diese Zeile weist das Betriebssystem an, diese Datei mit dem Interpreter Programm Node auszuführen. Der Vorteil ist, dass Dateien mit der Endung `*.js` so per Default mit Node ausgeführt werden, wenn die betreffende Zeile im Quellcode steht. Der Aufruf der relevanten Funktionen wäre verkürzt und würde sich so gestalten: `. <Dateiname> <Funktion>`. Ein Feature, das den Komfort der Anwendung steigern würde. Es ist auf einem Windows Betriebssystem nur über Umwege möglich, dasselbe zu erreichen. Man müsste ein `bin` Feld in der `package.json` nutzen und das Executable hier eintragen, damit npm einen `.cmd` Wrapper um das Script legt und dann den Wrapper aufruft⁴⁴. Dieses zu implementieren wäre ein zukünftiger Task.

Mit der Erfahrung aus der Entwicklung meines Tools würde ich zukünftig andere Zugänge zum Thema „visuelle Regressionstests“ wählen. Zunächst würde ich nicht mehr mit Graphicsmagick, sondern dem Fork Imagemagick arbeiten. Grund dafür ist, dass Imagemagick dazu instande ist, bessere Differenzbilder zu erzeugen, und zwar solche, auf denen nur die Abweichungen farblich kenntlich gemacht werden. Die Unterschiede sind so auf einen Blick schneller für das menschliche Auge zu erfassen. Die Gefahr die Graphicsmagick evoziert ist, dass Highlight Farbe identisch mit Hintergrundfarbe des Bildes, bzw. des Elements, ist, dass die Fehler hervorrief.

Für ein zukünftiges Unterfangen der gleichen Art würde ich außerdem darauf verzichten, den gm-Kern mit einem begrenzt-funktionalen und wenig dokumentierten Github Projekt zu ummanteln, von dessen Kern man im Grunde nur wenige Funktionen nutzt. Stattdessen würde ich die Funktionalität selbst schreiben, also eine eigene Implementation von Graphicsmagick bzw. dem vorzuziehenden Imagemagick in die Node-Environment umsetzen.

Abschließend:

Das Programm VRTT kann nur als Hilfsmittel dazu gelten, eine große Anzahl von Seiten und Unterseiten, repräsentiert durch Bilder, miteinander zu vergleichen. Dem Entwickler ist es möglich, sich einen schnellen Überblick über die Qualität seiner Anpassungen zu verschaffen. Ein ausgiebiges Testing der QA kann das Tool nicht ersetzen. Es reiht sich damit in die Werkzeuge des Testers ein.

⁴³ [https://en.wikipedia.org/wiki/Shebang_\(Unix\)](https://en.wikipedia.org/wiki/Shebang_(Unix)), Stand 25.05.2016.

⁴⁴ Weiterführend dazu <http://stackoverflow.com/questions/10396305/npm-package-bin-script-for-windows> Stand 25.05.2016.

