

Projektdokumentation

Entwicklung einer Mobile-App-Anwendung zum Annehmen von verschiedenen Aufgaben, die für die Auszubildenden im Unternehmen bereitgestellt werden

Auszubildender Fachinformatiker für Anwendungsentwicklung
Toni Kozarev
Projektzeitraum vom 11.10.2018 bis 08.11.2018



Auszubildender

Name: Toni Kozarev

Ausbildungsbetrieb

Dataport Anstalt öffentlichen Rechts

Niederlassung: Bremen

Anschrift: Utbremer Str. 90
28217 Bremen

Inhaltsverzeichnis

1 Einführung	1
1.1 Unternehmensprofil	1
1.2 Themenschwerpunkte der Ausbildung	1
1.3 Projektbeschreibung	1
2 Projektvorbereitung	3
2.1 Ist-Analyse	3
2.2 Soll-Konzept	3
2.3 Kosten-Nutzen-Analyse	3
2.4 Testplanung	4
3 Projektdurchführung	6
3.1 Auswahl der Software	6
3.1.1 Auswahl der Entwicklungsumgebung	6
3.1.2 Auswahl der Frameworks	6
3.1.3 Auswahl der Tests Frameworks	6
3.2 Entwicklung des UI Designs der Mobile-App	7
3.3 Implementierung der Klassen	7
3.3.1 Implementierung der Register	8
3.3.2 Implementierung der DatabaseHelper	9
3.3.3 Implementierung der Login	10
3.3.4 Implementierung der Session	10
3.3.5 Implementierung der AdminPlattform	11
3.3.6 Implementierung der Menu	11
3.3.7 Implementierung der List	12
3.3.8 Implementierung der Tasks	12
3.4 Entwurf der UML-Klassendiagramme	13
3.5 Entwickeln der automatisierten Tests	13
3.6 Entwickeln des JUnit Tests	14
3.7 Deployen der Mobile-App	14
4 Projektabschluss	15
4.1 Testdurchführung	15
4.2 Soll-Ist-Vergleich	15
4.3 Fazit	15
Anhang	16
A Glossar	16
B Literaturverzeichnis	B-1
C Code-Anhänge	C-1
D Abbildungen	D-1

Tabellenverzeichnis

1 Tesplanung der Integrationstests	5
2 Soll-Ist-Vergleich	15

Liste von Code-Anhängen

1 Android-Manifest.xml	C-1
2 Shake.xml	C-1
3 Strings.xml	C-2
4 Colors.xml	C-3
5 Styles.xml	C-3
6 Rand der Informationen	C-4
7 Rand der Listelementen	C-4
8 Rand der Anmelden EditTexts	C-5
9 Button für das Anmelden	C-5
10 Hintergrund-Bildschirm der Spinner	C-5
11 activity-login.xml	C-6
12 activity-register.xml	C-9
13 activity-menu.xml	C-13
14 activity-admin-plattform.xml	C-16
15 activity-list.xml	C-19
16 activity-task.xml	C-20
17 spinner-items.xml	C-23
18 LoginTests.java	C-23
19 RegisterTests.java	C-26
20 SharedPreferencesTests.java	C-29
21 AdminPlattformTests.java	C-30
22 MenuTests.java	C-33
23 ValidEmailTest.java	C-35
24 User.java	C-36
25 Session.java	C-36
26 DatabaseHelper.java	C-37
27 Login.java	C-41
28 Register.java	C-43
29 AdminPlattform.java	C-45
30 Menu.java	C-46
31 List.java	C-48
32 Task.java	C-50
33 GradleBuild	C-52

Abbildungsverzeichnis

1 Screenshots: Splashscreen, Anmeldung, Registrierung	8
2 Screenshots: Splashscreen, Anmelden, Registrieren	D-1
3 Screenshots: Admin, Admin-Plattform	D-1
4 Screenshots: Benutzer, Listenergebnisse, Leere Liste	D-2
5 Screenshots: Offen, In Bearbeitung, Erledigt	D-2

6	Screenshots: Menu, Mockup Beispiel, Mockup Realisierung	D-3
7	Rund Icon	D-3
8	ERD, Tabelle users, Tabelle tasks	D-4
9	Alle automatisierte Tests	D-5
10	Anwendungsfalldiagramm	D-6
11	Alle Klassen, Funktionen und Variablen	D-7

1 Einführung

Diese Projektdokumentation beschreibt die Planung, Implementierung und das Testen einer **Mobile-App** zum Annehmen von verschiedenen Aufgaben, die für die Auszubildenden im Unternehmen bereitgestellt werden. Die Projektdokumentation berücksichtigt, dabei neben der Auswahl passender **Frameworks**, wichtige Aspekte, wie eine Kosten-Nutzen-Analyse und den Datenschutz.

1.1 Unternehmensprofil

Das Unternehmen Dataport stellt für die öffentliche Verwaltung IT bereit und bietet den Behörden ein umfassendes Angebot von Dienstleistungen wie IT-Beschaffung, Fortbildungen und Schulungen [1]. Neben den oben aufgelisteten Dienstleistungen werden Datensicherheitskonzepte, sowie E-Government-Lösungen und Anwendungen für Verwaltungsaufgaben angeboten. Dataport ist eine Anstalt des öffentlichen Rechts.

Der Betrieb wurde am 1. Januar 2004 durch den Zusammenschluss der Datenzentrale Schleswig-Holstein mit dem Landesamt für Informationstechnik und der Abteilung für Informations- und Kommunikationstechnik des Senatsamtes für Bezirksangelegenheiten gegründet und hat seinen Sitz in Altenholz. Neben den Sitz in Altenholz hat unser Betrieb Niederlassungen in Hamburg, Bremen, Rostock, Lüneburg, Halle und Magdeburg.

Die Aufgabe unseres Unternehmens ist die Versorgung von Kommunikations- und Informationstechnik für die Trägerländer Hamburg, Bremen, Niedersachsen, Mecklenburg-Vorpommern, Schleswig-Holstein und Sachsen-Anhalt als **Full Service Provider**. Dataport hat momentan über 2.700 Mitarbeiterinnen und Mitarbeiter und erzielte 2017 einen Jahresumsatz von 547 Millionen Euro [2].

1.2 Themenschwerpunkte der Ausbildung

Die Schwerpunkte während der Ausbildung zum Fachinformatiker für Anwendungsentwicklung bei Dataport lagen in dem Testen von Software mit C# und in der Entwicklung von Programmen mit **Java**. Daneben waren weitere Ausbildungsinhalte wie **Virtualisierung**, Netzwerktechnik bzw. Netzwerkinfrastrukturen, **Linux**, Auftragssteuerung und Accounting, die am Standort in Bremen vermittelt wurden.

1.3 Projektbeschreibung

Es soll ein Prototyp einer Mobile-App für die Abstimmung einer betrieblichen Aufgabe von den Auszubildenden aller Standorte unserer Firma entwickelt werden. Diese soll auf dem **Android**-Betriebssystem [3] lauffähig sein.

Alle Benutzerdaten und wichtige Informationen für die Aufgaben sollen lokal gespeichert werden. Zu diesem Zweck sollte eine relationale SQL-Datenbank erstellt werden. Diese wurde auch für die Authentifizierung, Datenänderungen und Tests benutzt.

Im Betrieb werden die Auszubildenden in verschiedenen Teams verteilt und können sich ständig bei unterschiedlichen Tätigkeiten engagieren. Jedes Team ist für einen bestimmten Bereich verantwortlich, z.B. Technik-, Messe-, Software-, und Marketing-Team. Jede/r Auszubildende/r soll mindestens einem Team zugeordnet werden und regelmäßig sein/ihr Teil für dieses Team beitragen.

Die Idee dieser Anwendung basiert auf der **Kanban**-Methode. Die Aufgaben sollen in drei verschiedenen Zuständen organisiert werden - *Offen*, *In Bearbeitung*, *Erledigt*. Siehe Abbildung unten.

Es werden oft neue Aufgaben in der Datenbank von dem Administrator hinzugefügt, welche angenommen werden können. Manche Aufgaben brauchen mehrere Auszubildende, deswegen sollen sie mehrmals von dem Verteiler hinzugefügt werden, damit die notwendige Anzahl von Teilnehmern erreicht werden kann.

Alle Auszubildenden, die diese Android-App benutzen sollen, müssen zuerst registriert und erfolgreich angemeldet werden, sodass sie das Hauptmenü erreichen können. Dort sollen sie die Suchkriterien eingeben. Es soll eine Liste mit dem entsprechenden Team und Zustand erzeugt werden. Die ältesten Aufgaben werden am Anfang der Suchliste angezeigt, weil sie am dringendsten sind.

Der Benutzer kann jede Aufgabe von den gefundenen Ergebnissen öffnen und ihre Details durchlesen. Er hat die Möglichkeit den Titel, die Beschreibung, das Datum der Veröffentlichung, das Team und den Status anzuschauen, bevor er die Aufgabe annimmt. Die offenen Aufgaben sollen von allen registrierten Benutzern sichtbar sein. Nur die Person, die die Aufgabe zuerst akzeptiert hat, kann diese auch später als erledigt markieren, weil nur sie die Verantwortung für diese Tätigkeit trägt. Wenn eine Aufgabe angenommen wurde, soll sie ihr Status von „Offen“ auf „In Bearbeitung“ ändern. Solange eine Aufgabe fertig ist, kann ihren Status auf „Erledigt“ eingestellt werden und der Benutzer wird diese Aufgabe nachher unter „Erledigte“ Aufgaben finden können. Ihr Zustand sollte nicht mehr änderbar sein. Sie wird nur für Informationszwecke angezeigt werden.

Eine Sitzung wird nach der erfolgreichen Anmeldung eröffnet. Der Nutzer kann sich jederzeit sorglos die Mobile-App anschließen. Sobald der Benutzer angemeldet ist, wird seine Sitzung aktiv bleiben und er konnte die Mobile-App unbeschränkt von seinem Profil nutzen. Eine neue Anmeldung ist nur dann notwendig, wenn ein Nutzer sich von dem Hauptmenü explizit abmeldet und seine Sitzung beendet.

Aus dem Projekt soll hervorgehen, ob eine Fortführung des Projektes zu einem späteren Zeitpunkt sinnvoll und wirtschaftlich ist.



2 Projektvorbereitung

In diesem Kapitel wird der aktuelle Zustand des Projekts beschrieben, sowie der Soll-Zustand, der nach dem Projekt erreicht werden soll.

