

Wie UI-Component Tests unsere E2E-Tests schneller und robuster machen

E2E-Tests reduzieren ohne Testabdeckung einzubüßen? UI-Komponenten Tests können hier helfen.

Testen einer App in der hybriden Welt

Die Thalia-App konsumiert von anderen Teams entwickelte Endpunkte. Wir beleuchten die technologischen und organisatorischen Herausforderungen beim Testen.

Bereitstellung von Testdaten einmal smart

Wir, das Team AiM (Artikel im Mittelpunkt), haben mit dem Testdatentool eine Anwendung entwickelt, um anderen Teams Testartikel zur Verfügung zu stellen. Diese Anwendung möchte ich in diesem Blogartikel vorstellen.

Das Problem der Beschaffung und

Bereitstellung von Testdaten

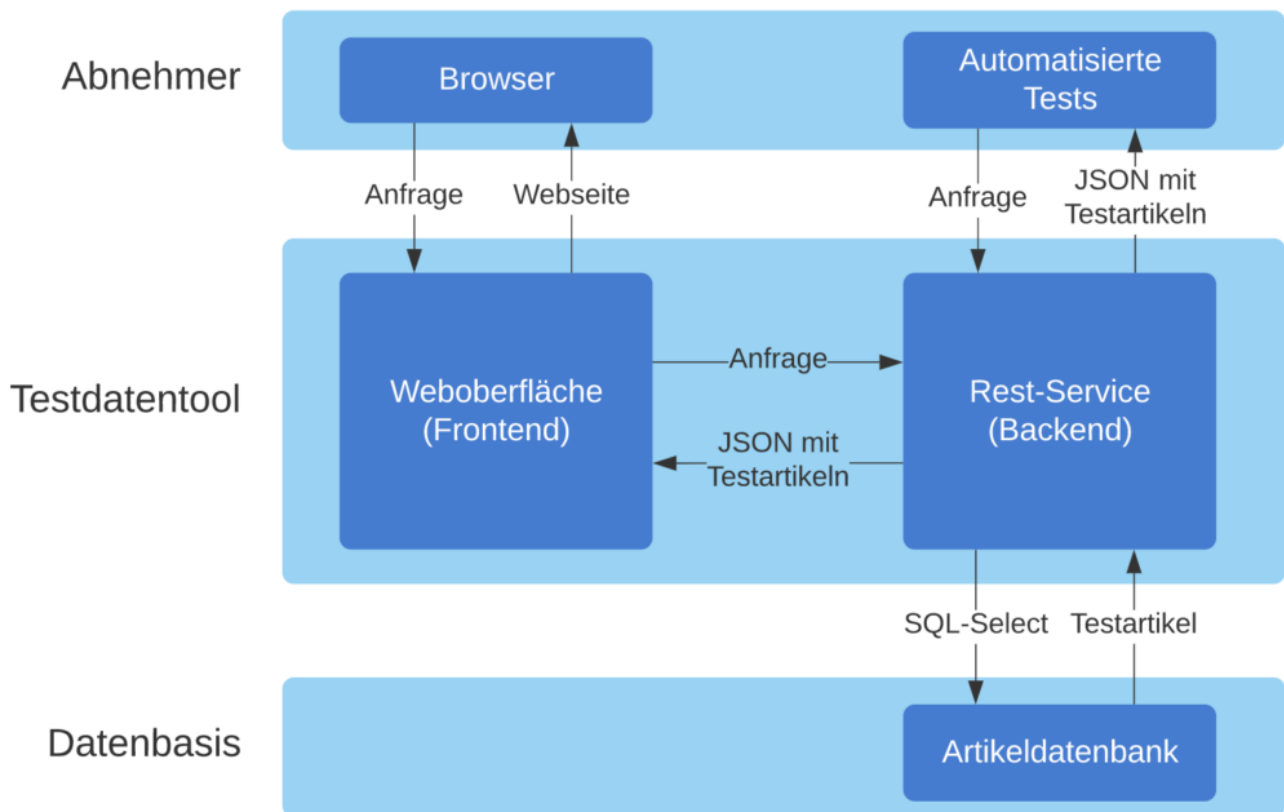
In meinen sieben Jahren als QA-ler bei der Thalia ist mir ein Problem immer wieder begegnet: Sobald Testdaten in anderen Teams liegen, nimmt deren Beschaffung viel Zeit und Energie in Anspruch. Nicht zu vergessen die Aufwände auf Seiten der Teams, die Testdaten bereitstellen.

Über viele Jahre hinweg haben wir im Team AiM für andere Teams Testartikel über individuelle SQL-Abfragen herausgesucht und über die Herausgabe der Artikel-ID oder EAN bereitgestellt. Dieser Prozess hat sich mit der Zeit als sehr ineffizient herausgestellt. Um diesen Prozess zu optimieren haben wir das Testdatentool entwickelt, über welches alle Teams eigenständig Testartikel beziehen können.

Das Testdatentool stellt zwei Dienste zur Verfügung. Zum einen bietet es eine **Weboberfläche**, welche die Suche nach Testartikeln über alle gängigen Browser ermöglicht. Zum anderen stellt das Testdatentool einen **REST-Service** bereit, den Entwicklerteams im Rahmen ihrer automatisierten Tests einbinden können, um dynamisch Testartikel zu beziehen.

Aufbau

Die Weboberfläche und der REST-Service bilden zusammen das **Frontend** und das **Backend** des Testdatentools (siehe Abbildung). Während das Frontend die Suchanfrage durch den Anwender über eine Weboberfläche entgegennimmt, führt das Backend die Suche über einen REST-Service auf der Datenbank aus, um die gefundenen Treffer im JSON-Format zurückzugeben. Die Begriffspaarungen Weboberfläche/REST-Service und Frontend/Backend werden in diesem Blogartikel synonym verwendet.



Weboberfläche

Die Weboberfläche des Testdatentools ermöglicht es dem Anwender über wenige Klicks einen passenden Testartikel zu finden (siehe Abbildung). Insgesamt bietet die Weboberfläche 23 Kategorien, die für die Suche nach einem Testartikel zur Verfügung stehen. Für jede Suche können die Marke (thalia.de, thalia.at, bol.de oder orellfuessli.ch), die Umgebung (Integrations- oder Produktivumgebung) und die Anzahl der benötigten Testartikel über entsprechende Comboboxen eingestellt werden. Einige Kategorien können zusätzlich über weitere, individuelle Filter konkretisiert werden. Die Abbildung veranschaulicht einen solchen zusätzlichen Filter für die Kategorie „FSK-Artikel“. Nachdem die Suche ausgeführt und die gefundenen Artikel in der Trefferliste angezeigt werden, kann der Anwender sich das zu Grunde liegende SQL über den Button „SQL“ anzeigen lassen. Weiter kann der Anwender über den Button „COPY EANS“ alle EANS aus der Trefferliste in den Zwischenspeicher des Betriebssystems kopieren. Über diesen können die EANS in jedes andere Programm übertragen werden. Die Buttons „COPY ARTIKEL-IDS“ und „COPY MATNRS“ verhalten sich entsprechend analog zu „COPY EANS“.