9. Abbildungsverzeichnis

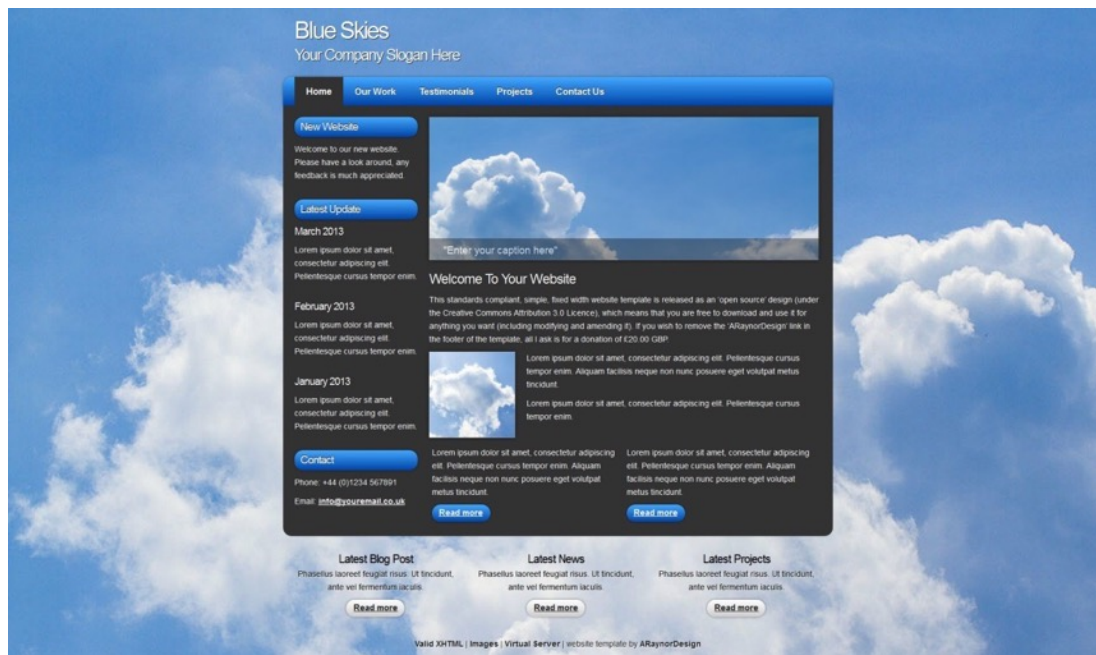


Abbildung 9 - BaselineBild des Testprojekts - selbst erstellt 28.05.2016.

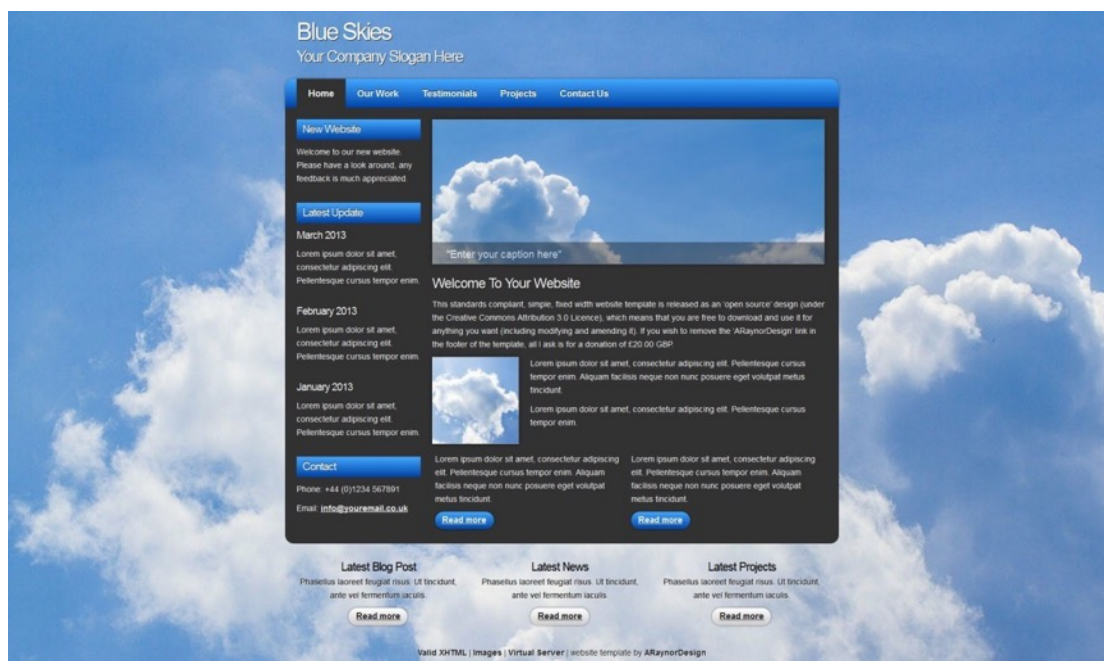


Abbildung 10 - Testlinebild des Testprojekts - selbst erstellt 28.05.2016.