2.1 Ist-Analyse

Momentan verfügt Dataport über keine technische Umsetzung, wie bspw. eine Mobile-App für die Verteilung von Aufgaben durch die Auszubildenden. Es ist lediglich möglich über MS-Outlook am PC Nachrichten mit allen Tätigkeiten, die gemacht werden sollen, an eine bestimmte Gruppe von Personen zu versenden. Der Ersteller wird von allen Auszubildenden Zusage/Absage-Emails für jedes Angebot bekommen und sein Posteingang wird überflutet. Wenn es Unklarheiten über die Aufgaben gibt, dann soll der Verteiler die Fragen der Auszubildenden beantworten und diese Diskussionsrunde wird durch Emails durchgeführt, welche viel Zeit verbrauchen werden.

2.2 Soll-Konzept

Zweck dieses Projekts soll eine lauffähige Mobile-App sein, die auf dem Android-Betriebssystem verwendet werden kann. Dazu soll das Android SDK verwendet werden. Dieses Projekt soll darauf abzielen, eine Mobile-App, welche auf die Idee von Kanban-Brett basiert, darzustellen. Die Informationen für die Aufgaben und alle Benutzerdaten sollen in einem relationalen Datenbankmodell gespeichert werden. Diese Datenspeicherung soll authentisch, korrekt und sicher ablaufen.

Damit neue Benutzer sich anmelden können, wurde die Registrierung zuerst implementiert, welche nicht Teil des Projektes war, stellt sich aber sehr entscheidbar für die Weiterentwicklung der Mobile-App heraus, da die Passwörter der registrierten Benutzern geschützt in der Datenbank gespeichert werden sollen. Danach konnte die Anmeldung programmiert werden, wo das gespeicherte und das eingegebene Passwort, welches zuerst verschlüsselt werden soll, verglichen werden muss. Dazu sollen bestimmte Bibliotheken ausgewählt werden, um beiden Funktionen eine sichere Datenverschlüsselung bereitzustellen. Nachdem ein Benutzer erfolgreich angemeldet ist, kann er eine Liste mit allen verfügbaren Aufgabentiteln durch die Suchkriterien erstellen. Danach sollte es möglich sein, dass der Nutzer durch die angezeigten Titel eine Aufgabe aussuchen und eröffnen kann. Dementsprechend sollen alle Informationen für diese Tätigkeit angezeigt werden. Damit der Administrator neue Aufgaben in der Datenbank hinzufügen kann, wurde eine Admin-Plattform implementiert. Diese konnte nur mit der Email-Adresse „admin@dataport.de“ erreicht werden. Das war auch kein Teil des Projektes, aber wird sehr hilfreich für den Verteiler der Aufgaben sein, damit er nicht stets die Datenbank korrigieren muss. Diese Funktionalität soll den Prozess erleichtern und beschleunigen. Es sollen neben der Durchführung der eigentlichen Mobile-App auch automatisierte Tests entwickelt und implementiert werden. Darüber hinaus soll es möglich sein, dass jeder angemeldete Benutzer seine Sitzung bei Schließung der Applikation nicht verliert und angemeldet bleibt.

2.3 Kosten-Nutzen-Analyse

Für dieses Projekt werden insgesamt 70 Arbeitsstunden geplant. In diesem Zeitraum muss das ganze Projekt entworfen, implementiert und getestet werden. Am Ende soll es ausführlich dokumentiert werden. Wenn die App fertiggestellt ist, muss sie nur als .apk Datei verteilt und von dem Benutzer auf dem privaten Smartphone installiert werden. Dieser Ablauf sollte kaum mehr als 5 Minuten dauern.

Es werden durch die Entwicklung der Mobile-App weder Anschaffungskosten neben dem Arbeitslohn des Anwendungsentwicklers noch laufende Kosten anfallen. Die Rechnung entspricht

$70 \text{ Arbeitsstunden} \cdot 70 \frac{\text{Euro}}{\text{Arbeitsstunden}} = 4900 \text{ Euro}$. Die Entwicklungskosten für dieses Projekt belaufen sich auf 4900 Euro.

Die Zeit, die durch die Nutzung der App gespart wird, ist nur ungenau zu berechnen, da jeder Nutzer unterschiedlich lang braucht, um die Details einer Aufgabe durchzulesen.

Zum Ersten soll der Verteiler der Aufgaben nicht mehr sorgen, wer von den Auszubildenden Zeit und Lust hat, eine Tätigkeit anzunehmen und die Verantwortung für sie zu tragen. Diese wurde komplett von der ganzen Mobile-App übernommen. Außerdem kann der Administrator jederzeit einen Blick in der lokalen Datenbank für Details werfen. Ein anderer Vorteil ist, dass es meistens mehrere Freiwillige gibt, die eine Aufgabe annehmen möchten. Solche Situationen, mit beschränkten Plätzen einer Aufgabe, führen zu Überschneidungen, wenn der Verteiler mit den Auszubildenden nur durch Emails kommunizieren kann.

Erfahrungswerte zeigen, dass es monatlich ungefähr 25-30 neue Aufgaben von verschiedenen Teams gibt, die gemacht werden müssen. Jede Aufgabe wurde vor der Implementierung der Mobile-App ca. 10 Minuten von dem Verteiler und dem/der Auszubildende/r diskutiert. Das sind 30 Auszubildenden, die 10 Minuten jeden Monat und 1 Verteiler, der 300 Minuten in solcher Diskussionen verbringen. Insgesamt sind das 600 Minuten, was auch ca. 10 Stunden monatlich sind oder ungefähr 120 Stunden pro Jahr. Mit der Applikation konnte der Benutzer in ca. 3-5 Minuten die Beschreibung einer Aufgabe lesen und entscheiden, ob er die Aufgabe annehmen möchte oder nicht. Das Hinzufügen von Aufgaben monatlich wird auch nicht mehr als 1 Stunde dauern.

Die **pessimistische Berechnung** soll entsprechen:

$$30 \text{ Auszubildenden} \cdot 5 \text{ Minuten} = 150 \text{ Minuten}$$

$$150 \text{ Minuten} + 60 \text{ Minuten} = 210 \text{ Minuten} \text{ (addiert die verbrachte Zeit des Verteilers)}$$

$$210 \text{ Minuten} \cdot 12 \text{ Monate} = 2520 \text{ Minuten pro Jahr}$$

$$\text{Insgesamt } \frac{2520 \text{ Minuten}}{60 \text{ Minuten}} = 42 \text{ Stunden/jährlich}$$

Die **optimistische Berechnung** soll so aussehen:

$$30 \text{ Auszubildenden} \cdot 3 \text{ Minuten} = 90 \text{ Minuten}$$

$$90 \text{ Minuten} + 60 \text{ Minuten} = 150 \text{ Minuten} \text{ (addiert die verbrachte Zeit des Verteilers)}$$

$$150 \text{ Minuten} \cdot 12 \text{ Monate} = 1800 \text{ Minuten pro Jahr}$$

$$\text{Insgesamt } \frac{1800 \text{ Minuten}}{60 \text{ Minuten}} = 30 \text{ Stunden/jährlich}$$

Also mit dieser Applikation sollten alle Nutzer jährlich zwischen $120 - 42 = 78$ Stunden und $120 - 30 = 90$ Stunden ersparen.

Eine objektive Einschätzung der Ersparnis wäre eine Verbesserung der Arbeitszeit um ca. 65-75%. Bei geschätzten Lohnkosten von 70 Euro pro Stunde belaufen sich die Entwicklungskosten so auf 4900 Euro und die jährlichen Einsparungen auf 5460 Euro bis 6300 Euro. Mit der Amortisierung der App ist so bereits nach weniger als ein Jahr nach Einführung zu rechnen.

2.4 Testplanung

Die Testfälle für diese Android-App sind so erstellt, dass es für einige Java Klassen eine Testsuite gibt. Alle Tests sind automatisiert und werden mithilfe des Frameworks Espresso gemacht. Es gibt Tests, die zum Beispiel die Aktionen: ein Benutzer registrieren oder anmelden, eine Aufgabe mit bestimmten Suchkriterien suchen oder Aufgaben erstellen (nur als Administrator möglich), testen. Darüber hinaus soll überprüft werden, ob eine Aufgabe von dem angemeldeten Benutzer korrekt

angenommen werden kann. Dabei werden ebenfalls auch triviale Tests durchgeführt, die Tests sollen überprüfen, ob jede Aktion, die an einen Button gebunden ist, erfolgreich ausgeführt wird. Es werden noch verschiedene UI Elemente getestet, u.a.:

- Layout
- Button
- Edit Text
- List View

Weitere Informationen bzgl. der Tests, siehe [Unterabschnitt 3.5](#).

#	Testszenarien	einzelne Testfälle
1	Start der App	Sind alle UI Elemente korrekt geladen und angezeigt?
2.1	Anmelden als Administrator	Kann der Administrator anmelden?
2.2	Aufgaben hinzufügen	Kann der Administrator neue Aufgaben in der Datenbank hinzufügen?
3	Anmelden als Benutzer	Ist eine Anmeldung erfolgreich oder nicht? Ist die Email/Passwort gültig?
4	Registrieren	Ist die Registrierung erfolgreich oder wurde die Email schonmal benutzt?
5	Menü	Funktionieren den Abmeldungsbutton und das Menü fehlerfrei?
6	Listenergebnisse	Zeigt die Liste die richtige Ergebnisse an?
7	Aufgabe anzeigen	Kann ein/e Auszubildende/r eine Aufgabe annehmen/erledigen?
8	Hinzugefügte Aufgaben	Können die neuen Aufgaben auf die Listenergebnisse gefunden werden?

Tabelle 1: Tesplanung der Integrationstests

3 Projektdurchführung

Im folgenden Kapitel werden alle Schritte bei der Implementierung der Android-App beschrieben und welche unerwarteten Hindernisse bzw. Probleme bei der Realisierung der geforderten Anforderungen auftraten.

3.1 Auswahl der Software

Die Auswahl der richtigen Software und Frameworks ist ein wichtiger Teil des Projektes. Die Entwicklungsumgebung **Android Studio** [4] ist eine sehr beliebte Umgebung für die Erzeugung von Android Applikationen und wird meist empfohlen. Außerdem musste eine lokale Datenbank mit der Software **Sqliteman** [5] erstellt werden. Die Dokumentation [6] in **Sqlite** wurde sehr verständlich und ausführlich geschildert, sowie öffentlich zugänglich und kommt mit vielen Beispielen.

