Hackathon 2023: Gute Ideen sind nicht gern allein!

Die diesjährigen Thalia Innovation Days & Hackathon hat Mitarbeitende aus drei Standorten vereint, neue Wege zum Jedi-Meister hervorgebracht und neue Rekorde gebrochen.

Nach den Erfolgen der Thalia Hackathon & Innovation Days 2022 und der Umsetzung der Idee des Live-Commerce Teams wurde dieses Jahr noch mehr Fokus auf Innovation gesetzt. Am 6. und 7. September haben 73 Teilnehmende in interdisziplinären Teams insgesamt 7 Ideen gepitcht und ihre ersten Prototypen direkt vorgestellt.

Die Teams hatten genügend Ansporn, um die gegebene Zeit für die Ausarbeitung ihrer Ideen möglichst effektiv zu nutzen. Dieses Jahr gab es insgesamt drei Preise zu gewinnen. Die sechsköpfige Fachjury hat einen Preis für die innovativste Idee und deren Umsetzung verliehen. Hierbei wurde das Thema einerseits danach bewertet, wie hoch sein Wert für die Kunden und Kundinnen von Thalia ist. Andererseits sollte das Thema auch realistisch umsetzbar sein.

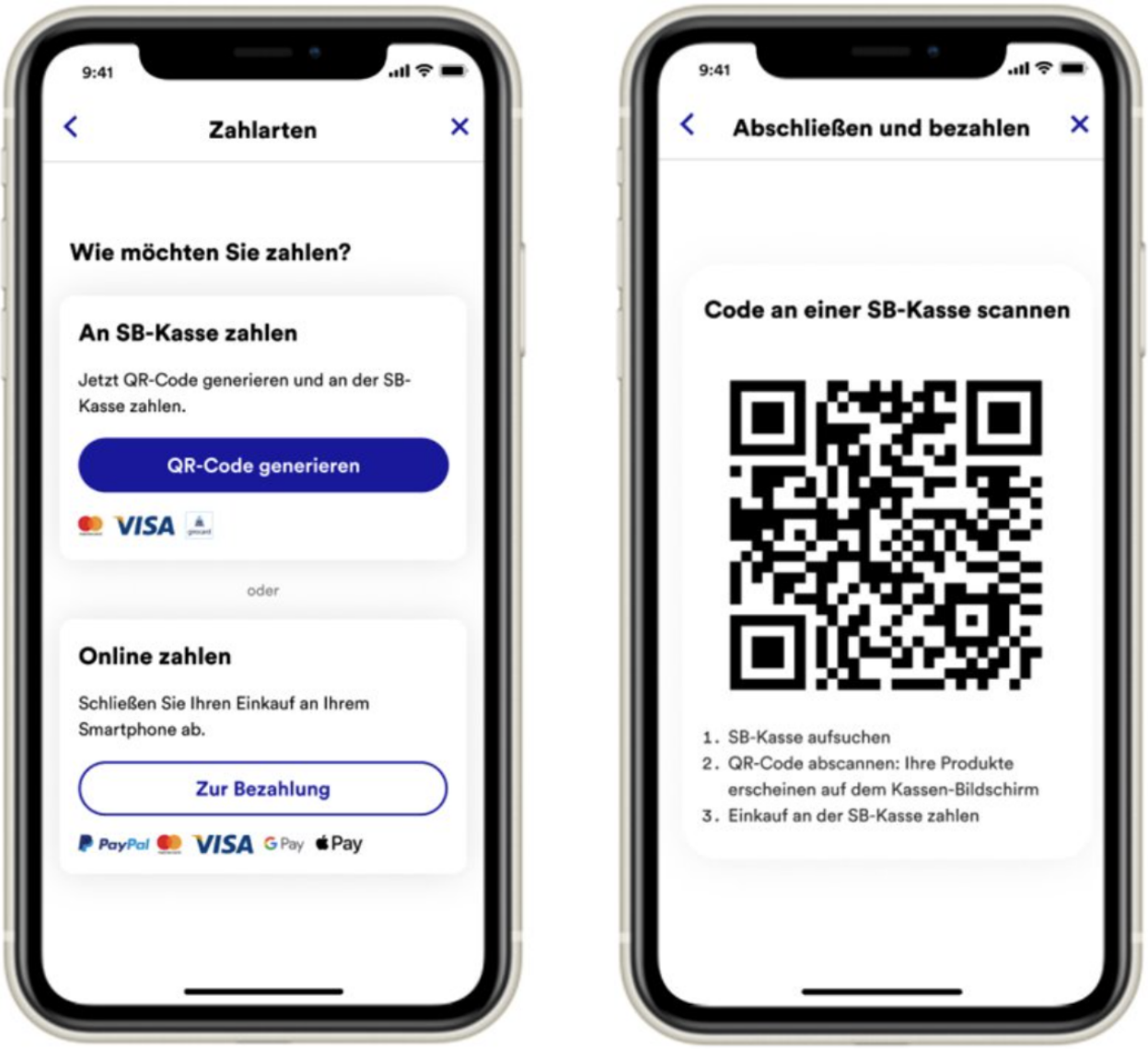
Beim zweiten Preis dagegen wurde mehr auf die Originalität der Idee geachtet. Ein weiteres Kriterium war das Aufgreifen von aktuellen technologischen Trends bei der Umsetzung. Zusätzlich wurde das Publikum auch dieses Jahr eingebunden und durfte für sein Lieblingsthema stimmen.

Gewinner des Innovation-Preises: Team Krasse Kasse mit „Scan & Go meets SCO“

Niemand mag lange Warteschlangen an der Kasse. Deswegen bietet Thalia Scan & Go an, und erlaubt damit Kund*innen, die in der Buchhandlung unterwegs sind, ihre geliebten Bücher ohne Anstehen mit der Thalia App zu zahlen. Das Team „Krasse Kasse“ hat gemeinsam an der bestehenden Lösung weitergearbeitet, um auch Kund*innen, die auf ein Kundenkonto verzichten, oder z.B. mit ihrer

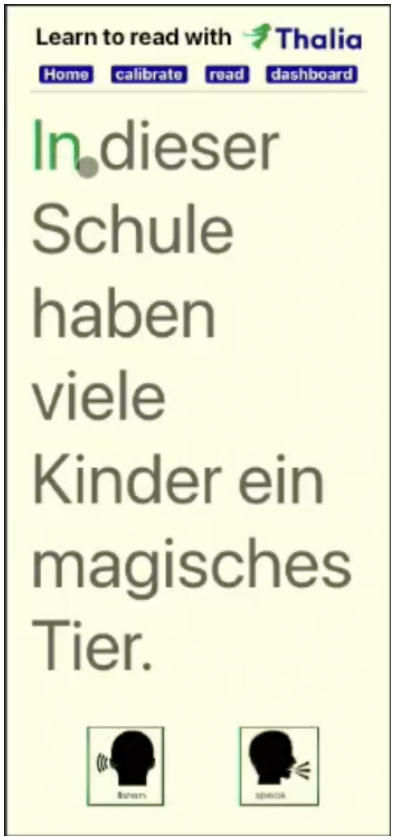
Girocard zahlen möchten, Scan & Go anzubieten.

Das Vorgehen kennt man aus anderen großen Handelsketten: man scannt einfach die Artikel mit der Thalia App, und wählt die Option „an der Kasse zahlen“ aus. Aus dem digitalen Warenkorb wird ein QR-Code generiert, das an der Self-Checkout Kasse gescannt werden kann. Somit lassen sich die Artikel blitzschnell in die Kasse eintragen, und die Kundin darf nun mit allen Kartentypen zahlen, als Rechnung erhält sie den Kassenbonn.



Damit läßt sich ein Self-Checkout Vorgang im Durchschnitt um 25% schneller abwickeln, was neben den Kunden natürlich auch Thalia freut, da damit die mobilen Kassen weiter reduziert werden können. Neben einer voll funktionalen Demo hat das Team bereits Pläne für einen möglichen Rollout und dessen Vermarktung vorgestellt. Eine wirklich krasse Leistung!

Gewinner des Hackathon-Preises: Learn to read with Thalia

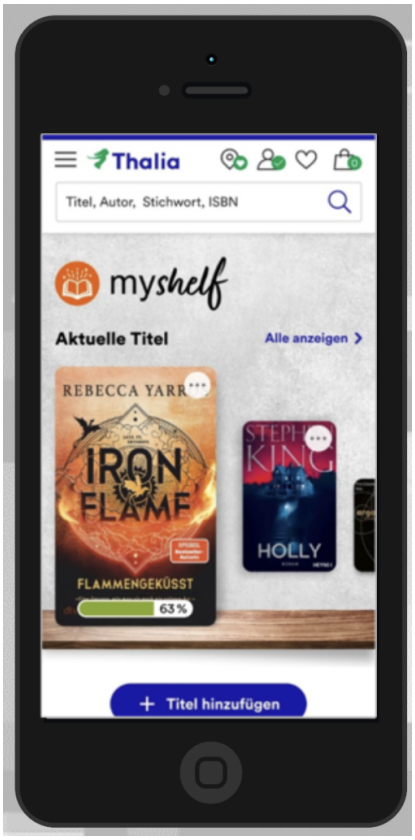


Dieses Team wollte eine Lösung entwickeln, um das gesellschaftliche Problem des Analphabetismus in Deutschland zu bekämpfen. Laut Studien sind heute 12.1% der Erwachsenen in Deutschland funktionale Analphabeten. Thalia hat sich bereits als Ziel gesetzt, die Anzahl der Nichtleser zu halbieren. Das Team hat ein Konzept für eine Lese-App entwickelt, und Teile davon als Prototypen umgesetzt.

Zuerst sollen Nutzer anhand des aktuellen Lesegrades in die passende Stufe (von „Padawan“ bis „Jedi Meister“) eingeordnet werden. Natürlich gibt es für jede Stufe Trainingsangebote: von dem Erkennen von einzelnen Buchstaben durch das Vorlesen von langen bzw. zusammengesetzten Wörtern bis hin zu kürzeren Leseproben aus Büchern bekommt jeder Nutzer und jede Nutzerin Hilfe beim Lesenlernen. In der Demo-Lektion konnte die App die Korrektheit des vorgelesenen Satzes bewerten und der Nutzerin eine prozentuale Erfolgsquote anzeigen.

Gewinner des Publikumspreises: Die Bookshelfies mit „Das Bücherregal, von dem du schon immer geträumt hast“

Fast jeder, der liest, besitzt heutzutage Bücher in sowohl physischer als auch digitaler Form. So verliert man schnell den Überblick über die eigene mentale Büchersammlung. Hatte ich das letzte Fitzek Buch auf meinem Tolino, oder als Paperback? Und warum finde ich das nicht in meinem Regal? Für solche Probleme und für vieles mehr möchten die Bookshelfies eine Lösung bieten, und haben das Bücherregal unserer Träume geschaffen.



Die vorhandenen Bücher können gescannt und blitzschnell ins virtuelle Bücherregal einsortiert werden, bei Käufen im Thalia-Shop passiert das selbstverständlich automatisiert. Handelt es sich um ein digitales Buch, werden Lesefortschritte getracked, können aber natürlich auch manuell eingetragen werden.

Für Vielleser*innen lassen sich die Werke in verschiedene Listen einordnen, die einfach geteilt werden können. Denn unser Bücherregal dient nicht nur zum Überblick, sondern auch zur Bildung der Community: Mitglieder oder einzelne

Listen können gefolgt werden, Kunden und Kundinnen können sich für Leseziele und andere Herausforderungen anmelden. So kann man nicht nur das eigene Leseverhalten im Blick behalten, aber sich auch von anderen Mitgliedern inspirieren lassen.

Die Umsetzung hat die Thalia-Community mit Sicherheit überzeugt: die Bookshelfies haben die meisten Stimmen aus dem Publikum erhalten und konnten somit den Publikumspreis abräumen.

Insgesamt haben wir auch dieses Jahr eine inspirierende Veranstaltung erleben können, die die Kreativität der Teilnehmenden auf beeindruckende Weise zeigte. Die Zusammenarbeit zwischen den Vertreter*innen diverser Bereichen ermöglichte es, frische Ideen und innovative Lösungen für die Herausforderungen der Buchbranche zu entwickeln. Wir sind gespannt darauf, welche bahnbrechenden Ideen nächstes Jahr aus dieser Veranstaltung hervorgehen werden.

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

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

App-Technik, die begeistert! Über

App-Architektur und Testing

Technische Einblicke am Beispiel von Scan & Go – Die mobile Einkaufslösung für die Thalia-App.

Einleitung

Wir, bei der Thalia Bücher GmbH, möchten unseren Kund*innen ein optimales Einkaufserlebnis anbieten. Aus diesem Grund stellen wir verschiedene Produktlösungen bereit, welche wir kontinuierlich verbessern.

Eines dieser Produkte ist die App „Thalia – Bücher entdecken“ für das Smartphone & Tablet (Android, iOS): https://www.thalia.de/vorteile/thalia-app.

Abb. 1: QR-Code zur App-Installation (Quelle: Thalia Bücher GmbH)

Um für unseren Kund*innen den Kauf zu vereinfachen, haben wir bei Thalia das Feature Scan & Go entwickelt. Ziel ist der kontaktlose, schnelle und mobile Einkauf in unseren Buchhandlungen mit der App. In Zeiten von Pandemie oder, um lange Schlangen an den Kassen in der Weihnachtszeit zu vermeiden, ein Mehrwert für unsere Kund*innen.

Allgemeine Informationen sind unter folgenden Webseiten zu finden: https://www.thalia.de/vorteile/scan-go oder https://www.youtube.com/watch?v=jCHEASuHVAc.

Alle Abbildungen stammen vom Autor, sofern keine Quelle angegeben ist.

Abb. 2: Scan & Go-Aufsteller mit QR-Startcode

So funktioniert Scan & Go

Nach der Installation bzw. nach dem Start der App kann in der Buchhandlung der QR-Startcode auf den hierfür bereitgestellten Aufsteller eingescannt werden (siehe Abbildung 2). Anschließend können Artikel anhand des Barcodes – z.B. auf der Buchrückseite – mit der App erfasst werden. Zum Schluss kann der Kauf

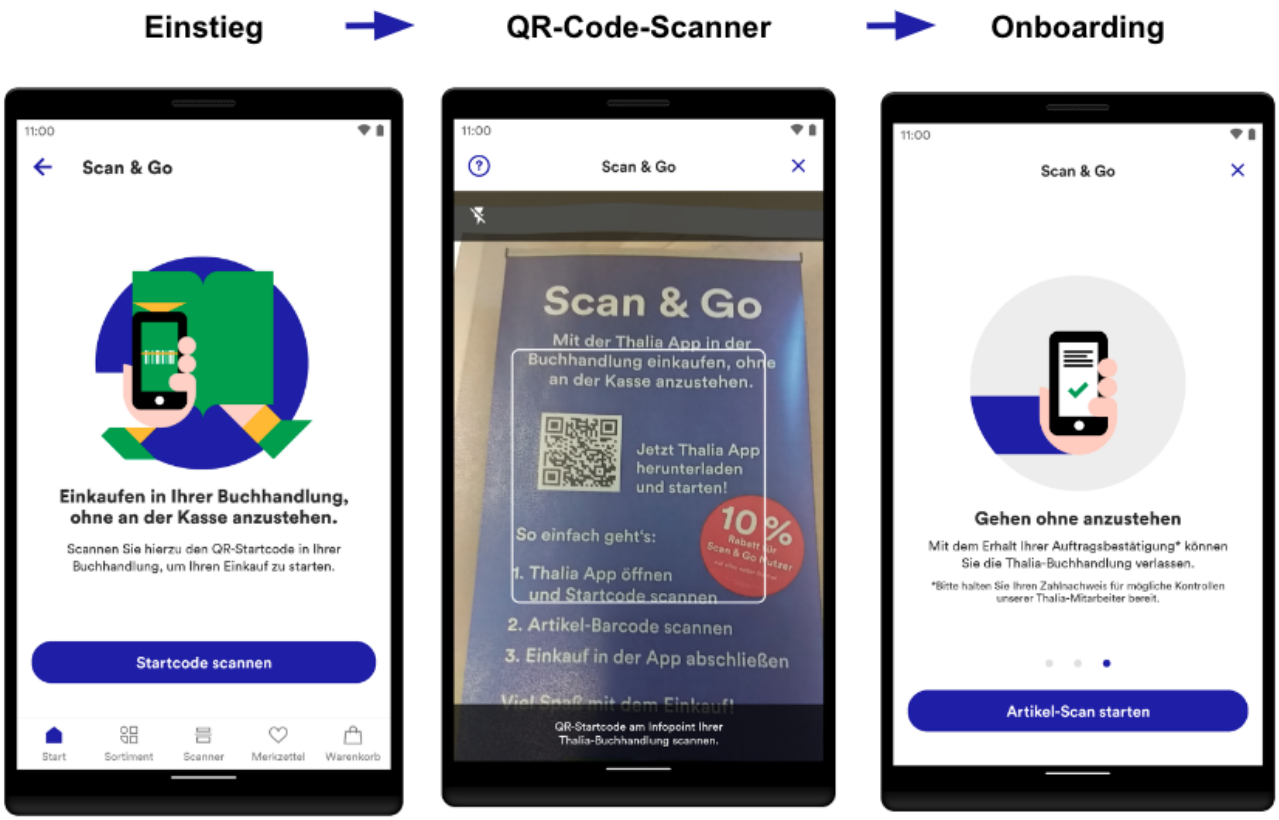
innerhalb der App bequem getätigt und die Buchhandlung, ohne an der Kasse anstehen zu müssen, sorgenfrei verlassen werden.

