SonarQube Integration in einem Android Projekt

Um eine bessere Code Qualität an unseren Softwareprodukten zu gewährleisten haben wir uns im Unternehmen entschlossen eine statische Code Analyse einzuführen.
Ein besonderer Augenmerk liegt hierbei auf der Code Coverage und das Einhalten von zuvor festgelegten Programmier-Richtlinien.

In diesem Artikel geht es um das Erstellen+Anzeigen von Testreports auf einem SonarQube Server mittels Jenkins, um das Herunterladen+Anzeigen dieser in Android Studio und um das Anzeigen von lokalen, neuen SonarQube Issues. Desweiteren wird noch kurz auf die SonarQube-Benutzer-Oberfläche eingegangen.

SonarQube ist ein Tool für die statische Code Analyse. Hierfür werden zuvor erstellte Test Reports von SonarQube eingelesen und nach bestimmten Richtlinien und Regeln ausgewertet.

Vorbereitung des lokalen Projekts

Zuerst muss das Android-Projekt so konfiguriert werden, dass es Jacoco Test Reports erstellt. Dafür wird eine jacoco.gradle Datei erstellt:

```
apply plugin: 'jacoco'

ext {
    coverageSourceDirs = 'src/test/java'
}

jacoco {
    toolVersion = "0.8.3"
    reportsDir = file("$buildDir/reports")
}

task jacocoTestReport(type: JacocoReport, dependsOn: "testDebugUnitTest") {
    group = "Reporting"
    description = "Generate Jacoco coverage reports for Debug build"

    reports {
        xml.enabled = true
        html.enabled = true
    }

    def excludes = ['**/R.class',
                    '**/R$*.class',
                    '**/build/generated/**',
                    '**/*$ViewBinder*.*',
                    '**/*$InjectAdapter*.*',
                    '**/*Injector*.*',
                    '**/BuildConfig.*',
                    '**/Manifest*.*',
                    '**/*Test*.*',
                    '**/CiMattersApplication*.*',
                    'android/**/*.*']

    def debugTree = fileTree(
            dir: "$buildDir/intermediates/javac/debug/compileDebugJavaWithJavac/classes",
            excludes: excludes)
    def kotlinDebugTree = fileTree(
            dir: "$buildDir/tmp/kotlin-classes/debug",
            excludes: excludes)
    def mainSrc = "${project.projectDir}/src/main/java"

    classDirectories = files([debugTree], [kotlinDebugTree])
    executionData = files("$buildDir/jacoco/testDebugUnitTest.exec")
    sourceDirectories = files([mainSrc])
}
```

In dieser Datei wird der Gradle-Task zur Erstellung des Test-Reports angelegt und die entsprechenden Pfade für die Source- und Class Dateien für die Java- und Kotlin Klassen angegeben.

In der build.gradle Datei des Moduls muss dann noch die jacoco.gradle Datei hinzugefügt werden.

```
apply from: '../jacoco.gradle'

android {
```

Zum Schluss müssen noch einige Konfigurations-Einstellungen für Sonar in den gradle.properties angegeben werden.

```
sonar.projectVersion=3.2.2.0.1
# ------------------
sonar.projectKey=gsp
sonar.projectName=Douglas Beauty Tab (Android)
sonar.modules=gsp
sonar.sources=src/main/java/de/douglas/gsp
sonar.scm.provider=git
sonar.java.binaries=build/intermediates/javac/debug/compileDebugJavaWithJavac/classes/de/douglas/gsp
sonar.coverage.jacoco.xmlReportPaths=build/reports/jacocoTestReport/jacocoTestReport.xml
```

Nun können mit den folgenden Gradle-Tasks die Test-Reports erstellt werden.

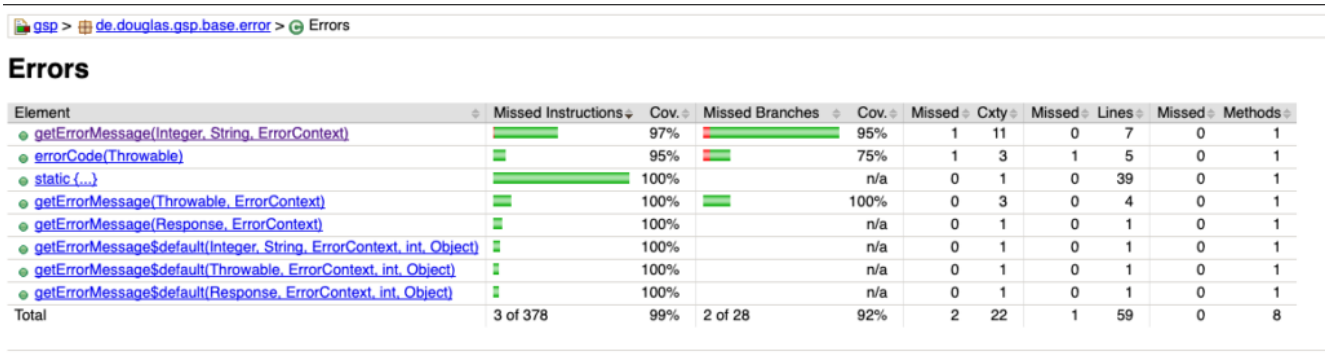
1. **Build**

– baut das Projekt und generiert die benötigten Class-Datein

2. **testDebugUnitTest**
   – führt die Tests aus und erstellt einen jacoco Test-Report (modul/build/jacoco/testDebugUnitTest.exec)
3. **jacocoTestReport**
   – erstellt die Test-Reports im HTML Format (modul/build/reports/jacocoTestReport/)

Es
kann auch nur der jacocoTestReport-Task ausgeführt werden, da dieser eine Abhängigkeit zu den anderen Tasks beinhaltet. Bei einem cleanen Projekt benötigt dies aber mehr Zeit.

Nach dem Ausführen der Gradle-Tasks liegen die Reports im Modul-Ordner unter build/reports/jacoco/TestReports im HTML-Format vor und können im Browser angezeigt werden.


gsp > de.douglas.gsp.base.error > Errors

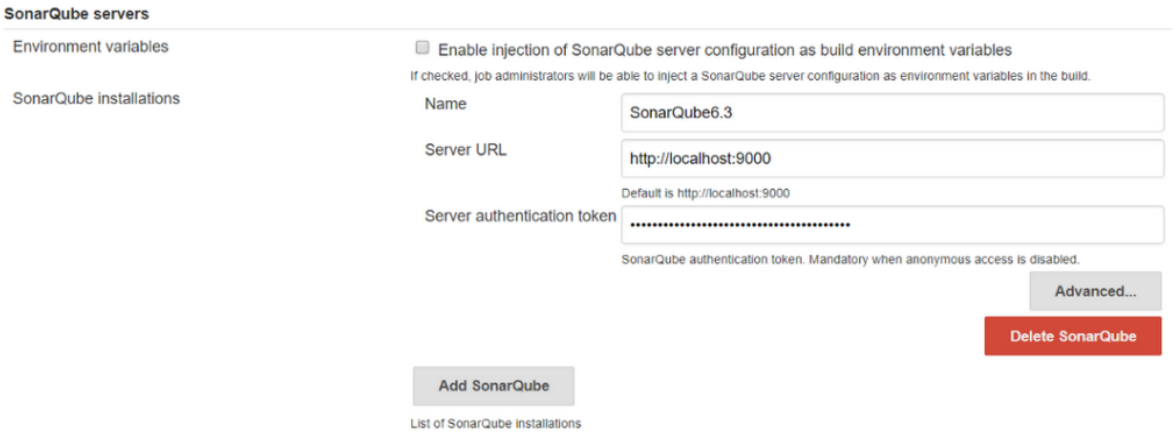
Errors

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getErrorMessage(Integer, String, ErrorContext)		97%		95%	1	11	0	7	0	1
errorCode(Throwable)		95%		75%	1	3	1	5	0	1
static {...}		100%		n/a	0	1	0	39	0	1
getErrorMessage(Throwable, ErrorContext)		100%		100%	0	3	0	4	0	1
getErrorMessage(Response, ErrorContext)		100%		n/a	0	1	0	1	0	1
getErrorMessage$default(Integer, String, ErrorContext, int, Object)		100%		n/a	0	1	0	1	0	1
getErrorMessage$default(Throwable, ErrorContext, int, Object)		100%		n/a	0	1	0	1	0	1
getErrorMessage$default(Response, ErrorContext, int, Object)		100%		n/a	0	1	0	1	0	1
Total	3 of 378	99%	2 of 28	92%	2	22	1	59	0	8

