Java Entwickler / Software Developer (m/w/d)

Bücher eröffnen Welten – wir tun es ihnen gleich. Als Deutschlands größter Sortimentsbuchhändler verknüpfen wir echte Kundennähe mit richtungsweisender Digitalinnovation. Wir denken das Buch als Erlebnis – ob vor Ort, online, als App oder über den e-Reader. Und versammeln rund 4000 Expertinnen und Experten mit einer Mission: Ein Kulturgut auf allen Kanälen erfahrbar zu machen.

Für unsere Zentrale in Münster suchen wir ab sofort einen

Java Entwickler / Software Developer (m/w/d)

Welche Aufgaben erwarten dich:

- Du betreibst die eigenverantwortliche Weiterentwicklung unserer **eCommerce-Plattform** mit Spaß an Software-Engineering und Clean Code in einem agilen Produktteam
- Du betreust als **Full-Stack-Entwickler** die ganze Kette vom Datenbankzugriff/ Serviceaufruf bis zur Gestaltung und Auslieferung des Frontends

- Du sicherst die Qualität durch **Code Reviews**, Unit-/Integrations-/End-To-End-Tests sowie die Auslieferung über **Jenkins** und **Git**
- Du misst anhand von **Kennzahlen** die Produktivität, Performance und Qualität eurer Software, um sie stetig zu verbessern
- Du entwickelst zusammen mit dem Product Owner die beste Lösung für unsere **Kunden**

Was bringst du mit:

- Mehrjährige Berufserfahrung in der Software-Entwicklung mit Java, gerne im Bereich **eCommerce**
- Sehr gute Kenntnisse aktueller Frameworks und Konzepte (z.B. **Spring** MVC/Spring Boot, JPA, Hibernate, REST API, Microservices, Self-contained Systems) und Entwicklungswerkzeuge (Git, Maven, Jenkins)
- Know How im Umgang mit dem **Apache Tomcat**, Apache Webserver, Linux, Docker und Co
- Erfahrungen mit **Datenbanksystemen** (MySQL, Postgres) und **Frontend-Technologien** (HTML5, JavaScript, CSS3)
- DDD, TDD, BDD – die Abkürzungen sind Dir nicht unbekannt und idealerweise hast Du hier bereits praktische Erfahrung gesammelt
- Konzeptionelle Stärke, analytisches Denken, Kommunikationsfähigkeit und **sehr gute Deutschkenntnisse**

Diese Benefits bieten wir dir:

- Ein **motiviertes und selbstorganisiertes Produktteam**, dem ein tolles Miteinander und Freude an der Arbeit wichtig ist
- Ein modernes Arbeitsumfeld und die Möglichkeit neue Technologien oder Vorgehensmodelle einzuführen
- Ein **aktueller Technologie Stack** (Java, Apache Tomcat, Spring, Spring Boot, JPA, Maven, Git, Jenkins, Nexus, Gitlab, Sonar, Jira, Eclipse, IntelliJ, Postgres, MySQL, RabbitMQ, Apache Kafka, Handlebars, Grafana, Graylog)
- Besuch und Mitgestaltung von **Fachkonferenzen, Hackathons** und die Möglichkeit auf dem **Thalia Tech Blog** zu veröffentlichen

- Ein familienfreundliches Unternehmen mit **flexiblen Arbeitszeiten** und 30 Tagen Urlaub im Jahr sowie eine Vielzahl an Sozialleistungen

Haben wir dein Interesse geweckt?

Dann bewirb dich unter Angabe deines Gehaltsrahmens und deines frühestmöglichen Eintrittstermins über unser Online-Portal oder über bewerbungen@thalia.de!

Thalia Bücher GmbH | Deine Ansprechpartnerin: Stefanie Klein

Java Entwickler eCommerce/PIM (m/w/d)

Bücher eröffnen Welten – wir tun es ihnen gleich. Als Deutschlands größter Sortimentsbuchhändler verknüpfen wir echte Kundennähe mit richtungsweisender Digitalinnovation. Wir denken das Buch als Erlebnis – ob vor Ort, online, als App oder über den e-Reader. Und versammeln rund 4000 Expertinnen und Experten mit einer Mission: Ein Kulturgut auf allen Kanälen erfahrbar zu machen.