Allgemein Ebooks von Libri Artikelverknüpfungen Bevorzugter Lieferant Fsk-Artikel Lieferbarkeit Sortiment Tolino Vorbesterler Hoertyp	FSK-ARTIKEL					Marke thalia.de	Umgebung Integration	Anzahl 100
	SUCHE  SQL COPY ARTIKELIDS COPY EANS COPY MATNRS							
Medien Bilder Hoerprobe Leseprobe Trailer	Altersfreigabe Freigegeben ab 6 Jahren							
Preis Streichpreis Preisbindung ThaliaClub Verkaufspreis	artikelId	ean	matnr	titel	preis	sortiment		
	1788013	0743218997495	A1004922755	Asterix & Obelix - Mission Kleopatra	9900	Film		
Abo Abo - Ebook Abo - Hoerbuch Download	6722291	0828766845198	A1001412833	In 80 Tagen um die Welt	999	Film		
	13566646	0828767673691	A1001372839	Good Woman - Ein Sommer in Amalfi	999	Film		
Slider auf Artikeldetailseite Lernhilfen zu diesem Titel Weitere Baende von Weitere Serientitel zu	15657456	0886972973098	A1001621513	Unsere Erde	1299	Blu-Ray		
	17236799	0886973825792	A1007631486	Wächter der Wüste	899	Film		
mehr-von-Suche mehr-von-Autor	21108135	0886973826294	A1011626401	Mitte Ende August	999	Film		
	16202906	0886974172093	A1002077437	Das Superhirn	9900	Film		
	19502897	0886976511999	A1010552125	Das Wunder von Bern	1299	Blu-Ray		
	22650005	0886976832292	A1013009820	Unsere Ozeane	9900	Film		
	28821137	0886978997593	A1017744074	Das Ende ist mein Anfang	1299	Blu-Ray		
	29056832	0886979197695	A1018104624	Nach fünf im Urwald	999	Film		
	33935866	0887654346292	A1021150179	LEGO Ninjago - Staffel 2 [2 DVDs]	9900	Film		
	19671018	4006680052540	A1010632951	Charlie Chaplin - Moderne Zeiten	1699	Blu-Ray		
	4427848	4010324021199	A1002077383	Spy Kids 2	9900	Film		
	14536693	4010324025661	A1001343424	Die Queen	1399	Film		
	32995487	4010884256017	A1019831211	Catch Me If You Can	9900	Blu-Ray		
	14672666	4010884528251	A1001289664	Elvis Presley - Blaues Hawaii	699	Film		
	14041681	4010884532739	A1001343113	Die Möwe Jonathan	1099	Film		
	28845928	4010884540949	A1017714995	Rango	1099	Film		
	32488101	4010884543902	A1020023701	SpongeBob Schwammkopf - Geisterdeppen	999	Film		

REST-Service

Der REST-Service besitzt für jede Kategorie, welche die Weboberfläche anbietet, eine REST-Ressource. Hinter jeder REST-Ressource verbirgt sich wiederum ein SQL, über das die Testartikel auf der Artikeldatenbank gesucht werden. In das SQL werden die Werte eingesetzt, die der Aufrufer über URL-Parameter mitliefert. Die gefundenen Testartikel werden in einem definierten JSON-Format zurückgeliefert.

Wir haben uns dazu entschlossen, den Funktionsumfang des REST-Service stets mit dem der Weboberfläche synchron zu halten. Denn während die Implementierung einer neuen Kategorie in der Weboberfläche ohne eine Anpassung des REST-Service nicht möglich ist, wäre es sehr wohl möglich, den REST-Service für einen automatisierten Test um eine neue Kategorie zu erweitern, ohne diese in die Weboberfläche aufzunehmen. Indem wir die Weboberfläche und den REST-Service synchron halten, stellen wir sicher, dass die Entwicklerteams die Weboberfläche als bildhafte Dokumentation des REST-

Service nutzen können.

Designentscheidungen

In diesem Kapitel beschreibe ich Designentscheidungen, die wir für das Testdatentool getroffen haben.

Bei der Implementierung der Weboberfläche haben wir besonders viel Wert auf eine einfache Erweiterbarkeit gelegt, damit die Weboberfläche auch ohne tiefgreifende Vue.js-Kenntnisse von allen Entwicklern des Teams erweitert werden kann. Das nachfolgende Codebeispiel verdeutlicht, dass lediglich die Erstellung einer neuen Klasse notwendig ist, um die Weboberfläche um eine neue Kategorie zu erweitern. Nachdem diese Klasse erstellt ist, muss diese nur noch in der main.js eingetragen werden, um die Erweiterung im Frontend abzuschließen.

```
Fsk.vue x main.js x
1 <template>
2 <div>
3 <theme :headline="headline"
4 :resource="resource"
5 :comboboxes="comboboxes"/>
6 </div>
7 </template>
8
9 <script>
10
11 export default {
12   created() {
13     // Legt fest, welcher Eintrag in der Combobox zu Beginn angezeigt wird.
14     this.comboboxes[0].selectedValue = this.comboboxes[0].defaultValue
15   },
16   data() {
17     return {
18       // Bestimmt, welche Überschrift fuer das Suchthema im Frontend angezeigt wird.
19       headline: 'Fsk-Artikel',
20       // Bestimmt, wie die Rest-Ressource des Rest-Service aufgerufen wird (hier: /testdatentool-backend/api/fsk/)
21       resource: 'fsk',
22       comboboxes: [
23         {
24           // Eintraege, die in der Combobox angezeigt werden. Die Variable "name" legt fest, wie die Eintraege
25           // in der Combobox heissen. Die Variable "value" definiert, wie genau das Backend fuer den Eintrag
26           // aufgerufen wird. Waehlt der Nutzer in der Combobox den Eintrag "Freigegeben ab 18 Jahren", dann wird
27           // das Backend mit dem Query-Parameter ?alter=18 aufgerufen. Der Name des Query-Parameters (hier: alter)
28           // wird ueber die Variable "queryParamName" festgelegt.
29           entries: [
30             {name: 'Freigegeben ab 6 Jahren', value: '6'},
31             {name: 'Freigegeben ab 12 Jahren', value: '12'},
32             {name: 'Freigegeben ab 16 Jahren', value: '16'},
33             {name: 'Freigegeben ab 18 Jahren', value: '18'}
34           ],
35           // Verweist auf den Eintrag, den der Nutzer in der Combobox ausgewaehlt hat. Diese Variable ist eine
36           // Referenz auf die Variable "value" in dem Array "entries". Hat der Nutzer den Eintrag
37           // "Freigegeben ab 18 Jahren" ausgewaehlt, dann traegt die Variable "selectedValue" den Wert "18". Die
38           // Variable "selectedValue" wird immer dann aktualisiert, wenn der Nutzer ueber die Combobox eine neue
39           // Auswahl trifft. Die Variable wird zum ersten Mal in der Methode created() mit einem Wert versehen.
40           selectedValue: '',
41           // Verweist auf den Eintrag, der in der Combobox zu Beginn ausgewaehlt ist. Diese Variable ist eine
42           // Referenz auf die Variable "value" in dem Array "entries".
43           defaultValue: '18',
44           // Beschriftung der Combobox.
45           label: 'Altersfreigabe',
46           // Name des Query-Parameters, ueber den die Auswahl in der Combobox an das Backend uebergeben wird (hier: ?alter).
47           queryParamName: 'alter'
48         }
49       ]
50     }
51   }
52 }
53 </script>
54
55
```

Die zweite Designentscheidung betrifft die Optimierung des Antwortzeitverhaltens des Backends. Von Beginn an war klar, dass einige Suchen bis zu 10 Sekunden dauern würden, da die jeweiligen SQLs sehr komplex sind. Um die Antwortzeiten zu optimieren, wurde das Backend um einen Cache inklusive Cacheaufwärmmechanismus erweitert. Hierdurch wird sichergestellt, dass das Testdatentool auch bei Langläufer-SQLs von über 10 Sekunden, in unter 500 Millisekunden antwortet. Anders ausgedrückt garantiert der Cacheaufwärmmechanismus, dass für jede Langläufer-Suche zu jedem Zeitpunkt ein gültiger Cache-Eintrag existiert und die Suche aus dem Cache bedient werden kann.