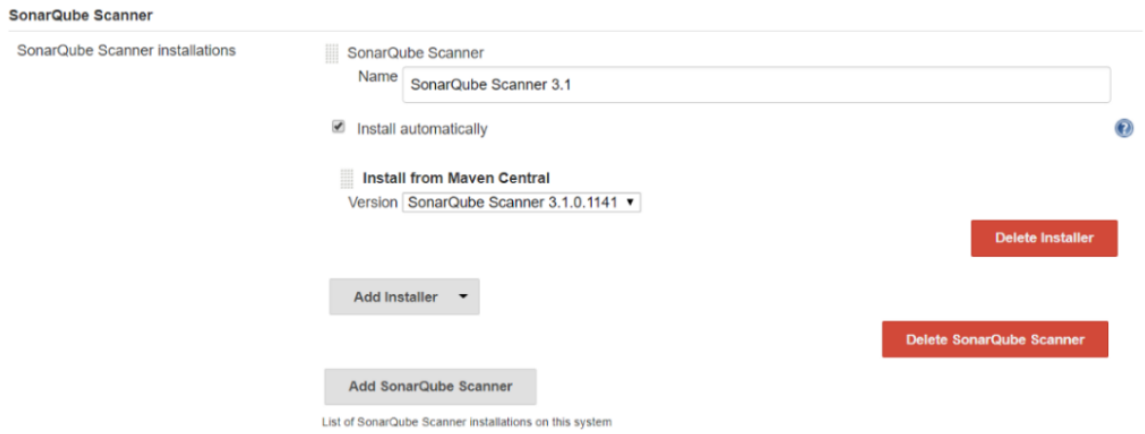
Vorbereitung des Jenkins-Servers

Um sich die Reports im Sonar an zu schauen, kann man diese über den Jenkins-Server zu Sonar übertragen.
Im Jenkins muss zuerst das Sonar-Plugin installiert werden, bei dem die URL des Servers eingetragen wird und der Sonar Runner ausgewählt werden.

Die Einstellung für das Sonar-Plugin finder man unter:
Manage Jenkins -> Configure System -> SonarQube

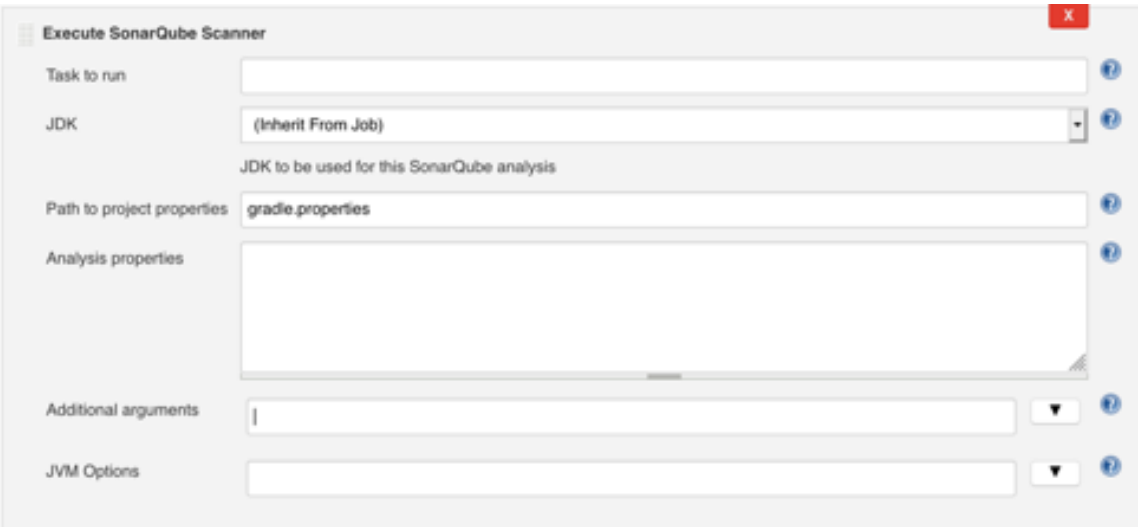


Die Einstellung für den Sonar-Runnar befindet sich unter:
Manage Jenkins -> Global Tool Configure -> SonarQube Scanner



In dem Jenkins-Job müssen dann noch die folgenden Gradle-Tasks eingetragen werden:

Anschließend wird der SonarQube Scanner konfiguriert. Hier muss die Datei mit den Properties für Sonar angegeben werden.

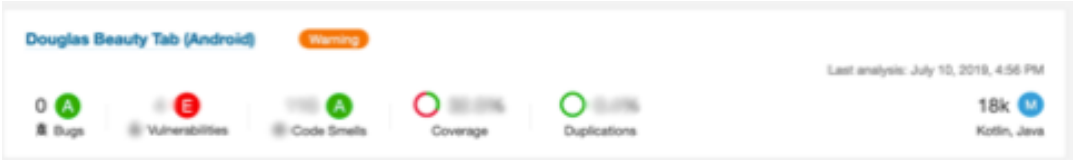


Nun wird jedes Mal, wenn der Jenksins-Job für das Projekt ausgeführt wird, auch der Sonarqube Scanner ausgeführt und veröffentlicht seinen Report auf dem SonarQube-Server.

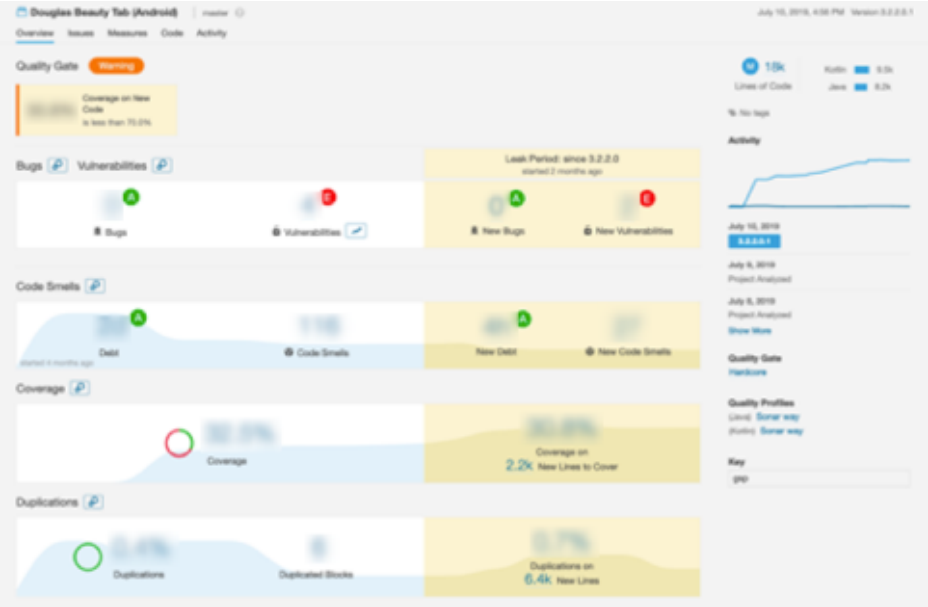
Benutzeroberfläche des Sonar-Servers

Damit Sonar die Testreports empfangen kann und diese korrekt angezeigt werden, müssen hier noch das SonarKotlin-, das SonarJava- und das Git-Plugin installiert werden.

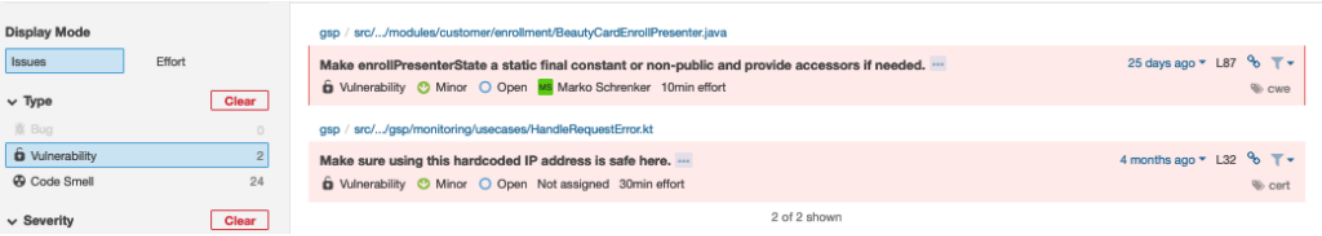
Auf der Startseite von Sonar werden alle Projekte aufgelistet und eine Übersicht der Analyse angezeigt.