Für unsere Zentrale in Münster suchen wir ab sofort einen

Java Entwickler eCommerce/PIM (m/w/d)

Welche Aufgaben erwarten dich:

- Du betreibst die eigenverantwortliche Weiterentwicklung unserer **eCommerce-Plattform** mit Spaß an Software-Engineering und Clean Code in einem agilen Produktteam
- Du betreust als **Full-Stack-Entwickler** die ganze Kette vom Datenbankzugriff/Serviceaufruf bis zur Gestaltung und Auslieferung des Frontends
- Du sicherst die Qualität durch **Code Reviews**, Unit-/Integrations-/End-To-End-Tests sowie die Auslieferung über **Jenkins** und **Git**
- Du misst anhand von **Kennzahlen** die Produktivität, Performance und Qualität eurer Software, um sie stetig zu verbessern
- Du entwickelst zusammen mit dem Product Owner die beste Lösung für unsere **Kunden**

Was bringst du mit:

- Mehrjährige Berufserfahrung in der Software-Entwicklung mit Java, gerne im Bereich **eCommerce/PIM Systeme**
- Sehr gute Kenntnisse aktueller Frameworks und Konzepte (z.B. **Spring** MVC/Spring Boot, JPA, Hibernate, REST API, Microservices, Self-contained Systems) und Entwicklungswerkzeuge (Git, Maven, Jenkins)
- Know How im Umgang mit dem **Apache Tomcat**, Apache Webserver, Linux und Co.
- Erfahrungen mit **Datenbanksystemen** (MySQL, Postgres) sowie **Frontend-Technologien** (HTML5, JavaScript, CSS3)
- Konzeptionelle Stärke, analytisches Denken, Kommunikationsfähigkeit und **sehr gute Deutschkenntnisse**

Diese Benefits bieten wir dir:

- Ein motiviertes und selbstorganisiertes **Produktteam,** dem ein tolles Miteinander und Freude an der Arbeit wichtig ist
- Ein modernes Arbeitsumfeld und die Möglichkeit **neue Technologien** oder Vorgehensmodelle einzuführen
- Ein **aktueller Technologie Stack** (Java, Apache Tomcat, Spring, Spring Boot, JPA, Maven, Git, Jenkins, Nexus, Gitlab, Sonar, Jira, Eclipse, IntelliJ, Postgres, MySQL, RabbitMQ, Apache Kafka, Handlebars, Grafana, Graylog)
- Hohe **Gestaltungsspielräume** und viele spannende Themen
- Besuch und Mitgestaltung von Fachkonferenzen, Hackathons und die Möglichkeit auf dem Thalia Tech Blog zu veröffentlichen
- Ein familienfreundliches Unternehmen mit **flexiblen Arbeitszeiten und 30 Tagen Urlaub** im Jahr sowie eine Vielzahl an **Sozialleistungen**

Haben wir dein Interesse geweckt?

Dann bewirb dich unter Angabe deines Gehaltsrahmens und deines frühestmöglichen Eintrittstermins über unser Online-Portal oder über bewerbungen@thalia.de!

Thalia Bücher GmbH | Deine Ansprechpartnerin Stefanie Klein

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]thalia.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: Waehle 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. Zalenium (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 Zalenium 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

