

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.

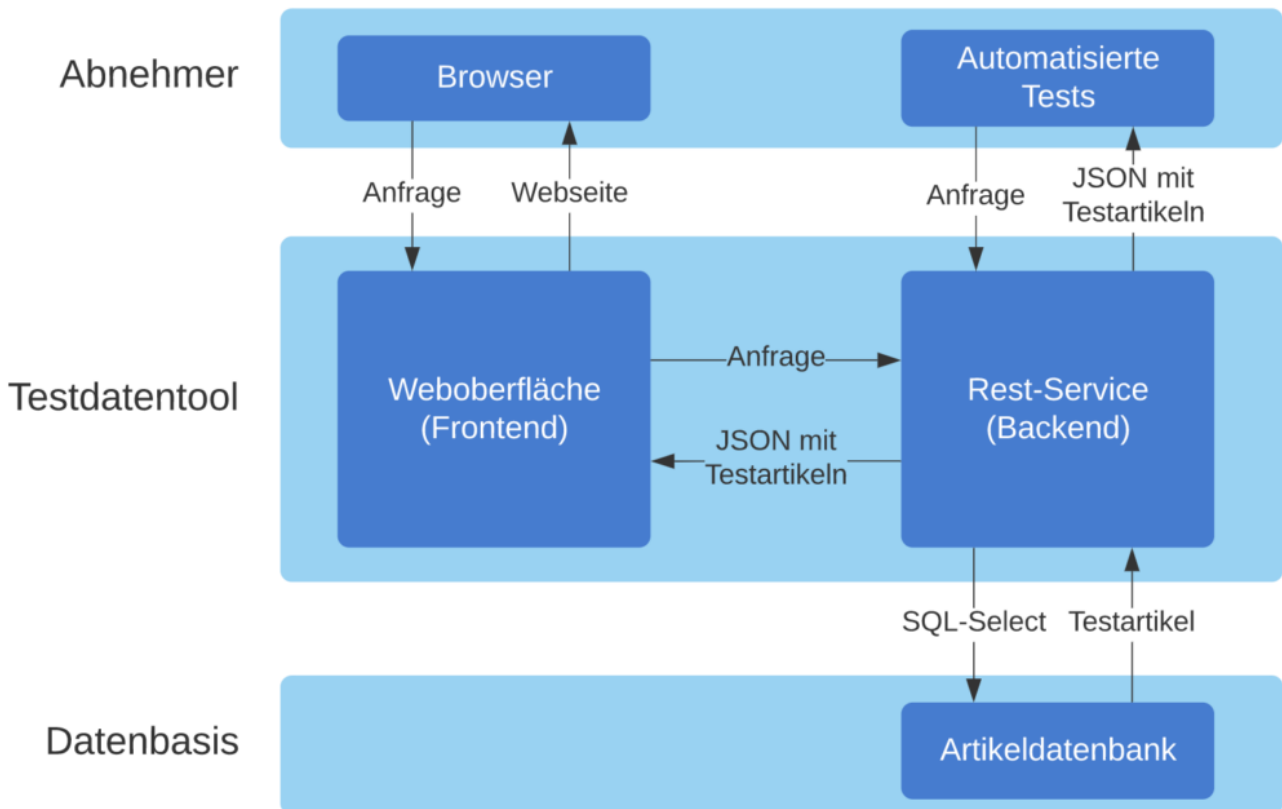
Über viele Jahre hinweg haben wir im Team AiM für andere Teams Testartikel über individuelle SQL-Abfragen herausgesucht und über die Herausgabe der Artikel-ID oder EAN bereitgestellt. Dieser Prozess hat sich mit der Zeit als sehr ineffizient herausgestellt. Um diesen Prozess zu optimieren haben wir das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

Vorbesteller

Hoertyp

Medien

Bilder

Hoerprobe

Leseprobe

Trailer

Preis

Streichpreis

Preisbindung

ThaliaClub

Verkaufspreis

Abo

Abo - Ebook

Abo - Hoerbuch Download

Slider auf Artikeldetailseite

Lernhilfen zu diesem Titel

Weitere Baende von

Weitere Serientitel zu

mehr-von-Suche

mehr-von-Autor

FSK-ARTIKEL

Marke
thalia.de

Umgebung
Integration

Anzahl
100

SUCHE

✓

SQL

COPY ARTIKELIDS

COPY EANS

COPY MATNRS

Altersfreigabe

Freigegeben ab 6 Jahren

artikelid	ean	matnr	titel	preis	sortiment
1788013	0743218997495	A1004922755	Asterix & Obelix - Mission Kleopatra	9900	Film
6722291	0828766845198	A1001412833	In 80 Tagen um die Welt	999	Film
13566646	0828767673691	A1001372839	Good Woman - Ein Sommer in Amalfi	999	Film
15657456	0886972973098	A1001621513	Unsere Erde	1299	Blu-Ray
17236799	0886973825792	A1007631486	Wächter der Wüste	899	Film
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 Ueberschrift 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           // Eintraege, die in der Combobox angezeigt werden. Die Variable "name" legt fest, wie die Eintraege
24           // in der Combobox heissen. Die Variable "value" definiert, wie genau das Backend fuer den Eintrag
25           // aufgerufen wird. Waehlt der Nutzer in der Combobox den Eintrag "Freigegeben ab 18 Jahren", dann wird
26           // das Backend mit dem Query-Parameter ?alter=18 aufgerufen. Der Name des Query-Parameters (hier: alter)
27           // wird ueber die Variable "queryParamsName" festgelegt.
28           entries: [
29             {name: 'Freigegeben ab 6 Jahren', value: '6'},
30             {name: 'Freigegeben ab 12 Jahren', value: '12'},
31             {name: 'Freigegeben ab 16 Jahren', value: '16'},
32             {name: 'Freigegeben ab 18 Jahren', value: '18'}
33           ],
34           // Verweist auf den Eintrag, den der Nutzer in der Combobox ausgewaehlt hat. Diese Variable ist eine
35           // Referenz auf die Variable "value" in dem Array "entries". Hat der Nutzer den Eintrag
36           // "Freigegeben ab 18 Jahren" ausgewaehlt, dann traegt die Variable "selectedValue" den Wert "18". Die
37           // Variable "selectedValue" wird immer dann aktualisiert, wenn der Nutzer ueber die Combobox eine neue
38           // Auswahl trifft. Die Variable wird zum ersten Mal in der Methode created() mit einem Wert versehen.
39           selectedValue: '',
40           // Verweist auf den Eintrag, der in der Combobox zu Beginn ausgewaehlt ist. Diese Variable ist eine
41           // Referenz auf die Variable "value" in dem Array "entries".
42           defaultValue: '18',
43           // Beschriftung der Combobox.
44           label: 'Altersfreigabe',
45           // Name des Query-Parameters, ueber den die Auswahl in der Combobox an das Backend uebergeben wird (hier: ?alter).
46           queryParamsName: 'alter'
47         ]
48       }
49     }
50   }
51 </script>
52
53
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.

The screenshot displays the Swagger UI for a service named 'Testdatentool'. The header is green with the Swagger logo, a version dropdown set to 'default (/v2/api-docs)', and an 'Explore' button. Below the header, the title 'Testdatentool' is followed by the description 'Rest-Ressourcen des Testdatentools.' and 'Created by Thalia'. A section titled 'rest-service : Rest Service' includes links for 'Show/Hide', 'List Operations', and 'Expand Operations'. The main content is a list of 18 REST API endpoints, each with a 'GET' method, a URL path containing a placeholder '{mandantId}', and a corresponding operation name.

Method	URL	Operation Name
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.