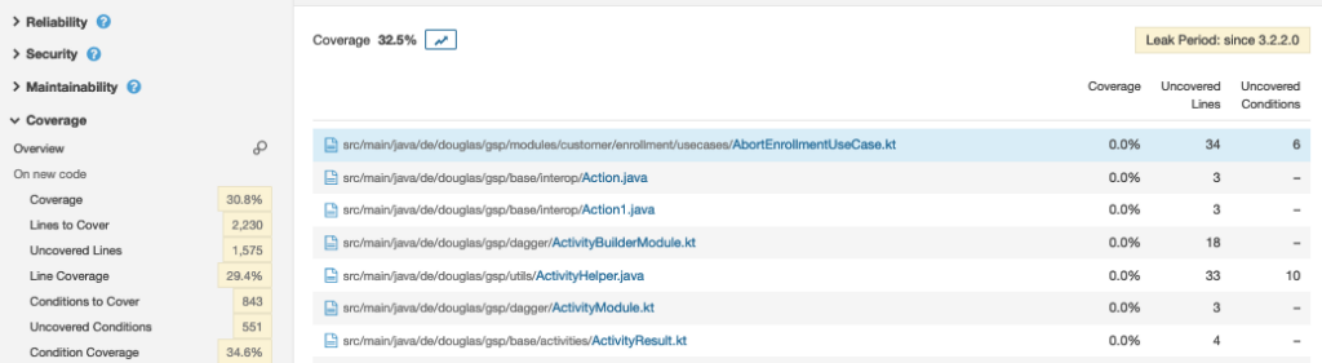
Klickt man auf ein Projekt, werden detailliertere und zusätzlich Informationen angezeigt.
Man kann die Analyse in 2 Kategorien einteilen. Das sind zum Einen Messungen (Code Coverage, Duplication) die SonarQube anhand des Codes durchführt und zum Anderen Issues (Bugs, Vulnerabilities, Code Smells), die durch Verletzung von zuvor festgelegten Code Richtlinien auftreten. Eine zusätzliche Richtlinie kann man für ein sogenanntes Quality Gate festlegen. Dies kann zum Beispiel eine Mindestanforderung von der Code Coverage des neu entwickelten Codes sein.



Über die einzelnen Analyse-Kategorien gelangt man zu einer detaillierten Ansicht der einzelnen Issues oder Messungen.



Isseus (Bugs, Vulnerabilities, Code Smells)



Messungen (Code Coverage, Duplications)

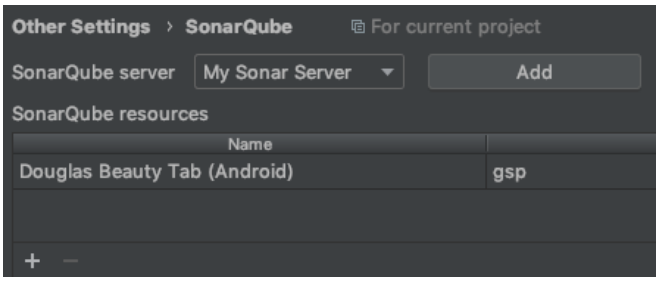
SonarQube Community Plugin

Für
Android-Studio gibt es das SonarQube Community Plugin, welches 2 Funktionen mit sich bringt:

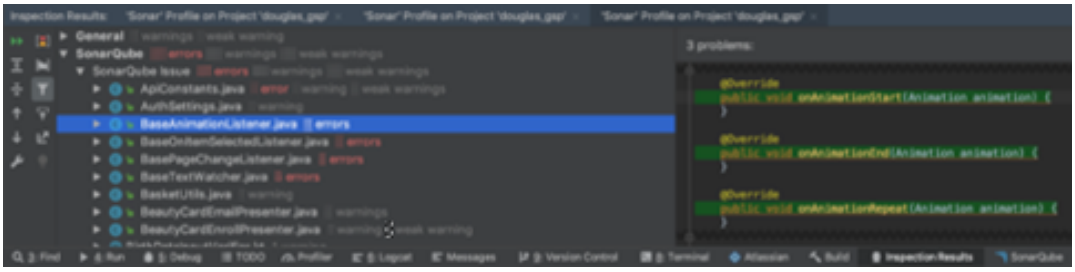
1. Anzeigen von vorhandenen Issues vom Sonar-Server
2. Anzeigen von neuen Issues mittels einer lokalen Projekt-Analyse

Vorhandene Issues vom Sonar-Server

Um sich die vorhandenen Issues vom Server herunterzuladen, muss das Plugin zuerst konfiguriert werden. Hierzu ruft man dieses in den Einstellungen von Android-Studio auf und trägt den Sonar-Server ein. Anschließend wählt man das Projekt in dem Abschnitt Resourcen aus.

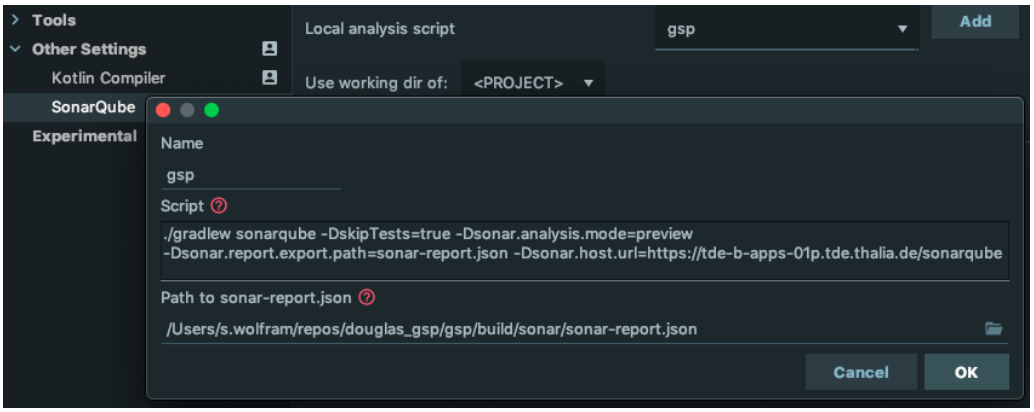


Unter dem Reiter Analyse -> Inspect Code muss in dem Inspection Profile SonarQube ausgewählt werden. Nun werden bei der Code Inspection die Sonar-Issues vom Server heruntergeladen und in der IDE angezeigt.



Lokale Projekt-Analyse mit dem Sonar-Gradle-Plugin

Um das lokale Projekt nach Sonar-Issues zu analysieren muss zunächst das SonarQube-Script konfiguriert werden. Hier muss der Sonar-Gradle-Task und der Pfad zum sonar-report eingestellt werden. Der Sonar-Report ist vor dem ersten Ausführen noch nicht vorhanden, da er erst durch den Gradle-Task erstellt wird.



Als nächster Schritt muss lokal SonarQube konfiguriert werden. Hierzu wird

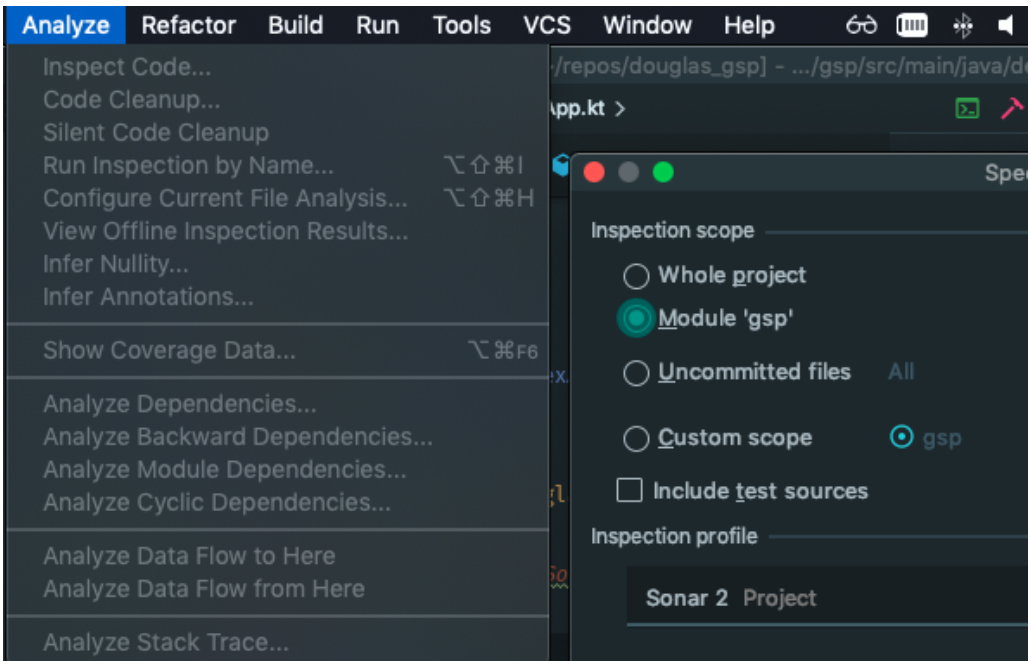
zuerst in der build.gradle Datei des Projekts sonarQube als Dependencie hinzugefügt.

```
dependencies {
    classpath "org.sonarsource.scanner.gradle:sonarqube-gradle-plugin:2.7.1"
```