3.1.1 Auswahl der Entwicklungsumgebung

Für die Implementierung der Mobile-App wurde sich für die Entwicklungsumgebung Android Studio entschieden. Diese Umgebung wird empfohlen für den Einsatz der **Android SDK** und bringt bereits **Gradle** [7] mit, das für die Anwendung der Mobile-App benötigt wird.

3.1.2 Auswahl der Frameworks

Am Anfang des Projektes wurde sich für bestimmte Frameworks und Bibliotheken entschieden, die für die Durchführung des Projektes fundamental sind.

Dazu zählt die Bibliothek **BCrypt**, die für die Verschlüsselung des Benutzer-Passworts benutzt werden sollte, bevor dieses in der lokalen Datenbank gespeichert wird.

Damit die ausgewählten Frameworks/Bibliotheken funktionieren, müssen sie in der **Gradle**-Konfiguration eingerichtet werden. Nach der Implementierung einer **Dependency** haben alle Klassen und Methoden Zugriff auf die enthaltenen Funktionalitäten in den Frameworks.

```
1 implementation 'dependency'
```

Hierbei muss **dependency** durch den Namen ersetzt werden, dessen Dependency hinzugefügt werden soll. Diese ist üblicherweise beim Anbieter des Frameworks oder der Bibliotheken zu finden. In diesem Fall soll die **build.gradle** Konfigurationsdatei mit der unten stehenden Zeile erweitert werden.

```
1 implementation 'org.mindrot:jbcrypt:0.4'
```

3.1.3 Auswahl der Tests Frameworks

Außerdem wird für das Testen ein sehr populäres Framework benutzt: **Espresso** [8]. Dieses Framework wurde für die Erstellung der automatisierten Tests ins Spiel gebracht. Diese UI-Tests liefern auf einem Smartphone oder Emulator und überprüfen, ob bei einer Änderung am User Interface das gleiche Ergebnis vorkommt. Diese Tests werden auch Instrumented Tests genannt. Mit den nächsten Zeilen können sie in der **Gradle**-Konfiguration eingerichtet werden:

```
1 androidTestImplementation 'com.android.support.test:runner:1.0.2'  
2 androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'  
3 androidTestImplementation 'com.android.support.test:rules:1.0.2'
```

Für die erstellte JUnit Tests musste noch eine Zeile hinzugefügt werden:

```
1 testImplementation 'junit:junit:4.12'
```

Die verwendeten Frameworks sind [Open Source Software](#) und es fallen keine Kosten hierfür an.

3.2 Entwicklung des UI Designs der Mobile-App

Als Erstes wird ein sogenanntes [Mockup](#) der App erstellt. Ein Mock-Up bedeutet einen Wegwerfprototyp der Benutzerschnittstelle einer zu erstellenden Software. Die Abbildung [6](#) stellt ein Beispiel für ein [Mockup](#) dar, das für das Projekt angefertigt wurde. Dieser Prototyp wurde in [Adobe Spark](#) [9](#) erzeugt. Auf dem Screenshot kann man sehen, wie die Positionierung der einzelnen Elemente geplant war und berücksichtigt wird, dass die Elemente sich nicht überschneiden. Die Schriftgröße der Texte und die Größe der Menü Buttons müssen auch entsprechend gewählt werden, um das Lesen und das Navigieren in der Mobile-App zu erleichtern. Die Implementierung dieses [Mockups](#) wird in der Abbildung [6](#) repräsentiert.

Das Hintergrundbild wurde mithilfe [Background Image Generator](#) [10](#) erstellt. Dies wurde dann mit dem [Batch Import Plugin](#) im Android Studio zu dem Projekt hinzugefügt, damit es für alle Größe des Bildschirms eines Smartphones das entsprechende Bild angepasst wurde.

Das runde Icon [7](#) wurde mithilfe [Launcher Icon Generator](#) [11](#) gemacht. Außerdem wurden verschiedene Layouts mit allen notwendigen UI Elementen gemacht. Alle Elemente werden hauptsächlich durch die XML-Dateien generiert, aber sie können auch in dem Java-Code geändert werden.

Um die App universell und speziell für unser Unternehmen zu machen und keine populären Layouts zu benutzen, wurde das ganze Projekt in einem Vollbildansicht-Thema gemacht. Dazu wurde ein [Splashscreen](#) mit einer Animation hinzugefügt. Die Navigation durch die verschiedenen Aktivitäten wurde leicht und intuitiv gemacht, um die App nutzerfreundlicher zu machen. Es werden meistens Farben benutzt, die im Unternehmens Logo, auf der Webseite und in den Dokumenten von Dataport verwendet werden.

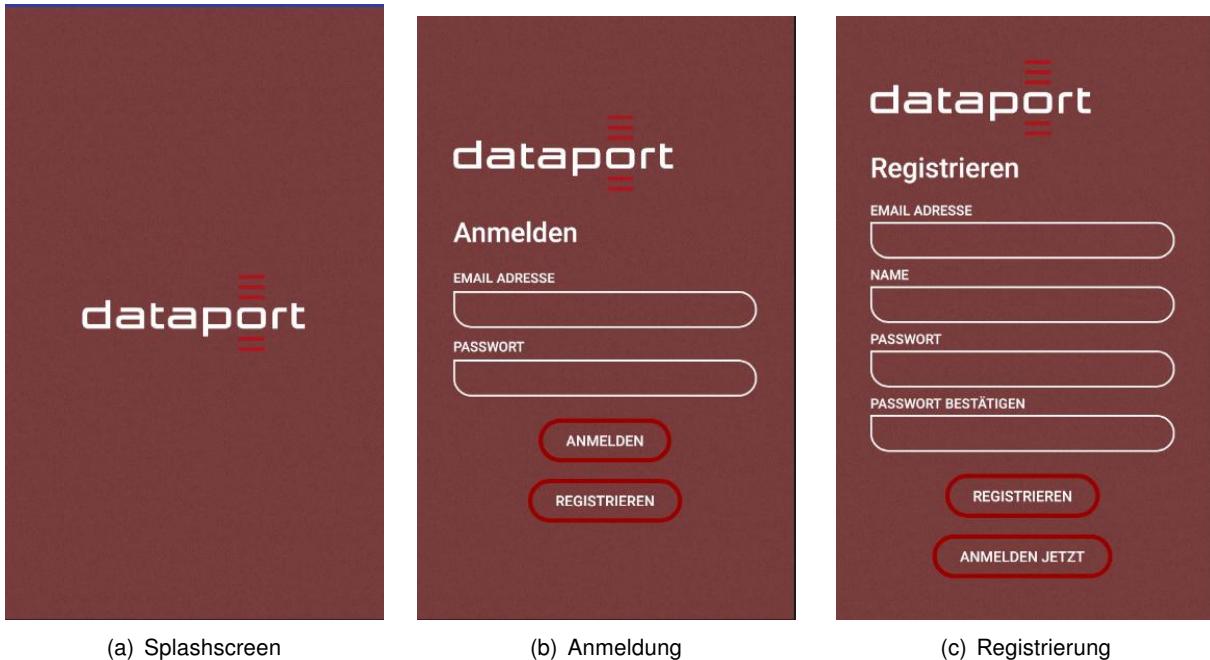
3.3 Implementierung der Klassen

In diesem Kapitel wird die ganze Implementierung des Projektes beschrieben. Beim Starten der App wird ein [Splashscreen](#) aktiviert. Nach dem Laden der Applikation sollte eine Anmeldungsmaske erzeugt werden. Um ein/e Auszubildende/r anzumelden, sollte er/sie vorher registriert werden sein. Dazu wird es einen Button geben, der den Benutzer zu einer anderen [Aktivität](#) weiterleitet, wo die Registrierung möglich ist. Wenn ein Konto erstellt wird, sollte die Person zunächst versuchen, sich mit ihren Benutzerdaten anzumelden. Wenn die Anmeldung erfolgreich war, wird eine Sitzung gestartet und das Menü angezeigt.

Dort kann jeder Benutzer eine Suche mit den Spinner durchführen. Als Ergebnis wird eine neue Seite mit den Listenergebnisse geöffnet. In dieser Liste werden alle gefundene Aufgaben mit den ausgewählten Suchkriterien aufgezählt. Beim Klicken auf einen Aufgabentitel werden die Details für sie angezeigt. Danach kann diese Tätigkeit angenommen werden. Am Ende sollte einen Abmeldungsbutton in dem Hauptmenü implementiert werden, der die Sitzung des Benutzers beendet.

Screenshots: [3](#)

Abbildung 1: Screenshots: Splashscreen, Anmeldung, Registrierung



(a) Splashscreen

(b) Anmeldung

(c) Registrierung

3.3.1 Implementierung der Register

Beim Starten der Registrierungsmaske wird ein **Splashscreen** aktiviert. Die Registrierung von neuen Benutzern soll in dieser Klasse durchgeführt werden. Damit die Registrierung erfolgreich ist, sollen einige Bedingungen erfüllt werden:

- 1) Die Email-Adresse sollte nicht registriert werden. (Die *checkIfExists*-Methode von der *DatabaseHelper.java*, welche eine SQL-Anfrage enthält, überprüft diese Bedingung.)
- 2) Die Email sollte gültig sein. (siehe unten gezeigte *isEmailValid*-Methode)
- 3) Die Passwörter sollen übereinstimmen.
- 4) Alle Felder müssen ausgefüllt sein.
- 5) Das Passwort soll mindestens 6 Zeichen enthalten.

Wenn alles erfüllt ist, wird das Passwort mit der **BCrypt**-Verschlüsselung [12] geschützt und in der Datenbank gespeichert.

```
1 String generatedPasswordHash = BCrypt.hashpw(passwordStr, BCrypt.gensalt(12));
```

Die **hashpw** ist eine Methode von der BCrypt-Bibliothek, welche für die Verschlüsselung einer Zeichenkette benutzt wird. Diese wurde mit der nachfolgende Zeile importiert:

```
1 import org.mindrot.jbcrypt.BCrypt;
```

Danach wird ein Objekt vom Typ *User* erzeugt. Dort werden mit allen *setter*-Funktionen, die in der Klasse *User.java* vorhanden sind, die Informationen von den Felder - den Name, das Passwort und die Email-Adresse in diesem Objekt gespeichert. Das ganze Objekt wird dann in der Funktion *insertUser* hinzugefügt, welche aus der Klasse **DatabaseHelper.java** vorkommt. Die SQL-Anfrage in dieser Methode wird sich darum kümmern, die Informationen dieses Benutzers korrekt und sicher in

der lokalen Datenbank zu speichern.