Die letzte Designentscheidung betrifft ebenfalls das Backend und besteht in der Nutzung von Swagger zur Dokumentation der REST-Ressourcen. Swagger bietet unserem Team vor allem zwei Vorteile. Zum einen spart unser Team durch Swagger Zeit, da die Dokumentation mit wenig Aufwand erstellt werden kann. Zum anderen ermöglicht Swagger eine inhaltlich ausdrucksstarke Dokumentation jeder einzelnen REST-Ressource. Dies ist besonders wichtig, da der REST-Service durch andere Teams genutzt wird und dessen Verwendung weitestgehend selbsterklärend sein soll. Die Einbindung von Swagger gestaltet sich dabei denkbar einfach. Nach der Aufnahme der Swagger-Dependency in die pom.xml und der Anreicherung der REST-Ressourcen um Metadaten zu jedem URL-Parameter, genügt eine Annotation über der REST-Service-Klasse, um Swagger einzubinden. Wie die Swagger-Dokumentation für den REST-Service aussieht ist der linken Abbildung zu entnehmen. Die rechte Abbildung zeigt die Detailansicht einer einzelnen Ressource.

Testdatentool

Rest-Ressourcen des Testdatentools.

Created by Thalia

rest-service : Rest Service

Show/Hide | List Operations | Expand Operations

GET	/api/abo-ebook/{mandantId}/	getAboEbook
GET	/api/abo-hoerbuch-download/{mandantId}/	getAboHoerbuchDownload
GET	/api/abo/{mandantId}/	getAbo
GET	/api/artikelverknuepfung/{mandantId}/	getArtikelverknuepfung
GET	/api/baende/{mandantId}/	getBaende
GET	/api/bevorzugterLieferant/{mandantId}/	getBevorzugterLieferant
GET	/api/bilder/{mandantId}/	getBilder
GET	/api/ebooks/{mandantId}/	getLibriEbooks
GET	/api/fsk/{mandantId}/	getFsk
GET	/api/hoerprobe/{mandantId}/	getHoerprobe
GET	/api/hoertyp/{mandantId}/	getHoertyp
GET	/api/lernhilfen/{mandantId}/	getLernhilfen
GET	/api/leseprobe/{mandantId}/	getLeseprobe
GET	/api/lieferbarkeit/{mandantId}/	getLieferbarkeit
GET	/api/mehr-von-autor/{mandantId}/	getMehrVonAutor
GET	/api/preisbindung/{mandantId}/	getPreisbindung
GET	/api/serientitel/{mandantId}/	getSerientitel
GET	/api/sortiment/{mandantId}/	getSortiment
GET	/api/streichpreis/{mandantId}/	getStreichpreis
GET	/api/thaliacub/{mandantId}/	getThaliaClub

GET
/api/fsk/{mandantId}/
getFsk

Response Class (Status 200)

string

Response Content Type
/

Parameters

Parameter	Value	Description	Parameter Type	Data Type
mandantId	2	2, 4, 5 oder 37	path	integer
anzahl	10	1 bis 100	query	integer
umgebung	integ	integ oder prod	query	string
alter	18	6, 12, 16 oder 18	query	integer

Response Messages

HTTP Status Code	Reason	Response Model	Headers
401	Unauthorized		
403	Forbidden		
404	Not Found		

Try it out!

Technologien

In diesem Kapitel möchte ich kurz die Technologien nennen, die für die Entwicklung des Testdatentools verwendet wurden. Weiter erläutere ich stichpunktartig die Gründe, welche für die Auswahl der Technologien ausschlaggebend waren.

Vue.js als Framework für die Weboberfläche - Vorteile:

- Vue.js setzt auf bekannte Technologien wie JavaScript, HTML und CSS
- Eine sehr aktive Community und eine umfassende Dokumentation mit vielen Best-Practice-Beispielen
- Steile Lernkurve, da das Konzept hinter Vue.js leicht zu verstehen ist
- Sehr gute Performance aufgrund einer effizienten Implementierung des virtual DOM

Spring-Boot als Framework für den REST-Service - Vorteile:

- Unser Team hatte bereits Erfahrungen im Umgang mit Spring-Boot
- Aufsetzen eines REST-Service ohne langwierige Konfigurationsarbeiten
- Einfacher Datenbankzugriff durch Spring-Boot-Funktionalitäten

Erfahrungsbericht

Dieser Erfahrungsbericht soll denen helfen, die eine Anwendung im Stile des Testdatentools für ihr eigenes Team in Erwägung ziehen.

Als wir das Testdatentool eingeführt haben, war die Resonanz durch die anderen Teams durchweg positiv. Insbesondere die Weboberfläche wurde von Beginn an intensiv genutzt und überzeugt seitdem durch ihre übersichtliche und einfache Bedienung. Auf der anderen Seite hat es einige Zeit gedauert, bis der REST-Service als Testartikel-Lieferant für automatisierte Tests bei den Teams Einzug erhalten hat. Tatsächlich gibt es immer noch Teams, die auf die Nutzung des REST-Service im Rahmen ihrer automatisierten Tests verzichten und stattdessen darauf zurückgreifen, Testartikel statisch in ihre Tests einzubinden. Dies ist für viele Teams völlig ausreichend, wenn Testartikel in den Tests nur eine untergeordnete Rolle spielen.

Das Testdatentool hat unserem Team dabei geholfen, die Anzahl der Anfragen, die sich um die Bereitstellung von Testartikeln drehen, um den Faktor vier zu reduzieren. Hierdurch hat sich die Entwicklungszeit des Testdatentools nach kurzer Zeit bezahlt gemacht. Nicht zu vergessen, dass es auch auf Seiten der Teams, die nach Testartikeln suchen, zu Zeiteinsparungen kommt. Abschließend kann ich mit Überzeugung sagen, dass sich dessen Einführung für uns und alle Beteiligten gelohnt hat.

True Grid oder wie wir unsere Testausführung durch Parallelisierung 6x schneller machten

Bei Thalia gibt es nicht nur crossfunktionale Teams mit QAlern, es gibt auch die

zentrale QA. Wir testen prozesskettenübergreifend – d.h. während die Kollegin aus Team Kaufen alle Eingabemöglichkeiten der Zahlarten durchtestet oder der Kollege aus Team Kunde alle möglichen (und unmöglichen) Lieferadressen hinterlegt, legen wir in einem Test einen Neukunden an, ändern die Zahlart, hinterlegen eine Lieferadresse, suchen einen Artikel und kaufen ihn dann. Alles natürlich automatisiert, und für 4 verschiedene Mandanten.