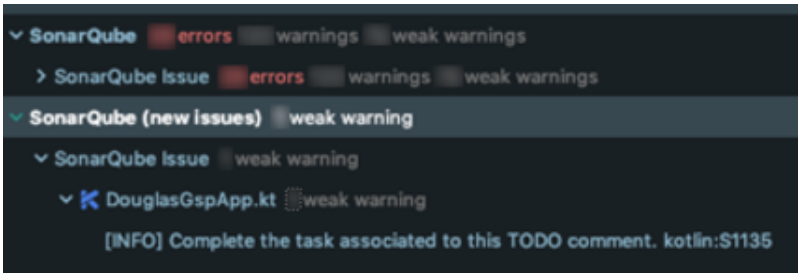
In der build.gradle Datei des Moduls wird SonarQube dann applied und konfiguriert.

```
apply plugin: 'org.sonarqube'
sonarqube {
    properties {
        property 'sonar.projectName', 'Douglas Beauty Tab (Android) - From Local'
        property "sonar.host.url", "https://tde-b-apps-01p.tde.thalia.de/sonarqube"
        property "sonar.jacoco.reportPaths", "build/jacoco/testDebugUnitTest.exec"
        property "sonar.projectVersion", version
        property "sonar.scm.provider", "git"
    }
}
```

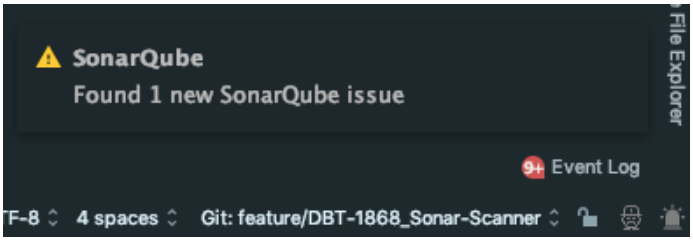
Nachdem SonarQube konfiguriert wurde, kann das Projekt unter Analyze -> Inspect Code analysiert werden. Hierzu muss nur noch das Sonar-Profile (mit Sonar Issues + new Sonar Issues) ausgewählt und mit OK bestätigt werden.



Nach der Analyse werden die Sonar-Issues vom Server und die aus der lokalen Analyse unten im Reiter Inspection Results angezeigt.



Neu gefundenen lokalen SonarQube Issues werden zusätzlich noch in einem kleinen Popup unten rechts von Android Studio angezeigt.



Nach der Einführung von SonarQube in unser Android-Projekt haben wir eine zentrale Anlaufstelle für unsere Code Qualität und kontrollieren regelmäßig diese in der SonarQube-Benutzer-Oberfläche.
Desweiteren bekommen wir über Jenkins ein schnelles Feedback ob unser Quality Gate eingehalten wurde.
Mit dem Community-Plugin laden wir uns die SonarQube-Issues vom Server herunter und beheben diese direkt in Android Studio. Bevor wir ein Feature pushen, kontrollieren wir mittels des Plugins im Vorfeld das Einhalten der Programmier Richtlinien und können neu aufgetretene Issues direkt in Android Studio beheben.

Hackathon 2019 in Berlin

- 

Unseren **3. Hackathon** haben wir im Juni 2019 zusammen mit unserem B2B-Partner Douglas ausgerichtet. An zwei Tagen pitchten wir mit 25 Teilnehmern Ideen, probierten neue Methoden und Techniken aus und teilten unser Wissen. Das alles gut versorgt, mit jeder Menge Spass und Motivation. Mit dem Thema **Personalisierung** haben wir den Hackathon dieses Mal inhaltlich geschärft. Das Publikum hat nach den **Abschluss-Präsentationen** zwei Projektideen prämiert und die Gewinner durften sich über tolle **Preise** freuen. Über die Themen und Eindrücke möchten wir euch über unseren Techblog teilhaben lassen.

Thalia B2B-Entwicklung in Berlin

In Berlin entwickeln wir seit 2014 für B2B-Partner digitale Produkte im Retail-Umfeld. In der kontinuierlichen Produktentwicklung sind uns agile Arbeitsweisen und stabile Teams sehr wichtig. Unser Schwerpunkt liegt in der Frontend-Entwicklung für mobile Geräte auf Basis von Android und iOS.

Teilnehmer und Partner beim gemeinsamen Abschlussfoto – Yeahh!

- Vanessa Stützle – Managing Director Ecommerce & Omnichannel Douglas Group begrüßte die Teilnehmer mit einer Videobotschaft

Personalisierung – Unser Schwerpunktthema

Mit unseren Teams in Berlin entwickeln wir seit mehreren Jahren erfolgreich Apps für unseren B2B-Partner Douglas. In Absprache mit Douglas haben wir den

inhaltlichen Schwerpunkt auf das Thema **Personalisierung** gelegt und uns überlegt, wie wir diesbezüglich Projektideen generieren können. Die Teilnehmer haben wir vorab mit einer **Lightning-Demo Session** eingestimmt, in der aktuelle Apps bzw. Websites vorgestellt wurden, die inspirierende Personalisierungsfunktionen anbieten.

Lightning Demos

Eine Methode mit der sich ein Team mit Produkten und Services für die Konzeptphase inspiriert. Wird in der Methodik „Design Sprint" angewendet.

Ablauf
– Jeder Teilnehmer recherchiert selbst 2-3 inspirierende Beispiele
– Jeder Teilnehmer präsentiert die Beispiele
– Die Kernideen werden auf Klebezetteln geschrieben und an ein Board angebracht

Am Ende hat das Team 10-20 Ideen zu dem Thema gesammelt. Es geht darum von allen Teilnehmern die besten persönlichen Ideen aufzunehmen und dabei nicht zu diskutieren.

https://www.sessionlab.com/methods/lightning-demos
https://www.thesprintbook.com

Die Projekte

Am Morgen des ersten 1. Tages stellten die Teilnehmer 6 Projektideen vor. Neben 5 technischen Vorschlägen aus dem Mobile-Umfeld hatten wir mit dem Design-Sprint-Projekt auch ein Methode aus dem UX-Umfeld am Start. So starteten wir nach dem Pitch mit 4 Teams in die Projekte.

- 

Der Pitch – Nach den Vorstellungen wählten die Teilnehmer ihr jeweiliges Projekt

- 

Team Mini Design Sprint – Solve Big Problem And Test New Ideas

Das Thema Personalisierung sind wir methodisch mit einem Mini Design-Sprint angegangen. Ein Design-Sprint ist **Konzeptionsmethode** für Produkte, die innerhalb von 5 Tagen mit dem Team durchgeführt wird. Da nur 2 Tage zur Verfügung standen, wurde das Format „kreativ“ komprimiert. Ziel der 2 Tage war das Kennenlernen der Methode und die Erstellung von interaktiven Prototypen für die Kundenapp unseres Partners. Bei diesem Non-Coding-Projekt stand also nicht die Programmierung, sondern die Methodik zur Generierung neuer Produktideen im Fokus.

Design Sprint

„The Design Sprint is a five-day process for solving problems and testing new ideas.“

Invented at Google by Jake Knapp, perfected with more than 150 startups at GV, then shared with the world in the bestselling book “Sprint – How To Solve Big Problems and Test New Ideas in Just Five Days”.