Die `isValidEmail`-Funktion [13] validiert, ob eine Email-Adresse gültig für eine Registrierung ist:

```
1 public static boolean isValidEmail(String email) {  
2     String emailPattern = "^[a-zA-Z0-9_\\*-]+(?:\\.[a-zA-Z0-9_\\*-]+)*@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,7}$";  
3     Pattern pat = Pattern.compile(emailPattern);  
4     return email != null && pat.matcher(email).matches();  
5 }  
6  
7 }
```

Diese Funktion benutzte dieses Muster [14] und sollte mit der folgenden Zeilen in der Klasse importiert werden, bevor der eigentlichen Implementierung der Methode:

```
1 import java.util.regex.Pattern;
```

3.3.2 Implementierung der DatabaseHelper

Damit die Klasse **DatabaseHelper** erstellt wurde, sollten einige Importe gemacht werden:

```
1 import android.content.ContentValues;  
2 import android.content.Context;  
3 import android.database.Cursor;  
4 import android.database.sqlite.SQLiteDatabase;  
5 import android.database.sqlite.SQLiteOpenHelper;
```

In dieser Klasse werden alle Datenbankänderungen implementiert. Hier sind alle SQL-Anfragen beschrieben, die das Programm benutzt. In der Datenbank sollen zwei Tabellen vorhanden sein - **Tasks** [8], **Users** [8]

Das Objekt vom Typ **ContentValues** sollte für das Hinzufügen neuer Werte in der Datenbank benutzt werden. Am Anfang wird auf das **SQLiteDatabase db**-Objekt die Methode `getWritableDatabase` aufgerufen. Der Cursor wird mithilfe der SQL-Anfragen gefundene Werte in der Datenbank hinzugefügt. Danach wird auf das **db**-Objekt die Methode `insert` aufgerufen und der Cursor bzw. die Datenbank anschließen. Als Beispiel dient eine Methode von dieser Klasse, die in der Datenbank Informationen hinzufügen sollte:

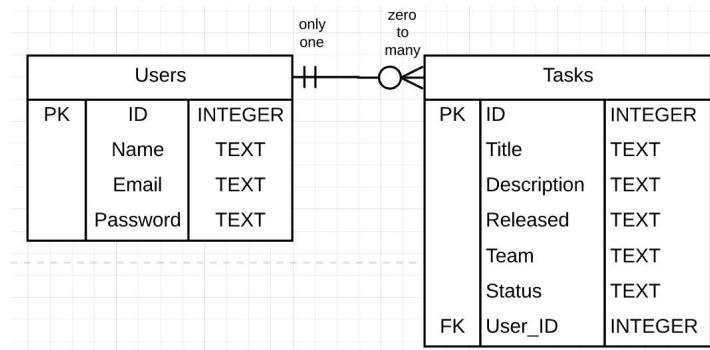
```
1 public void insertUser(User user) {  
2     db = this.getWritableDatabase();  
3     ContentValues cv = new ContentValues();  
4     String query = "SELECT * FROM " + TABLE_NAME;  
5     Cursor crs = db.rawQuery(query, null);  
6     int count = crs.getCount() + 1;  
7     cv.put(COLUMN_ID, count);  
8     cv.put(COLUMN_EMAIL, user.getEmail());  
9     ...  
10    db.insert(TABLE_NAME, null, cv);  
11    crs.close();  
12    db.close();  
13 }
```

Das Lesen von der Datenbank sollte auch nicht kompliziert sein. Zuerst wird auf das **SQLiteDatabase db**-Objekt die Methode `getReadableDatabase` aufgerufen. Dann wird der Cursor alle Werte, die die entsprechende SQL-Anfrage findet, durchgehen. In dem folgenden Beispiel wird den Cursor, der die Ergebnisse hat, zurückgegeben. Dieser kann später in einer anderen Java-Datei benutzt und mit einer WHILE-Schleife durchgelaufen werden:

```

1 public Cursor queryList() {
2     db = this.getReadableDatabase();
3     String query = "SELECT * FROM " + TABLE_NAME2;
4     return db.rawQuery(query, null);
5 }
```

Die Datenbank hat die Struktur von dem Bild unter.



3.3.3 Implementierung der Login

Beim Starten der Anmeldungsmaske wird ein **Splashscreen** aktiviert. Danach sollte diese Klasse ermöglichen, dass sich ein Benutzer in der Anwendung anmeldet. Damit die Anmeldung erfolgreich ist, sollen einige Bedingungen erfüllt werden:

- 1) Die Email-Adresse muss gültig sein.
- 2) Die Email-Adresse oder das Passwort dürfen nicht leer sein.
- 3) Die Passwörter müssen übereinstimmen.

Wenn es alles erfüllt ist, wird das eingegebene Passwort mit der **BCrypt** verschlüsselt und mit dem gespeicherten Passwort dieses Benutzers verglichen. Diese wird mit der Methode **checkpw** durchzuführen.

```
1 boolean ifMatchPass = BCrypt.checkpw(passwordStr, pass);
```

Die **checkpw** benutzt die BCrypt-Bibliothek, dementsprechend soll diese auch importiert werden. Wenn der Vergleich erfolgreich ist, wird eine Sitzung mit diesem Benutzer eröffnet:

```

1 session.setLogIn(true); // start the session
2 session.saveId(idInt); // save user_ID
```

Beide Methoden **setLogIn** und **saveId** sind in der Klasse **Session.java** implementiert und benutzen **SharedPreferences**, um Benutzerdaten lokal auf dem Smartphone zu speichern.

3.3.4 Implementierung der Session

In dieser Klasse wird die Sitzung eines Benutzers implementiert. Um eine Session zu eröffnen, musste das folgende Zeile importiert werden:

```
1 import android.content.SharedPreferences;
```

SharedPreferences [15] wird die Rohdaten in der Form eines Schlüssels in der Datei der Mobile-App speichern. Es konnte den privaten Speicherungsmodus ausgewählt werden, damit die anderen Anwendungen in dieser Datei nicht zugreifen. Der Editor wurde sich um die sichere Änderungen der Informationen kümmern.

```
1 preferences = c.getSharedPreferences("KanAzubi", Context.MODE_PRIVATE);  
2     editor = preferences.edit();  
3     editor.apply();
```

In dem **SharedPreferences** werden die Email-Adresse, der Namen und die ID des Benutzers gespeichert. Mit den *getter*-Methoden dieser Klasse können diese Informationen immer abgerufen werden, wenn es eine aktive Sitzung gibt.

3.3.5 Implementierung der AdminPlattform

Diese Klasse wird es ermöglichen, dass der Administrator neue Aufgaben direkt von der Android-App in der Datenbank hinzufügen kann, indem er die Felder mit der Informationen für die Aufgabe ausfüllt. Diese Admin-Plattform kann nur mit der Email-Adresse „admin@dataport.de“ erreicht werden. Die spezielle Implementierung hier ist die Methode *addTask* aus der *DatabaseHelper*-Klasse, welche die Informationen von dieser Klasse nimmt und die in der Datenbank hinzufügt.

```
1 public void addTask(..., String team) {  
2     db = this.getWritableDatabase();  
3     ContentValues cv = new ContentValues();  
4     String query = "SELECT * FROM " + TABLE_NAME2;  
5     Cursor crs = db.rawQuery(query, null);  
6     int count = crs.getCount() + 1;  
7     cv.put(COLUMN_ID, count);  
8     ...  
9     cv.put(COLUMN_STATUS, "Offen");  
10    cv.put(COLUMN_TEAM, team);  
11    cv.put(COLUMN_USERID, 0);  
12    db.insert(TABLE_NAME2, null, cv);  
13    crs.close();  
14    db.close();  
15 }
```

3.3.6 Implementierung der Menu

Beim Starten von dieser Aktivität wird geprüft, ob der angemeldeten Benutzer der Administrator ist. Wenn das der Fall ist, dann wird der *Suchen* Button in dem *Hinzufügen* Button geändert, welcher zu der *AdminPlattform* Aktivität weiterleitet.

```
1 if (session.getId() == 1 && session.getEmail().equals("admin@dataport.de")) {  
2     String addTask = "Hinzufügen";  
3     searchBtn.setText(addTask);  
4     ...  
5 }
```

Ansonsten wird die Klasse wie üblicherweise funktionieren und die Aktivität soll für Benutzer bereit sein. In dieser Klasse wurde auch der Abmeldungsbutton implementiert. Der sollte die aktive Sitzung beenden und den Benutzer abmelden. Beim Starten der Aktivität wird überprüft, ob eine Session vorhanden ist. Wenn eine solche nicht existiert, wird der Benutzer automatisch abgemeldet:

```
1 session = new Session(this);  
2 if (!session.loggedin()) {  
3     logout();  
4 }
```

Wenn es aber eine aktive Sitzung gibt, dann kann beim Klicken von dem Abmeldungsbutton diese beendet werden. Dies ist nur mit einer Zeile möglich:

```
1 session.setLogIn(false);
```

In dieser Klasse gibt es noch zwei Spinner, welche die Suchkriterien entscheiden und an die nächsten Aktivität weiterleiten:

```
1 i.putExtra("status", statusStr);
2 i.putExtra("team", teamStr);
```

Und diese zwei Variablen soll in der Liste so behandelt werden, dass es nur eine Liste mit diesen Kriterien geöffnet wurde.

3.3.7 Implementierung der List

In dieser Klasse sollte eine Liste (ArrayList) mit den Ergebnissen von den Suchkriterien der *Menü*-Klasse erzeugt werden. In dem Code unten ist zu sehen, wie werden die durchgehende Werte mit dem „Intent“ für das Team und den Status funktionieren.

```
1 Intent i = getIntent();
2 Bundle bundle = i.getExtras();
3 if (bundle != null) {
4     status = (String) bundle.get("status"); //selected Status from Menu.java
5     team = (String) bundle.get("team"); //selected Team from Menu.java
6 }
```

Wenn es keine Ergebnisse gibt, sollte der Text „Die Liste ist leer“ anzeigen. Hier sollte auch die Benutzer ID geholt werden, damit die Liste nur für den richtigen Benutzer erstellt wird.