Verwendete Startseiten-URL: [https://\[REDACTED\]chalia.de/shop/home/show/](https://[REDACTED]chalia.de/shop/home/show/)

Einzelne Testschritte:

1. Testschritt: Aufruf der ShopStartseite
2. Testschritt: Aufruf der LoginSeite
3. Testschritt: Aufruf Registrierungsformular.
4. Testschritt: Neukundenregistrierung. Kunde ID: 217 | Anrede: Frau | Vorname: Selenium | Nachname: Regression | E-Mail: [REDACTED] | Passwort: [REDACTED] | Strasse: Am Sportplatz | Hausnummer: 6 | Geburtstag: 1.2.1975 | FaxNr: | TelefonNr: [REDACTED] | Benutzername: [REDACTED] | PLZ 48480 | Stadt: Spelle | Staat: 81
5. Testschritt: Aufruf Zahlarten-Formular
6. Testschritt: Wähle Vorkasse.
7. Testschritt: Klick auf Abmelden.
8. Testschritt: Aufruf der Artikeldetailseite fuer EAN: 4260036673159
MyInfoId: 5841031
Beschreibung: FSK-18-Artikel
Titel: X-Pilation 3 - Experiment Erzählung
9. Testschritt: Lege Artikel 4260036673159 in den Warenkorb.
10. Testschritt: Gehe zum Warenkorb.
11. Testschritt: Gehe weiter zur Kasse.
12. Testschritt: Login mit validen Daten
13. Testschritt: Schliesse Bestellung ab.
14. Testschritt: Fehler, deswegen Abbruch: org.openqa.selenium.TimeoutException: Expected condition failed: waiting for element to be clickable: Proxy element for: DefaultElementLocator 'By.id: divPostident-label' (tried for 60 second(s) with 400 milliseconds interval)

Insgesamt haben wir rund 100 Tests, die wir auch noch in 2 Browsern mit unterschiedlichen Auflösungen ausführen. Bis vor wenigen Wochen noch dauerte der größte Integrationstest ca. 6 Stunden... Wenn dann aber ein Hotfix aufgespielt werden soll und jemand schnell die Antwort haben möchte, ob denn die automatischen GUI-Tests fehlerfrei durchgelaufen sind, ist „Frag morgen nochmal nach“ keine angenehme Antwort – unsere Tests mussten schneller werden.

Wir überlegten kurz, ob weniger Tests in weniger Browsern oder ohne Screenshots eine Alternative wären, stellten aber schnell fest, dass unser Problem leichter mit mehr Rechnern und paralleler Testausführung zu lösen war. [Selenium Grid](#) mit allen Möglichkeiten zur einfachen Parallelisierung von ferngesteuerten Browsern war die Antwort.

Alle machen Docker, warum nicht wir? Bisher hatten wir unsere Tests auf Windows durchgeführt (die Mehrzahl der Kunden nutzt Windows), aber wenn man z.B. [Zalenum](#) (einen Docker-basierten, dynamischen Selenium Grid) nutzen will, muss man Linux als Plattform wählen. Leider scheint es mit Firefox unter Linux bei automatisierten Tests manchmal Probleme mit unserem Hauptmenü zu geben, was sich an unerwarteten Stellen über die wichtigen Elemente legt und damit den Testweitergang behindert. Außerdem kann man mit Zalenum Chrome

nicht in der kleinsten Auflösung unserer Webseite benutzen – das ist uns aber sehr wichtig, daher nahmen wir wieder Abstand von einer Docker- und Linux-basierten Lösung.

Die Anzahl unserer Tests ändert sich nicht so häufig, und wie viele Nodes wir parallel brauchen, können wir einfach pro Testinstanz im Jenkins festlegen – daher entschlossen wir uns, mit einer festen Anzahl Windows-Rechner zu arbeiten, die sich jeweils mit 4 Node-Prozessen an einem Selenium-Grid (der ausnahmsweise unter Linux läuft) anmelden. Um die genaue Anzahl von verfügbaren Browser-Handles und damit die Anzahl von Threads, die pro Test verwendet werden können, zu verwalten, sei hier noch löblich auf das [Lockable Resource-Plugin](#) im Jenkins verwiesen, mit dem wir recht einfach Flaschenhälse bei der gleichzeitigen Ausführung des Tests verhindern konnten.

Lockable Resources

Resource	Status	Labels	Action
GridNode1	LOCKED by [REDACTED] #971	GridNode	Unlock
GridNode2	LOCKED by [REDACTED] #971	GridNode	Unlock
GridNode3	LOCKED by [REDACTED] #428	GridNode	Unlock
GridNode4	LOCKED by [REDACTED] #428	GridNode	Unlock
GridNode5	LOCKED by [REDACTED] #428	GridNode	Unlock
GridNode6	LOCKED by [REDACTED] #428	GridNode	Unlock
GridNode7	LOCKED by [REDACTED] #428	GridNode	Unlock
GridNode8	FREE	GridNode	Reserve
GridNode9	FREE	GridNode	Reserve
GridNode10	FREE	GridNode	Reserve
GridNode11	FREE	GridNode	Reserve
GridNode12	FREE	GridNode	Reserve
GridNode13	FREE	GridNode	Reserve
GridNode14	FREE	GridNode	Reserve
GridNode15	FREE	GridNode	Reserve