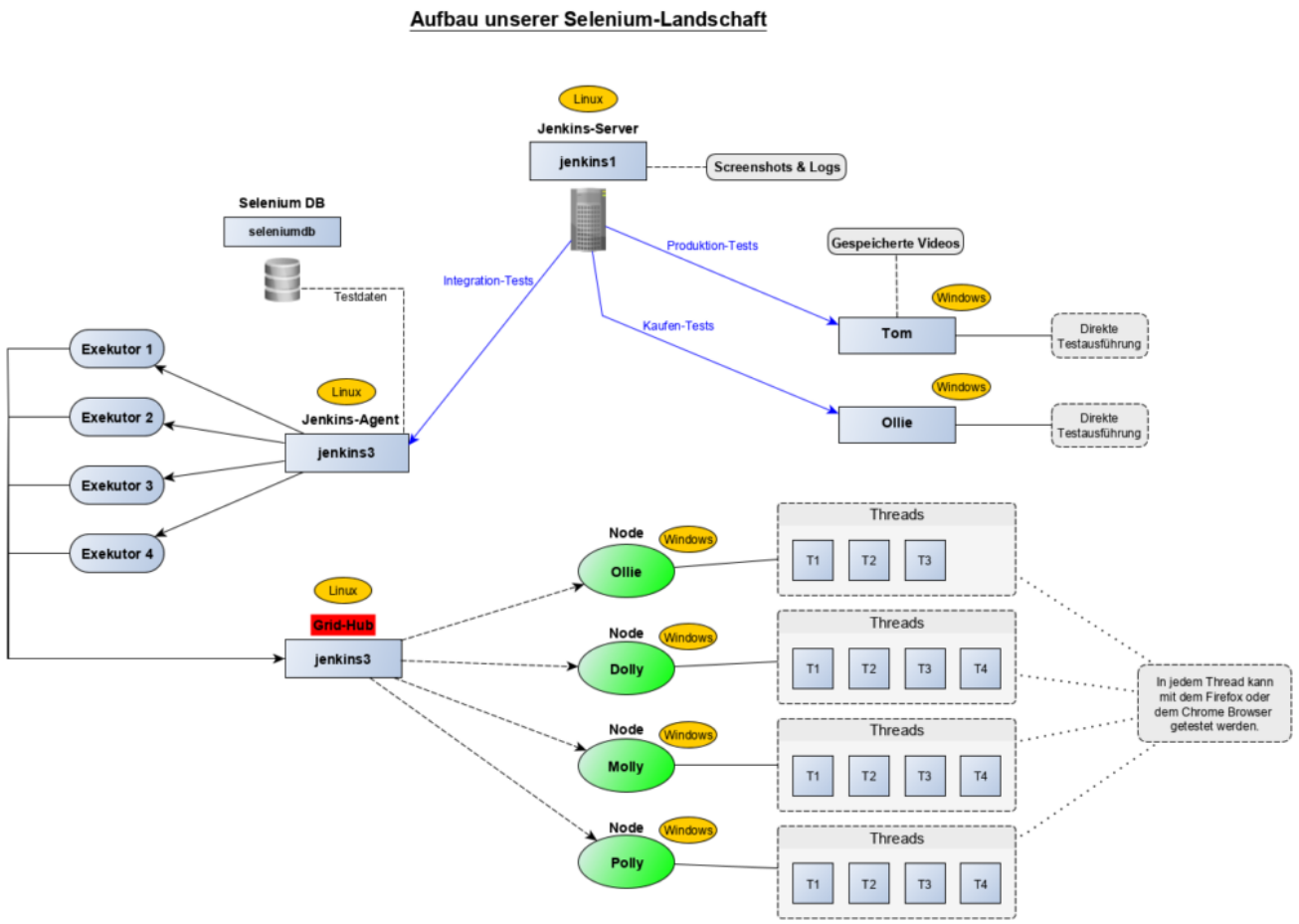
Resource	Status	Labels	Action
GridNode1	LOCKED by [redacted] #971	GridNode	Unlock
GridNode2	LOCKED by [redacted] #971	GridNode	Unlock
GridNode3	LOCKED by [redacted] #428	GridNode	Unlock
GridNode4	LOCKED by [redacted] #428	GridNode	Unlock
GridNode5	LOCKED by [redacted] #428	GridNode	Unlock
GridNode6	LOCKED 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.