3.3.8 Implementierung der Tasks

In dieser Klasse sollen alle Informationen einer Aufgabe angezeigt werden. Jeder Benutzer soll auch die Möglichkeit haben, eine Aufgabe anzunehmen, wenn sie offen ist, oder als erledigt zu markieren, wenn diese schon von ihm angenommen wurde. Wenn eine Aufgabe „Erledigt“ ist, soll nur ihre Informationen zeigen, ohne einen Button unten. In dem Code unten sollte die richtige Aufgabe ID gefunden werden und diese auch in der Methode *addData* gesendet werden, damit die richtige Informationen für diese Tätigkeit ausgefüllt wird.

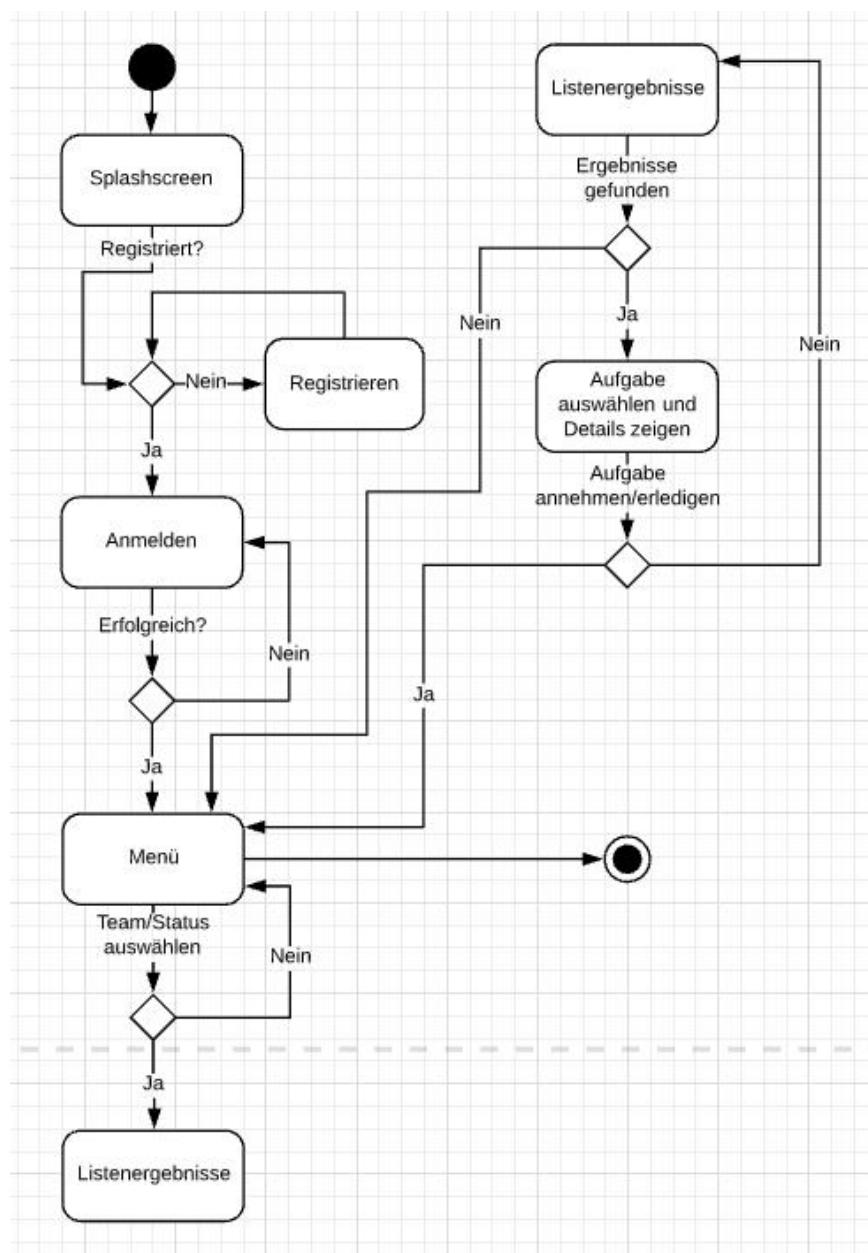
```
1 final Bundle bundle = getIntent().getExtras();
2 if (bundle != null) {
3     taskID = bundle.getInt("taskID");
4     listID = bundle.getIntegerArrayList("listID");
5 }
6 for(int i = 0; i < listID.size(); i++) {
7     if(i == taskID) {
8         taskNr = listID.get(i);
9         addData(taskNr);
10    }
11 }
```

Die anderen zwei Methoden, die sehr wichtig für diese Klasse sind, sind *acceptTask* und *doneTask*. Die werden in der *DatabaseHelper*-Klasse implementiert, da beide einige Werte in der Datenbank updaten müssen. Die SQL-Anfrage soll mit dem Wort „UPDATE“ anfangen und sie soll noch irgendwelche Bedingung in der „WHERE“-Anweisung hat, um nur ein Ergebnis zu aktualisieren. Siehe das Beispiel unten mit der erledigten Aufgabe:

```
1 String query = "UPDATE " + TABLE_NAME2 + " SET " +
2 COLUMB_STATUS + " = '" + doneTask +
3 "' WHERE " + COLUMB_ID2 + " = '" + taskNr + "'";
4 db.execSQL(query);
```

3.4 Entwurf der UML-Klassendiagramme

Vor der Implementierung der Klassen wurde ein UML-Klassendiagramm per Hand gezeichnet. Nach dem Entwurf aller Java Klassen mithilfe des Diagramms wurde ein Plugin **SimpleUML** [16] im Android Studio benutzt, um alle Variablen und Funktionen jeder Klasse darzustellen. Diese werden in Abbildung 11 gezeigt. Mithilfe eines Klassendiagramms konnte berücksichtigt werden, welche der geplanten Methoden implementiert und welche Variablen benötigt werden. Außerdem wurden noch ein Aktivitätsdiagramm (siehe Abbildung unten) und ein Anwendungsfalldiagramm Abbildung 10 erstellt, welche die Verbindung aller Aktionen und die Anwendungsfälle bzw. Akteure in diesem System darstellen. Diese werden mit der Webseite **Lucidchart** [17] erzeugt.



3.5 Entwickeln der automatisierten Tests

In diesem Ausschnitt wird die Erstellung der Tests betrachtet. Anhand des fertigen Designs und der implementierten Java Klassen konnten einige Tests entworfen werden. Es handelt sich um automa-

tisierte Tests, die nicht nur die Funktionalität der Oberfläche, sondern auch die Funktionalität der Methoden überprüfen können. Der Code-Anhang im **Unterabschnitt C-9** zeigt, dass ein Test nicht nur einen Fehler auslösen kann, wenn ein Wert nicht korrekt ist, sondern auch die Richtigkeit der Funktion bei der Eingabe von korrekten Werten überprüfen kann. Einige Tests überprüfen, ob zum Beispiel die Anmeldung erfolgreich ist oder ob ein Benutzer sich mit einer gültigen Email-Adresse registrieren kann. Es wird auch geprüft, ob die Email-Adresse schon benutzt wurde oder ob das Passwort die Bedingungen erfüllt. Wenn ein Passwort nicht vorhanden ist oder weniger als sechs Zeichen hat, ist keine Registrierung möglich und es wird eine Fehlermeldung angezeigt. Beispiel: Anmelden und Abmelden als Administrator (siehe unten).

```
1    onView(withId(R.id.et_email)).perform(typeText("admin@dataport.de"));
2    closeSoftKeyboard();
3    onView(withId(R.id.et_password)).perform(typeText("123456"));
4    closeSoftKeyboard();
5    onView(withId(R.id.loginBtn)).perform(ViewActions.click());
6    onView(withText(R.string.successfulLogin))
7    .inRoot(withDecorView(not(is(LoginActivity.getWindow().getDecorView()))))
8    .check(matches(isDisplayed()));
9    onView(withId(R.id.logoutBtn)).perform(ViewActions.click());
```

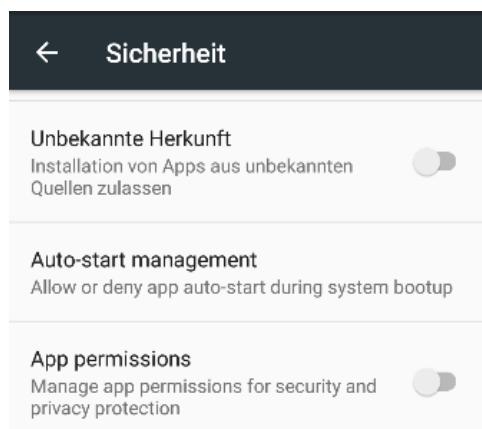
3.6 Entwickeln des JUnit Tests

Die automatisierten Tests haben die Funktionalität der Methoden in jeder Klasse geprüft. Es wurde eine Testsuite für einen **JUnit Test** [18] erstellt. Dieser Test soll prüfen, ob alle Emails, die für die Registrierung benutzt werden, gültig sind und ob die gestellten Bedingungen eingehalten werden. Er wird die *isEmailValid*-Funktion in der *Register*-Klasse validieren:

```
1  @Test
2  public void emailValidator_CorrectEmail() {
3      assertTrue(Register.isEmailValid("administrator@hotmail.co.uk"));
4      assertFalse(Register.isEmailValid("da$s?@a.da"));
5      ...
6  }
```

3.7 Deployen der Mobile-App

Um eine Android-App zu installieren und anwenden zu können, muss zuerst eine **.apk**-Datei erzeugt werden. Es gibt eine Release-Version und eine Debug-Version. In beiden Fällen wird **Gradle** als Buildsystem benutzt. Die App soll intern genutzt werden und von daher soll die Sicherheitseinstellung für unbekannte Quellen aktiviert werden. Siehe Abbildung unten.



4 Projektabschluss

4.1 Testdurchführung

Um die Tests durchzuführen muss ein Emulator installiert oder ein Android-Gerät angeschlossen sein. Alle Tests sind automatisiert und werden mithilfe des Espresso Frameworks durchgeführt. Diese sogenannten Instrumented Tests können mit dem Befehl **gradlew connectedAndroidTest** ausgeführt werden. Test-Ergebnisse werden mit dem Einsatz vom **Gradle**-Befehl produziert. Für die Tests wurde ein Emulator mit der Android-Version 7.0 („Nougat“) [19] verwendet. Auf der Konsole wurden alle Testszenarien und alle einzelnen Testfälle durchgeführt. Dort ist zu sehen welche und wie viele Tests erfolgreich bzw. fehlgeschlagen sind. Dazu werden die Tests nochmal auf einem Android-Handy mit der Version 6.0 („Marshmallow“) [20] durchgeführt. Auf der Abbildung 9 sind alle Tests zu sehen. Die Durchführung der Testszenarien verlief wie geplant. Alle 18 Tests sind automatisiert und waren auf Smartphone und Emulator 100% erfolgreich.