Ziel

Mit diesem Tech-Blog-Artikel möchten wir aufzeigen, wie wir aus technischer Sicht Scan & Go entwickelt haben und welche Hürden wir zu bewältigen hatten. Des Weiteren wird ein Überblick über unser automatisches Testen von Scan & Go beschrieben. Der Artikel richtet sich an alle App-Entwickler*innen und technisch versierten Leser*innen.

Überblick

Die wesentlichen Bestandteile von Scan & Go werden mit Hilfe der folgenden Screenshots analog der User Journey dargestellt. Aufgrund des Umfangs werden nicht alle Screens vorgestellt (z.B. Hilfe oder Login).



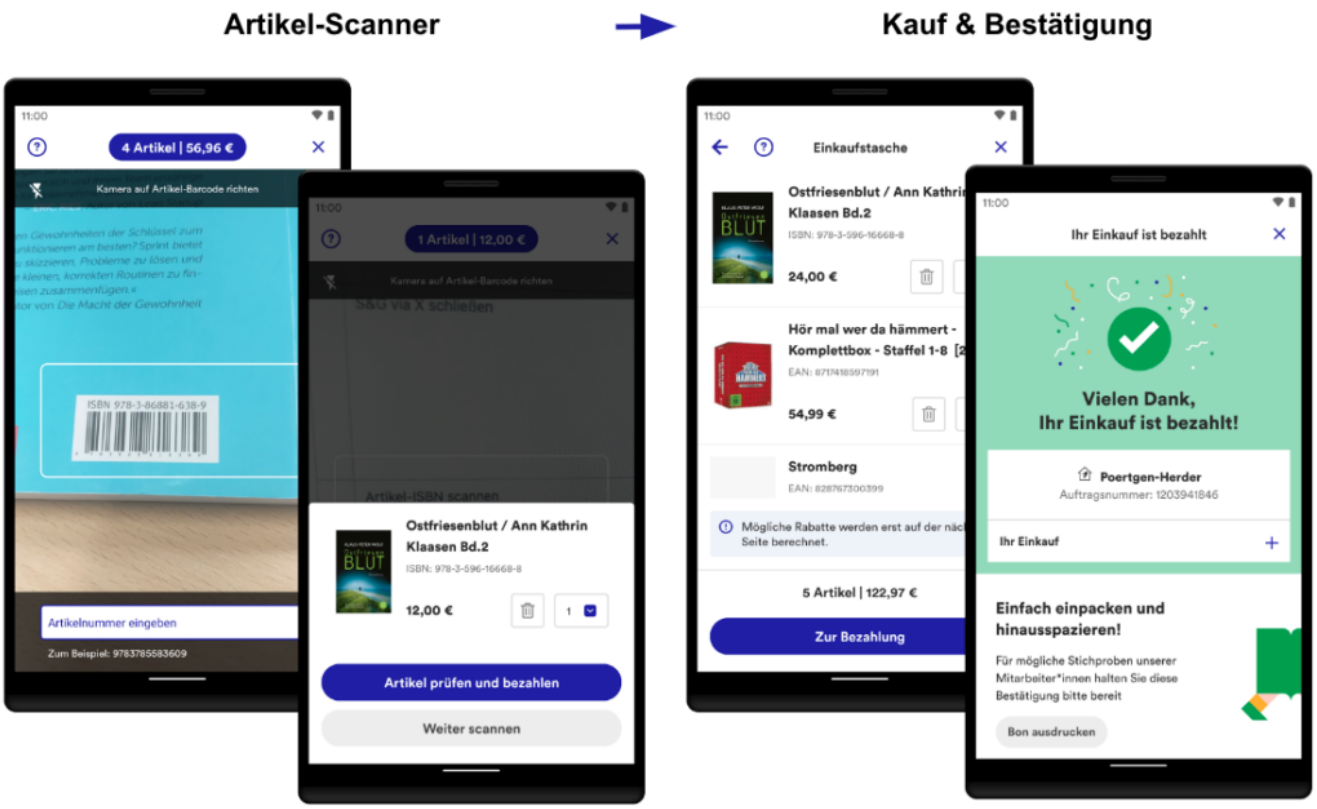


Abb. 3: Übersicht von Scan & Go

Einstieg: Scan & Go kann über diverse Wege angesteuert werden: Über die Startseite der App, den internen Scanner in der App oder über die Kamera des Gerätes.

QR-Code-Scanner: Der Zugang zu Scan & Go wird durch das Einscannen von auf diversen Werbemitteln gedruckten QR-Codes in der Buchhandlung gewährt. Der Zugang ist zwei Stunden gültig.

Onboarding: Nach dem Einscannen des QR-Codes werden mit Hilfe eines optionalen, dreistufigen Onboardings die wichtigsten Funktionen und Informationen erklärt. Sie werden erst wieder angezeigt, nachdem der Zugang abgelaufen ist.

Artikel-Scanner: Das Herzstück: Im nachfolgenden Scanner werden die Artikel anhand des Barcodes erfasst. Im Bestätigungsdialog kann der Artikel begutachtet und auf Wunsch bearbeitet oder entfernt werden. Alle Artikel werden automatisch in der digitalen Einkaufstasche gespeichert.

Kauf & Bestätigung: In der Einkaufstasche werden alle Artikel in einer Liste zusammengefasst. Einzelne Artikel können geändert werden. Abschließend erfolgt nach der Bezahlung eine Bestätigung auf der Dankeseite. Der Einkauf ist damit erfolgreich abgeschlossen.

Implementierung

Für die Implementierung haben wir einen modernen Technologie-Stack gewählt, damit Scan & Go stabil läuft, einfach erweiterbar ist und sich über einen langen Zeitraum bewahren kann. Im Folgenden wird das technische Konzept erläutert und anhand von ausgewählten Beispielen vorgestellt. Scan & Go haben wir für iOS und Android entwickelt. Eingesetzte Technologien befinden sich im Anhang.

Architektur

Als übergreifendes Architekturkonzept haben wir die sog. “Clean Architecture” herangezogen (vgl. https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html, Buchempfehlung: https://www.thalia.de/shop/home/artikeldetails/A1039840971). Ziel des Konzeptes ist, eine skalierbare, einheitliche und wartbare Implementierung anzustreben. Des Weiteren ist einer der wichtigsten Merkmale: Wir können verständlichen und gut testbaren Code entwickeln. Erst dadurch sind wir in der Lage, eine unerlässliche Testautomatisierung umzusetzen (siehe “Abschnitt Sicherstellung der Qualität & Automatisiertes Testen”).

Unsere Architektur ist in drei Schichten gegliedert: **UI**, **Data** und **Domain** (z.B. https://developer.android.com/topic/architecture).

UI: Die UI beinhaltet die Logik zur Präsentation von aufbereiteten Daten. Sie ist abhängig von Data und Domain und sollte keine Business-Logik beinhalten. Die UI ist eng an das Betriebssystem (z.B. Android) gekoppelt bzw. interagiert mit diesem.

Data: Beinhaltet die Business-Logik der App und implementiert die Datenhaltung und -beschaffung z.B. über REST-Services. Data ist lediglich von Domain abhängig.

Domain: Domain ist technisch das Bindeglied zwischen UI und Data. Ziel ist die Vermeidung von Redundanzen bzw. Bereitstellung von wiederverwendbaren Funktionalitäten (Use Cases) und weitaus komplexeren Business-Logiken. Dadurch wird die Interaktion zwischen UI und Data stark vereinfacht. Domain ist komplett unabhängig von UI und Data und kennt keine umgebende Infrastruktur (z.B. das Betriebssystem Android). Aus diesem Grund ist es sehr gut automatisch

testbar.

Für jeden Screen in Scan & Go wurde eine separate UI-Klasse implementiert und das übliche Entwurfsmuster „Model View Viewmodel“ verwendet. Der Unterbau für jede UI-Klasse setzt sich wie folgt zusammen (vereinfacht):

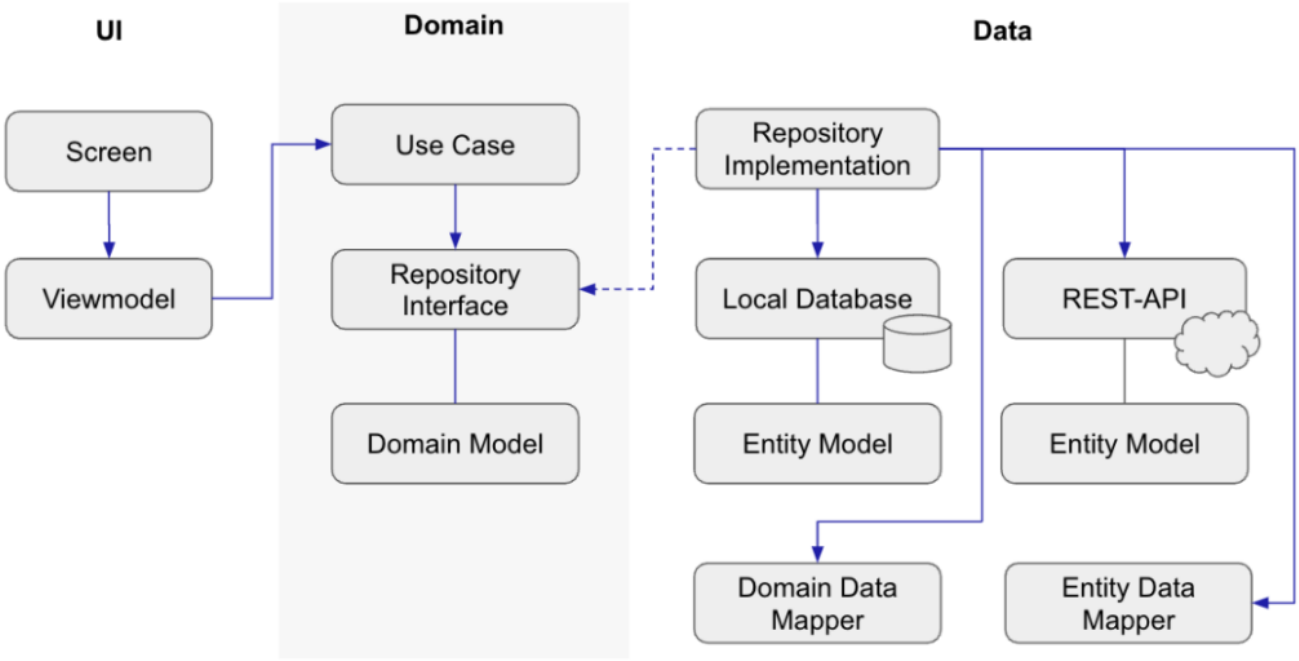


Abb. 4: Grober Aufbau einer UI-Klasse

- Die UI-Klasse *Screen* arbeitet nur mit einem *Viewmodel* (durchgezogener Pfeil). Sie observiert das *Viewmodel* und präsentiert bei jeder Änderung eins zu eins die Daten.
- Ein Viewmodel operiert mit *Use Cases* und verwendet lediglich die Daten aus dem Domain Model.
- Ein *Use Case* greift ausschließlich auf abstrakte *Repositories* (Interfaces) zu und bereitet gemäß der gewünschten Business-Logik die Daten auf.
- Data implementiert die Interfaces aus Domain (gestrichelte Linie), regelt den lokalen und externen Datenzugriff und bildet über *Mapper*-Klassen die entsprechende Datenstruktur ab. Dadurch hat beispielsweise eine Änderung von Attributen der *Entity*-Klasse in der Regel keinen Einfluss auf Domain und UI.

Beispiel: Implementierung des Artikel-Scanners

(Android)

Nachfolgend werden die oben beschriebenen Zusammenhänge exemplarisch für den Artikel-Scanner-Screen mit Android-Code illustriert. Aufgrund der Komplexität und Schutz des Urheberrechts sind nicht alle Details abgebildet.

UI + Domain (Fragment, Viewmodel und Use Case): Für den Artikel-Scanner haben wir eine Klasse *StorePaymentScannerFragment* implementiert. Die Verdrahtung der Klassen erfolgt mittels Dependency Injection auf App-Ebene mit Dagger.

Abb. 5: Zusammenhang zwischen Fragment, Viewmodel und Use Case

- (A) Mit Hilfe der Kamera des Gerätes kann der Barcode des Artikels erfasst werden. Die Erkennung des Barcodes erfolgt beispielsweise in Android mit dem ML Kit und mit CameraX von Google.
- (B) Alternativ kann die EAN bzw. ISBN oberhalb des Barcodes manuell eingetragen werden.
- (1) Nach Erkennung der EAN-Nummer wird diese an *loadArticle* übergeben.
- (2) Der übergebene Code wird an den *HandleScannedEanUseCase* weitergereicht, welcher asynchron ausgeführt wird und die Beschaffung der Artikeldaten sowie die Speicherung kapselt.
- (3) *HandleScannedEanUseCase* übergibt intern den Code an das Interface (siehe nächsten Abschnitt).

Data (Repository, Mapper und API): In der nächsten Abbildung wird anhand einer eingescannten EAN-Nummer der passende Artikel aus dem Backend besorgt.

Abb. 6: Zusammenhang zwischen Use Case, Repository, Mapper und API

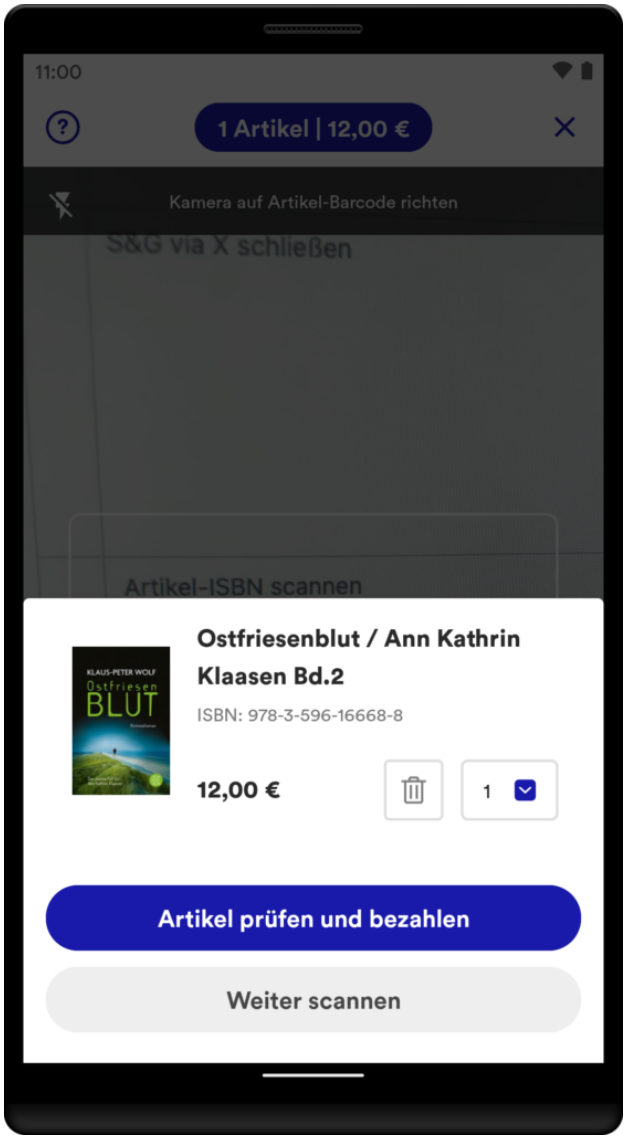


Abb. 7: Erfolgreich eingescannter Artikel

- (1) Der *HandleScannedEanUseCase* greift auf das ihm bekannte Interface *getArticle* zu und übergibt u.a. die eingescannte EAN-Nummer. Implementierungsdetails sind dem Use Case nicht bekannt.
- (2) Die Klasse *StorePaymentArticleRepositoryImpl* implementiert das Interface und greift auf die externe Datenquelle mit *getStoreArticle* zu. Die Implementierung unserer Schnittstelle *StoreArticleAPI* wird generisch mit Retrofit realisiert.
- (3) Im letzten Schritt wird die Entität *ArticleEntity* von der API in die domänenspezifische Datenklasse *StorePaymentArticle* mit einem Subset notwendiger Daten abgebildet und an das Use Case zurückgeliefert.

Final wird im Fragment der Artikel angezeigt. Es können weitere Artikel eingescannt werden. Dann werden erneut die drei beschrieben Schichten

durchlaufen.

Sicherstellung der Qualität & Automatisiertes Testen

Um dauerhaft die Qualität, Stabilität und Zukunftssicherheit von Scan & Go zu gewährleisten, haben wir umfangreiche Unit- und UI-Tests implementiert. Die Tests sollen rechtzeitig Fehler aufdecken bzw. die fachlichen Anforderungen bewahren. Zudem wird ganz erheblich der manuelle Abnahmeprozess für ein neues App-Release reduziert. Somit sind wir in der Lage, schneller zu agieren und neue Features oder Bugfixes zu releasen.

Das Schreiben von Unit-Tests ist komplett in unserem Scrum-Prozess verankert. Es ist ein bedeutender Teil unserer Definition of Done (DoD). UI-Tests hingegen sind komplexerer Natur und werden in separaten Tickets entwickelt. Diese werden plattformübergreifend gemeinsam mit der QA spezifiziert und anschließend realisiert.

Unit-Tests

Aufgrund der gewählten Architektur und der strikten Trennung in einzelne Klassen ist es leicht, Unit-Tests zu schreiben. Der große Vorteil ist, dass diese Art von Tests sehr schnell direkt in einer JVM ausführbar sind. Ein Unit-Test ist deterministisch, wird isoliert ausgeführt und testet einen kleinen Bereich des Codes (eine „Unit“) ab.