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

Continous Integration in der App-Entwicklung

Am Standort Berlin entwickeln wir für unseren B2B-Partner Douglas unter anderem die Kunden-App [2], [3]. Einhergehend mit dem Ausbau der App-Entwicklungsaktivitäten haben wir in den letzten Monaten den CI-Ansatz überarbeitet. Ein zusätzliches Team sollte am selben Produkt – der KundenApp – mitarbeiten und die App sollte öfter veröffentlicht werden. Mit Methoden und Tools aus dem Bereich Continous Integration wollten wir dafür sorgen weiter zuverlässig und mit hoher Qualität zu liefern. Und das natürlich automatisiert. Neben der Technik geht es auch um die Teams. Wie sind sie vorgegangen und welche Hürden haben sie genommen.

Komplexität und Feedback

Je mehr Personen gleichzeitig an einem Produkt entwickeln, desto größer wird die Wahrscheinlichkeit, dass unbeabsichtigte Seiteneffekte auftreten. Gleichzeitig steigt der Umfang der App, da wir konstant neue Funktionen hinzufügen und bestehende Funktion ändern. Um die Komplexität weiter zu beherrschen, ist schnelles Feedback zu Änderungen ein entscheidender Faktor, um Probleme schnell zu korrigieren. Wie wäre es, automatisiert ein Feedback nach jedem Commit zu bekommen und darauf nur kurze Zeit warten zu müssen? Genau hier setzten wir an.

Schnelles Feedback erhalten wir durch den Einsatz von **Feature-Toggles** und durch die Ausführung von **automatischen Tests** im CI-Prozess.

Feature Toogles

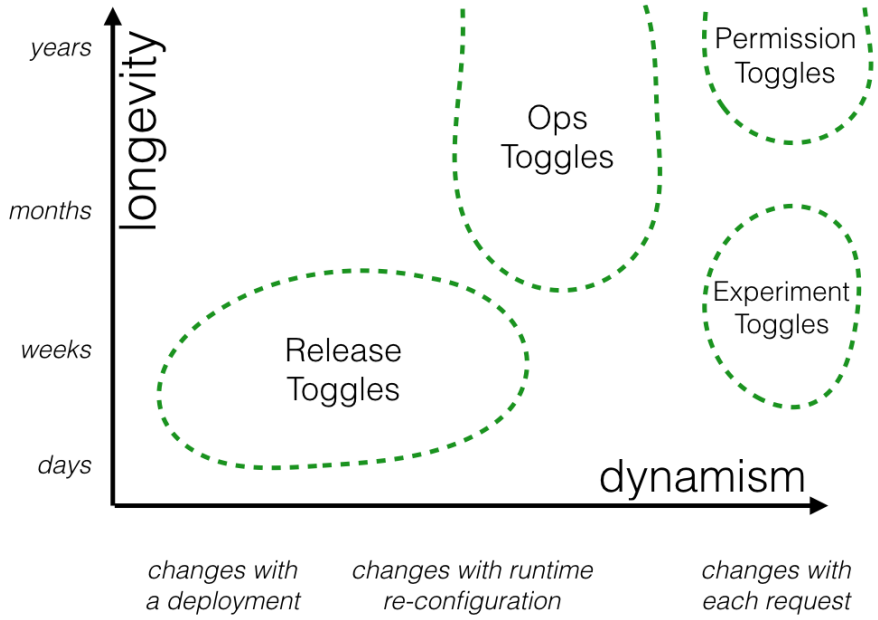
Feature Toogles ermöglichen es uns Codeänderungen aller Entwickler kontinuierlich in einen gemeinsamen Integration-Branch zusammenzuführen. Und das auch, wenn Features noch nicht fertig bzw. für den Kunden nicht sichtbar

sein sollen. In der Vergangenheit haben wir solche Features für mehrere Tage, manchmal Wochen, in separaten Branches entwickelt und erst am Ende der Entwicklung in den Integration-Branch gemergt. Das Feedback kam entsprechend spät. Traten Probleme auf, war es durch die Vielzahl der Änderungen mitunter schwer festzustellen, welche konkrete Änderung zum Bruch geführt hat. Die Integration hatte das Potential unsere Zeitplanung empfindlich zu stören. Diese Bing-Bang-Szenarien sollen durch Toggles und kontinuierliche Integration abgefedert werden.