“On Monday, you make a map of the problem. On Tuesday, each individual

sketches solutions. On Wednesday, you decide which sketches are strongest. On Thursday, you build a realistic prototype. And on Friday, you test that prototype with five target customers."

https://www.thesprintbook.com/
https://www.youtube.com/watch?v=K2vSQPh6MCE

In den 2 ziemlich durchgeplanten Tagen hat das Team im ersten Schritt in **Experteninterviews** die **Challenges** zum Thema Personalisierung zusammengetragen und priorisiert. Die Interviews wurden mit 2 Personen aus den Bereichen Business, Markting/Sales und Tech durchgeführt.

> ***How we might …***
> *„Wie könnten wir persönlich Kunden ansprechen, ohne das es creepy wird"?*

Es wurden ambitionierte 2 **Jahresziele** und mögliche **Hindernisse** auf dem Weg identifiziert. Auf dieser Basis wurden **Lösungskontext** und **Fokusgebiet** geschärft und das Team hat eine **Lightning Demo** Session durchgeführt. Anschließend hat jeder Teilnehmer eine **eigene Produktidee** ausgearbeitet und als **Paper-Prototype** in **3 Schritten** visualisiert. Zwei Ideen wurden nach intensiven Diskussionen ausgewählt, um **Flows** auszuarbeiten und final **interaktive Prototypen** zu erstellen.

Mit den Ideen „Find your Style" und "Love me Tinder" ging das Team in die Abschlusspräsentation. Der Hackathon-Designsprint war damit abgeschlossen. Die Phase der Nutzer-Validierung möchte das UX-Team im Nachgang zusammen mit der Product-Ownerin angehen.

Die 8 Teilnehmer waren von der Methode und den Ergebnissen begeistert. Unser UX/UI-Lead Nicolai hat den Design-Sprint sehr gut vorbereitet und moderiert. Das Team bestehend aus Designern, Product Owner, Entwickler, QA, Scrum Master und Team-Manager hat viele Perspektiven in die Konzeptphase eingebracht und so für unseren Kunden einen wertvollen Input für die Produktentwicklung geben können.

WORKSHOP

MONDAY	TUESTDAY
Define the Challenge	Map the User Flow
Envision Long Term Goal	Design and build the Prototype
Produce a mass of solutions	Pitch Outcome
Curate and vote on best solutions	

"SOLVE BIG PROBLEMS AND TEST NEW IDEAS"

From the guys who brought you **Lightening Demos**

MINI DESIGN SPRINT

PERSONALIZATION

Thalia **AppTech**

HACKATHON

2019

Powered by **Douglas**

Paper Prototypes

Team KRAS – Konferenz Raum Anzeige System

Mit der Projektidee sollte ein ganz praktisches Problem in unserem Office gelöst werden. Steht man vor einem unserer Meetingräume und möchte wissen ob oder ab wann dieser belegt ist, muss man erst eine Buchungsanfrage am Rechner durchführen, um an die Information zu gelangen. Da gehen schon mal ein paar Minuten ins Land. Das geht besser!

Die Informationen sollen mit einem kleinem Display an jedem Raum angezeigt werden. Darauf soll die aktuelle Tagesbelegung dargestellt werden.

Das Team hat sich für **Angular** als Frontendtechnologie entschieden. Die UI ist responsiv und kann auch auf kleinen **eInk-Devices** dargestellt werden. Neben der Darstellung der Rauminformationen hat das Team auch damit begonnen, eine Schnellbuchungsfunktion für den Raum zu implementieren. Die Web-UI wurde mit einer **Android-App** auf einem modifizierten **Tolino eInk-Reader** und auf einem Android-Tablet dargestellt. Aufgrund der deutlich langsameren Geschwindigkeit der eInk-Displays müsste die UI im nächsten Schritt weiter optimiert werden. Z. B. ist eine Alternative für das Scrolling durch Listen nötig.

Den Code hat das Team auf Github bereitgestellt:

Conference room assisting system
https://github.com/jenszech/cras

ePaper interface for Conference Room Assistent System (CRAS)
https://github.com/jenszech/crasBadgeIt

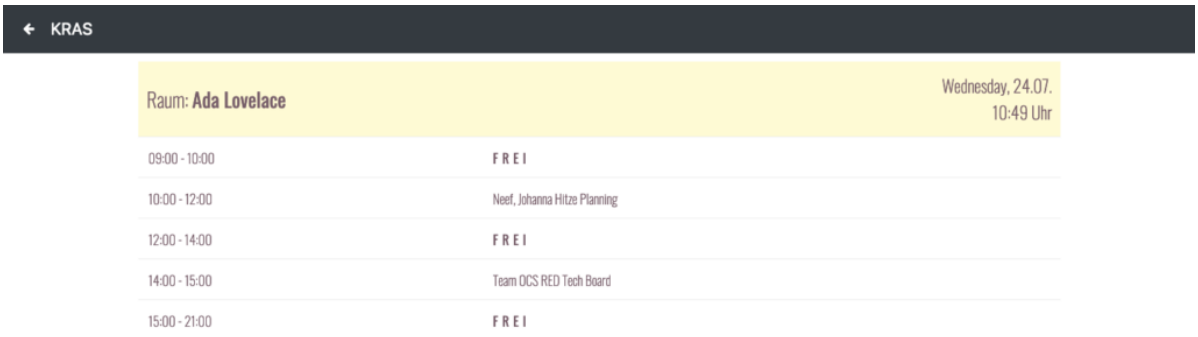
Neben der Web-UI hat das Projektteam auch eine **Arduino-basierte ePaper-Hardwarelösung** mit Mikrokontroller evaluiert. Die Lösung ist äußert energiesparsam.

Die REST-API wurden über einen **Node.js-Server** realisiert, der auf einem **Raspberry Pi** läuft. Eine Herausforderung lag in der Anbindung des **Thalia Exchange**-Servers, der die Informationen zu den Räumen bereitstellt. Zusätzlich wurden Metadaten zu den Räumen wie z. B. vorhandene Meeting-Technik und Größe hinterlegt.

In den 2 Tagen konnte sich das Team gut in das umfangreiche Thema einarbeiten. Ein weiteres Learning war, dass für die Koordination der Subteams entsprechend Zeit eingeplant werden muss. Alle im Team waren hoch motiviert für ihre

Bereiche Lösungen zu erarbeiten und diese mit den anderen zu integrieren. Die erarbeitete Lösung ist nahezu fertig, um sie mit einem ersten Meetingraum zu testen.

- javascript:history.back()