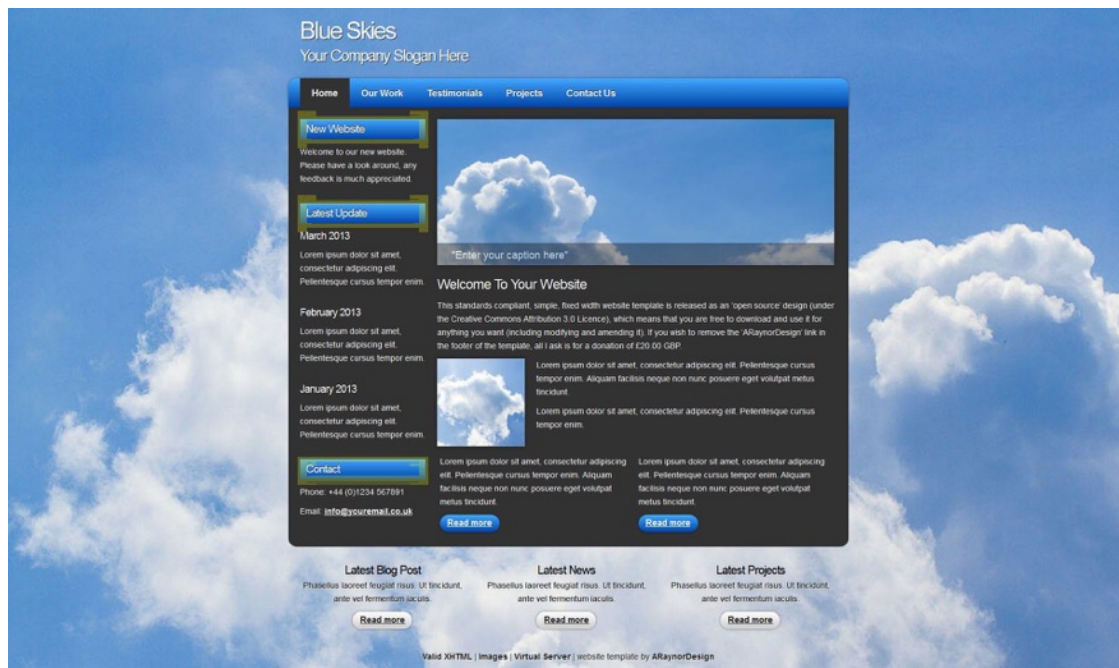


Abbildung 11 - Differenzbild mit farblichen Hervorhebungen - selbst erstellt 28.05.2016.