Toggles und Diskussionen

Der Einsatz der Feature-Toggles wurde im Team intensiv diskutiert, denn die Einführung erhöht erstmal die Komplexität – und liebgewonnen Pfade, wie die isolierte Arbeit im Feature-Branch, standen auf einmal auf dem Prüfstand. Es gab diverse Pros und Cons. Auch musste ein gemeinsames Verständnis beim Thema Toggle-Mechanik erarbeitet werden. In Bezug auf Dynamik und Langlebigkeit der Toggles gab es unterschiedliche Auffassungen, da viele schon mal irgendetwas mit Toggels gemacht hatten.

Wir haben uns am Ende auf die Nutzung von Feature-Toggels zur Entwicklungszeit – auch Release-Toggles genannt – geeinigt. Sie werden für den Zeitraum weniger Tage/Wochen genutzt. Ist das Feature fertig entwickelt, wird der Toggle aus dem Code komplett entfernt. Der Artikel auf martinfowler.com [1] sei dem interessierten Leser an der Stelle empfohlen.



https://martinfowler.com/articles/feature-toggles.html
2 bis 3 Feature-Toggle sind im Durchschnitt parallel im Einsatz. In unserem Jenkins haben wir durch einen manuellen Schritt in der Build-Pipeline die Möglichkeit geschaffen, einen einzelnen Toggle zu aktivieren und somit App-Artefakte für Features (apk, ipa) für das Testing zu bauen. Ist das Feature komplett entwickelt, wird der Toggle aus dem Code entfernt. Mit dem nächsten App-Release ist das Feature dann für den Kunden sichtbar.

Was ist ein Feature Toggle?

Ein Feature Toggle ist eine Progammiertechnik, die es erlaubt ein Feature oder eine Funktion vor Kunde ein- bzw. auszuschalten. Also die Sichtbarkeit zu ändern. Entwicklungsteams aktivieren Features beispielsweise um noch nicht fertige Funktionen integrieren und testen können. Ist ein Feature fertig, kann es ohne großen zusätzlichen Merge-Aufwand veröffentlicht werden, da die Arbeit in separaten Branches entfällt. Feature Toggles können auch dafür genutzt werden, die Sichtbarkeit von Funktionen zur Laufzeit der Anwendung zu ändern. Z.B. im Rahmen von A/B Tests oder wenn die Sichtbarkeit zu einer bestimmten Zeit geändert werden soll.

Toogles im Code

iOS

Unter iOS wird ein Feature in der App über eine Environment-Variable in der **Launch-Konfiguration** aktiviert (z.B.: USE_NATIVE_PRODUCT_LIST = 1). Im Code wird dann an relevanten Stellen über eine Abfrage entschieden, ob bestimmte Codestellen zur Ausführung kommen oder nicht.

```
if toggleRouter.isNativeProductListEnabled() {
  // Feature Code
}
```

Android

Es gibt ein Interface, in dem alle Toggles als Methoden definiert werden. Diese Methoden werden mit Java-Annotations annotiert und geben immer ein Boolean zurück – TRUE für Feature aktiv, FALSE für nicht aktiv.

```
@ReleaseToggle(BuildConfig.FEATURE_PRODUCT_LIST)
fun isProductListEnabled(): Boolean
```

Eine eigens dafür entwickelte Library mit einem **Annotation-Prozessor** wird während der Build-Phase ausgeführt: Dieser schaut in einer Konfigurations-Datei (json) nach, ob das jeweilige Feature getoggelt werden soll. Wenn das Feature eingeschaltet werden soll, muss der String, der sich in der Annotation befindet, hier eingetragen werden.

```
[
    "FEATURE_PRODUCT_LIST"
]
```

Der Prozessor baut dann jeweils die Implementation für das Interface zusammen. In diesem Fall würde die implementierte Methode TRUE zurück liefern. Wäre der String FEATURE_PRODUCT_LIST nicht in der Datei, wäre es FALSE.