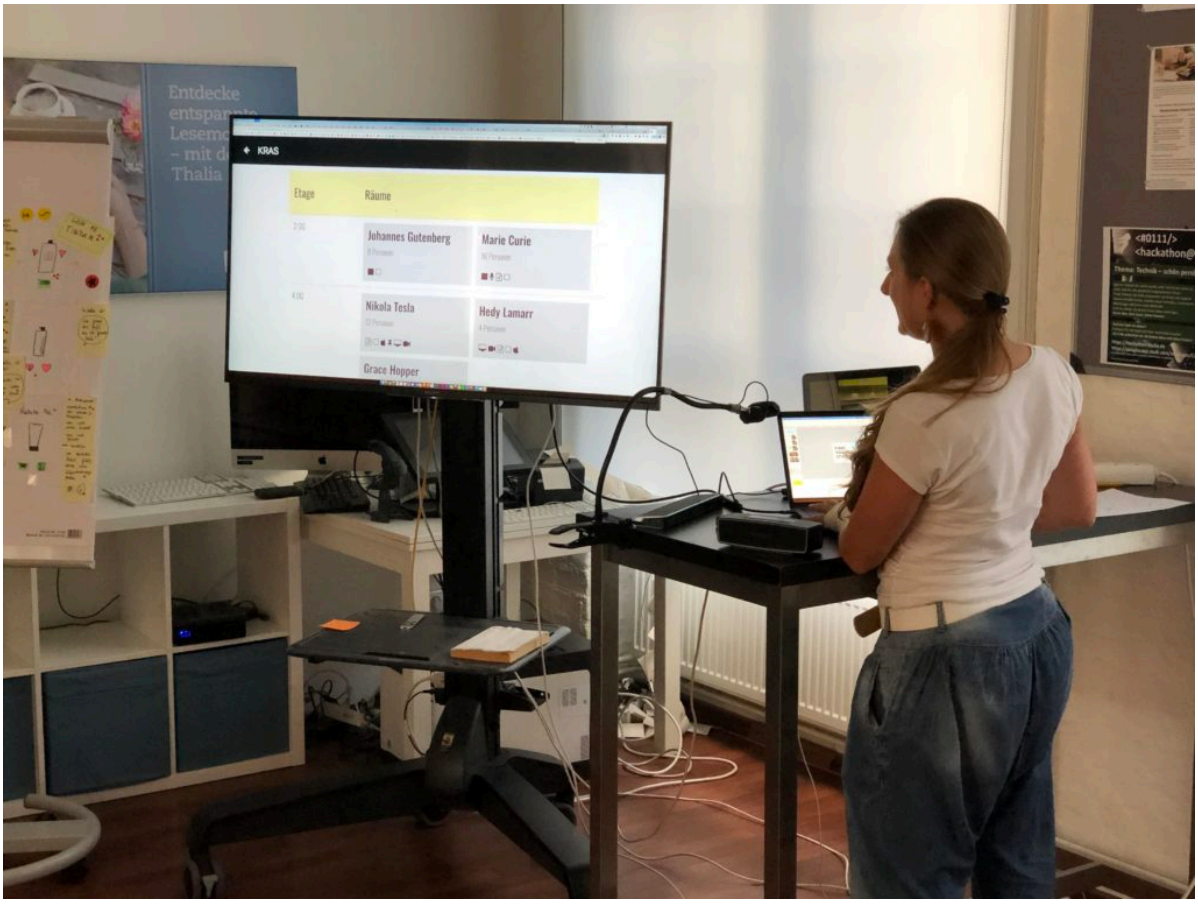


- KRAS

Etage	Räume		
2 OG	**Johannes Gutenberg** 8 Personen	**Marie Curie** 16 Personen	
4 OG	**Nikola Tesla** 12 Personen	**Hedy Lamarr** 4 Personen	**Grace Hopper** 6 Personen
3 OG	**Ada Lovelace** 10 Personen	**AlanTuring** 4 Personen	

- 

-

- 
- 

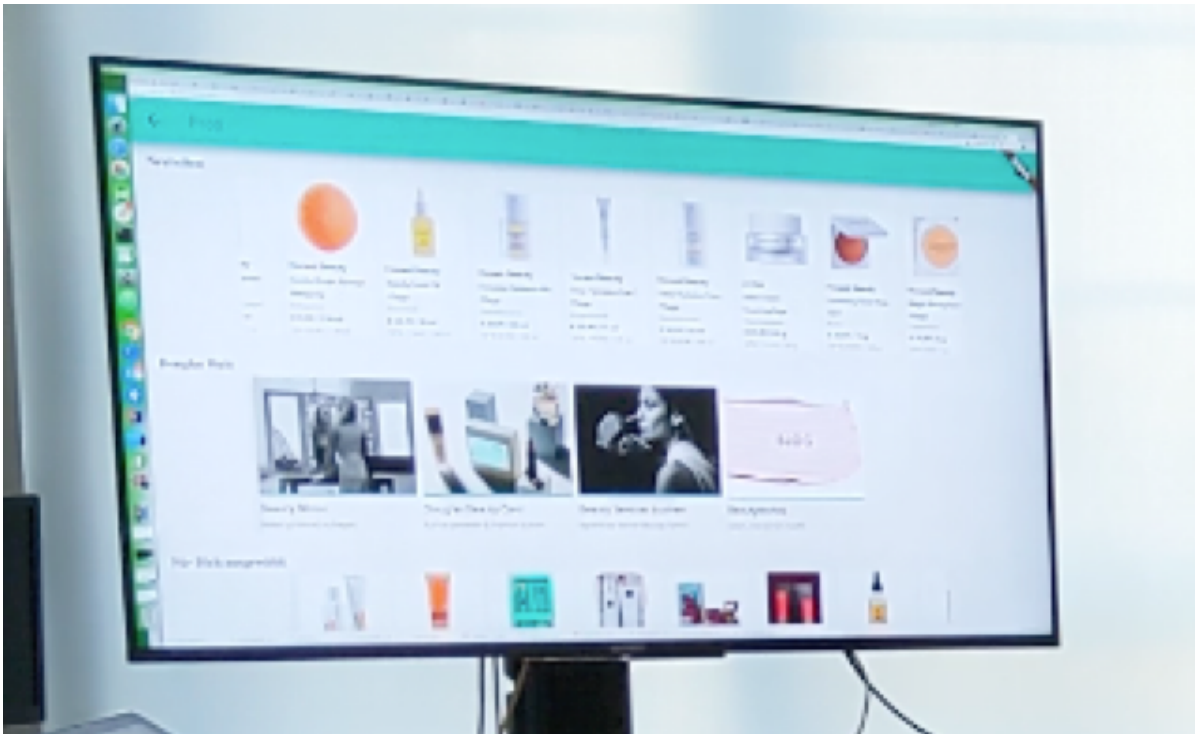
Team Flutter Touch – We want to Flutter you!

Mit dem Thema Cross-Plattform Entwicklung unter Anwendung von **Google's Flutter** hat sich auch dieses Jahr ein Team beschäftigt. Flutter ist mittlerweile in Version 1.7 veröffentlich. Neben iOS und Android existiert auch ein **Technology Preview für das Web**. Seit Ende 2018 ist Flutter production-ready und die Flutter Community wächst kontinuierlich. Das Team hatte 2 Teilnehmer die bereits Projekterfahrung mit Flutter hatten.

Unter dem Topic „**Feed-Configurator**" hat das Team für die Kundenapp unseres Partners eine Individualisierungsfunktion geschaffen, um die Produktbereiche wie „Für dich ausgewählt", „Kategorien" oder „Marken" in der Reihenfolge zu ändern. Mit dem „**Tinder-Swipe für Produkte**" Feature kann der Kunde im Tinder-Stil Produkte durch Wischen nach links oder rechts als interessant bzw. uninteressant markieren. Die Daten könnten später in eine Recommendation-Engine überführt werden, um bessere Produktvorschläge zu machen. Präsentiert wurden die Funktionen im **Web**, auf **iOS** und auf **Android**.

Das Tinder-Feature hat das Team sogar auf einem **Android-Wearable** (Uhr) im Emulator gezeigt. Großartig!


WE WANT TO FLUTTER YOU!
NEAREST RECRUITING STATION JUST AROUND THE CORNER!
THALIA HACKATHON 17./18. JUNI 2018 - MELDE DICH BEIM
TEAM FLUTTER TOUCH

- 
NOPE
YEAH
Dior
DOLCE & GABBANA
- 

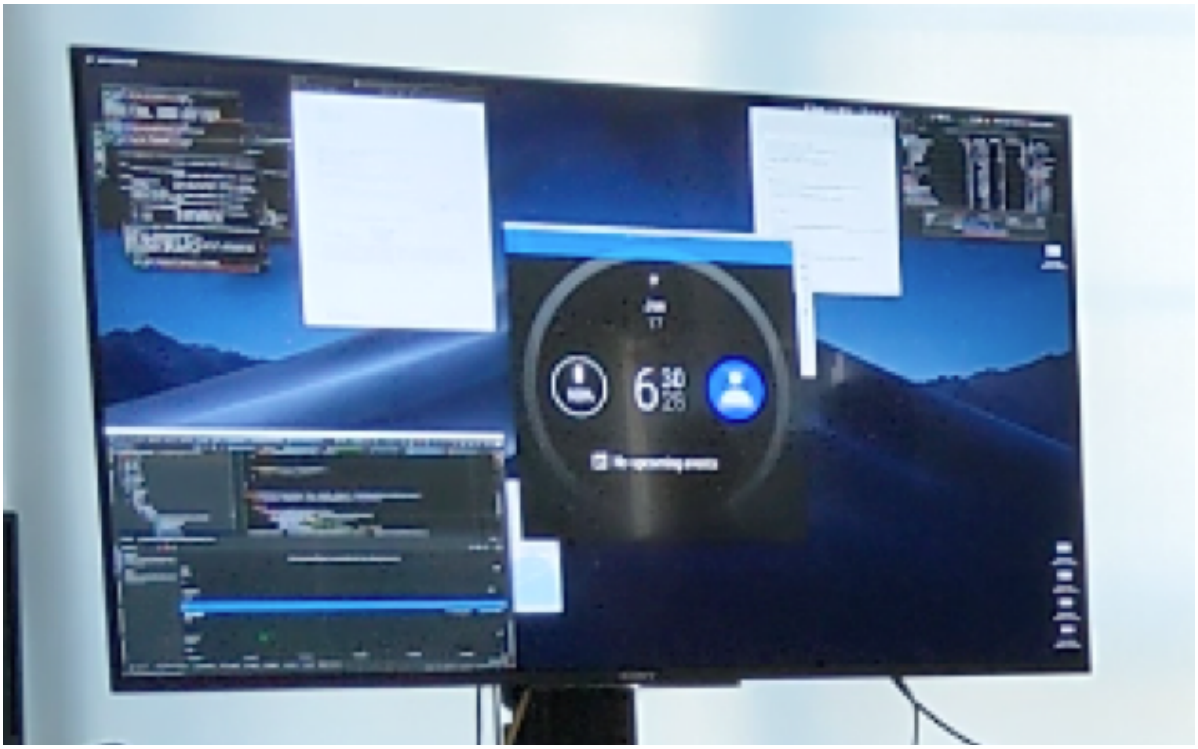
Wearables – Wie können wir Wearables im Douglas Kontext einsetzen?