Während der Entwicklung werden bei jedem Commit einer Codeänderung auf den Entwicklungsbranch (vgl. Gitflow) automatisch alle Unit-Tests auf unserem Jenkins ausgeführt. Dadurch erhalten wir als Entwickler*innen schnelles Feedback, falls der Code Regressionsfehler enthält oder wichtige fachliche Anforderungen ausgehebelt wurden. Dementsprechend können wir rechtzeitig korrigierende Schritte einleiten.

Wir haben für Scan & Go alle Nicht-UI-Bereiche mit Unit-Tests abgedeckt. Im folgenden Codesnippet ist ein in Kotlin entwickelter Test exemplarisch abgebildet. Der Test prüft, ob falsch ausgezeichnete Barcodes (EAN, ISBN, …)

identifiziert werden.

```
class EanDigitCheckerTest {

    private val eanDigitChecker = EanDigitChecker()

    @Test
    fun givenInvalidEanCodes_whenCodeIsVerified_thenCodeIsDetectedAsInvalid() {
        assertInvalidEanCode(eanDigitChecker.verifyEan(""))
        assertInvalidEanCode(eanDigitChecker.verifyEan("-"))
        assertInvalidEanCode(eanDigitChecker.verifyEan("T"))
        assertInvalidEanCode(eanDigitChecker.verifyEan("544900009624T"))
        assertInvalidEanCode(eanDigitChecker.verifyEan("test"))
        assertInvalidEanCode(eanDigitChecker.verifyEan("9783406628178"))
    }

    private fun assertInvalidEanCode(result: Result) {
        Assert.assertFalse("Expect invalid code.", result.isValid)
        Assert.assertNotNull("Expect not empty code.", result.outputEan)
    }
}
```

Abb. 8: Unit-Test zur Prüfung von invaliden EAN-Codes

UI-Tests

Technisch anspruchsvoller ist es, die Scan & Go-Screens mittels UI-Tests automatisiert zu testen. Grund ist nicht der zu schreibende Code, sondern vielmehr, dass die Tests auf physischen Geräten bzw. Emulatoren ausgeführt werden und eine deutlich längere Ausführungszeit brauchen. Zudem wird eine stabile Infrastruktur benötigt, wie z.B. Mockserver, Testmaschinen und ein dediziertes WLAN, um nur einige Punkte zu nennen.

Aufgrund der Infrastruktur und der externen Einflüsse und Abhängigkeiten können Tests scheitern, obwohl der Code nicht geändert wurde. Diese Tests sind sog. „flaky Tests“ und sollten vermieden werden. Für das Scheitern der UI-Tests sind folgende Einflussfaktoren maßgeblich:

- Ausgelastetes Test-WLAN/Timeouts in den Service-Requests
- Stabilität von REST-Services in der Testumgebung
- Änderungen von REST-Services oder Webseiten von anderen Teams

- App-Animationen (z.B. Einblendung von Bannern)
- Dialoge (z.B. das BottomSheetDialogFragment von der Android-API)
- (Ungeplante) Systemdialoge vom Betriebssystem
- Zugriff auf die GPS-Position (z.B. Auffinden der nächsten Buchhandlung)

Wir sind die genannten Einflussfaktoren über einen längeren Zeitraum angegangen und haben für beide Plattformen Android und iOS inzwischen eine gute Erfolgsquote an positiven Tests (ca. 98 %, siehe Abbildung 9). Unser Ziel ist dennoch 100 %, um eine hohe Zuverlässigkeit mit den UI-Tests zu gewährleisten. Tests sollten ausschließlich scheitern, wenn der Code Regressionsfehler enthält oder wichtige fachliche Anforderungen ausgehebelt wurden.

Abb. 9: Testreport für unsere Kundenapp (Android)

Auf Basis dieser Vorarbeiten haben wir für jeden Scan & Go-Screen einen Test entwickelt und häufig genutzte Funktionalitäten getestet („happy path“). Aufgrund der langen Ausführungszeit haben wir auf Edge-Cases verzichtet (z.B. testen, ob ein Artikel 100 Mal in die Einkaufstasche gelegt werden kann).

Damit wir für beide Plattformen Android und iOS identische Testfälle haben, haben wir die zu testenden Schritte und deren Reihenfolge in gemeinsame Interfaces ausgelagert (siehe Abbildung 10). Ein Interface ist als Contract zwischen der Qualitätssicherung (QA) und der Entwicklung zu verstehen.

```
public protocol InstorePaymentManageArticleOnScannerFeatureInterface {
    func givenIAmAtTheInstoreArticleScannerWith(storeId: Int)
    // changed parameter eanString to ean
    func whenIAddArticleWith(ean: String)
    func thenTheArticleInformationIsDisplayedWith(displayLabel: String)
    func andTheAmountAndTotalAreUpdated()

    func thenTheFirstTimeUnknownArticleAlertIsDisplayedWith(displayLabel: String)
    func whenITapEnterManually()
    func thenTheManualInputIsSelected()
    func thenTheUnknownArticleAlertIsDisplayed()
}

public extension InstorePaymentManageArticleOnScannerFeatureInterface {

    func runTestScenarioAddKnownArticle() {
        // Is article information displayed correctly when I add a known article?
        givenIAmAtTheInstoreArticleScannerWith(storeId: 5198)
        whenIAddArticleWith(ean: "9783596166688")
        thenTheArticleInformationIsDisplayedWith(displayLabel: "ISBN: 978-3-596-16668-8")
        andTheAmountAndTotalAreUpdated()
    }

    func runTestScenarioAddKnownArticleWithHyphens() {
        // Is article information displayed correctly when I add a known article with hyphens?
        givenIAmAtTheInstoreArticleScannerWith(storeId: 5198)
        whenIAddArticleWith(ean: "978-3-596-16668-8")
        thenTheArticleInformationIsDisplayedWith(displayLabel: "ISBN: 978-3-596-16668-8")
        andTheAmountAndTotalAreUpdated()
    }
```

Abb. 10: Interface für einen gemeinsamen Testfall

Die jeweilige Plattform (Android, iOS) implementiert das Interface als normalen UI-Test (siehe Abbildung 11). Dabei müssen lediglich die atomaren Schritte implementiert werden. Die Reihenfolge der Schritte gibt das Interface in Form einer Methode vor (z.B. *runTestScenarioAddKnownArticle*). Diese Methode muss in der Testklasse für das entsprechende Testframework referenziert werden.

Im folgenden Codesnippet ist eine Implementierung mit Kotlin exemplarisch abgebildet.

```
class StorePaymentManageArticleOnScannerTest
    : StorePaymentBaseTest(), InstorePaymentManageArticleOnScannerFeatureInterface {

    @Test
    fun runTestScenarioAddKnownArticleTestCase() {
        runTestScenarioAddKnownArticle()
    }

    @Test
    fun runTestScenarioAddKnownArticleWithHyphensTestCase() {
        runTestScenarioAddKnownArticleWithHyphens()
    }

    override fun givenIAmAtTheInstoreArticleScannerWith(storeId: Int) {
        navigateToArticleScanner(storeId)
    }

    override fun whenIAddArticleWith(ean: String) {
        waitHandler.waitForView(R.id.enter_ean_search_input)
        BaristaEditTextInteractions.writeTo(R.id.enter_ean_search_input, ean)
        BaristaKeyboardInteractions.pressImeActionButton(R.id.enter_ean_search_input)
    }

    override fun thenTheArticleInformationIsDisplayedWith(displayLabel: String) {
        waitHandler.waitForText(displayLabel)
        // close dialog
        BaristaClickInteractions.clickBack()
    }

    override fun andTheAmountAndTotalAreUpdated() {
        BaristaVisibilityAssertions.assertDisplayed(RegexViewMatcher(articleSummaryRegex))
        screenshotProvider.screenshot("amount_and_total")
    }
```

Abb. 11: Implementierung eines gemeinsamen Testfalls (Android)

Für Scan & Go haben wir 24 Tests konzipiert und in fachliche Bereiche gruppiert, z.B.:

- Funktioniert der Einsprung nach Scan & Go?
- Kann zur Hilfe navigiert werden und zurück?
- Funktioniert der komplette Kaufprozess?
- Wie verhält sich das Einscannen von bekannten/unbekannten Artikeln?
- Funktioniert das Löschen eines Artikel ordnungsgemäß?

Insgesamt wurden in etwa 1200 Codezeilen implementiert. Die Tests werden

täglich nachts ausgeführt und mit einem Report begutachtet:

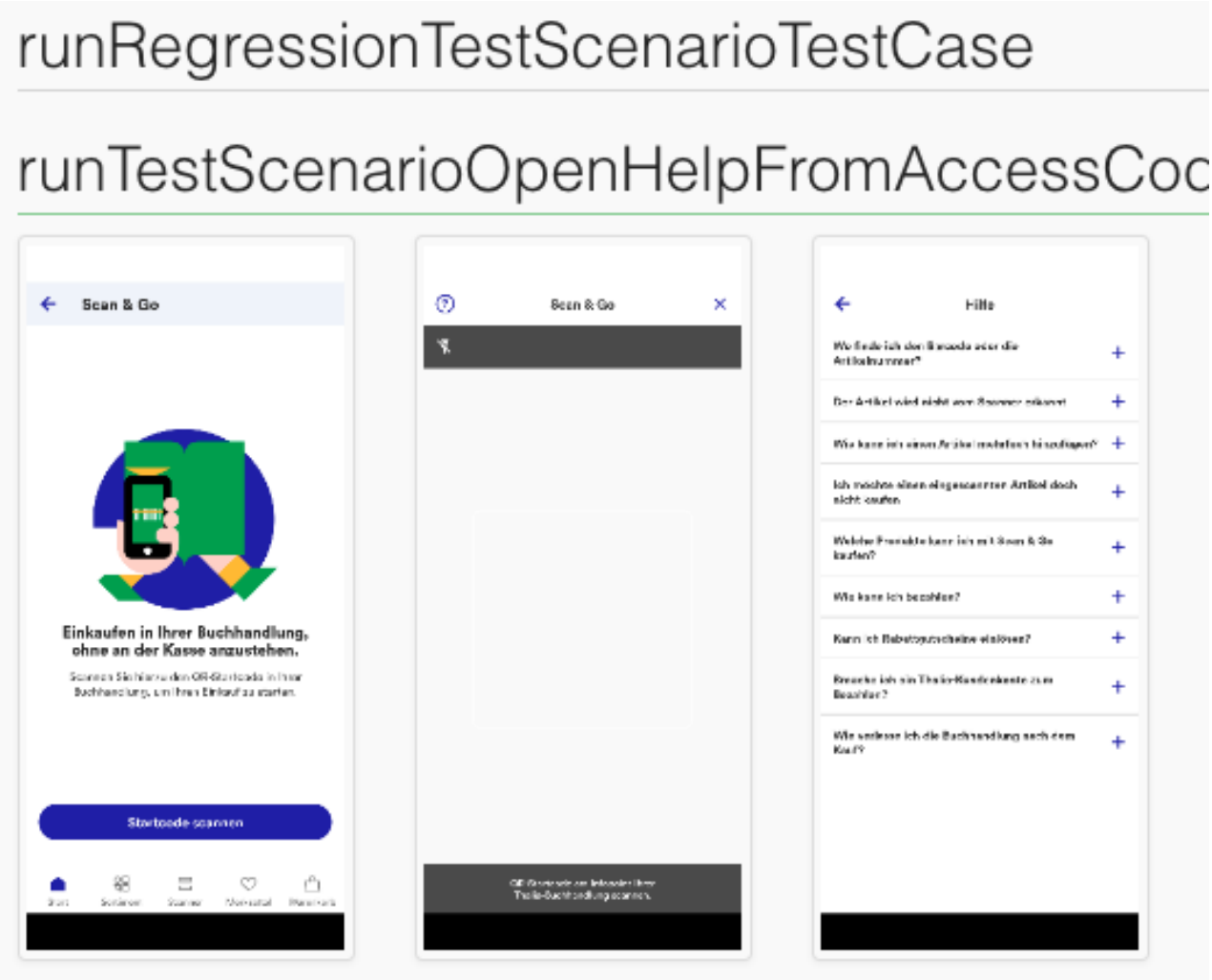


Abb. 12: Auszug aus dem Testreport (Android)

Um das Schreiben von Tests für unsere QA einfach zu halten, haben wir uns entschieden, die Testinterfaces in Swift zu implementieren. Alle Testinterfaces werden als Library in einem Nexus-Repository deployed und in Android und iOS als übliche Dependency eingebunden. Für Android wird der Swift-Code automatisch im Buildprozess nach Kotlin konvertiert.

Weiterführende Informationen rund um unsere allgemeine Teststrategie haben wir in einem separaten Blog beschrieben: https://tech.thalia.de/testen-einer-app-in-der-hybriden-welt/.

Probleme & Lösungen während der Implementierung

Lichtverhältnisse

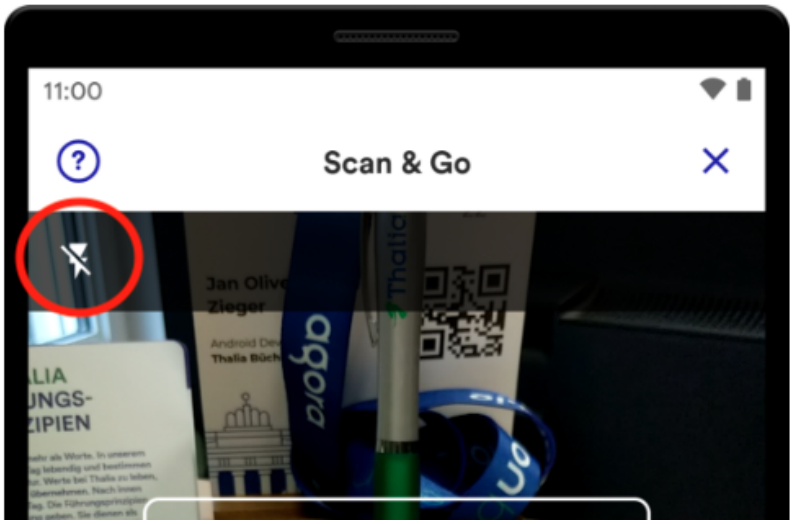


Abb. 13: Aktivierung des Lichts

Die Lichtverhältnisse in den verschiedenen Buchhandlungen variieren und sind aus technischer Sicht nicht immer optimal. Die Kamera des Gerätes hat unter Umständen Schwierigkeiten, den Barcode auf den Artikeln zu erkennen. Um dieses Problem zu lösen, bieten wir den User*innen eine einfache Möglichkeit, direkt beim Einscannen des Artikels das Licht des Gerätes zu aktivieren. Das technische Aktivieren des Lichts ist mit der Standard-API des jeweiligen Betriebssystems gelöst (z.B. *androidx.camera.coreCameraControl*).

Reduzierung der Serverlast

Um den User*innen ein bequemes und flüssiges Einscannen zu ermöglichen, werden während des Scanvorgangs die Aufnahmen der Kamera fortwährend automatisch analysiert und versucht, den Barcode des Artikels zu erfassen. Für jede erfolgreiche Erfassung eines Barcodes wird dieser an unser REST-Backend gesendet. Bei einer erfolgreichen Suchanfrage werden die Artikeldaten instantan geliefert und in der UI präsentiert.

Die Optimierung besteht darin, nicht valide Barcodes zu erkennen und dem Backend vorzuenthalten. Das mindert nicht nur die Serverlast, sondern spart auch Datenvolumen der Kund*innen ein. Die Prüfung erfolgt direkt in der App und verifiziert den Barcode anhand der letzten Zahl im Barcode, der sog. Prüfziffer (vgl. https://de.wikipedia.org/wiki/European_Article_Number).

Für Android haben wir den Algorithmus für die Prüfung beispielhaft in Kotlin wie

folgt implementiert:

```
fun verifyEan(eanCode: String): Result {
    val cleanedDigits = eanCode.replace(" ", "").replace("-", "")

    if (cleanedDigits.toLongOrNull() == null) {
        return Result(outputEan = eanCode, isValid = false)
    }

    val eanDigits = toEan(cleanedDigits)
    val digitsToCheck = eanDigits.dropLast(n = 1).reversed()
    val detectedChecksum = Character.getNumericValue(eanDigits.last())

    val computedChecksum = computeChecksum(digitsToCheck)
    val isValid = computedChecksum == detectedChecksum

    return Result(outputEan = eanDigits, isValid = isValid)
}
```

Abb. 14: Implementierung der Prüfung von Barcodes

Die App wertet das Ergebnis aus und sendet entsprechend die Ausgabe an das REST-Backend oder bricht ab und setzt den Scanvorgang fort.

WLAN

Sofern das Gerät nicht mit einem WLAN verbunden ist, kann eine App träge wirken und zu einer geringeren Zufriedenheit führen. Insbesondere ist die Netzabdeckung innerhalb der Buchhandlungen und der Shoppingcenter oft unzureichend. Aus diesem Grund bieten wir in den Buchhandlungen kostenlose WLAN-Zugänge an. Einmal mit dem WLAN verbunden, kann unsere App optimal performen, da die Artikeldaten signifikant schneller geladen werden können. Zudem wird kein mobiles Datenvolumen verbraucht.

Nach der Entwicklung der ersten Version für Scan & Go haben wir das Feedback erhalten, dass das Einscannen schneller sein könnte. Des Weiteren war den User*innen nicht bekannt, dass ein WLAN-Zugang in den Buchhandlungen vorhanden ist. Diese Probleme sind wir zügig angegangen. Ziel war in einer nächsten App-Version den Scan & Go-Vorgang flüssiger von der Hand gehen zu lassen und den User*innen eine verbesserte User Experience anzubieten.