So kann man auf jedem lokalen Rechner die Features beliebig ein- und ausschalten. Auf dem Jenkins kann man das genauso machen, hier editieren wir nicht manuell die Datei sondern sagen ihm über das **Blue Ocean Plugin**, welches Feature getoggelt werden soll.

Und die jeweiligen Code-Stellen togglen wir, in dem wir die Interface-Implementation prüfen:

```
if  (ReleaseToggleConfiguration_Impl().isProductListEnabled())
{
     // Mach was mit der Product list
}
```

Ein gemeinsames Traffic Light für Build und Test

Eine weitere zentrale Komponente im CI-Prozess stellt die Testautomatisierung dar. Das Feedback, dass Build und Test erfolgreich nach einem Commit auf Integration durchgelaufen sind, wird durch eine Ampel visualisiert. Diese ist für jedem im Team sichtbar. Ist sie rot, ist das **gesamte Team angehalten** den Grund zu ermitteln und die Ampel wieder auf „grün zu bekommen“. Also Fehler zu korrigieren, Tests oder die Automatisierung anzupassen.

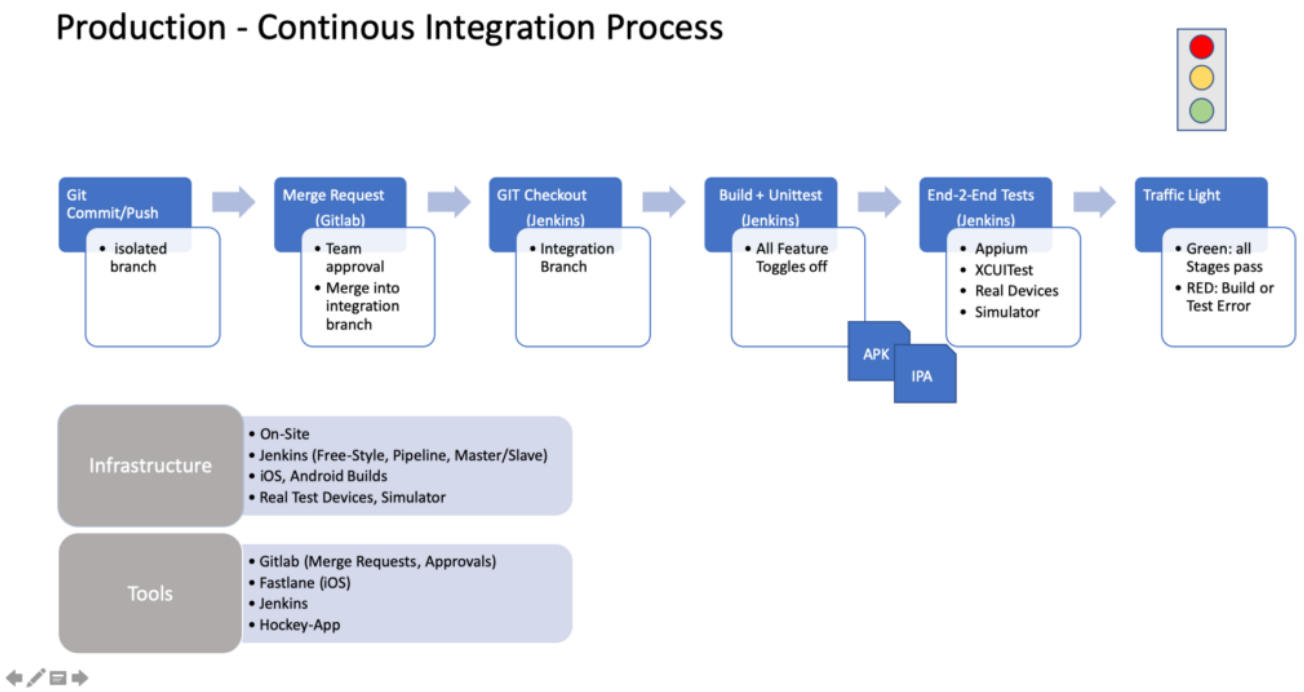
CI-Build-Status für Android und iOS

Die Tests sind eine Kombination aus Unit-Tests und End-2-End-Tests (Akzeptanztests). Die End-2-End-Tests laufen auf echten Geräten bzw. Simulatoren im Zusammenspiel mit dem Backend.