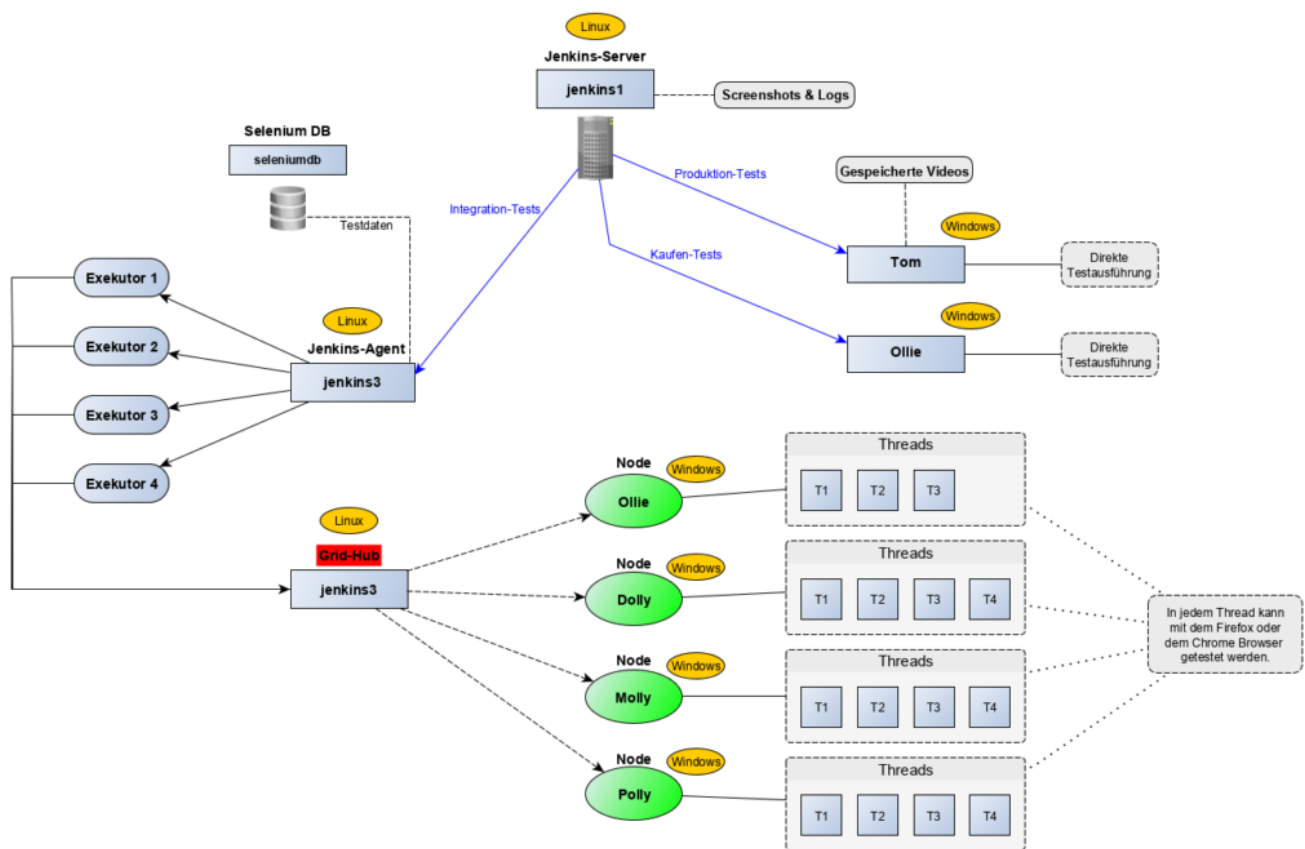
Labels

Label	Free resources
GridNode	8

Jetzt haben wir also 4 Windows-Rechner als Selenium Nodes. Ein weiterer

Windows-Rechner wird für stündliche Tests gegen die Produktion verwendet. Und auf denen soll man jetzt Firefox- bzw. Chrome-Versionen im Blick behalten und die passenden Treiber für Selenium (geckodriver oder chromedriver) updaten? Sowas macht man heutzutage nicht mehr per Hand, es musste eine automatisierte Update-Lösung her. Dank Git und einem Batch-Update-Skript, was nächtlich läuft, kann ich auf meinem lokalen Rechner das Zusammenspiel von Browser-Treiber und Browser testen, einchecken und am nächsten Tag davon ausgehen, dass die Updates auf alle Rechner verteilt sind.

Aufbau unserer Selenium-Landschaft



Und wie lange brauchen unsere Tests jetzt? Wir sind tatsächlich mit Tests für alle 4 Mandanten und noch einigen wenigen komplexen Testszenarien jetzt in rund 1 Stunde mit allen Tests durch - wenn alles gut läuft und die Testumgebung nicht zu sehr verändert wurde.

Build	↑	Duration	Agent
#1186		1 hr 1 min	um-qajenkins3
#1185		1 hr 1 min	um-qajenkins3
#1184		1 hr 20 min	um-qajenkins3
#1183		1 hr 3 min	um-qajenkins3

Akzeptanztests bei Thalia

Im Team „Kaufen“ und im Team „Kunde im Mittelpunkt“ setzen wir seit über einem Jahr automatisierte Akzeptanztests zur Sicherung unserer Qualität ein. Beide Teams sind begeistert von diesem Vorgehen. Höchste Zeit für einen Beitrag in unserem Techblog ☐

Was ist ein Akzeptanztest?

Ein Akzeptanztest ist ein funktionaler Test, der das Benutzerverhalten beschreibt, um User Story und Akzeptanzkriterien zu überprüfen.

Dabei steht die Sicht des Benutzers im Vordergrund. Es ist wichtig nicht irgendwas zu testen, sondern genau das, was wir mit einer User Story umsetzen wollen. Durch die Akzeptanztests wird sichergestellt, dass die Software aus Sicht des Benutzers wie gewünscht funktioniert.

Ein Akzeptanztest besteht aus drei Teilen: Vorbedingungen (Angenommen), Aktionen die durchgeführt werden (Wenn) und erwartete Ergebnisse (Dann). Diese sogenannten Szenarien werden später zu automatisierten Tests auf verschiedenen Ebenen der Testpyramide. Zur Definition der Szenarien verwenden wir in unseren Teams das Framework [Cucumber](#).

Beispiel eines einfachen Szenarios:

Szenario: Anzeige der Zahlungsarten unter mein Konto mit Rechnung als präferierte Zahlungsart

Angenommen es existiert ein angemeldeter Kunde bei Thalia.de

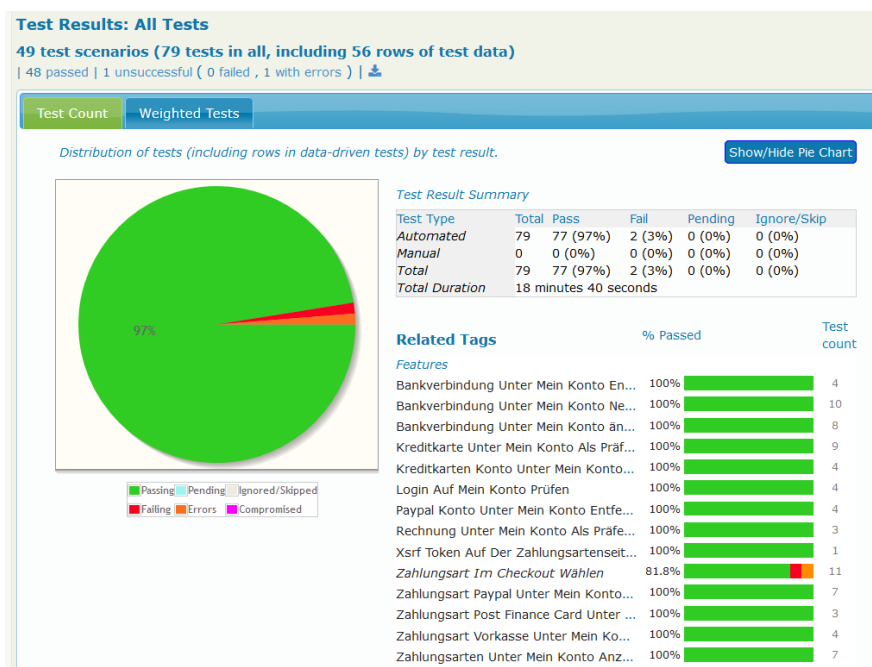
Angenommen der Kunde hat die präferierte Zahlungsart Rechnung