Abb. 15: Einblendung Hinweisbanner für das WLAN

Wir haben in der App einen Hinweisbanner entwickelt, der angezeigt wird, wenn der QR-Code- oder der Artikel-Scanner geöffnet wird. Der Hinweisbanner wird nach ein paar Sekunden von oben eingeblendet. Er macht darauf aufmerksam, das WLAN in der Buchhandlung zu nutzen. Für den Absprung in die WLAN-Einstellungen wird die Standard-API des jeweiligen Betriebssystems verwendet.

Fazit & Ausblick

Aufgrund der gewählten technischen Architektur und der Teststrategie können wir mit dem Feature Scan & Go unseren Kund*innen einen Mehrwert anbieten, um schnell und intuitiv in einer Buchhandlung einkaufen zu können. Wir arbeiten weiter fokussiert an der Thematik und möchten aus Sicht des Kunden Scan & Go kontinuierlich erweitern und verbessern.

Langfristig wollen wir den Scan & Go-Prozess verschlanken und noch einfacher für unser Kund*innen gestalten. Beispielsweise möchten wir zukünftig das initiale Einscannen des QR-Startcodes obsolet machen. Damit wir wissen, in welcher Buchhandlung Scan & Go verwendet wird, müsste eine Ortungstechnologie herangezogen werden (WLAN-Ortung, Bluetooth Beacon, GPS, Geofencing, …). Beim Start von Scan & Go wird automatisch die Buchhandlung ermittelt und das Einscannen des QR-Startcodes entfällt. Das ist zudem nachhaltiger, da die Werbemittel zum Bedrucken des QR-Codes (Aufsteller, Flyer, …) eingespart werden können.

Weitere Ideen werden gesammelt und priorisiert umgesetzt. Im Entwicklungsprozess halten wir uns an Scrum und durchlaufen vereint alle Fachdisziplinen (UX, PO, QA, DEV), um ein stimmiges Endprodukt zu erreichen.

Anhang

	Android	**iOS**
Sprache	Kotlin, teilweise Java (legacy Code)	Swift, SwiftUI, teilweise Objective C (legacy Code)
OS-Version	>= Android 8	>= iOS 14
IDE	Android Studio	Xcode
CI/CD	Jenkins, git, GitLab, gradle, Bash, Pipeline-Scripting	Jenkins, git, GitLab, xcodebuild, Bash, Fastlane
Netzwerk, Datenbank	Retrofit, Room	Alamofire, CoreData
QR-Code- und Text-Erkennung	Google ML Kit, CameraX	AVFoundation
Unterstützte Scan-Codes	FORMAT_EAN_13, FORMAT_EAN_8 FORMAT_UPC_E, FORMAT_QR_CODE FORMAT_CODE_128	EAN13Code, EAN8Code, UPCECode, QRCode, Code128Code
Test	Espresso, Roboletric, Mockito, JUnit Barista	XCTestCase, XCTest
Animation	Lottie	Lottie

- https://www.thalia.de/vorteile/thalia-app
- https://www.thalia.de/vorteile/scan-go
- https://tech.thalia.de/
- https://tech.thalia.de/testen-einer-app-in-der-hybriden-welt/
- https://lottiefiles.com/what-is-lottie

- https://developer.android.com/training/testing/espresso
- https://github.com/AdevintaSpain/Barista
- https://robolectric.org/
- https://site.mockito.org/
- https://developers.google.com/ml-kit
- https://developer.android.com/training/camerax
- https://developers.google.com/ml-kit/reference/android
- https://developer.apple.com/av-foundation/
- https://developer.apple.com/documentation/avfoundation/avmetadataobject/objecttype
- https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
- https://developer.android.com/topic/architecture
- https://de.wikipedia.org/wiki/European_Article_Number
- https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
- https://www.youtube.com/watch?v=jCHEASuHVAc
- https://www.thalia.de/shop/home/artikeldetails/A1039840971
- https://www.esri.com/en-us/arcgis/products/arcgis-ips/overview

---

Hackathon & Innovation Days Münster 2022

Nach dem Erfolg des rein digitalen Hackathons 2021 haben wir uns dieses Jahr größere Ziele gesetzt. Mehr Teilnehmende, mehr Zeit, und das Ganze vor Ort.

Kubernetes mal anders mit Tanzu

Container waren lange Zeit dem Warentransport vorbehalten – auch bei Thalia. Aber heutzutage wächst auch der Bedarf an Containern im Bereich der IT. Das vollständige Potential lässt sich erst erzielen, wenn auch ein entsprechender Orchestrator verwendet wird, um die Verwaltung zu übernehmen. Wo liegen jetzt die Vorteile von Containern und was bringt mir ein Orchestrator außer einer zusätzlichen und womöglich zugleich unbekannten Technologie?

Ein immer schnellerer Entwicklungszyklus und damit auch häufigere Deployments sind ein Treiber. Hierbei bieten Container die Möglichkeit, einen vordefinierten und unveränderlichen Stand, auch als „Immutable Image“ bezeichnet, in unterschiedlichen Architekturen auszurollen. Somit ist es auch simpel, diese Images sowohl auf den Geräten der Entwickler, als auch in den unterschiedlichen Entwicklung-Stages zu verwenden.

Ein weiterer Treiber sind die Skalierungsmöglichkeiten, die durch einen Container Orchestrator wie Kubernetes entstehen. Anwendungen können hierbei anhand von Auslastungsmerkmalen wie der CPU- oder der RAM-Verbrauch automatisch skaliert werden, was ein manuelles Eingreifen überflüssig macht. Bei

hohen Auslastungen können hierdurch weitere Container bereitgestellt, andersherum bei wenig Auslastung auch wieder reduziert werden, was Einsparungen ermöglicht.

Ideenfindung/Werdegang

Kubernetes ist in der heutigen Zeit als Orchestrator für Container eine gesetzte Technologie. Aber gleichzeitig birgt diese auch das Risiko, einen gewaltigen Mehraufwand im Betrieb zu erzeugen. Schnelle Entwicklungszyklen, die häufige Updates von Kubernetes selber erzwingen, um am Ball zu bleiben. Viele Komponenten innerhalb von Kubernetes, die ebenfalls verwaltet und aktualisiert werden wollen. Und das bei immer mehr Abstraktionsschichten, von der eigentlichen Hardware hin zu der eigentlichen Basis für die Anwendung. Insgesamt entsteht also ein hilfreiches, aber auch komplexes Konstrukt, dessen Management sichergestellt sein muss, um sich nicht kontraproduktiv auszuwirken.

Wie bekommt man also den schmalen Grad hin, bei einer vordefinierten Anzahl von Administratoren den Aufwand möglichst gering zu halten und trotzdem die Vorteile gewinnen zu können? Hierzu gibt es unterschiedliche Varianten, die wir uns angeschaut haben, bevor wir schlussendlich bei der jetzigen Lösung angekommen sind.

Die erste Herangehensweise war, sich mit Kubernetes selbst auseinanderzusetzen. Bezeichnet als „Vanilla Kubernetes“ machten wir uns mit entsprechender Automatisierung dran, Kubernetes auszurollen, miteinander zu verknüpfen und auch zu nutzen. Die ersten Anwendungen waren schlussendlich bereits in den Entwicklungsstages ausgerollt und aktiv, bevor wir die Reißleine ziehen mussten. Mit der in der Zwischenzeit gesammelten Erfahrung war ein Betrieb in „Produktion“ schlichtweg nicht realistisch. Zu hoch war der Aufwand und das hiermit verbundene Betriebsrisiko.

Der nächste logische Schritt war, dass der Betrieb komplett ausgelagert wird. Somit könnten wir uns auf das Deployment der Anwendungen konzentrieren und müssten uns um die Verwaltung keine Gedanken machen. Somit würden auch keine Ressourcen unsererseits in Beschlag genommen. Klingt doch soweit super, könnte man denken? Aber auch das hat am Ende leider nicht die gewünschten

Ergebnisse gebracht. Bei solchen „managed“ Lösungen gilt es einige Punkte zu beachten. Da wären z.B. das Kosten-Nutzen-Verhältnis aber auch der Abgang von Erfahrungen im Betrieb und damit das Verständnis für tiefere Ebenen der Technologie. Zusätzliche benötigte Funktionen müssen ebenfalls auf eigene Faust gestemmt werden oder gehen weiter zu Lasten der Kosten.

Schauen wir uns also einmal die Probleme an. Häufige Updates und Abhängigkeiten sind ein großer Treiber der Technologie. Auf der anderen Seite benötigen wir mehr als eine „managed“ Lösung uns im ersten Schritt bieten kann. Lässt sich also die grundlegende Verwaltung, wie z.B. Updates und Management von Abhängigkeiten, simpler gestalten? Aber gleichzeitig der eigentliche Betrieb und somit auch das tiefergreifende Wissen erhalten? Und das bei einem sinnvollen Kosten-Nutzen-Verhältnis?

Hier kommt „VSphere with Tanzu“ als Produkt von VMWare ins Spiel, welches im Nachfolgenden nur noch als „Tanzu“ bezeichnet wird.

Planung

Für die Recherche und die Prüfung des Produkts gab es verschiedene Schritte. Die Idee war aber immer eine pragmatische Herangehensweise. In einem ersten kurzen Test haben wir uns die Grundlagen angesehen, einen Cluster aufgesetzt, erste Objekte angelegt und dann Anwendungen hineingesetzt und die Funktionen geprüft. Soweit so gut, die erste Hürde war gemacht und bisher noch keine Blocker entdeckt.

Im nächsten Schritt haben wir uns dann noch einmal genauer mit den Details auseinandergesetzt, den technologischen Aufbau rekapituliert, Testfälle heruntergeschrieben und zusätzliche Kollegen eingeweiht und einbezogen. Mit dieser Vorplanung ging es dann in den PoC (Proof of Concept). Das Setup konnten wir zum Glück noch wiederverwenden, hier gab es keine nötigen Anpassungen. Um einige Funktionen erweitert und die Testfälle abgeklappert ging die Stimmung weiter nach oben. Jetzt galt es noch die Entwicklungsteams abzuholen, eine ungenutzte Hülle hilft ja schließlich keinem. Hier kamen also die DevOps-Kollegen aus unseren Teams ins Spiel, um die finalen Bereiche abzuklären. Und wäre das nicht mit Erfolg gekrönt, würden diese Zeilen wohl nie gelesen werden.

Jetzt ging es also an den größeren Brocken der Arbeit. Das Projekt muss geplant

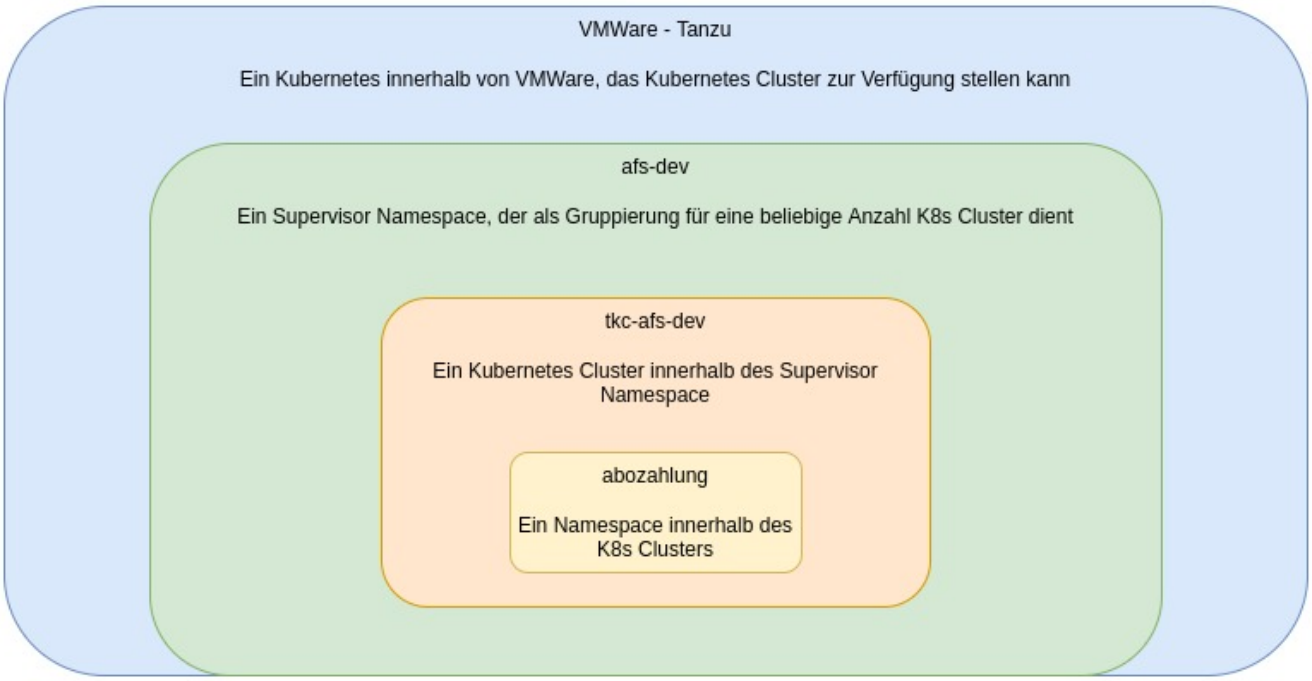
werden, um die Technologie auch sinnvoll für die Entwickler bereitzustellen. Viele Punkte wurden im PoC schon abgedeckt, aber der Teufel steckt ja bekanntlich im Detail. Auch Gedanken zu internen Abläufen, Prozesse zur Nutzung und das spätere Onboarding der Kollegen sollten noch die ein oder andere Stunde verbrauchen.

Was ist Tanzu eigentlich?

Aber noch einmal einen Schritt zurück. Erst ist von Kubernetes die Rede, dann Tanzu. Wo liegt denn jetzt eigentlich der Unterschied?

Wie zu beginn schon erwähnt, bringt Kubernetes im Container Kontext viele Vorteile. Im Management aber auch einige Hürden mit sich. Tanzu bildet eine Abstraktionsschicht um einige dieser Hürden und versucht so, den Management Overhead zu minimieren. Es bildet unter anderem ein Rahmenwerk aus verschiedenen Technologien, die nach Prüfung von VMWare in einem Zusammenspiel gut miteinander funktionieren und auch unterstützt werden. Somit werden zwar ein paar Standards vorausgesetzt, was die ein oder andere Einschränkung mit sich bringt, aber im großen Ganzen noch alle Anforderungen erfüllt. Gleichzeitig gibt es jemanden, der einem bei Problemen weiterhelfen kann. Genau diese Standards ermöglichen es ebenfalls, dass Updates über wenige Klicks in Tanzu abgebildet sind. Im VCenter habe ich mit Tanzu einen Überblick über alle mit Tanzu gebauten Kubernetes Cluster und auch deren Versionsstand. Ebenso finde ich hier eine simple Möglichkeit alle Cluster auch auf einen neuen Stand zu bringen. Was in dem Kontext nur fehlt, sind die Updates für Plugins und Erweiterungen. Aber auch hier bietet Tanzu einen Download kompatibler Versionen inkl. detaillierter Anleitung zur Umsetzung, was das Vorgehen deutlich vereinfacht.

Was es nicht einfacher macht? Abstraktionsschicht um Abstraktionsschicht wird die Komplexität trotzdem nicht geringer. Um es sich zu verdeutlichen, hier einmal der Aufbau der mittlerweile entstandenen Schichten für die Kubernetes Cluster:



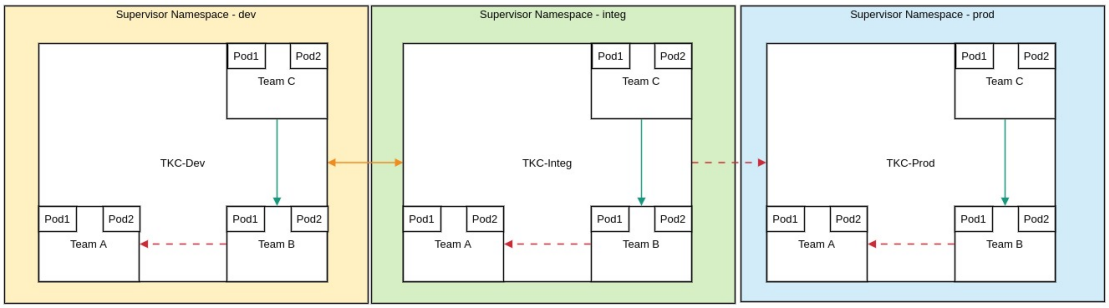
Architektur und Umsetzung

Die grundsätzlichen Überlegungen zu Abläufen etc. sind getroffen. Die Arbeitspakete geschnürt. Aber auch die Architektur soll wohl überlegt sein, bevor Sie Fallstricke enthält. Wie viele Cluster werden wir brauchen? Wie bauen wir das Netzwerk zwischen den Clustern auf? Brauchen wir ein Staging Konzept für die Cluster selber und nicht nur die Anwendungen? Das und vieles mehr galt es jetzt im konkreten Kontext anzugehen und zu klären.