Das Team hat sich einen Usecase im Kontext Personalisierung überlegt. In diesem werden Produktempfehlungen als personalisierte **Push-Notification** auf dem **Wearable** dargestellt und sollen über **Smart-Actions** auf den Merkzettel oder in

den Warenkorb gelegt werden können. In der Filiale könnte man sich auch vorstellen die Produkte direkt über einen Paymentdienst wie Google Pay zu bezahlen.

Entwickelt wurden die Prototypen auf **iOS, Android und Tizen**. Über ein **Google Firebase**-Setup sollten die Push Notifications auf die Geräte gesendet werden. Die Anwendungen liefen jeweils in den Emulatoren der Plattformen. Die Produktdetails wurden von den Partner-APIs abgerufen und auf den kleinen Displays dargestellt. Die Entwickler waren davon überrascht, wie schnell sie auf den Wearables entwickeln können.

- 

- 

Unser Fazit

Das Feedback der Teilnehmer und unseres Partners Douglas war äußerst positiv. Den Hackathon auf 2 Tage zu erweitern, gab den Teams mehr Zeit sich inhaltlich intensiver mit den Themen zu beschäftigten und ihre Präsentationen vorzubereiten. Die Bandbreite der Themen war technologisch und inhaltlich groß und brachte den Teilnehmern viele interessante Einblick in die Themen, mit denen wir uns vielleicht schon morgen beschäftigen werden.

Da auch Douglas-Mitarbeiter in Berlin dabei waren, konnte wir uns persönlich besser kennenlernen und austauschen. Die Projekte waren allesamt spannendend und wir haben unser Ziel erreicht mit dem Schwerpunkt Personalisierung einen direkten Mehrwert in Form von **Protoytpen**, **Methodik** und **Inspiration für Produktideen** zu schaffen.

Die Abschlussphase mit den Präsentationen und die Preisverleihung hat den Hackathon-Event gekonnt abgeschlossen. Wir freuen uns schon auf das nächstes Jahr!

Und noch ein paar weitere Inspirationen …

Team KRAS – 1. Preis

Team miniDesignSprint – 2. Preis

- 
- 
Hackathon 2019
Hackathon 2019
Hackathon 2019

Möchtest du mehr über uns erfahren? Dann empfehle ich dir auch die folgenden Beiträge und Links:

B2B-Entwicklung in Berlin
Hackathon 2018
Hackathon 2017

Continous Integration

Douglas KundenApp Android (Playstore)
Douglas KundenApp iOS (App Store)

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.

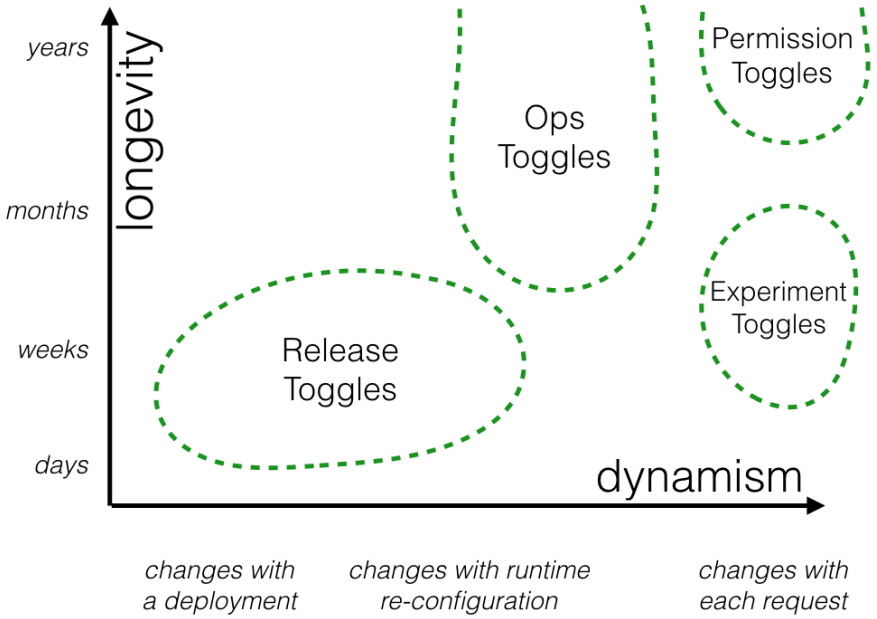
Feature Toogles

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.

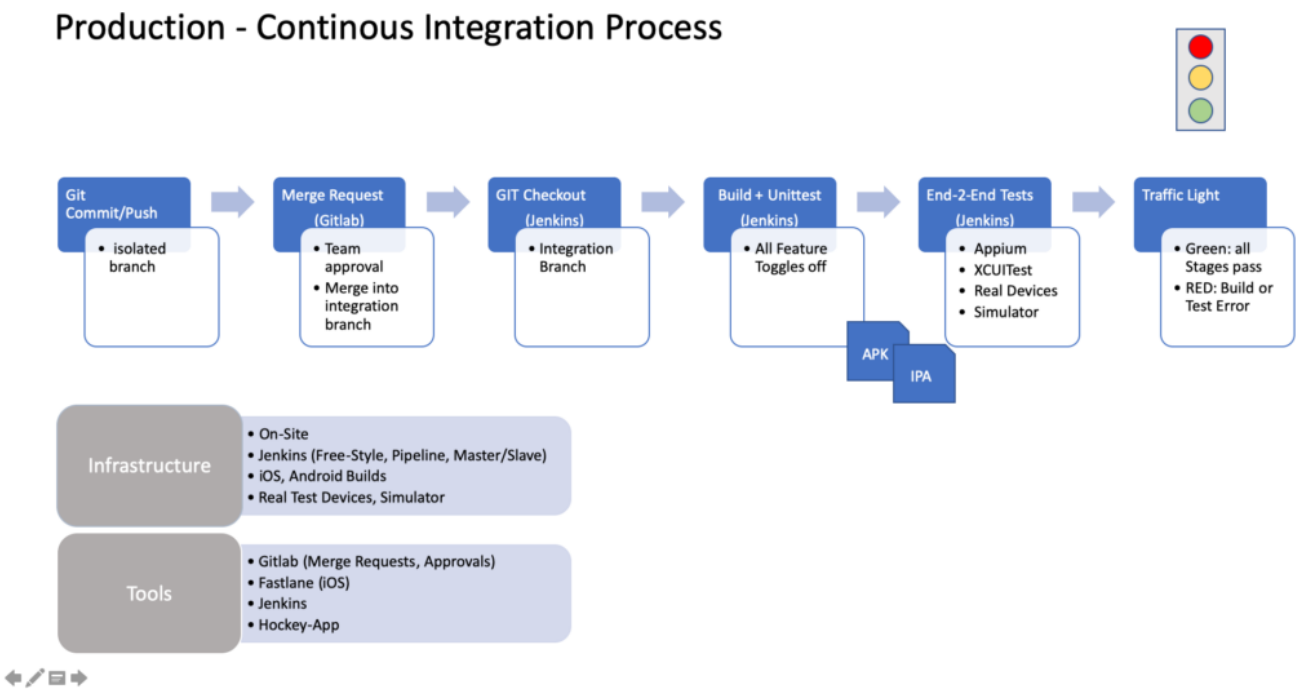
CI-Build-Status für Android und iOS

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.