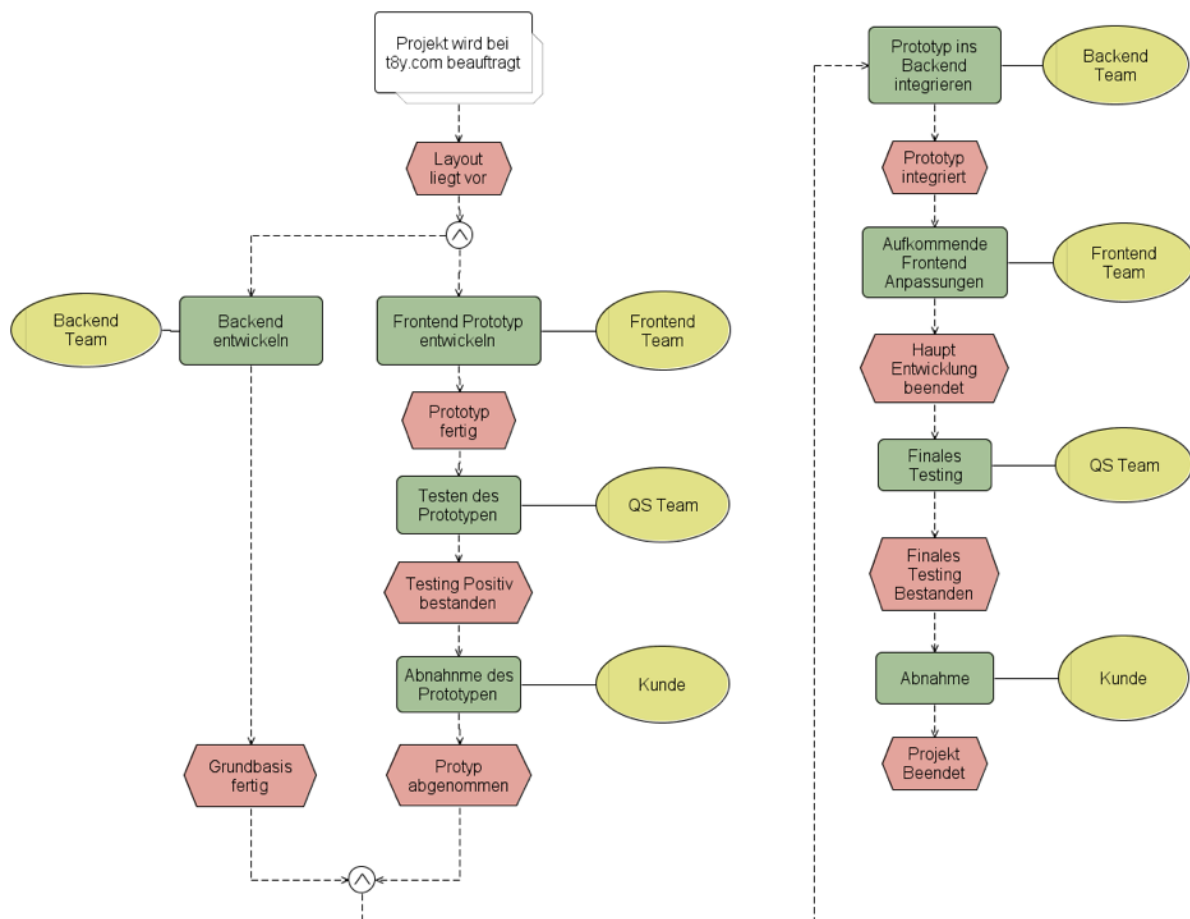


Abbildung 12 - Darstellung des Workflows bei t8y.com - erstellt von A.Hems / Mitarbeiter t8y.com - Stand 25.05.2016.

10. Quellenverzeichnis

10.1. Bücher:

Spillner, Andreas / Linz, Tilo: Basiswissen Softwaretest, Heidelberg, dpunkt.Verlag, 5. Auflage, 2012.

10.2. Onlineressourcen

Apache Friends Community: XAMPP. <https://de.wikipedia.org/wiki/XAMPP> (30.05.2016).

Diverse: Applitools Homepage. <https://applitools.com/> (30.05.2016).

Diverse : Magnolia Homepage. <https://www.magnolia-cms.com/> (30.05.2016).

Diverse: NodeJS Homepage. <https://nodejs.org/en> (23.05.2016).

Diverse: Node Paket Manager Homepage. <https://www.npmjs.com/> (23.05.2016).

Diverse: Command Line Interface. https://en.wikipedia.org/wiki/Command-line_interface (23.05.2016).

Diverse: PhantomJS Homepage. <http://phantomjs.org/> (23.05.2016).

Diverse: Headless Browser. https://en.wikipedia.org/wiki/Headless_browser (23.05.2016).

Diverse: Graphicsmagick Homepage. <http://www.graphicsmagick.org/> (23.05.2016).

Diverse: Event gesteuerte Programmierung. https://en.wikipedia.org/wiki/Event-driven_programming (27.05.2016).

Diverse: NTFS Dateisystem. <https://en.wikipedia.org/wiki/Filename> (23.05.2016).

Diverse: Lizenztypen – MIT Lizenz. <https://de.wikipedia.org/wiki/MIT-Lizenz> (30.05.2016).

Diverse: Architekturmodell Pipes und Filter. https://de.wikipedia.org/wiki/Pipes_und_Filter (30.05.2016).

Diverse: ECMAScript 6 Sprachfeatures. <http://es6-features.org> (30.05.2016).

Diverse: Promises und Futureprogrammierung. [https://de.wikipedia.org/wiki/Future_\(Programmierung\)](https://de.wikipedia.org/wiki/Future_(Programmierung)) (30.05.2016).

Diverse: Event-verkettung. https://en.wikipedia.org/wiki/Event_chain_methodology (30.05.2016).

Diverse: Mapping. <https://de.wikipedia.org/wiki/Mapping> (28.05.2016).

Diverse: Callbackfunktionen. <https://de.wikipedia.org/wiki/R%C3%BCckrufffunktion> (28.05.2016).

Diverse: Mittlere quadratische Abweichung. https://de.wikipedia.org/wiki/Mittlere_quadratische_Abweichung (25.05.2016).

Diverse: Peak Signal-to-noise Ratio. https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio (25.05.2016).

Diverse: Absoluter Fehler. https://en.wikipedia.org/wiki/Mean_absolute_error (25.05.2016).

Diverse: falsch-positiv Fehler. https://de.wikipedia.org/wiki/Fehler_1._Art (27.05.2016).

Diverse: Shebang. Executables. [https://en.wikipedia.org/wiki/Shebang_\(Unix\)](https://en.wikipedia.org/wiki/Shebang_(Unix)) (25.05.2016).

Fowler, Martin: Page Object Modell. <http://martinfowler.com/bliki/PageObject.html> (27.05.2016).

Heckmann, Aaron. Graphicsmagick for Node <https://github.com/aheckmann/gm> (23.05.2016).

10.3.Weiterhin genutzt

Stackoverflow – Entwicklerforum <http://stackoverflow.com/>

Urban Dictionary - Übersetzungstool. <http://www.urbandictionary.com/>

Google Translate- Übersetzungstool. <https://translate.google.de>