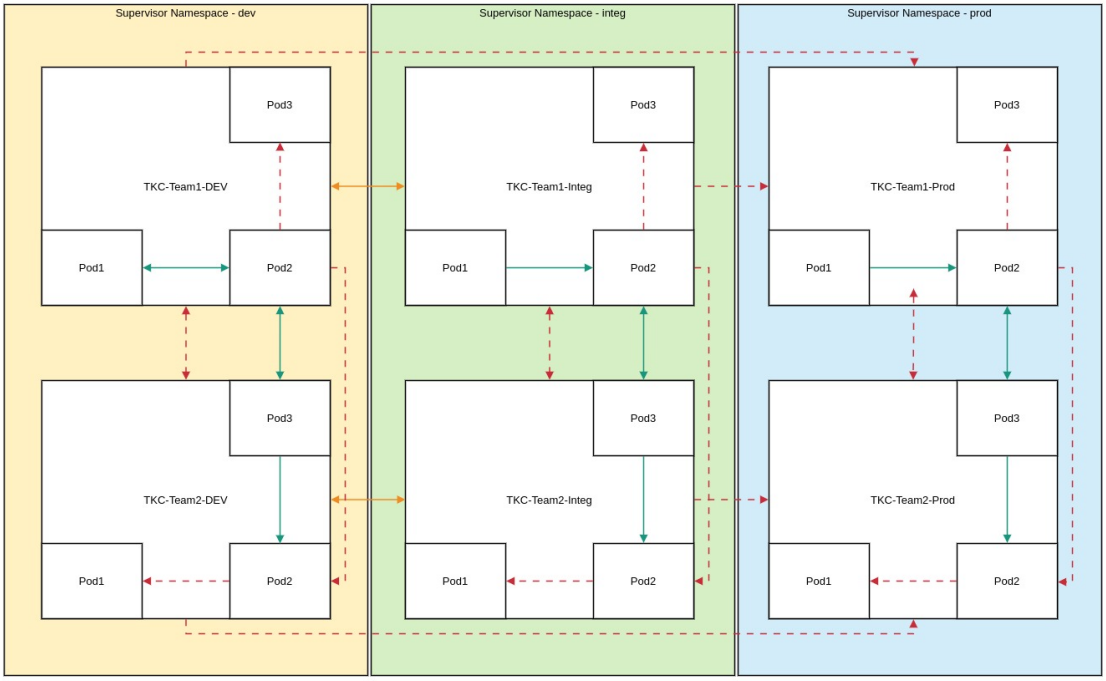
Einiges lässt sich einfach beantworten, anderes dann wiederum nicht. Schließlich sollen ja die unterschiedlichen Stages der Anwendungen wie Integration und Produktion bestmöglich voneinander getrennt sein. Auf der anderen Seite muss aber der Management Overhead überschaubar bleiben. Und dann gibt es ja nicht nur die unterschiedlichen Stages, sondern wir haben auch noch verschiedene Teams, die dann Ressourcen nutzen. Je mehr Cluster wir also aufbauen, desto komplizierter wird die Verwaltung des Konstrukts. Aber je weniger Cluster, desto schlimmer wird die Trennung – sowohl auf Netzwerk, der Ressourcen als auch der Berechtigungsebene.

Wie ein paar Überlegungen ohne nähere Erläuterungen aussehen können:

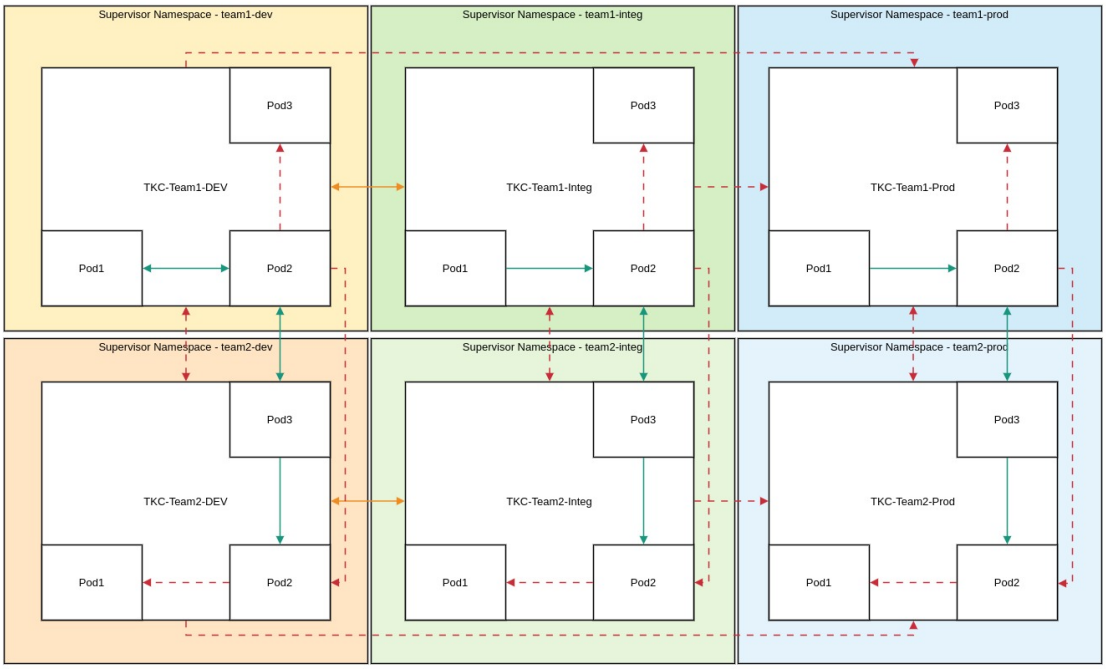
Option A



Option B



Option C



Architekturoptionen mit Kubernetes
Eines war klar, wir brauchen auf jeden Fall eine Automatisierungslösung. Eine Lösung, die uns so viel wie möglich von der Arbeit abnimmt und gleichzeitig dafür sorgt, dass wir die Infrastruktur reproduzieren können. Wie sonst auch soll dieser Ansatz basierend auf dem „Infrastructure as Code“ Prinzip abgebildet werden, womit die Wahl des Konstrukts am Ende unabhängig davon ist.

Ein wenig Recherche später qualmt der Kopf schon wieder. Sollten wir eine Lösung wie Flux oder ArgoCD nehmen? Genügt uns eine Jenkins oder Gitlab Pipeline? Ist Terraform die finale Lösung? Oder wird es womöglich eine Kombination aus mehreren Tools?

Wir haben ja schließlich mehrere Schritte, welche wir in der Provisionierung eines Clusters abdecken müssen. Zum einen der Bau eines Clusters über VSphere, die Einrichtung des Netzwerkes innerhalb des Clusters und auch die Anbindung der Services wie Monitoring, Logging und Alarming wollen in der Initialisierung bedacht werden. Und dann gibt es auch noch Sonderlocken von VMWare, wie z.B. Extensions, die einem teilweise die Arbeit abnehmen, durch die Standardisierung aber auch nicht für alle Fälle bei uns geeignet sind.

„Schuster bleib bei deinen Leisten“ heißt ein alter Spruch. Somit war die naheliegendste Lösung, vorerst eine Pipeline in unserem bestehenden Tool zu bauen. Die Daten sollen aus einem Git-Repository ausgelesen und alle nötigen Schritte hierin umgesetzt werden. Die Erfahrung wird zeigen, ob wir hier noch etwas nachzubessern haben und auf eine der anderen Lösungen schwenken, aber vorerst können wir alles abdecken. Und das ohne eine weitere neue Technologie, in die wir uns erst einarbeiten müssen.

Das Endergebnis des gesamten Setups im Tanzu Kontext kann sich aus unserer Sicht auf jeden Fall sehen lassen!

Onboarding

Der Tag der Tage ist gekommen. Nach all der Planung, Konzeptionierung und Umsetzung steht die Vorstellung vor der gesamten Entwickler-Community an. Doch was soll schon groß passieren? Die Kommunikation mit den Teams lief bereits vorher regelmäßig durch kurze Präsentationen des Standes. Unsere DevOps-Kollegen in den Teams haben uns im Piloten unterstützt. Und insgesamt

war das Feedback bisher durchweg positiv.

Was hiernach trotzdem noch fehlt? Die Migration. Wir freuen uns schon auf die Zusammenarbeit mit den Teams und sind gespannt, wie die Lösung im weiteren Verlauf auch unseren Betriebs-Alltag verbessern wird.

Ein Artikel von

Matthias Efker

Linux Systems Engineer

Suginthan Rushiyendra

Linux Systems Engineer

Du hast Lust ein Teil des Teams zu werden?

Neues Swift-Feature Async/Await

Auf der diesjährigen Swift Heroes Konferenz (April 2021) wurde in einem Vortrag von Vincent Pradeilles das neue Swift Feature Async/Await (ab Swift 5.5)

vorgestellt. Wir iOS Entwickler aus dem Thalia App Team haben uns den Vortrag angehört und waren begeistert von den neuen Möglichkeiten des Sprachfeatures, zur Vereinfachung von asynchronem Code. Andere Programmiersprachen (wie zB. C#) besitzen dieses Feature schon länger und wir sind froh, dass Swift jetzt nachgezogen hat.

Viele iOS Entwickler kennen das Problem, dass asynchroner Code schnell unleserlich, unübersichtlich und kompliziert werden kann. Genau dieses Problem soll Async/Await lösen, weshalb wir die Funktionsweise und Benutzung im folgenden Artikel mit Codebeispielen veranschaulichen wollen. Dabei stellen wir Async/Await auch den Alternativen gegenüber, um die Vorteile und Unterschiede zu bisherigen Verfahren deutlich zu machen.

Ziel des Artikels soll es sein, für Entwickler, die noch keine Berührung mit dem neuen Swift-Sprachfeature Async/Await hatten, die Grundlagen des Features zu erläutern. Für diejenigen, die sich schon ein bisschen mit dem Thema beschäftigt haben, empfehlen wir zur Vertiefung des Wissens die WWDC 2021 Session über Async/Await anzuschauen.

Problemstellung und Lösungsmöglichkeit mit Completions

In dem Beispiel aus dem Vortrag von Vincent Pradeilles soll der Anwender mit Vor- und Nachnamen begrüßt werden. (Konsolenausgabe) Dafür muss eine UserId geladen werden, um damit den Vor- und Nachnamen zu laden. Die klassische Lösungsmöglichkeit für dieses Problem in Swift sind Completions.

In Abb. 1 sehen wir, wie eine solche Problemlösung mit Completions aussehen könnte. Wir sehen das der Code nicht so gut leserlich ist, da er eine Verschachtelung von Methodenaufrufen enthält (Pyramid of Doom), was bei diesem Beispiel noch einigermaßen verständlich ist, aber bei mehr Aufrufen schnell sehr unleserlich und kompliziert werden kann.

```swift
func getUserData() {
    getUserId { userId in
        getUserFirstname(userId: userId) { firstname in
            getUserLastname(userId: userId) { lastname in
                print("Hello \(firstname) \(lastname)")
            }
        }
    }
}
```

Abb. 1: Asynchroner Swift Code mit Completions

Implementierung mit Async/Await

```swift
func greetUser() async {
    let userId = await getUserId()
    let firstname = await getUserFirstname(userId: userId)
    let lastname = await getUserLastname(userId: userId)

    print("Hello \(firstname) \(lastname)")
}
```

Abb. 2: Asynchroner Swift Code mit Async/Await

In Abb. 2 sehen wir, wie das Problem mit Async/Await gelöst werden kann. Das async-Keyword dient dazu dem Compiler zu signalisieren, dass in dieser Methode asynchrone Aufrufe stattfinden. Das await-Keyword bedeutet, dass hier auf das Ergebnis eines asynchronen Aufrufes gewartet werden muss. Im ersten Schritt wird über die Methode **getUserId()** die **userId** geladen. Erst wenn die **userId** geladen wurde, wird über die Methode **getUserFirstname** der **firstname** geladen. Im letzten Schritt wird der **lastname** geladen. D.h. die Aufrufe erfolgen **seriell** nacheinander.

Achtung: Enthält eine Methode ein await-Keyword, muss diese auch als asynchron über das async-Keyword markiert werden.

In Abb. 3 ist ein Beispiel zu sehen, um innerhalb der asynchronen Methode **getUserId()** eine Methode mit einer Completion aufzurufen (zB. um abwärtskompatibel zu bestehendem Code zu sein). Dazu kann man die Methode **withCheckedContinuation** aus der Swift Standard Library verwenden. Dabei wird die Rückgabe der UserId erst ausgeführt, wenn die **resume** Methode

auf dem Continuation-Objekt aufgerufen wird.

```
func getUserId() async -> Int {
    return await withCheckedContinuation({ continuation in
        getUserId { userId in
            continuation.resume(returning: userId)
        }
    })
}
```

Abb. 3: Aufruf einer Methode mit Completion aus einer mit async markierten Methode

In vielen Fällen ist es möglich asynchrone Methoden **parallel** auszuführen, um Zeit zu sparen. Mit Async/Await ist dies sehr einfach umzusetzen, durch Verwendung der Keywords **async let** wie in Abb. 4 zu sehen:

```
func greetUser() async {
    let userId = await getUserId()
    async let firstname = await getUserFirstname(userId: userId)
    async let lastname = await getUserLastname(userId: userId)

    await print("Hello \(firstname) \(lastname)")
}
```

Abb. 4: Parallele asynchrone Methodenaufrufe mit Async/Await

Zunächst wird, wie im ursprünglichen Code, die **userId** geladen. Durch die Deklaration der beiden Variablen **firstname** und **lastname** mit **async let** werden die beiden Methoden **getUserFirstname** und **getUserLastname** parallel ausgeführt. Erst wenn beide Methoden einen Rückgabewert liefern, wird das print-Statement ausgeführt. Dies wird erreicht durch das **await-Keyword** vor dem print-Statement.

Lösung mit Combine

Eine andere Lösungsmöglichkeit in Swift ist Combine zu nutzen.

In Abb. 5 sehen wir, wie das Problem mit Combine gelöst werden kann.

```swift
func greetUser() {
    _  = getUserId()
    .flatMap { userId in
        return getUserFirstname(userId: userId).zip(getUserLastname(userId: userId))
    }.sink(receiveValue: { firstname, lastname in
        print("Hello using Combine \(firstname) \(lastname)")
    })
}
```

Abb. 5: Asynchroner Swift Code mit Combine

Die Lösung mit Combine ist nicht so tief verschachtelt wie die klassiche Lösung mit Completions, allerdings für diesen Anwendungsfall auch nicht sehr leserlich und etwas kompliziert.

Was ist nun die bessere Lösung?

Die Frage lässt sich nicht so leicht beantworten. Combine ist ein Werkzeug, das viel mehr bietet als Async/Await. Allerdings löst Async/Await dieses spezielles Problem besonders gut. Je nach Anwendungsfall muss entschieden werden, ob eine Lösung mit Combine oder Async/Await die bessere Wahl darstellt.

Lösung mit besseren Server APIs

Quelle: Twitter (https://twitter.com/onmyway133/status/1407772929031651328?s=21)

Fazit

Wir haben gesehen, wie man mit Async/Await Code schreiben kann, der klassichen Code mit Completions ersetzt. Außerdem wie man Code mit Completions aus einer async-Methode aufrufen kann (Abwärtskompatibilität) und wie man mit async let mehrere Aufrufe parallel ausführen kann. Es wurde auch gezeigt das das Problem mit Combine lösbar ist.

Uns hat der Vortrag von Vincent Pradeilles auf der Swift Heroes 2021 und die WWDC 2021 Sessions zu Async/Await so gut gefallen, dass wir das Sprachfeature gerne in Zukunft in der Thalia App verwenden möchten. Dabei ist zu beachten, dass die Verwendung erst ab iOS 15 möglich ist.

Natürlich sind die anderen Lösungswege über Combine und Completions genauso valide und es muss von Fall zu Fall unterschieden werden, welche Lösung die Beste für das jeweilige Problem darstellt.

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.

---

hackathon@thalia in Münster 2021

Zusammen gegen den Corona Blues und für kreative Kundenlösungen:

Nach einem Jahr Pause haben wir 2021 wieder einen Hackathon veranstaltet, diesmal rein digital. Auch nach vielen Monaten üben und gestalten von neuen Online-Formaten sind diese noch immer spannende Experimente für uns.

Wir haben dieses sehr generische Motto gewählt, um möglichst viele Kolleg:innen anzusprechen und den Raum für innovative Perspektiven weit zu öffnen. Das hat funktioniert:

5 kreative Ideen mit Mehrwert für unsere (internen) Kund:innen konnten durch

ihren Pitch überzeugen und wurden an diesem Tag von spontan zusammengefundenen interdisziplinären Teams weiter gedacht.

Nicht nur ITler:innen, auch Expert:innen aus den Fachbereichen waren eingeladen sich zu beteiligen, was durch die unterschiedliche Ausrichtung der Ideen angenehm leicht gemacht wurde. So gab es sowohl eher fachlich als auch eher technisch motivierte Themen zur Auswahl.

5 kreative Ideen

1_Produktentwicklung auf Basis von Kundenfeedback

Worum ging es?

Nicht direkt die offensichtlichste Lösung übernehmen, sondern kreativ sein. Nicht direkt mit der Tür bzw. Lösung ins Haus fallen, sondern sich eingehend mit dem Problem auseinandersetzen. Das Problem verstehen, um die für den Kunden beste Lösung erarbeiten – das war das Ziel dieses Projektes.

Ein spontan zusammengefundenes Team aus verschiedenen Fachbereichen wurde dazu mit einer abgespeckten Variante der Methode Spark Canvas zunächst auf vorgefiltertes Kundenfeedback losgelassen.