Wenn dieser Kunde die Seite Mein Konto Zahlungsarten aufruft

Dann ist das Akkordeon an der Stelle Rechnung aufgeklappt

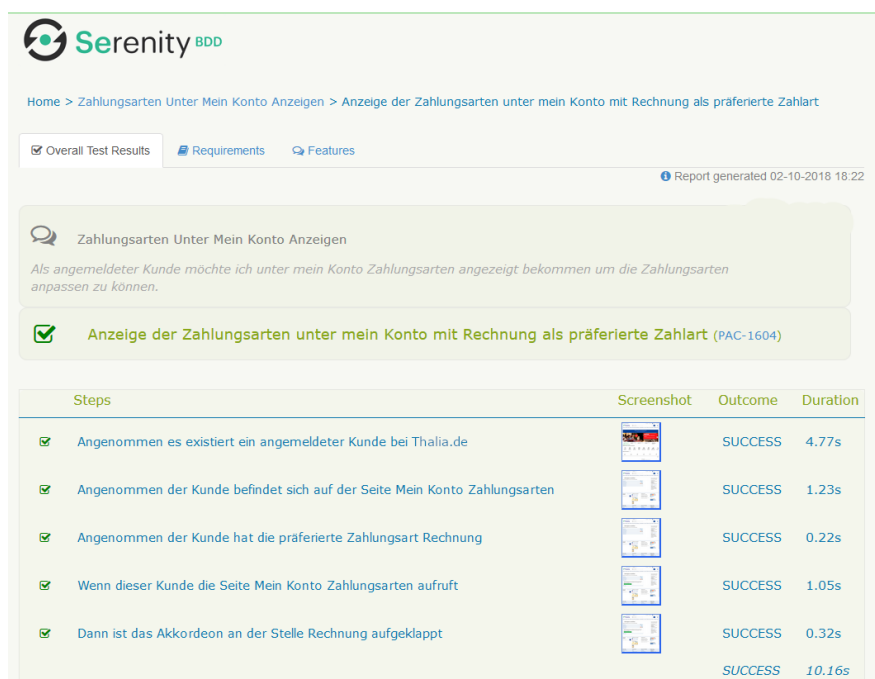
Testreport mittels Serenity

Bei jedem Testdurchlauf wird automatisch mittels [Serenity](#) ein Bericht mit den Testergebnissen erstellt.



Serenity Report - Übersicht

Zu jedem Szenario können die einzelnen Testschritte inklusive Screenshots betrachtet werden. Im Fehlerfall sieht man somit direkt, welcher Testschritt fehlgeschlagen ist.



Serenity Report - Übersicht

Zielbild Testpyramide

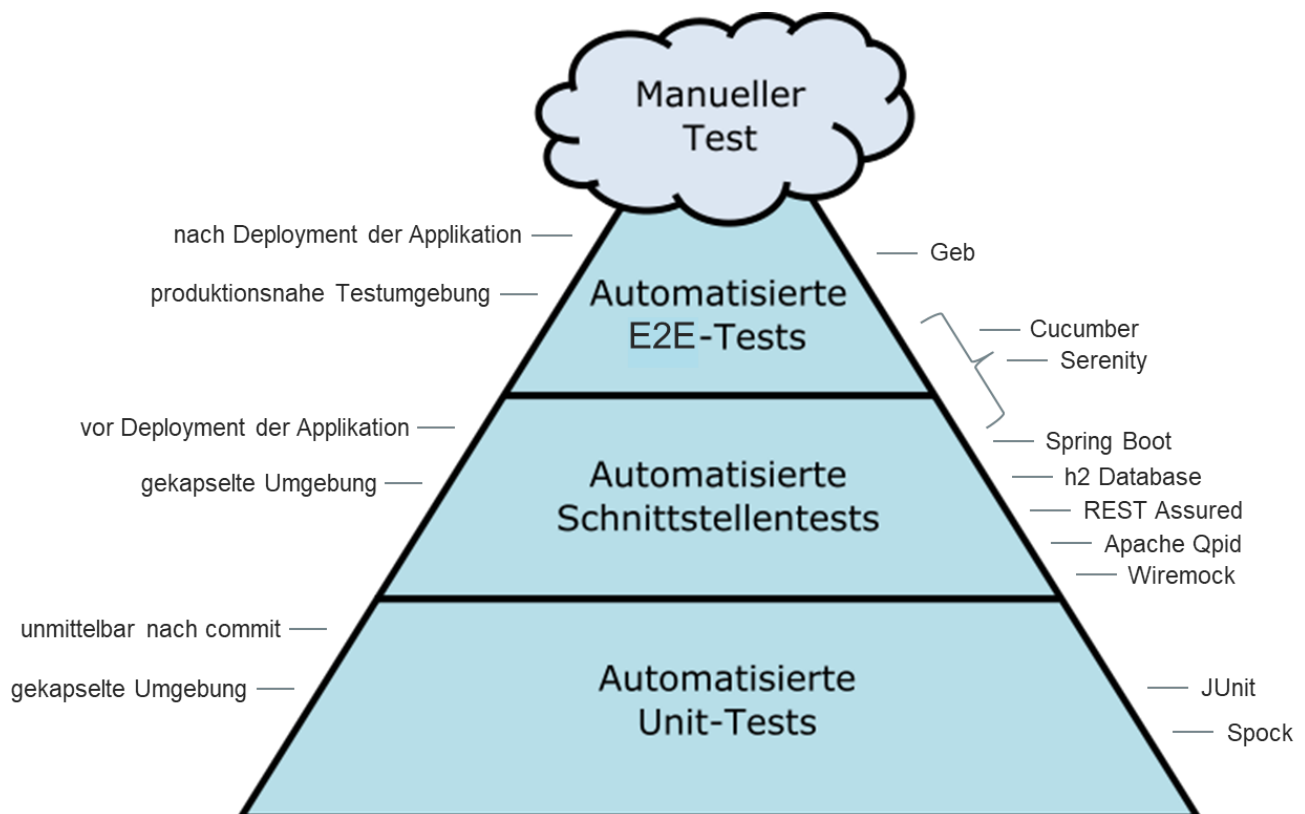
Unser Zielbild bei der Qualitätssicherung im Team „Kaufen“ orientiert sich an der

klassischen Testpyramide. Wir wollen unsere Fehler natürlich möglichst früh und möglichst vor dem End-to-End-Test finden. Da wir allerdings gerade dabei sind unser Frontend abzulösen und unseren Checkout neu zu gestalten, schreiben wir im Moment relativ viele End-to-End-Tests.

Um dem entgegen zu wirken, begeben wir uns als nächstes auf die Suche nach einem geeigneten JavaScript-Test-Framework. Wir erhoffen uns dadurch mögliche Fehler im Frontend bereits auf Unit-Test-Ebene aufzudecken und ggf. sogar einzelne End-to-End-Tests ablösen zu können.

Welche Frameworks verwenden wir und wann werden die Tests ausgeführt?

Die gleichen Szenarien nutzen wir sowohl für die automatisierten End-to-End-Tests, als auch für die Schnittstellentests.



Die automatisierten End-to-End-Tests erfolgen nach dem Deployment in einer produktionsnahen Testumgebung. Dabei verwenden wir das Framework [Geb](#), welches den Selenium WebDriver nutzt und der Browser-Automatisierung dient. Die Tests werden hierbei in der Programmiersprache groovy geschrieben.

Durch die Schnittstellentests soll die jeweilige Applikation inklusive Datenbank- und RabbitMQ-Prüfungen (wo notwendig) zur Laufzeit getestet werden. Um möglichst unabhängig zu sein, erfolgt der Test in einer gekapselten Umgebung. Dazu wird die Applikation mittels h2 Database, Apache Qpid und Spring Boot gestartet, so dass die Schnittstellentests zwar zur Laufzeit, aber noch vor dem Deployment ausgeführt werden können. Fremdsysteme werden dabei mit Wiremock simuliert. REST Assured wird zur Durchführung von HTTP Requests und Validierung der HTTP Response verwendet.

Welche Vorteile bringen uns die Akzeptanztests?

Die Akzeptanztests helfen uns offene Punkte und Ungereimtheiten aufzudecken und mit dem Product Owner zu besprechen.

Aktuell werden im Team „Kaufen“ die Szenarien meistens durch eine Person aus dem Dev-Team kurz vor oder zeitgleich zur Implementierung geschrieben. Dies kann, muss aber nicht, die QAlerin des Teams sein. Wie beim Code-Review erfolgt auch hier ein Review durch eine zweite Person des Dev-Teams.