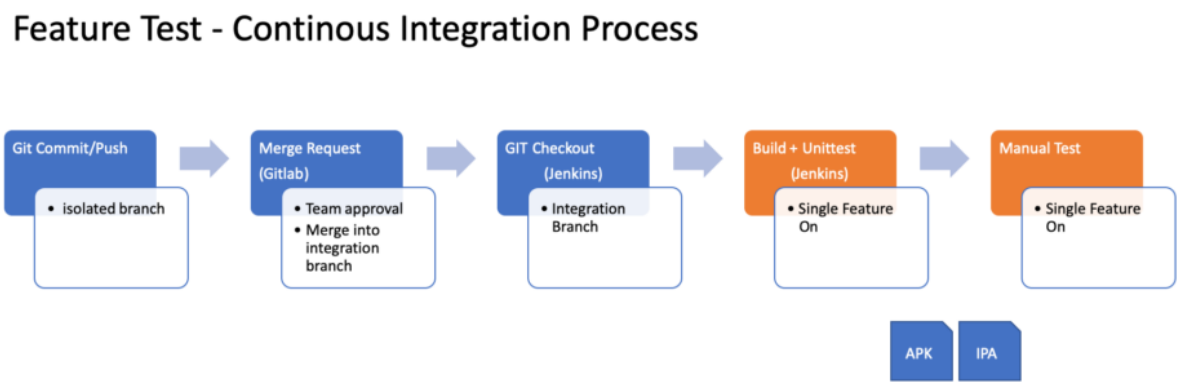
Continous Integration Process

Unser CI Prozess sieht wie folgt aus:

Nach dem Approval der Codeänderung in Gitlab und Integration in den Integrations-Branch baut der Jenkins das App-Artefakt, führt die Unittests aus und startet die End-2-End Tests. Das kombinierte Ergebnis aus Build/Unitests und End-2-End Test wird auf der Ampel dargestellt.



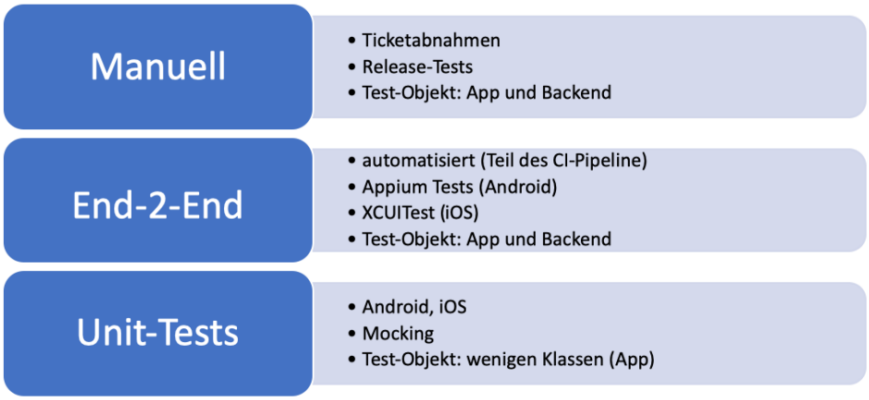
Für den Test eines Features, dass sich noch in der Entwicklung befindet, wird ein Feature-Toggle manuell im Jenkins aktiviert, die App gebaut und die Unittests ausgeführt. Die End-2-End Tests werden zum jetzigen Zeitpunkt noch nicht ausgeführt. Zum einen müssten die Tests für das Feature bereits angepasst und erweitert sein. Das ist noch nicht der Fall. Ein weiterer Grund sind die noch fehlenden Ressourcen in Form von Hardware und Testgeräten. Ein nächster Schritt.