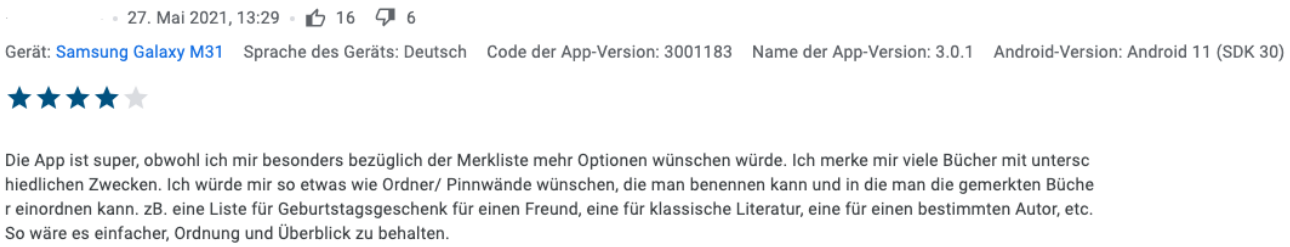


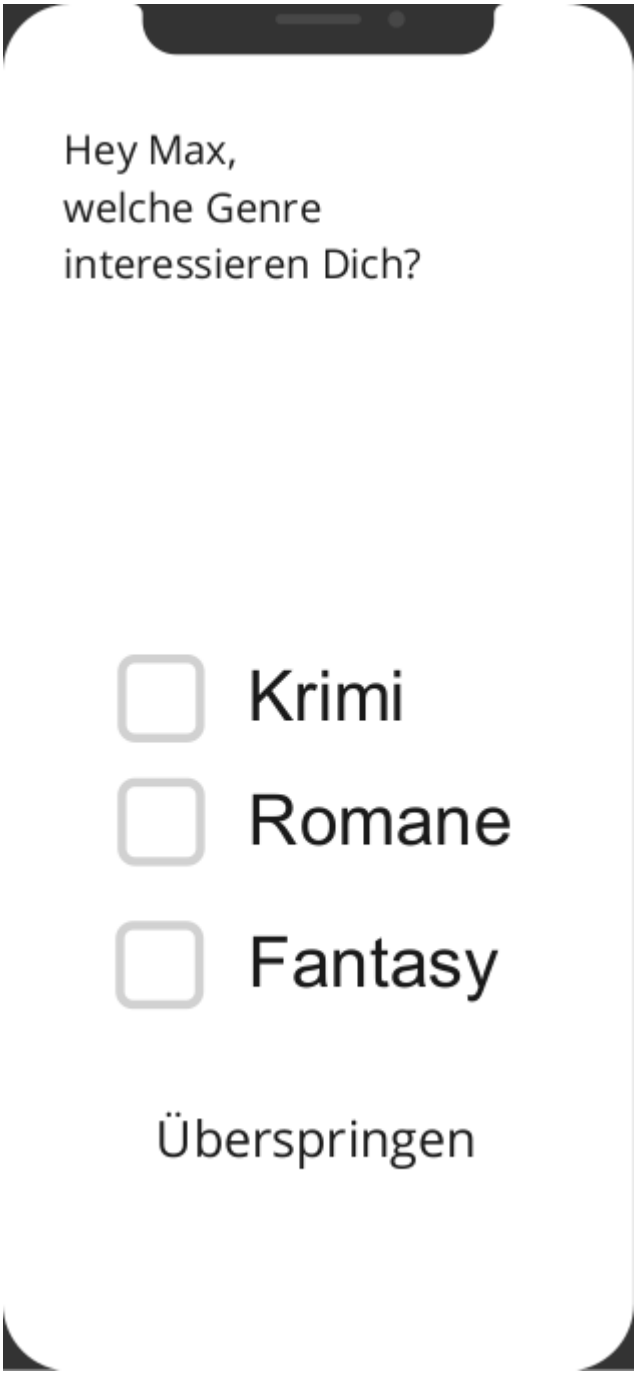
Abbildung: Eine berücksichtigte Rezension

Aus diesen wurde der Use Case „sich einen Überblick über interessante Inhalte bei Thalia verschaffen“ identifiziert und damit verbundene Challenges des Kunden erarbeitet. In einem zweiten Teil haben wir uns zunächst komplett vom

Produkt entfernt, um Inspiration an anderer Stelle zu suchen, z.B. bei der App „KptnCook“. Welche Probleme löst die App besonders gut? Können wir etwas von ihr lernen? Am Ende wurde als die für uns interessante Speciality die Begrenzung der Auswahl, bei KptnCook auf drei Rezepte pro Tag, gewählt.

Ergebnisse

Use Case, Challenges, Inspiration und Speciality wurden am Ende zusammengeführt zu dem Spark „Eine Startseite mit nur drei Büchern“:



Danke!
Und noch eine
Frage:

- [ ] liest du?
- [ ] hörst du?

Überspringen

Hey Max,
hier deine
Empfehlungen für
deine Woche

bereits gelesen | interessant

Die Geschichte vom Herz
Lorem larem ipsum dad
njidnkwendjwebn

Hey Max,
hier deine
Empfehlungen für
deine Woche

bereits gelesen | interessant

Die Geschichte vom Herz
Lorem larem ipsum dad
njidnkwendjwebn

ins Bücherregal

Notizen

★ ★ ★ ★

Hey Max,
hier deine
Empfehlungen für
deine Woche

bereits gelesen | interessant

Die Geschichte vom Herz
Lorem larem ipsum dad
njidnkwendjwebn

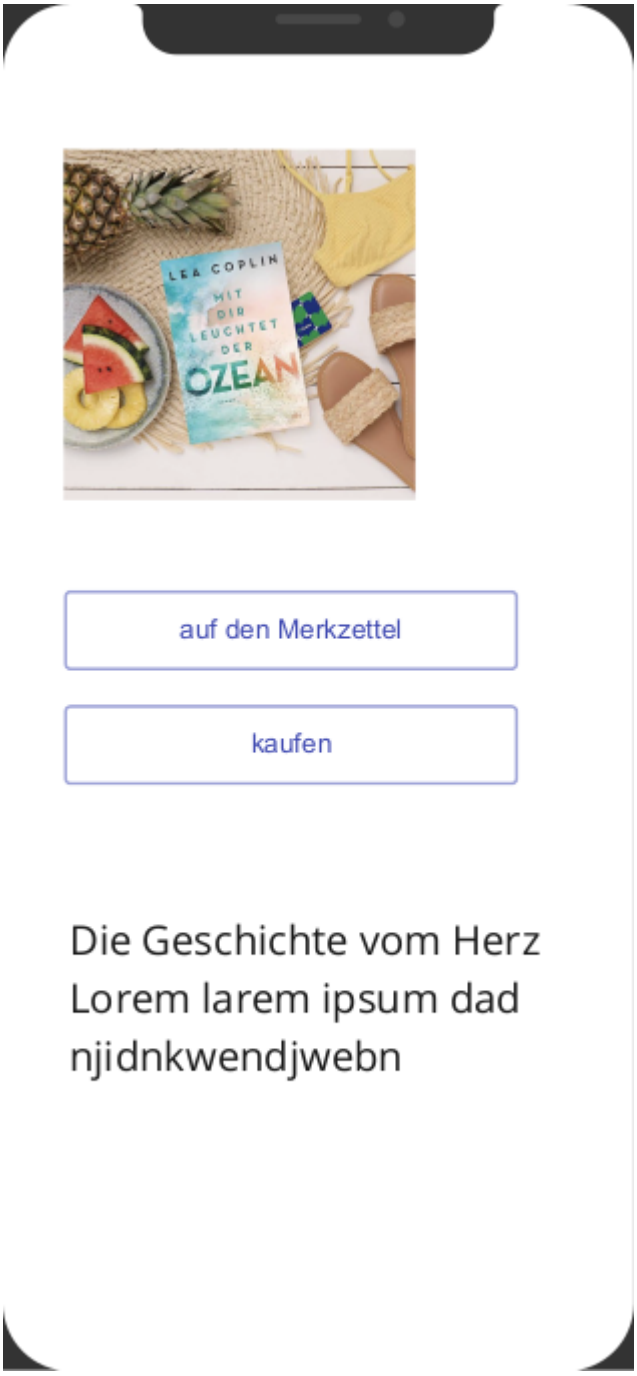


Abbildung: Prototypische Skizze des Sparks

Sich in einem Hackathon mit anderen Themen beschäftigen, oder, in diesem Fall, sich auf eine andere Art mit Themen beschäftigen hat allen viel Spaß gemacht. Es war toll zu sehen, was eine solche kreative Methode aus uns herausholen konnte. Wir sind davon überzeugt, dass dieses Feature unsere Kund:innen gut abholen kann, den Use Case und die identifizierten Challenges berücksichtigt. Er:sie bekommt regelmäßig „Futter“, das unserer Meinung nach gut passt und hat die Möglichkeit, dieses entsprechend einzusortieren.

Mal schauen, ob dieser Spark ein Feature-Feuer entzünden kann…

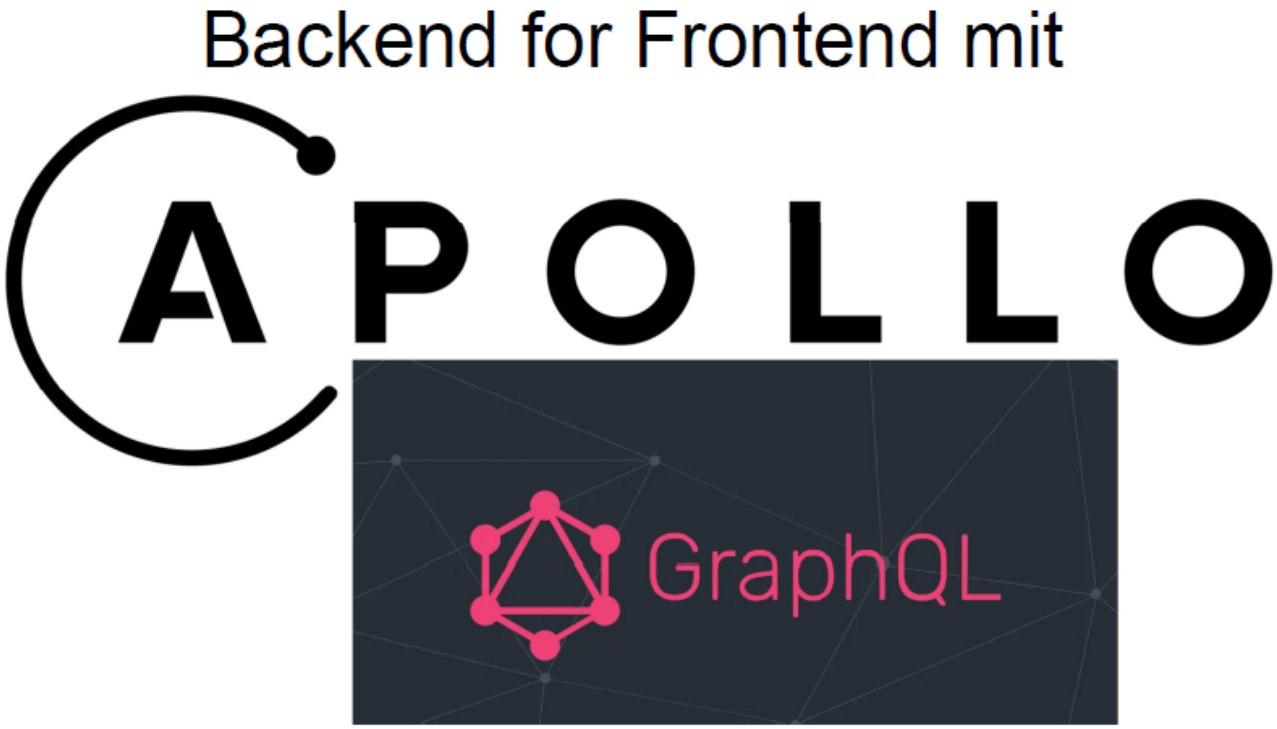
2_Backend for Frontend mit Apollo GraphQL

Worum ging es?

Ein Thema, welches uns in den letzten Monaten sehr stark beschäftigt hat. Treiber hierbei ist unser Thalia-App-Team. In der App werden sowohl native Elemente entwickelt als auch Komponenten und Klick-Strecken von anderen Entwicklungsteams eingebunden.

Das Problem am Beispiel beschrieben:

- Team A möchte in einem API-Aufruf alle (für sie) notwendigen Daten von der API von Team B bekommen, um diese in ihrem Frontend darzustellen
- Team B möchte die API nicht erweitern, um Daten „durchzuschleusen“, die nicht zur Domäne des eigenen Teams gehören.



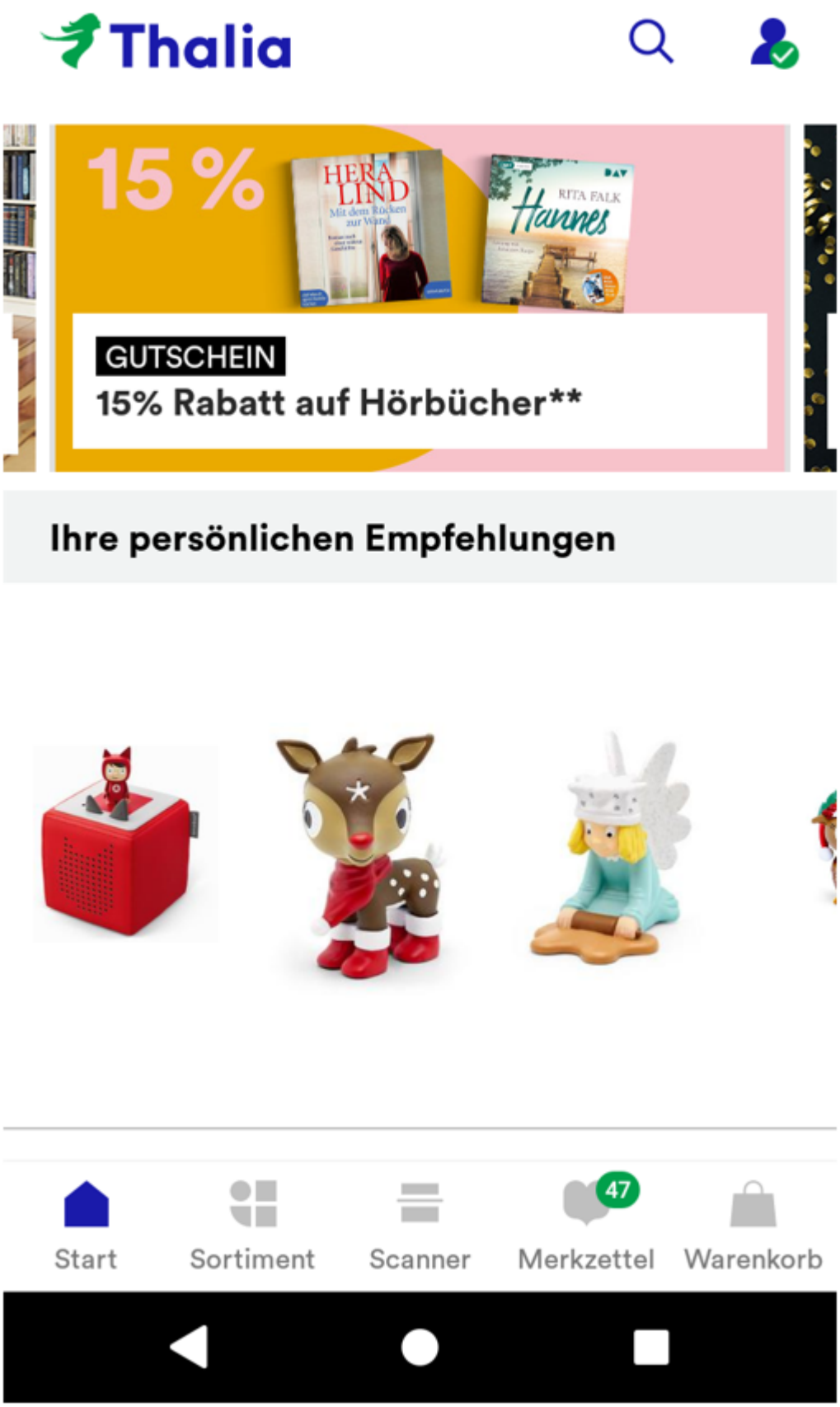
Kann diese Problemstellung mit Hilfe von Apollo und GraphQL vereinfacht werden?

Im Rahmen des Hackathon sollte versucht werden, ein Setup aufzubauen und

anhand einer einfachen Beispielapplikation die Daten von verschiedenen Thalia-Services, per Backend for Frontend aggregiert, zu konsumieren.

Ergebnisse

Unsere Thalia-App gibt es sowohl in der Android-Version, wie auch für iPhone und iPad-Geräte. An folgendem Use Case sollte nun die Verprobung stattfinden:



Das App-Team möchte über eine Schnittstelle nur Artikel inkl. Stammdateninformationen erhalten, für die es eine Empfehlung gibt.

Aktuell werden diese Daten bei uns von 2 unterschiedlichen Teams bereitgestellt

und – nicht überraschend – sind dementsprechend hierfür 2 Microservices verantwortlich:

Der Artikelservice stellt alle Artikelinformationen (EAN, Bilder, Texte, Preise etc.) zur Verfügung.

Der Empfehlungsservice kennt alle Artikel, für die es eine Empfehlung (Personalisierung, PaarKauf-Information, Tipps etc.) gibt.

Das Teams aus Java-Expert:innen, Android- und iOS-Spezialist:innen und Frontend-Entwickler:innen hat eine Umgebung aufgebaut, in der ein Proof of Technology auf beiden App-Versionen lauffähig war.

Per Apollo GraphQL konnten die Apps dann auf kombinierte Daten aus Empfehlungsservice und Artikelservice zugreifen. Der Empfehlungsservice hat in diesem Szenario dann nur ArtikeIds (Primärschlüsselinformation) von empfohlenen Artikeln und eben nicht alle Artikeldaten ausgespielt. Der Artikelservice hat dann den Rest der Daten bereitstellt.

Das Team war am Ende des Tages sehr zufrieden: Neben den gesammelten Erfahrungen hat sich die Einschätzung gefestigt, dass hiermit das beschriebene Problem gelöst werden konnte. Eine Fortsetzung im Daily Business ist mehr als wahrscheinlich. Ein prima Ergebnis!

3_Judge a book by its cover

Worum ging es?