Noch besser wäre es die Szenarien bereits vor dem Start der Software-Implementierung zu schreiben. Optimalerweise sogar in der „3 Amigos“ Konstellation, welche aus Product Owner, QAlerin/Analyst und Entwickler besteht, um die Anforderung aus unterschiedlichen Blickwinkeln zu betrachten. Dadurch können Ungereimtheiten und Komplexitätstreiber bereits während der Anforderungsanalyse aufgedeckt und besprochen werden. Bisher fehlte uns im Team „Kaufen“ leider die Zeit / der Vorlauf diese Variante auszuprobieren. Wir sind aber zuversichtlich, dass es dazu eine Gelegenheit geben wird ☐

Im Team „Kunde im Mittelpunkt“ wird bei der Erstellung der Szenarien bereits mit der Product Ownerin und der Fachseite zusammen gearbeitet. Dabei wurden bisher sehr gute Erfahrungen gemacht, da anschließend alle relevanten Personen den gleichen Wissenstand auf einer formalen Ebene haben. Siehe auch [hybride Testkonzeption](#).

Es gibt weitere Vorteile, die uns als Dev-Team zugutekommen.

Wir haben es geschafft die Teamsicht auf unsere Tests zu verbessern. Alle im Dev-Team schreiben Szenarien und implementieren die daraus resultierenden automatisierten Tests. Die Akzeptanztests sind Teil des Repositories der jeweiligen Applikation, werden in der Entwicklungsumgebung geschrieben und

bei Bedarf lokal ausgeführt. Durch die Verknüpfung zwischen Fachlichkeit und automatisierten Tests in Form der Szenarien, sind die Auswirkungen bei fehlgeschlagenen Testfällen außerdem direkt sichtbar und helfen uns so bei GoLive-Entscheidungen.

BDD + DDD = eine hybride Testkonzeption

Das agile Produktteam „KiM“, verwendet in Bezug auf die Qualitätssicherung seiner Produkte zwei wesentliche Ansätze, die wir im Rahmen dieses Blogs kurz skizzieren wollen. Dabei geben wir nicht nur einen konzeptionellen Einblick, sondern stellen auch aus unserer Sicht die Vorteile für die Anwendung eines hybriden Ansatzes heraus.

Behaviour Driven Development

Beim Behaviour Driven Development steht das erwünschte Verhalten der Software aus Sicht des Kunden im Vordergrund. In der Anforderungsanalyse werden die Ziele, Aufgaben und Ergebnisse der Software in Textform als Szenarien festgehalten. Die Szenarien werden unter Verwendung des „Given-When-Then“ - Konzepts erstellt (siehe auch: [Akzeptanztests bei Thalia](#)) und dienen als Basis für die Entwicklung automatisierter Tests. Somit wird bei den automatisierten Tests stark die Perspektive des Kunden eingenommen.

Dieser Ansatz bietet nicht nur den Vorteil der einfachen Lesbarkeit, sondern auch die Möglichkeit alle beteiligten Akteure (wie bspw. den Product Owner, die Fachseite, etc.) während der Testfallerstellung mit einzubeziehen und dadurch ein gemeinsames Verständnis vom System bzw. der Software aufzubauen.

Ein mögliches Beispiel für ein Szenario:

Szenario: Benutzer meldet sich über die Mobile-App an
Angenommen "Max Mustermann" benutzt die Mobile-App

Wenn "Max Mustermann" sich während des Bestellvorgangs erfolgreich anmeldet
Dann soll die Checkout-Seite angezeigt werden
Und "Max Mustermann" ist in der Mobile-App angemeldet

Ein weiteres Element, welches beim BDD-Konzept das gemeinsame Verständnis stärkt, ist der Einsatz einer „ubiquitous language“.

Die „ubiquitous language“ dient als gemeinsam definiertes und genutztes Vokabular, wodurch Fehlinterpretationen bei der Verwendung von Begriffen minimiert werden. Beispielsweise definieren Fachbereiche wie das CRM oder die Softwareentwicklung Begriffe wie „Kunde“ oder „Auftrag“ unterschiedlich. Der Kunde könnte zum einen als Kunde interpretiert werden, sobald er sich im Webshop eingeloggt hat oder zum anderen wenn er mindestens einen Auftrag durchgeführt hat. Dieses heterogene Verständnis kann zu Problemen bei der Testfallerstellung führen, beispielsweise durch fehlende oder kontroverse Testfälle.

Die eingenommene Perspektive beim BDD-Ansatz offenbart allerdings auch einige Schwächen, die im folgenden beschrieben werden und weshalb sich das Team KiM zu einem Einsatz eines weiteren Konzepts entschieden hat: Dem Domain-Driven Design.

Domain-Driven-Design in der Praxis

Um den eingenommenen Blickwinkel des Endkunden durch den BDD Ansatz im Softwareentwicklungsprozess zu erweitern und mögliche Grenzen zwischen IT und Business aufzuweichen, kombiniert das Team KiM während der Anforderungsanalyse den Behaviour-Driven-Development mit dem Domain-Driven-Design (DDD) Ansatz. Ausgehend vom Business eines Unternehmens, bis hin zu einer fachlichen Domäne (wie bspw. Vertrieb, Buchhaltung, CRM, etc.) bietet das strategic Design des DDD Ansätze zur Strukturierung und Aufteilung der Domäne. Das strategic Design, als einer- von vier Bestandteilen des DDD-Konzeptes, steht bei Team KiM im besonderen Fokus, da sich hier sehr schnell ein Mehrwert generieren lässt. Aufbauend auf der ubiquitous language bietet das strategic Design u.a. folgende Techniken bzw. Konzepte:

- Event Storming
- bounded Context
- Context Map

Event Storming

Nach der Bestimmung der Domäne können anhand des Event Storming die jeweilig relevanten Events innerhalb einer Domäne identifiziert und zugeordnet werden.

Am Beispiel eines sog. „Newsletter“-Projektes“ hat das Team KiM in enger Zusammenarbeit mit Product-Owner und dem Fachbereich die Events in der Domäne ermittelt und das Ergebnis festgehalten:



Event Storming - Newsletter

bounded Context

Die im Event Storming ermittelten Events lassen sich in bounded Contexts (sog. eigenständige Modelle) clustern. Folgende bounded Contexts wurden festgehalten:

- Kontaktverwaltung
- Registrierung
- Newsletter (An/-Abmeldung)
- Gewinnspiel
- Präferenzen

Der bounded Context (s. Bild unten) bietet leider noch keinen brauchbaren Systemüberblick unter Berücksichtigung des Kommunikations-, Daten- und Modellflusses. Diesen Part übernimmt die Context map, die den Zusammenhang zwischen Model und dem Kontext skizziert.

Context Map

Die Context Map bietet einen guten Ausgangspunkt für die nachfolgende Softwareentwicklung und die Testfallerstellung. Am Beispiel des Newsletter-Projektes hat das Team KiM folgende context map erstellt:



Context Map - Newsletter

Die Domain Events aus dem Event-Storming werden den einzelnen bounded contexts (blaue Post-its) zugeordnet. Darüber hinaus ist der Daten- und Modellfluss anhand von Pfeilen (rot + schwarz) skizziert.