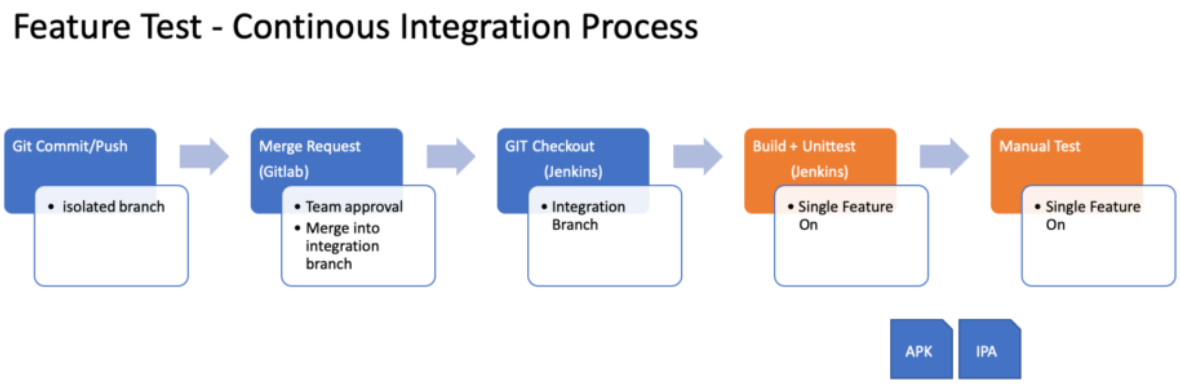
Continous Integration Process

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.



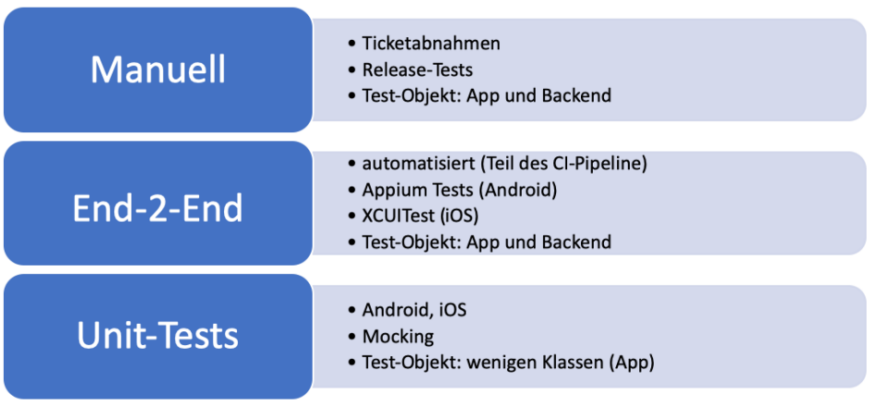
Für den Test eines Features, dass sich noch in der Entwicklung befindet, wird ein Feature-Toggle manuell im Jenkins aktiviert, die App gebaut und die Unittests ausgeführt. Die End-2-End Tests werden zum jetzigen Zeitpunkt noch nicht ausgeführt. Zum einen müssten die Tests für das Feature bereits angepasst 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

---

<#0100/> <hackathon@thalia/> in Berlin

Ende Mai fand der 2. Hackathon in Berlin statt. Zum Thema waren diesmal alle neuen Technologien gesetz.

<#0010/> hackathon@thalia.de in Berlin

<#0010/> <hackathon@thalia.de/> in Berlin zum Thema „Neue App Technologien“

Am 30. Juni 2017 fand der 2. Thalia Hackathon statt, diesmal mit dem Thema „Neue App-Technologien“ und am Standort Berlin. Mit dabei waren Kollegen aus den App- & Backend Teams, dem Digital Sales Management und dem Service Center Münster.

Wir starteten am Morgen gut gelaunt mit zwei **Lightning Talks** zum Thema „**Amazon Alexa**“ und „**Künstliche Intelligenz bei Thalia**“. Alexa ist seit Oktober 2016 in Deutschland verfügbar und hat bereits das Interesse einiger Mitarbeiter geweckt. Mit Alexa kann der Anwender digitale Dienste über eine Sprachsteuerung nutzen. Thalia beschäftigt sich im Service-Center schon seit längerer Zeit mit dem Thema „Künstliche Intelligenz“ (KI). Unter anderem im Kontext der Bearbeitung von Kunden-Emails. Wir gehen davon aus, dass die KI in Verbindung mit Sprachsteuerung und Chatbots in den kommenden Jahren für Thalia stark an Bedeutung zunehmen wird. So lag es nahe sich damit zu beschäftigen.

Nach den Vorträgen wurden 3 Projektvorschläge präsentiert, Teams gebildet und die Themen bearbeitet. Am späten Nachmittag trafen sich die Gruppen, um ihre Erkenntnisse zu teilen und die Arbeiten zu präsentieren.

Thema 1: The Voice of Thalia – Eine Buchsuche im Dialog (am Beispiel Alexa)

Wie wäre es, Alexa nach Buchempfehlungen zu fragen? Vielleicht nach Empfehlungen unserer Thalia-Mitarbeiter? „Alexa, empfehle mir ein Buch zu Harry Potter“. Gesagt, getan…

Eine Gruppe nahm sich der Aufgabe an, die Buch-Empfehlungen zu realisieren. Dazu wurden **Intents** definiert und bestehende Services aus der **Thalia-E-Commerce-Plattform** angesprochen.

Eine 2. Gruppe beschäftigte sich damit einen **Kaffee-Experten** zu entwickeln. Also Alexa dazu zu nutzen, ganz alltägliche Problemstellungen zu lösen. Dazu wurden Skills für die Domäne „Kaffee“ realisiert, indem passende **Intents** im Zusammenspiel mit **AWS-Lambda-Functions** erstellt wurden. Alexa wurde unter anderem darauf trainiert zu folgenden Fragen Antworten zu geben:

- Welchen Kaffee kannst du mir aus einer Rösterei in Leipzig empfehlen?
- Wieviel Wasser brauche ich für <XXX> Gramm Kaffee?

Thema 2: Buchempfehlungs-Agent für Messenger

Ein weiteres Team nahm die Herausforderung der **Chatbots** an. Auch hier ging es darum Buch-Empfehlungen auszuspielen. Dazu nutzen wir die **API.AI** Plattform, die im September 2016 von Google übernommen wurde und mit der es möglich ist „**Conversional User Interfaces**“ zu gestalten. Die Plattform bietet unter anderem eine **Slack-Chat-Integration** an, die vom Team zusammen mit **Heroku** (node.js) genutzt wurde, um die Business-Logik zur Abfrage der Buchempfehlung zu implementieren. Wir trainierten die KI darauf eine Antwort auf folgende Frage zu geben (<XXX> steht für einen beliebigen Buchtitel):

- Empfehle mir ein Buch zu <XXX>

Die Herausforderung lag darin, das Buch zu identifizieren und anschließend mit unserer Empfehlungs-Engine einen Vorschlag zu ermitteln.

Weiterführende Links:
https://api.ai/
https://www.heroku.com

Thema 3: Anwendungen und Dialoge

Eine dritte Gruppe beschäftigte sich mit dem Thema **Dialogerstellung** bei Sprachsteuerungen und Chatbots. Eine Erkenntnis war, dass es für ein rundes Produkt wichtig ist, erst die Dialoge zu erstellen und anschließend die Skills zu

entwickeln. Die Komplexität der Dialoge ist nicht zu unterschätzen. Unsere UX/UI- und Fachexperten waren begeistert, sich mit dieser neuen Art der Kommunikation zwischen Kunde und Services auseinanderzusetzen.

Fazit

Alle Gruppen konnten am Ende des Tages beindruckende Ergebnisse präsentieren. Das hatten die wenigsten zu Beginn erwartet. Die Ansteuerung von Alexa funktionierte ebenso, wie die Chatbot-Integration. Die Erkenntnisse aus der Dialog-Erstellung haben uns gezeigt, dass wir nicht nur technische Herausforderungen meistern müssen.

Vielen Dank an die Organisatoren des Events. Pizza, Snacks und Getränke lieferten die dringend benötigte Energie, um in der kurzen Zeit voranzukommen. Vielen Dank an Mirko, der uns mit Rat und Tat beim Thema Alexa zur Seite stand.