Meine Thalia Lieblingsfiliale in Münster→ Rolltreppe in die erste Etage nehmen → dann leicht rechts halten → links hinter dem Info-Point: Mein Lieblingsbüchertisch! Kuratiert von den Kolleg:innen in der Buchhandlung – Neues, Altes, Preisgekröntes, Ungewöhnliches, Mutiges.

Wie schaffen wir es, die Exzellenz der Auswahl in den Webshop, die App, vielleicht sogar den Tolino-eReader zu bringen? Wir möchten den Kund:innen ein vergleichbares „Stöber-Erlebnis“ bieten, an einem Sonntag oder in einer

Lockdownphase. Oder anders formuliert: Wie können unsere Kund:innen bequem und barrierefrei diese Erfahrung an allen Berührungspunkten zu und bei Thalia nutzen?

Ergebnisse

Ein bunt gemischtes Team aus den Bereichen Business Development, UX, Softwareentwicklung, Coaching und Produktmanagement hat sich dieser Fragestellung angenommen. Möglichst schnell sollte die Ideenphase in eine Entwicklungsphase übergeben.

Mit Hilfe eines digitalen Whiteboards (Brainstorming, thematisches Clustern und Voting) und einer straffen Agenda hat die Gruppe, aufgeteilt in 2 Teams, die folgenden 2 Fragestellung bearbeitet:

- *Wie werden eigentlich die Tische zusammengestellt und wie kommen wir an die genaue Zusammenstellung der Artikel? Pro Tisch und pro Filiale?*
- *Wie kann so ein digitaler Tisch oder Regal aussehen?*

Für beide Fragestellungen haben wir Lösungen im eigenen Umfeld bei Thalia und Best Practices auf dem Markt angeschaut. Virtuelle Buchhandlungen, Raumbilder mit klickbaren Elementen & Augmented Reality: Bei der Recherche sind uns viele innovative Lösungen begegnetet, die wir uns mit mehr Zeit sicher genauer angeschaut hätten. Als Inspiration für einen ersten MVP aber schon sehr hilfreich.

Themenwelten, Empfehlungen unserer Lieblingsbuchhändler:innen, kuratierte Sortimentslisten – schon heute hätten wir etliche Möglichkeiten die Artikelzusammenstellung über Schnittstellen bereitzustellen. Verknüpft mit der eindeutigen Filialkennzeichnung, dann in einem weiteren Schritt über die Tischkennzeichnung haben wir für uns festgestellt, das kann funktionieren. Eine gute Nachricht. Mehr konnten und wollten wir an der Stelle nicht erreichen.

Für den Büchertisch selbst hatten wir schnell die notwendigen Artikelattribute definiert. Es sind genau die, die Kund:innen in der Situation sehen: Titel, Autor:in, Klappentext, Preis, Cover-Bild und, um nun den Schwenk auf eCommerce zu schaffen: Bestandsverfügbarkeit in der Filiale (8 Bücher auf dem Stapel), ein Link zur Artikeldetailseite und natürlich die Möglichkeit zu kaufen bzw. zu merken.

Das Ergebnis mit viel UX- und Frontend-Magic kann sich sehen lassen.

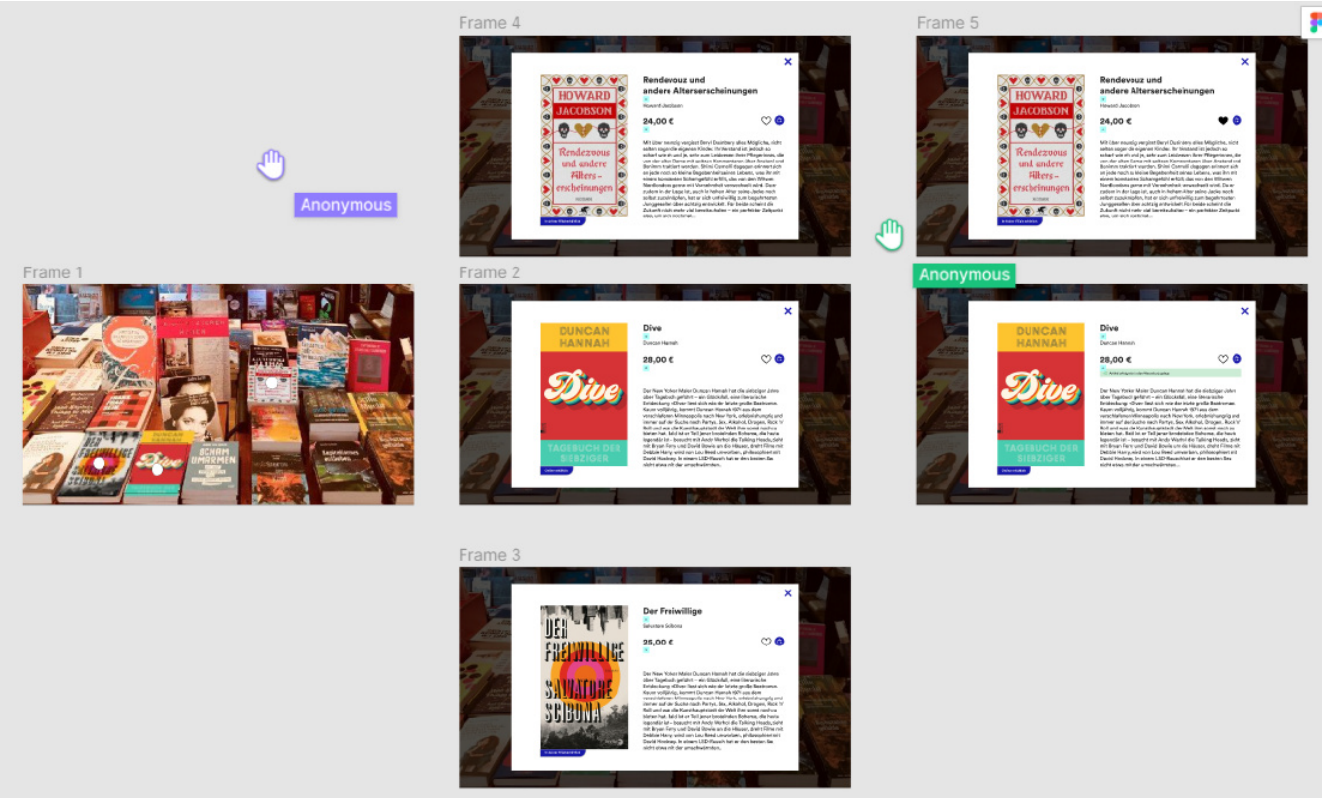


Abbildung: Ergebnispräsentation am Beispiel von 3 Artikeln des Büchertisches.

4_Frequent Pattern Mining FTW!

Worum ging es?

Personalisierung mit dem Ziel der perfekten Produkt-Empfehlungen sind für jeden eCommerce-Händler sowohl eine große Herausforderung wie a*uch eine signifikante Möglichkeit Kundenzufriedenheit und Umsatz zu steigern.*

Dabei konzentrieren wir uns hier auf den Zusammenhang zwischen gleichzeitig gekauften Produkten. Wie finden wir heraus, dass die Ähnlichkeit zwischen einem Schachspiel und einem Lehrbuch sehr wahrscheinlich größer ist als zwischen einer Schürze und einer DVD?

Oder um es mit den Worten eines Kollegen zu beschreiben: Frequent Pattern Mining ist ein Weg für faule Leute (z.B. Informatiker:innen) diese Zusammenhänge zu finden und nutzbar zu machen, z.B. über Paarkaufempfehlungen.

Abbildung: Paarkaufbeispiele auf Thalia.de
Ergebnisse

In einer überaus unterhaltsamen Präsentation wurden spannende Erkenntnisse geteilt:

- Datenanalyse, -bereinigung, -filterung im Vorfeld sind unabdingbar, um gute Ergebnisse zu erhalten. Schlicht 8 Millionen Transaktionen (mit

mehr als 2 Artikeln in einem Kauf) zu nehmen und dann auf ein Wunder zu hoffen, funktioniert nicht.
- Insbesondere die Filterung der richtigen Artikel erhöht signifikant die Ergebnisqualität. So blieben im Testlauf nur ~ 4.000 Artikel übrig, die in mind. 0,01% aller Transaktionen enthalten waren.
- Somit sind neben zu überprüfenden Artikelstammdaten (Stichwort kostenlose eBooks) auch übergreifende Sortimentsinformationen relevant.

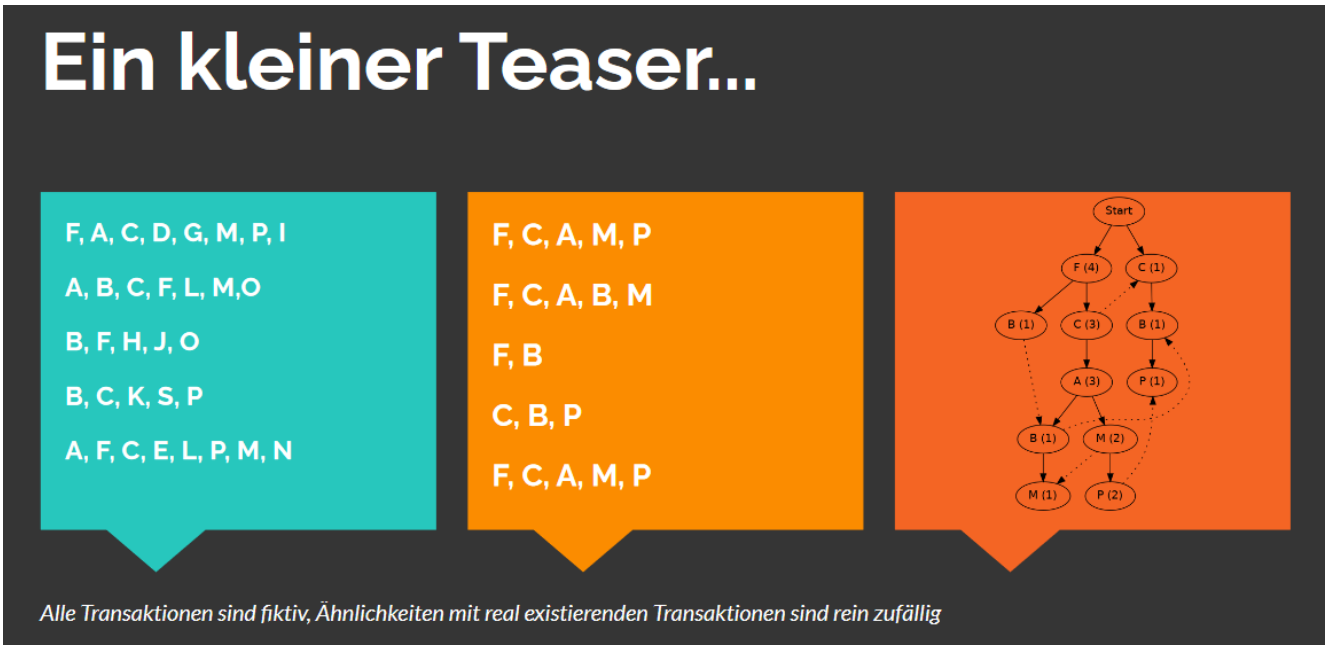


Abbildung: Transaktionen und Ähnlichkeiten

Unterschiedliche technische Ansätze im Bereich FPGrowth wurden ausprobiert und bewertet:

- Python #1 (aus Pandas): Schlüssig war diese Implementierung nicht und funktioniert hat sie für unseren Use Case auch nicht. Daher keine Empfehlung, da „irgendetwas“ gemacht wurde, aber nicht FPGrowth.
- Python #2 (github.com/chonyy): Funktions- und lauffähig, aber nicht optimiert auf unsere Anforderung. Auch hier: Viel gelernt, aber keine 5 Sterne.
- SparkML: Ein vielversprechender Ansatz, der ein wenig kompliziert zu implementieren ist. Jedoch überzeugt hier die Möglichkeit der Parallelisierung.

Wir haben gelernt, dass die Berechnung an sich mit einfachen Mitteln machbar ist, die Datenaufbereitung jedoch der Potentialbringer sein wird.

Eine Live-Demo vervollständige die gelungene Präsentation und hinterließ

weniger fragende Gesichter, als das doch sehr technische Thema im Vorfeld vermuten ließ. Well done!

5_Spring-Cloud-Config: Konfigurationsmanagement für Microservices

Worum ging es?

Konfigurationsmanagement ist häufig eine fehleranfällige und lästige Aufgabe in der Software-Entwicklung. Insbesondere in einer Microservice-Architekur, in der gesamtheitliche Funktionen von mehr als einem Produktteam implementiert, getestet und deployed werden.

Dieses betrifft dann nicht nur mehrere Service-Instanzen, sondern eben auch unterschiedliche Services. Für Integrationstest ist es wichtig, dass auch auf den Stages vor der produktiven Umgebung diese konsistent und effizient gepflegt werden. Auf die Frage, ob dies auch ohne das Durchführen einer Deployment-Pipeline oder Datenbankänderung erfolgen kann, wollte dieses Team eine Antwort geben und ist bereits mit einer konkreten Idee gestartet: Spring-Cloud-Config

Im Rahmen des Hackathons sollten Erfahrungen mit dieser Implementierung gesammelt werden und eine Machbarkeitsstudie anhand einer konkreten Beispielanwendung erfolgen.

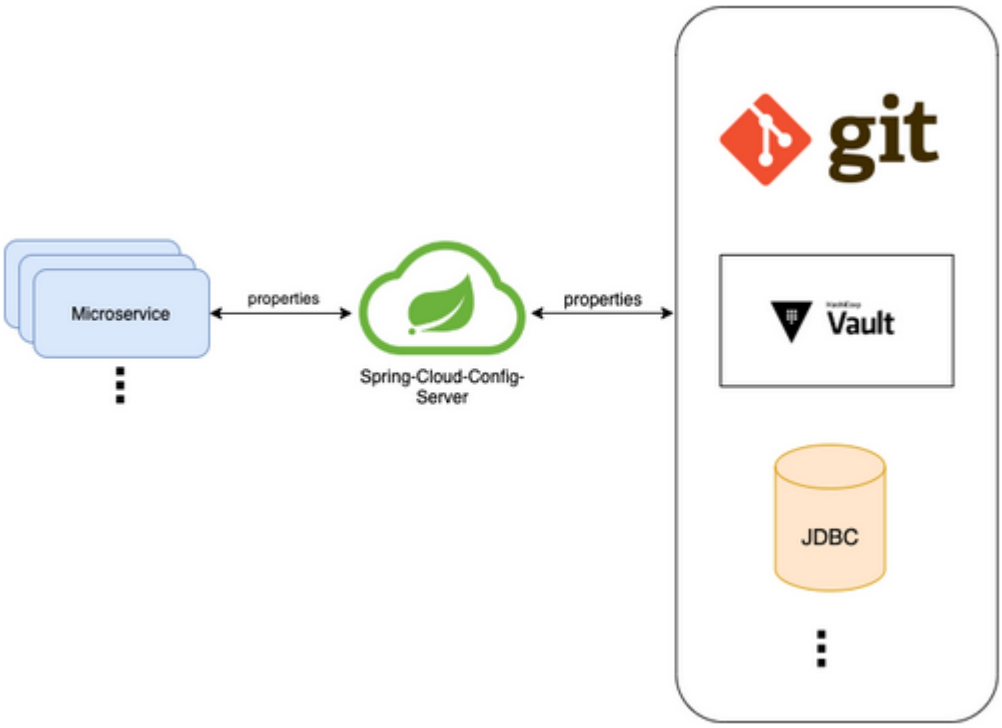


Abbildung: Infrastruktur Idee, s. auch https://cloud.spring.io/spring-cloud-config/reference/html/

Ergebnisse:

Das Team hat sich eine fachliche Konfiguration als Beispiel genommen: Eine generelle Versandkostenfreiheit soll Webshop-spezifisch eingestellt werden.

Als Anmerkung: Wir betreuen auf einer einheitlichen IT-Plattform unterschiedliche Shops, neben unserer Marke Thalia.de auch weitere, wie z.B. bol.de, Thalia.at oder orellfuessli.ch.

Die Versandkosten-Information werden auf der Prüfen&Bestellen-Seite im Rahmen des Checkouts interpretiert. Technisch übernimmt hier der bestelluebersichtsservice die Orchestrierung und ist somit der Hauptakteur des Experiments. Als Messaging-System ist RabbitMQ im Einsatz, die Datenbank ist MySql und das Ganze läuft in einem Docker-Container.