4.2 Soll-Ist-Vergleich

In der folgenden Tabelle wird der Verlauf des Projekts und die aufgewendeten Stunden mit der Planung verglichen. Der Teil der Projektplanung und der Dokumentation und Tests benötigten jeweils zwei Stunden mehr Zeit als gedacht, wodurch die Implementierung der App von 47 auf 43 Stunden gesunken ist. Diese Stunden wurden in der Durchführungsphase eingespart, da die Implementierung schnell und fehlerfrei verliefen. Entgegen der Planung im Projektantrag konnten die Funktionen Registrieren von Benutzern und Hinzufügen von Aufgaben (nur als Administrator) erfolgreich umgesetzt werden, da dieses den geplanten Zeitrahmen nicht überschritten hätte. Die Planung des Projektes wurde auch zeitlich länger als geplant sein, da das Soll-Konzept und die Kosten-Nutzen-Analyse mehrmals geändert wurden. Für die Dokumentation und die Tests wurde mehr Zeit genutzt, sodass die Implementierung gut beschrieben und getestet werden kann. Somit wurden die 70 Arbeitsstunden eingehalten.

Projektphase	Soll-Stunden	Ist-Stunden
Projektplanung	5	7
Projektdurchführung	47	43
Test und Dokumentation	18	20
Gesamtsumme	70	70

Tabelle 2: Soll-Ist-Vergleich

4.3 Fazit

Das Projekt „Entwicklung einer Mobile-App-Anwendung zum Annehmen von verschiedenen Aufgaben, die für die Auszubildenden im Unternehmen bereitgestellt werden“ konnte mit erhöhtem Leistungsumfang erfolgreich abgeschlossen werden. Als Resultat ist eine Android-App entstanden, mit dessen Hilfe viele Auszubildenden verschiedene Aufgaben annehmen und erledigen können.

A Glossar

.apk Die Installationsdatei für eine Android-Applikation.

Android Ein Betriebssystem für Smartphones.

Android SDK Android Entwicklerwerkzeug.

Aktivität Die Ansicht einer Android-Applikation.

Dependency Bibliotheks-Abhängigkeit für ein Framework.

Deployen Die Bereitstellung einer Android-Anwendung.

Full Service Provider Vollständige Übernahme von Geschäftsprozessen.

Framework Eine Art externer Bibliotheken.

Gradle Build-Management-Tool.

Integrationstests Eine Menge an Tests, die die Funktionalität einer Software als ganzes testen soll.

Java Eine höhere Programmiersprache

Kanban ist eine Methode, bei der die Anzahl gleichlaufender Arbeiten begrenzt und somit kürzere Durchlaufzeiten erreicht bzw. Hindernisse und Probleme schnell erkennbar gemacht werden sollen.

Linux Ein freies, unix-ähnliches Mehrbenutzer-Betriebssystem.

Mockup Ein Vorführmodell, das die Funktionen eines fertigen Produktes repräsentieren soll.

Mobile-App Eine Applikation für das Smartphone.

MS-Outlook Eine Software von Microsoft zum Verwalten von Terminen und Aufgaben, sowie zum Empfangen und Versenden von E-Mails.

Open Source Software Eine Software, deren Quelltext öffentlich zugreifbar ist und vom Dritten eingesehen und genutzt werden kann.

Splashscreen Eine Aktivität, die beim Starten einer Applikation dargestellt wird.

SQL ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten und Abfragen von darauf basierenden Datenbeständen.

UML-Aktivitätsdiagramm Ein Aktivitätsdiagramm stellt die Verbindung von elementaren Aktionen und deren Vernetzungen mit Kontroll- und Datenflüssengrafisch dar.

UML-Klassendiagramm Ein Klassendiagramm ist ein Strukturdiagramm der Unified Modeling Language (UML) zur grafischen Darstellung von Klassen und ihren Beziehungen.

UML-Anwendungsfalldiagramm Ein Anwendungsfalldiagramm stellt Anwendungsfälle und Akteure mit ihren jeweiligen Abhängigkeiten und Beziehungen dar.

Virtualisierung Abstraktion von physikalischen IT-Ressourcen in Form von Hardware, Software oder Netzwerken zu virtuellen Komponenten.

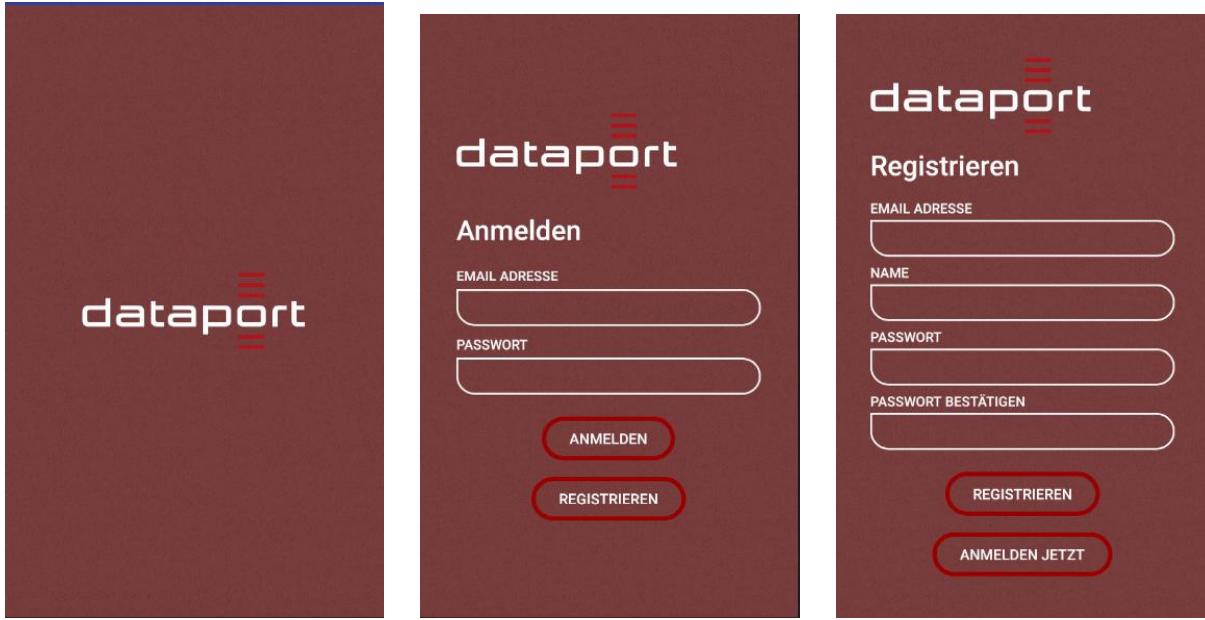
B Literaturverzeichnis

- [1] D. AöR, "Dataport lösungen a-z." <https://www.dataport.de/Seiten/L%C3%BCsungen/LC3%BCsungen-A-bis-Z.aspx>. [Online, letzter Abruf: 02-November-2018].
- [2] D. AöR, "Unternehmensportät." <https://www.dataport.de/Seiten/Unternehmen/%C3%9Cber-uns.aspx>. [Online, letzter Abruf: 02-November-2018].
- [3] Google, "Android." <https://www.android.com/>. [Online, letzter Abruf: 11-Oktober-2018].
- [4] "Android studio." <https://developer.android.com/studio/index.html>. [Online, letzter Abruf: 11-Oktober-2018].
- [5] "Sqliteman." <http://sqliteman.yarpen.cz/>. [Online, letzter Abruf: 12-Oktober-2018].
- [6] "Sqlite dokumentation." <https://www.sqlite.org/docs.html>. [Online, letzter Abruf: 17-Oktober-2018].
- [7] G.Inc., "Gradle." <https://gradle.org/>. [Online, letzter Abruf: 13-Oktober-2018].
- [8] Google, "Ui-tests mit espresso." <https://developer.android.com/training/testing/espresso/index.html>. [Online, letzter Abruf: 01-November-2018].
- [9] "Adobe spark." <https://spark.adobe.com/sp/>. [Online, letzter Abruf: 11-Oktober-2018].
- [10] "Background." <http://bg.siteorigin.com/>. [Online, letzter Abruf: 13-Oktober-2018].
- [11] "Round icon." [https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html#foreground.type=clipart&foreground.clipart=assignment_turned_in&foreground.space.trim=1&foreground.space.pad=0.2&foreColor=rgba\(161%2C%20184%2C%20194%2C%200\)&backColor=rgb\(148%2C%2068%2C%2068\)&crop=1&backgroundShape=square&effects=shadow&name=ic_launcher](https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html#foreground.type=clipart&foreground.clipart=assignment_turned_in&foreground.space.trim=1&foreground.space.pad=0.2&foreColor=rgba(161%2C%20184%2C%20194%2C%200)&backColor=rgb(148%2C%2068%2C%2068)&crop=1&backgroundShape=square&effects=shadow&name=ic_launcher). [Online, letzter Abruf: 01-November-2018].
- [12] "bcrypt." <https://www.mindrot.org/files/jBCrypt/jBCrypt-0.1-doc/BCrypt.html>. [Online, letzter Abruf: 19-Oktober-2018].
- [13] "Email validation." <https://www.geeksforgeeks.org/check-email-address-valid-not-javascript/>. [Online, letzter Abruf: 20-Oktober-2018].
- [14] "Struktur einer email-adresse." <https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html>. [Online, letzter Abruf: 16-Oktober-2018].
- [15] "Shared preferences." <https://developer.android.com/reference/android/content/SharedPreferences>. [Online, letzter Abruf: 14-Oktober-2018].
- [16] "Simpleuml." <https://plugins.jetbrains.com/plugin/243-simpleuml>. [Online, letzter Abruf: 31-Oktober-2018].
- [17] "Lucidchart." <https://www.lucidchart.com/>. [Online, letzter Abruf: 12-Oktober-2018].
- [18] "Junit tests." <https://developer.android.com/training/testing/unit-testing/local-unit-tests>. [Online, letzter Abruf: 28-Oktober-2018].