Learnings zu Tools, Integrationslevel und Verantwortung

Auf drei Learnings möchte ich an der Stelle speziell eingehen.

Der Leser soll dazu wissen, dass unsere Entwicklungsteams Cross-funktional aufgebaut sind. Ein Entwicklungs-Team besteht aus iOS-Entwicklern, Android-Entwicklern, Backend-Entwicklern und einem Quality Engineer. Die Backend-Entwickler sitzen an einem anderen Standort und sind wie wir Dienstleister für den B2B-Partner. Folgende Tests führen wir zur Zeit durch:



Testtypen

Beim Tooling ging es vor allem um die Wahl des **Testautomatisierungstools**. Da die QA in der Vergangenheit auf Appium gesetzt hat, wollten wir die Technik auch für unsere CI-Tests im gesamtem Team nutzen (Dev+QA). Bisher wurden die Appium Test ausschließlich von der QA geschrieben und waren nicht in einem gemeinsamen CI-Prozess zusammen mit dem Build integriert. Es stellte sich heraus das die Akzeptanz des Tools für iOS unter den Entwicklern sehr gering war. Stabilität, Funktionsumfang, Integrierbarkeit und Ausführungsgeschwindigkeit überzeugten nicht. Unsere Teams haben sich daher entschieden, für iOS die End-2-End Tests auf Basis von XCUITest neu zu schreiben. Für den Android Bereich setzen wir vorerst weiter auf Appium.

Ein weiteres Learning gab es beim **Integrationslevel**. Unsere End-2-End Tests weisen ein hohes Integrationslevel auf: die App wird im Zusammenspiel mit dem Backend getestet. Fehler im Backend oder eine schlechte Verbindungsqualität können dazu führen, dass Tests fehlschlagen, obwohl die App „korrekt“ funktioniert. Die Ampel zeigt rot, obwohl „mit der App alles in Ordnung ist“. Flaky Tests bzw. instabile Tests senken die Aussagekraft der Ampel deutlich und führen dazu, dass die Teams einer roten Ampel weniger Aufmerksamkeit schenken. Neben dem End-2-End Test planen wir daher einen zusätzlichen Testtyp einzuführen, der vom Integrationslevel zwischen Unittest und End-2-End Test liegt. Ziel ist es die App ohne Backend zusätzlich zu verifizieren. Dafür sollen Backend-Responses „gemockt“ und die Tests auf der UI-Ebene durchgeführt werden. Die Test sollen eine Ergänzung zu den End-2-End Tests werden.

Beim Thema Verantwortung gehen wir mit dem CI-Ansatz ebenfalls neue Wege. Das Ergebnis aus Build + Test in einem gemeinsamen Ampelstatus zu visualisieren und damit jeden im Entwicklungs-Team zu aktivieren, Probleme im **Test oder Build** zu analysieren, erfordert, dass sich Entwickler mehr mit dem Thema Testing und sich die QA mehr mit der Automatisierung auseinandersetzt. Dieser Veränderungsprozess benötigt Zeit und den Willen aller Beteiligten sich zu verändern. In unserem Produktteams ist diese Veränderung explizit gewünscht und alle im Team sind angehalten für den Prozess **Verantwortung** zu übernehmen und ihn aktiv weiterzuentwickeln.

Ausblick

Im Bereich Testing steht der Ausbau der Testautomatisierung für die End-2-End

Tests und die Einführung der zusätzlichen Test-Verifikationsstufe für die Android-App mit Espresso als UI-Testing Tool an.

Um die Qualität zu steigern, möchten wir automatisiert statische Codeanalysen durchführen und Metriken wie beispielsweise die technische Schuld ermitteln.

Verweise

[1] https://martinfowler.com/articles/feature-toggles.html

[2] Douglas App iOS: https://itunes.apple.com/de/app/douglas-parfüm-kosmetik/id394685685?mt=8

[3] Douglas App Android: https://play.google.com/store/apps/details?id=com.douglas.main&hl=de