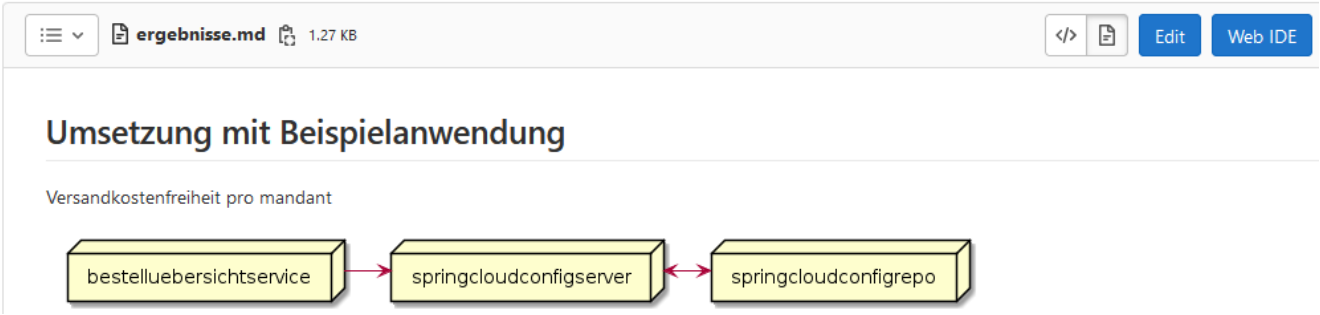


Abbildung: Konkrete Idee – Versandkostenfreiheit pro Webshop

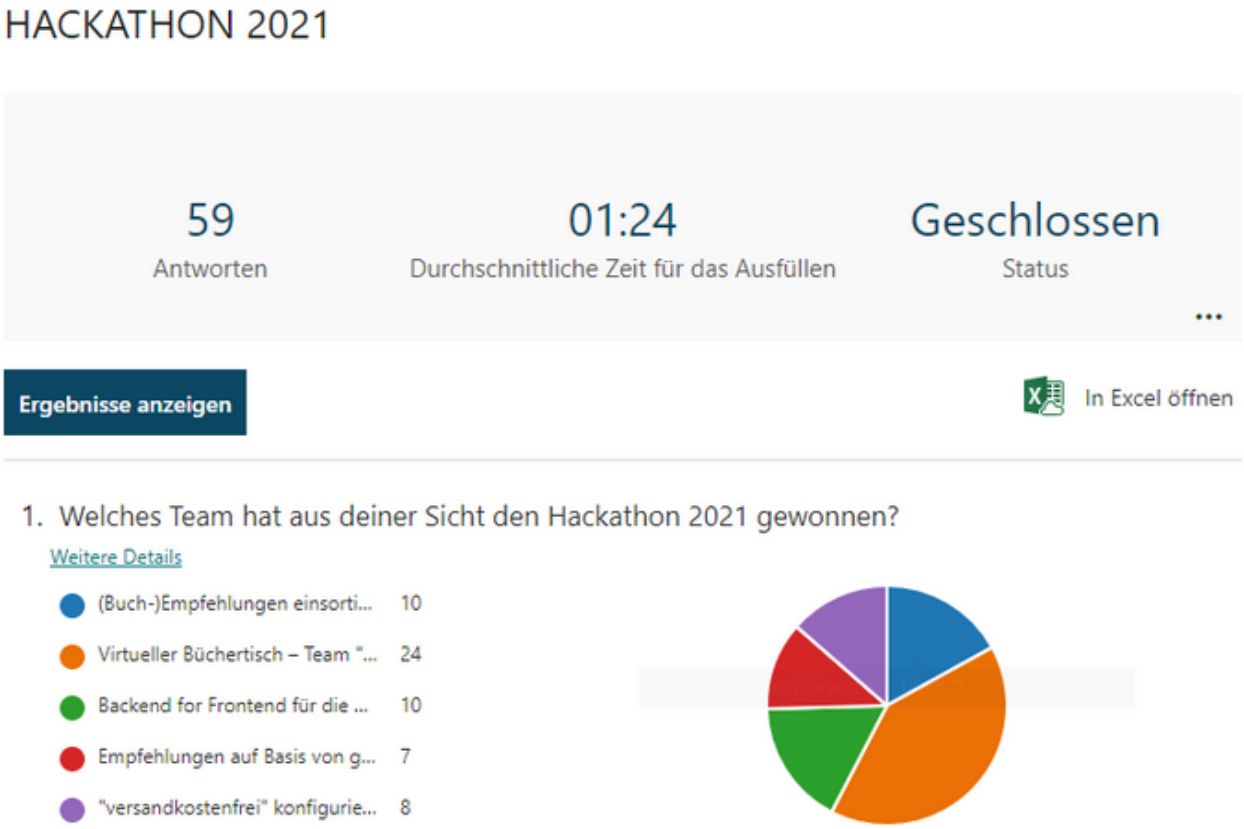
In GitLab wurde ein entsprechendes Projekt aufgesetzt und die einzelnen Schritte dokumentiert.

Mit Hilfe von Spring-Cloud-Config-Server konnten die im Use Case benötigten Konfigurationsänderungen zur Laufzeit und ohne Deployments durchgeführt werden. Das in der Vergangenheit bereits mit einem Spring Boot-Service gearbeitet wurde, war von Vorteil und hielt die Aufwände in Grenzen.

Im Experiment wurde auch deutlich, dass insbesondere der Anspruch, zentrale Änderungen service- und teamübergreifend zur Verfügung zu stellen, hier an seine Grenzen kommt. Nicht zwingend technisch, sondern organisatorisch. Wir halten unsere Microservice-Architektur bewusst flexibel, um schnell und effizient Veränderungen in Produktteams durchzuführen. Je mehr Shared-Services wir nutzen, desto höher sind die Kosten so einer Veränderung.

Auch wenn die Zeit etwas knapp wurde: Bewiesen wurde, wie einfach sich ein dynamischer Konfigurationsserver aufsetzen und einbinden lässt. Ob sich dies nun auch für weitere Anwendungsfälle rechnet und wir diese Lösung übergreifend einsetzten? Das Team hat mit diesem Proof of Concept eine gute Entscheidungsvorlage präsentiert.

Zusammenfassung



Das Experiment hat sich gelohnt. Ein digitaler Hackathon funktioniert. Auf die Frage, ob und wie sich die Thalia-Kolleg:innen einen Hackathon 2022 wünschen war die Antwort jedoch eindeutig: Machen, und zwar analog vor Ort. Wir geben unser Bestes.

Ach ja, war da nicht noch etwas mit einem Preis?! Ca. 60 Interessierte haben am Ende der Präsentationen ein Voting abgegeben. Unser neuer Wandpokal hat eine erste Signatur:

Die Teilnehmenden des siegreichen Teams Bookcave (# 2_Judge a book by its cover) konnten sich über diesen und ein kleines Überraschungspaket freuen.

Requirements Engineering –

Zurück zu den Wurzeln

In Retrospektiven und Lessons Learned wurde wiederholt Verbesserungspotential rund um das Thema Anforderungsmanagement (Requirements Engineering) benannt. Das Geschäftsmodell von Thalia bedingt eine enge Verzahnung von Prozessen und Touchpoints, um die Reise unserer Kund*innen zu einem inspirierenden und spielerisch einfachen Erlebnis zu machen. Und hält damit etliche Herausforderungen in der Produktentwicklung bereit:

- OmniChannel-Services – von der Filiale zum Webshop bis zum eReader
- Teamübergreifende Zusammenarbeit – von der Planung über den Schnittstellen-Kontrakt bis zum Regressionstest
- Integration vieler Stakeholder und deren Bedürfnisse – vom Marketing über den Kunden-Service bis zur Buchhaltung
- Priorisierung und Featureentscheidung – von der Idee über die Erfolgsmessung bis zur Weiterentwicklung

Aus diesem Wissen und Bedarf heraus haben wir vor einigen Monaten eine neue Community of Practise gegründet, mit dem Ziel, das Requirements Engineering (RE) bei Thalia weiter zu professionalisieren und dieses Wissen in die Organisation zu tragen.

Miro-Board – Was sind unsere wichtigsten Themen?

In einer der ersten Ideen, die wir in eine konkrete Maßnahme überführten, haben wir eine interne Schulung zu IREB® CPRE Foundation Level über einen externen Dienstleister organisiert. Wir wollten noch einmal besser verstehen, welche Werkzeuge uns zur Verfügung stehen und wie wir diese effektiver nutzen können. Hierbei haben wir uns bewusst für das Foundation-Level entscheiden, um eine gute Basis zu legen. Schwerpunkte dieses Levels sind das Kennenlernen von Terminologie, Methodik und Notation im Requirements Engineering.

Miro-Board – die erste Maßnahme

Für einige Kolleg*innen war es eine Auffrischung bereits bekanntem, aber manchmal auch verschollenem Wissen. Andere haben Neuland betreten.

Wie wichtig es ist, Wissen und Impulse aus von unterschiedlichen Rollen, Erfahrungen oder Kompetenzen zu bekommen, kennen wir aus der Arbeit in unseren crossfunktionalen Produktteams. Daher haben wir bei der Zusammensetzung der 10 Schulungsteilnehmenden auf diese Diversität geachtet. Konnten aber auch bei einigen Kolleg*innen den Wunsch nach individueller Weiterbildung inkl. einer Zertifizierung ermöglichen.

Welche Motivation hattest Du für die Schulung?

Felix (Software Developer): Grundlegend war meine Motivation, einen Überblick und eine Einordnung der Aufgaben und Werkzeuge des Requirements Engineering zu erhalten. Außerdem war mir der Erfahrungsaustausch mit Kolleg*innen aus anderen Teams besonders wichtig. Diese Fragestellungen waren dabei für mich primär relevant:

- Warum betreibt man Requirements Engineering?
- Wie geht man konkret an die Aufgaben heran?
- Was kann ich davon für mich in den Arbeitsalltag einbauen?
- Wie wird das Thema in den anderen Teams gelebt?

Lukas (QA Manager): Meine Motivation als QA Manager für das Thema Requirements Engineering liegt in der Tatsache, dass meine komplette Arbeit auf Anforderungen basiert. Eine meiner Hauptaufgaben ist es, Umsetzungen gegen

die erhobenen Anforderungen zu prüfen – aber auch die Sicherung der Qualität von Anforderungen vor deren Umsetzung ist ein Thema, das mir am Herzen liegt. Requirements Engineering hilft mir, den gesamten Lebenszyklus von Anforderungen mit einem Blick für Details zu begleiten.

Julian (Software Developer): Meine ursprüngliche Motivation war einen Überblick über die verschiedenen Aufgaben des RE zu bekommen. Außerdem wollte ich gerne die Möglichkeiten kennenlernen, wie RE in der agilen Vorgehensweise eingesetzt werden kann.

Christoph (App Developer): Als Touchpoint hat man bei Thalia fast zu jedem Team Berührungspunkte, sodass eine gute Kommunikation über Anforderungen unerlässlich ist. Dies habe ich nicht nur beim Schreiben der Tickets für andere Teams, sondern auch beim Bearbeiten unserer eigenen gemerkt. Nicht nur, dass das Verständnis für bestimmte Dinge nicht da war, oftmals waren die Grenzen für eine bestimmte Aufgabe nicht klar abgesteckt, sodass durch Nachfragen schon einiges an Zeit verloren ging. Aus diesem Grund habe ich mich für Techniken interessiert, welche ein gemeinsames Verständnis fördern und zudem den Aufgabenbereich klar definieren. Wichtig hierbei war es mir, dieses Wissen nicht nur theoretisch zu erlernen, sondern auch anhand praktischer Beispiele gezeigt zu bekommen.

> *Auch in agilen Vorgehensweisen kann davon ausgegangen werden, dass man durch die Anfangsiteration und die kontinuierlichen Aufwände für die Requirements-Themen über die gesamte Projektlaufzeit auf einen Aufwand von ca. 15%-20% des Gesamtprojektaufwands kommt. Das ist in etwa gleich oder etwas geringer als bei nicht agilem Vorgehen. Der wesentliche Unterschied ist, dass die Aufwände anderes verteilt sind und zielgerichtet genau dort anfallen, wo sie den größten Nutzen bringen.*
>
> *Johannes Bergsmann, Requirements Engineering in der agilen Softwareentwicklung, dpunkt.Verlag, 2. Aufl. 2018*

Wie korrespondiert nun ein vermeintlich klassischer Ansatz im Anforderungsmanagement mit den agilen Vorgehensmodellen innerhalb unserer

Produktentwicklung? Ziemlich gut. Was wir vorher schon ahnten, hat sich während der Schulung und den darauffolgenden Wochen manifestiert: Eine systematische, disziplinierte und strukturierte Herangehensweise unterstützt das iterative und inkrementelle Vorgehen. Die Community of Practise wird die Entwicklung mit großem Interesse in den nächsten Monaten verfolgen.

Hatte die Schulung schon Auswirkungen auf Deine tägliche Arbeit?

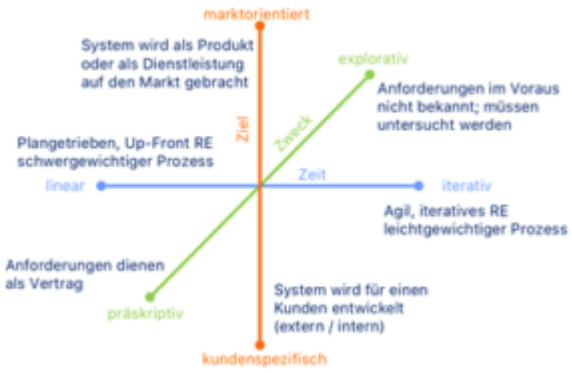
Jan (Product Owner): Allein wenn man die Prinzipien etwas deutlicher im Hinterkopf hat, hilft das schon in der täglichen Arbeit und der Überwindung des inneren Schweinehundes:

- Wertorientierung (Anforderungen sind Mittel zum Zweck, kein Selbstzweck)
- Stakeholder (Wünsche und Bedürfnisse der Stakeholder befriedigen)
- Gemeinsames Verständnis (Erfolgreiche Systementwicklung ist ohne eine gemeinsame Basis nicht möglich)
- Kontext (Systeme können nicht isoliert verstanden werden)
- Problem –Anforderung –Lösung (Ein unausweichlich ineinandergreifendes Tripel)
- Validierung (Nicht-validierte Anforderungen sind nutzlos)
- Evolution (Sich ändernde Anforderungen sind kein Unfall, sondern der Normalfall)
- Innovation (Mehr vom Gleichen ist nicht genug)
- Systematische und disziplinierte Arbeit (Wir können im RE nicht darauf verzichten)

Wenn ich Anforderungen bearbeite, achte ich stärker darauf und habe auch ein Gespür dafür, wo ich das gerade nicht tue, aber besser täte. Es fällt leichter Lösungen abzulehnen, die gar kein Problem lösen, oder wo dies nicht klar herausgestellt / weit genug gedacht ist Es macht die Entscheidung leichter, etwas zu tun, oder nicht zu tun.

Daniel (Software Developer): Mein erstes Aha-Erlebnis bei der Anwendung des Gelernten:
Ich habe den Kontext unseres Merkzettel-Service visualisiert und war sehr überrascht, wie groß und unübersichtlich dieser schon bei einer eigentlich recht überschaubaren Anwendung ist. Das bestätigt meine Annahme, dass es ohne systematisches RE schwierig ist, den Überblick zu behalten

Özge (QA Managerin): Die Schulung hat mich dazu gebracht, mir einige Fragen über die Art und Weise zu stellen, wie wir testen und vor allem, wie wir unsere Anforderungen validieren. Deshalb haben wir dieses Thema auch in unsere QA Community of Practice eingebracht, um herauszufinden, welchen Ansatz unsere Kollegen bei der Validierung verschiedener Arten von Anforderungen verwenden. Ich glaube, dass die Ergebnisse unseres kommenden Workshops uns dazu bringen werden, andere Perspektiven zu sehen, die uns bei der Planung unserer Iterationen helfen werden. Für mich war der interessanteste Moment, herauszufinden, dass die REs eine Schlüsselrolle im Stakeholder-Management spielen und dass es so viele Techniken gibt, um Konflikte zu lösen.



https://www.microtool.de/requirements/der-neue-lehrplan-ireb-cpre-foundation-level-3-0/

Deniz (Software Developer): Grundsätzlich denke ich, wir machen schon viel von dem Erlernten, könnten aber hier und da vielleicht noch präziser werden, da vieles nicht immer konsequent angewandt wird (z.B. Stories nur mit Akzeptanzkriterien). Es gibt auch Dinge, die wir gefühlt nicht machen: Zum Beispiel die Versionierung von Anforderungen zur Nachverfolgbarkeit oder die Validierung von Anforderungen. Einen großen Impact könnten wir denke ich auch erreichen, wenn wir einige Punkte an die Fachbereiche und die Geschäftsleitung spiegeln. Ich denke da z.B. an den angehängten Screenshot. Hier sieht man das wir häufig komplett komplementäre Ziele verfolgen. Wir wollen marktorientiert sein, sind aber häufig kundenorientiert.

Thalia bei der DTS-World

Am 25.02.2021 findet im virtuellen Showroom die DTS-World unseres Dienstleisters DTS-Systeme statt.

Im Rahmen der DTS-World wird es auch ein Interview mit unserem Kollegen Alexander Waltering zum Thema Security geben.

Wer sich dafür interessiert, der findet hier weitere Information und kann sich auch anmelden: https://www.dts.de/dts-world-anmeldung.html

25. Februar 2021

Agenda DTS World

09.30 – 10.00 Uhr **Opening Keynote**
Kai Mallmann, CEO
DTS Systeme GmbH

10.00 – 10.30 Uhr **Cyber Security Predictions**
Sergej Epp, Chief Security Officer I Cyber Security Advisor
Palo Alto Networks

10.45 – 11.30 Uhr **Ganzheitliche Cyber Security Plattform**
Markus Kohlmeier, Head of Cyber Security
DTS Systeme GmbH

13.30 – 14.00 Uhr **Keynote**
Kai Mallmann, CEO
DTS Systeme GmbH

14.00 – 14.45 Uhr **Aktuelle Bedrohungen aus dem Internet in DE**
Norbert Heuermann
Kriminalpolizei/Landeskriminalamt KHK

15.00 – 15.30 Uhr **Ihr Weg in die Cloud**
Timo Butenkemper, Head of Sales – Cloud
DTS Systeme GmbH

15.30 – 16.30 Uhr **Customers Stories**

- Frederik Neuhaus, Co-Founder & CEO, Clockin GmbH
- Julian Heinl, Leiter Security, Vedes AG
- Wolfgang Fey, Leiter IT, Aschendorff Medien GmbH & Co.
- Alexander Waltering, Microsoft Systems Engineer IT eCommerce – Operations, Thalia Bücher GmbH

Markus Nienaber, Regional Director & Johann Miske, Senior Account Manager DTS Systeme Münster GmbH