- [19] Google, “Android 7.0 nougat.” https://www.android.com/intl/de_de/versions/nougat-7-0/. [Online, letzter Abruf: 27-Oktober-2018].
- [20] Google, “Android 6.0 marshmallow.” https://www.android.com/intl/de_de/versions/marshmallow-6-0/. [Online, letzter Abruf: 27-Oktober-2018].

D Abbildungen

Abbildung 2: Screenshots: Splashscreen, Anmelden, Registrieren

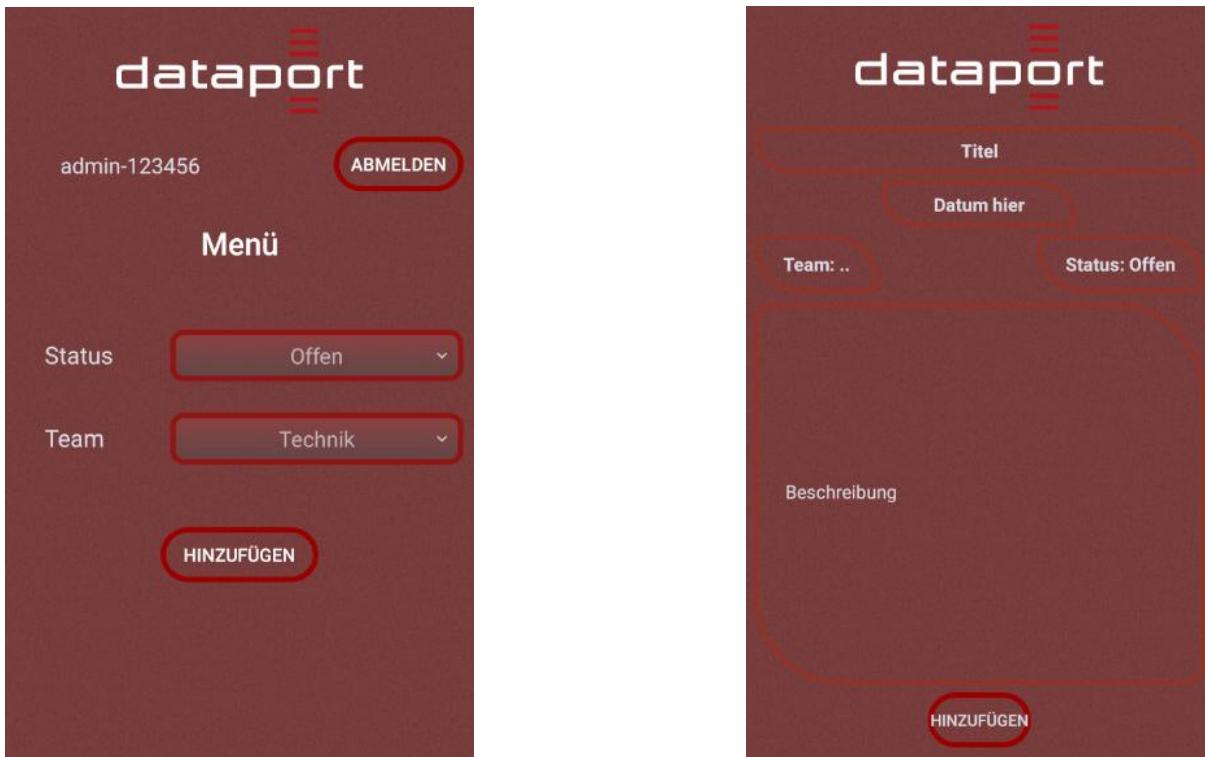


(a) Splashscreen

(b) Anmelden

(c) Registrieren

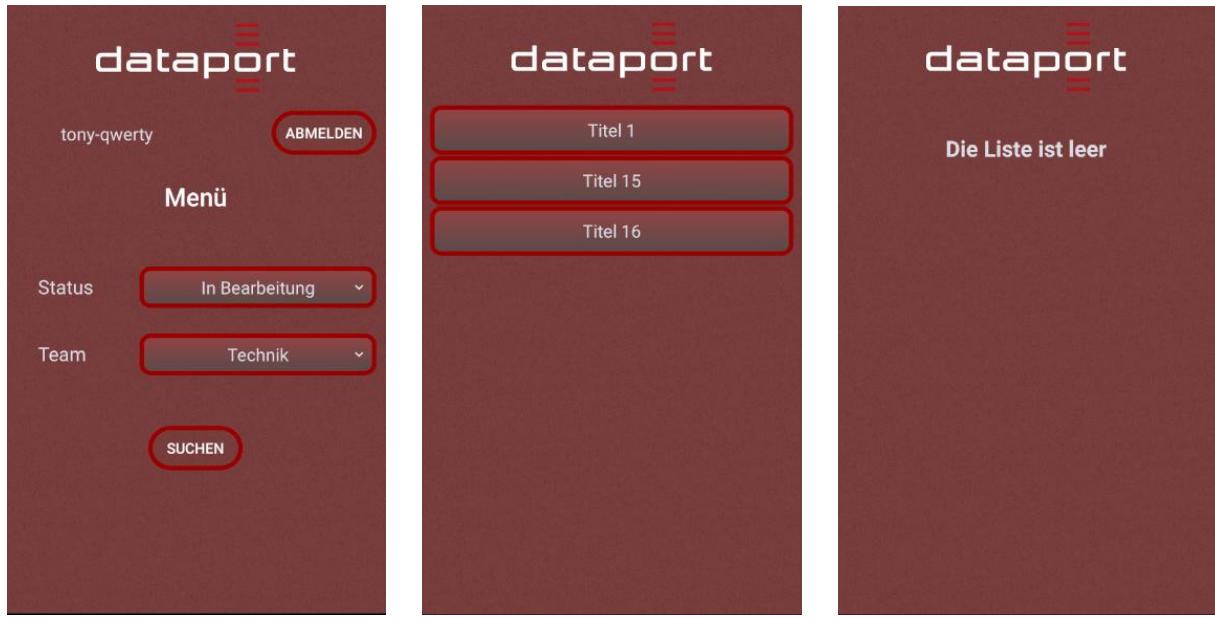
Abbildung 3: Screenshots: Admin, Admin-Plattform



(a) Admin

(b) Admin-Plattform

Abbildung 4: Screenshots: Benutzer, Listenergebnisse, Leere Liste



(a) Benutzer

(b) Listenergebnisse

(c) Leere Liste

Abbildung 5: Screenshots: Offen, In Bearbeitung, Erledigt



(a) Offen

(b) In Bearbeitung

(c) Erledigt

Abbildung 6: Screenshots: Menu, Mockup Beispiel, Mockup Realisierung



(a) Registrieren



(b) Mockup Anmeldung



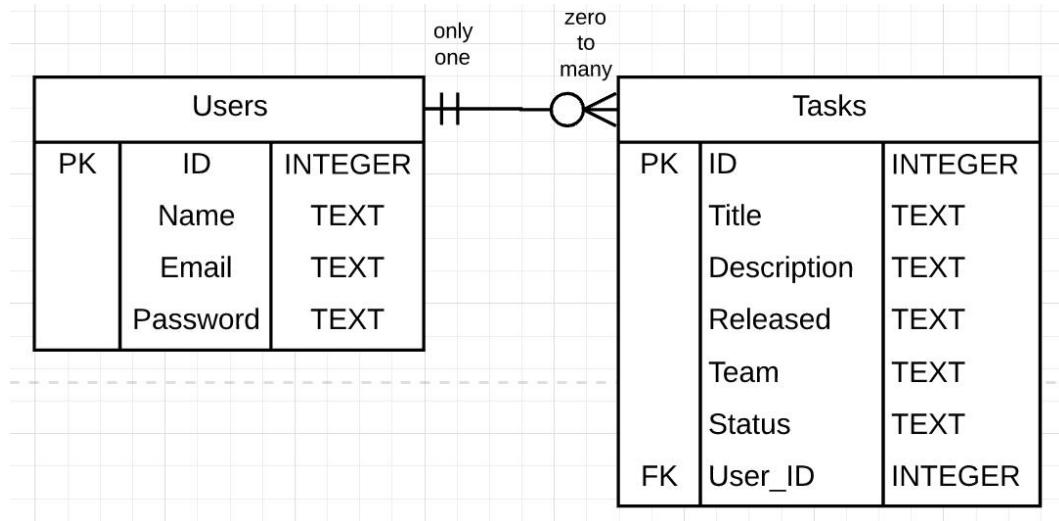
(c) Mockup Realisierung

Abbildung 7: Rund Icon



(a) Rund Icon

Abbildung 8: ERD, Tabelle users, Tabelle tasks



(a) ERD

	ID	Email	Password	Name
1	1	admin@dataport.de	\$2a\$12\$RioU.yn/Qu4WBYhlwKZ18uadT2bYyO6JF1O5XXulyjOWqgyKTphKS	admin-123456
2	2	tony@gmail.com	\$2a\$12\$zkTNjGy8UDW0lUeUly75GOiloUXjSkQhvEPZP1pQvbwlmXpGV/0SO	tony-qwerty
3	3	tonkata@gmx.de	\$2a\$12\$HoSs/eXOtVc3zEbikuUY2OullVAdN0rE1te1lqX6GU0PZRmTxbgi	tonkata-123456

(b) Tabelle users

	ID	Title	Description	Released	Team	Status	User_ID
9	9	Titel 9	Beschreibung 9	06.04.2018	Marketing	Erledigt	3
10	10	Titel 10	Beschreibung 10	13.11.2015	Software	Offen	0
11	11	Titel 11	Beschreibung 11	23.08.2014	Software	In Bearbeitung	3
12	12	Titel 12	Beschreibung 12	09.09.2018	Software	Erledigt	2
13	13	Titel 13	Technik; In Bearbeitung - 2,13,14	17.02.2017	Technik	In Bearbeitung	2
14	14	Titel 14	Technik; In Bearbeitung - 2,13,14	07.09.2017	Technik	In Bearbeitung	2
15	15	Titel 15	Technik; Offen - 1+15+16	14.12.2017	Technik	Offen	0
16	16	Titel 16	Technik: Offen - 1+15+16	12.01.2016	Technik	Offen	0