Die Abkürzungen CF, OHS, SUP und CUS stellen sogenannte context map patterns dar, mit der Bedeutung:

- CF = Conformist (Das Downstream Team übernimmt das Modell des Upstream Teams)
- OHS = Open / Host Service (der Host bietet einen vordefinierten Satz an Services an)
- SuP = Supplier (Supplier-Customer i.S.v. Kunde-Lieferantenbeziehung)
- CuS = Customer (Supplier-Customer i.S.v. Kunde-Lieferantenbeziehung)

Die erstellte Context Map bündelt nicht nur die Ergebnisse der hier vorgestellten Techniken, sondern unterstützt zugleich die Testfallentwicklung. Hierbei hilft nicht nur die Visualisierung des Systems, sondern auch die Gruppierung der Events und die Modellzuordnung. Dadurch lassen sich präzisere bzw. eindeutig definierte Testfälle erstellen.

Vorteile

Den hier skizzierten hybriden Ansatz hat das Team KiM bereits erfolgreich in den Softwareentwicklungsprozess integriert und aufgrund folgender Vorteile zu schätzen gelernt:

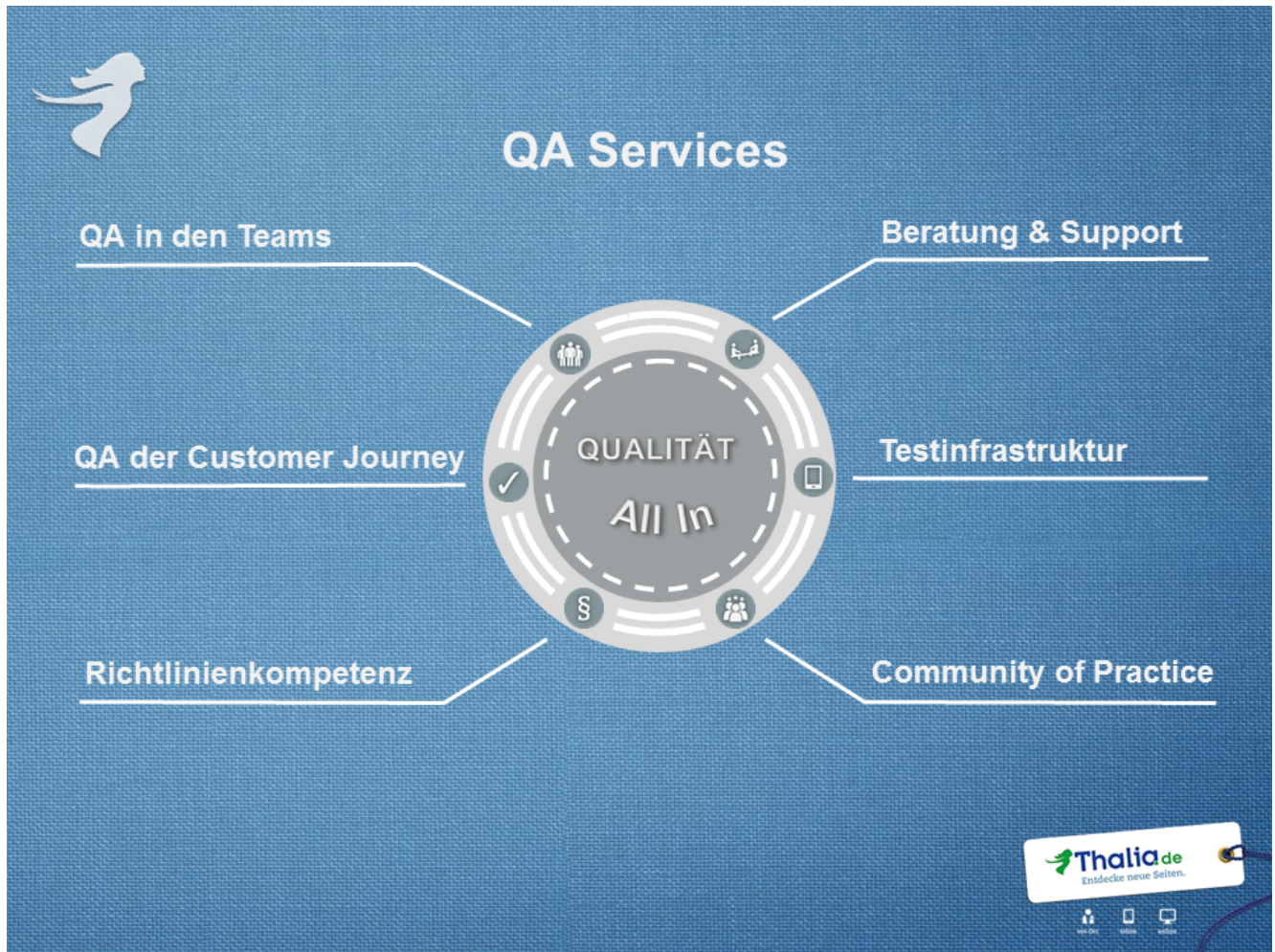
- Berücksichtigung mehrerer Perspektiven (Business und Endkunde)
- Integration der Testfallentwicklung
- Beteiligung aller relevanten Akteure am Entwicklungsprozess
- Reduzierung von Fehlinterpretationen bei Sachverhalten/Begrifflichkeiten (ubiquitous language)
- Reduzierung des Testaufwandes durch Testautomatisierung

Abschließend zu diesem Blog stellen sich dem Team die Fragen:

1. Wie ist eure Meinung zu dem o.g. Vorgehen ?
 2. Welche Erfahrungen habt ihr mit dem BDD- bzw dem DDD-Ansatz gemacht?
-

QA Services - Mittendrin statt nur am Ende

Der Bereich Thalia eCommerce Quality Assurance Services unterstützt die anderen Bereiche bei dem Ziel, gegenüber den Kunden und Kundinnen ein hochwertiges Produkt zur Verfügung zu stellen. Wir sind dabei entweder direkt einem Team zugeordnet oder in der zentralen QA tätig. Durch die permanente Weiterentwicklung von Produkten und Prozessen werden wir stets gefordert, neue passende Lösungen zu entwickeln. Derzeit stehen bei uns sechs Themen im Fokus:



QA in den Teams

Wir sind als QA-Lead direkter Bestandteil eines cross-functional Teams und nehmen vielfältige Aufgaben wahr:

- Abstimmung der teaminternen Qualitätskriterien
- Erster Ansprechpartner zum Thema Qualitätssicherung und Test
- Themen zur Testautomation im Team bündeln und vorantreiben
- und natürlich das normale Testgeschäft

Beratung und Support

Wir beraten die einzelnen Teams und Fachbereiche bei Themen wie Testfallermittlung, Testkonzeption, Tooleinsatz etc.

QA der Customer Journey

Wir prüfen zyklisch, dass die Customer Journey für unsere Kunden keine Stolpersteine enthält. Die Prüfung durch automatisierte Tests bauen wir stetig

aus.

Testinfrastruktur

Wo und wann kann ich testen? Woher bekomme ich Testdaten? Womit kann ich Lasttests ausführen? Das sind einige Fragestellungen, die wir beantworten. Und falls wir die Antwort nicht kennen, sorgen wir für Lösungen.

Richtlinienkompetenz

Wir definieren Spielregeln, damit die Flexibilität und hohe Innovationsgeschwindigkeit der agilen Produktentwicklung nicht zu Lasten von Stabilität und Qualität geht.

Community of Practice

Wir tauschen uns zu spannenden Themen im Bereich der QA aus, um voneinander zu lernen und besser zu werden.