(c) Tabelle tasks

Abbildung 9: Alle automatisierte Tests

Test Results		2m 30s 708ms
▼	OK com.example.techgeek.kanazubi.AdminPlatformTest	32s 751ms
	OK adminPlatformTest	32s 751ms
►	OK com.example.techgeek.kanazubi.ExampleInstrumentedTest	100ms
▼	OK com.example.techgeek.kanazubi.LoginTest	31s 340ms
	OK enterIncorrectEmail	4s 895ms
	OK loginAndMore	6s 555ms
	OK goToRegistrationAndBack	7s 533ms
	OK enterIncorrectPassword	5s 583ms
	OK enterEmptyEmailOrPassword	6s 774ms
▼	OK com.example.techgeek.kanazubi.MenuTest	34s 595ms
	OK menuSpinnerAndListResultTest	13s 491ms
	OK joinTask_AcceptReadyDone_Test	21s 104ms
▼	OK com.example.techgeek.kanazubi.RegistrationTest	51s 922ms
	OK registerPasswordLessThan6Chars	7s 834ms
	OK registerPasswordsNotMatching	8s 66ms
	OK registerAndMore	14s 15ms
	OK registerEmptyFields	6s 270ms
	OK registerInvalidEmail	7s 586ms
	OK registerEmailExists	8s 151ms
▼	OK com.example.techgeek.kanazubi.SharedPreferencesTest	0ms
	OK putAndGetID	0ms
	OK putAndGetName	0ms
	OK putAndgetEmail	0ms

Abbildung 10: Anwendungsfalldiagramm

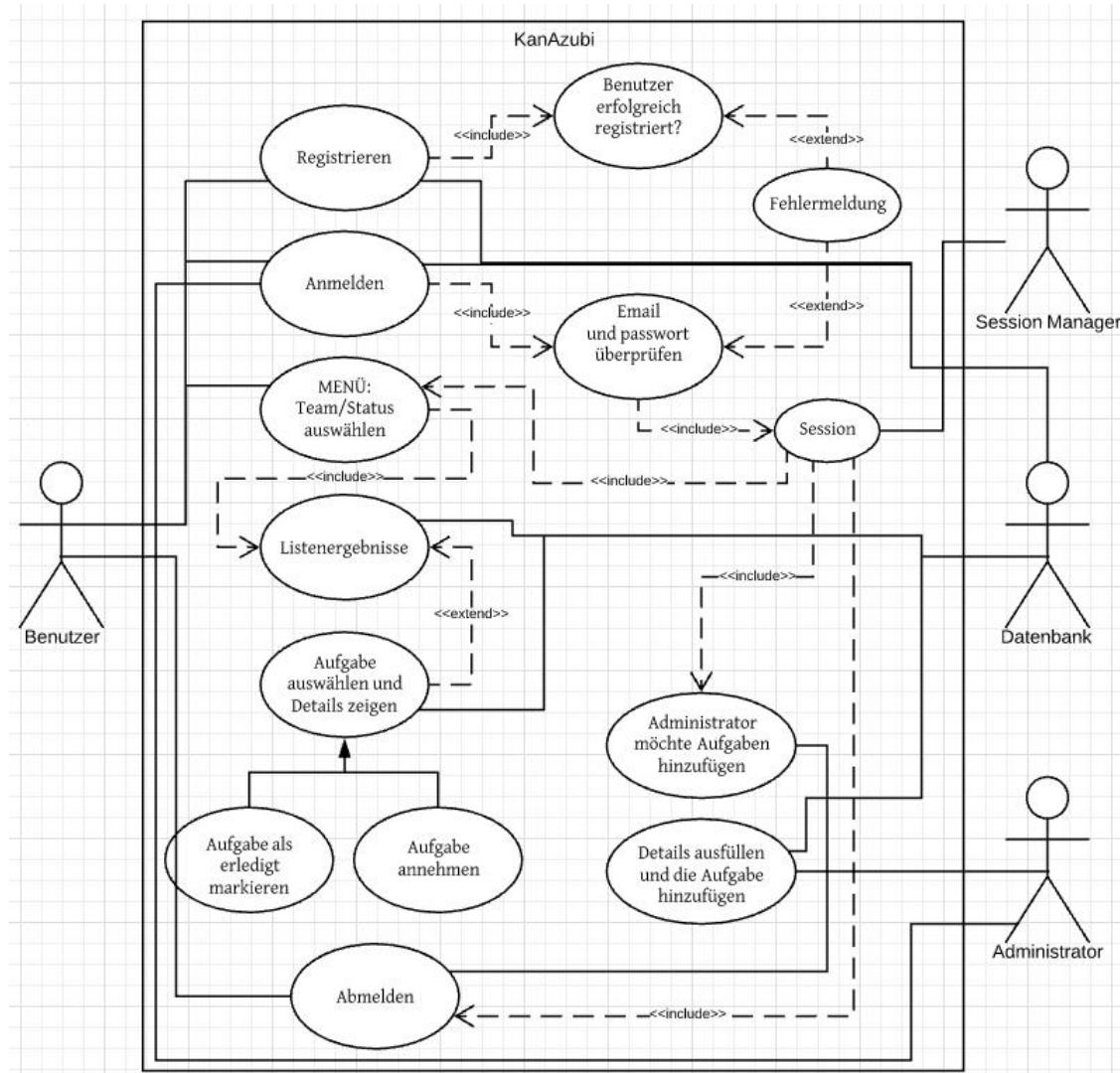


Abbildung 11: Alle Klassen, Funktionen und Variablen

+ DatabaseHelper extends SQLiteOpenHelper	+ Login extends AppCompatActivity	+ AdminPlatform extends AppCompatActivity	+ Task extends AppCompatActivity
<ul style="list-style-type: none"> fields - db : SQLiteDatabase - final DB_VERSION : int - final DB_NAME : String - final TABLE_NAME : String - final COLUMB_ID : String - final COLUMB_EMAIL : String - final COLUMB_PASSWORD : String - final COLUMB_NAME : String - final TABLE_NAME2 : String - final COLUMB_ID2 : String - final COLUMB_TITLE : String - final COLUMB_DESCR : String - final COLUMB_RELEASED : String - final COLUMB_TEAM : String - final COLUMB_STATUS : String - final COLUMB_USERID : String 	<ul style="list-style-type: none"> fields - relay1 : RelativeLayout - relay2 : RelativeLayout - loginBtn : Button - goToRegisterBtn : Button - email : EditText - password : EditText - DBHelper : DatabaseHelper - session : Session - handler : Handler - runnable : Runnable 	<ul style="list-style-type: none"> fields - session : Session - userID : int - title : String - descr : String - released : String - team : String - et_taskTitle : TextView - et_taskDescr : TextView - et_taskReleased : TextView - et_taskTeam : TextView - addTaskBtn : Button - DBHelper : DatabaseHelper 	<ul style="list-style-type: none"> fields - taskTitle : TextView - taskReleased : TextView - taskTeam : TextView - taskStatus : TextView - taskDescr : TextView - joinTaskBtn : Button - listID : ArrayList<Integer> - taskID : int - taskNr : int - userID : int - session : Session - openTask : String - inProgressTask : String - doneTask : String - buttonText : String - DBHelper : DatabaseHelper
<ul style="list-style-type: none"> constructors + DataBaseHelper (context: Context) 	<ul style="list-style-type: none"> methods # onCreate (savedInstanceState: Bundle) : void - loginUser () : void 	<ul style="list-style-type: none"> constructors 	<ul style="list-style-type: none"> methods # onCreate (savedInstanceState: Bundle) : void
<ul style="list-style-type: none"> methods + onCreate (db: SQLiteDatabase) : void + checkIfExist (email: String) : boolean + searchForID (email: String) : int + searchForName (email: String) : String + searchForPass (email: String) : String + insertUser (user: User) : void + addTask (title: String, descr: String, released: String, team: String) : void + queryList () : Cursor + acceptTask (userID: int, taskNr: int) : void + doneTask (taskNr: int) : void + onUpgrade (db: SQLiteDatabase, oldVersion: int, newVersion: int) : void 	<ul style="list-style-type: none"> fields - relayReg1 : RelativeLayout - relayReg2 : RelativeLayout - goToLogin : Button - registerBtn : Button - email : EditText - password : EditText - confirmPassword : EditText - name : EditText - DBHelper : DatabaseHelper - handler : Handler - runnable : Runnable 	<ul style="list-style-type: none"> fields - relay1 : RelativeLayout - relay2 : RelativeLayout - tvName : TextView - status : Spinner - team : Spinner - logoutBtn : Button - searchBtn : Button - statusStr : String - teamStr : String - session : Session - handler : Handler - runnable : Runnable 	<ul style="list-style-type: none"> fields - emptyList : TextView - listResult : ListView - session : Session - userID : int - status : String - team : String - listID : ArrayList<Integer> - listTitle : ArrayList<String> - adapter : ArrayAdapter - DBHelper : DatabaseHelper
<ul style="list-style-type: none"> constructors 	<ul style="list-style-type: none"> methods # onCreate (savedInstanceState: Bundle) : void + isEmailValid (email: String) : boolean + registerUser () : void 	<ul style="list-style-type: none"> constructors 	<ul style="list-style-type: none"> methods # onCreate (savedInstanceState: Bundle) : void - logout () : void + onBackPressed () : void
<ul style="list-style-type: none"> Session 	<ul style="list-style-type: none"> User 	<ul style="list-style-type: none"> List 	
<ul style="list-style-type: none"> fields - preferences : SharedPreferences - editor : Editor - c : Context 	<ul style="list-style-type: none"> fields - email : String - password : String - name : String 	<ul style="list-style-type: none"> fields - emptyList : TextView - listResult : ListView - session : Session - userID : int - status : String - team : String - listID : ArrayList<Integer> - listTitle : ArrayList<String> - adapter : ArrayAdapter - DBHelper : DatabaseHelper 	
<ul style="list-style-type: none"> constructors + Session (c: Context) 	<ul style="list-style-type: none"> constructors 	<ul style="list-style-type: none"> constructors 	
<ul style="list-style-type: none"> methods + saveEmail (email: String) : void + getEmail () : String + saveUserName (name: String) : void + getUserName () : String + saveId (id: int) : void + getId () : int + setLogin (login: boolean) : void + loggedIn () : boolean - getContext () : Context 	<ul style="list-style-type: none"> methods + getEmail () : String + setEmail (email: String) : void + getPassword () : String + setPassword (password: String) : void + getName () : String + setName (name: String) : void 	<ul style="list-style-type: none"> methods # onCreate (savedInstanceState: Bundle) : void - createList () : void 	