Projektdokumentation

Entwicklung einer mobilen Applikation zur Terminbestimmung und -verwaltung von Veranstaltungen zwischen Mitarbeitern im Unternehmen

Auszubildender Fachinformatiker für Anwendungsentwicklung
Toni Kozarev
Projektzeitraum vom 20.03.2018 bis 20.04.2018

Auszubildender
Name: Toni Kozarev

Ausbildungsbetrieb
Dataport Anstalt öffentlichen Rechts
Niederlassung: Bremen
Anschrift: Utbremer Str. 90
28217 Bremen

Inhaltsverzeichnis

Tabellenverzeichnis

Liste von Code-Anhängen

Abbildungsverzeichnis

1 Einführung

Diese Projektdokumentation beschreibt die Planung, Implementierung und das Testen einer Mobile-App zur Terminabstimmung und -verwaltung von Veranstaltungen zwischen Mitarbeitern im Unternehmen. Die Projektdokumentation berücksichtigt neben der Auswahl passender Frameworks dabei 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-Goverment-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, Lünenburg, Halle und Magdeburg.
Die Aufgabe unseres Unternehmens ist die Versorgung von Kommunikation- 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 Mitrarbeiter 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, Accounting und Linux vorhanden, die am Standort in Bremen vermittelt wurden.

1.3 Projektbeschreibung

Es soll ein Prototyp einer Mobile-App für die Vereinbarung und Bearbeitung eines betrieblichen Termins mit verschiedenen Personen entwickelt werden. Diese soll auf dem Android-Betriebssystem [3] lauffähig sein. Ein Event soll von jedem Mitarbeiter, der vom Ersteller eingeladen ist, bearbeitet und geändert werden. Jeder Mitarbeiter hat beschränkte Optionen, während ein Administrator sowohl alle Informationen ändern kann, als auch das Event komplett löschen oder eine neue Frist ansetzen kann. Die Android-App soll die Möglichkeit haben, dass mehrere Benutzer an verschiedenen Veranstaltungen teilnehmen und interaktiv neue Ideen einbringen können. Es ist auch möglich bei einem Event zu dem man eingeladen wurde, dieses abzulehnen. 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 die Bearbeitung eines betrieblichen Termins von mehreren Mitarbeitern bei Dataport über keine technische Umsetzung, wie bspw. eine Mobile-App. Es können derzeit Termine in MS-Outlook erstellt werden, diese können aber nur vom Ersteller geändert werden. Ein großer Teil der Mitarbeiter nutzt für die Terminabstimmung einer Veranstaltung die Webseite Doodle [4]. Die dort erstellten Events können nur vom Administrator geändert werden. Alle Personen, die den Link für das Event haben, können nur für die gegebenen Daten abstimmen, was die Vermittlung von interaktiven Ideen negativ beeinflusst.

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. Diese Mobile-App soll darauf abzielen, eine Art Event Manager darzustellen. Da die Informationen und Event Details dynamisch konfigurierbar sein sollen, soll die Mobile-App über eine Datenbank, die Informationen speichert und zusätzlich eine Dateiablage für Bilder haben. Damit alle Benutzer miteinander kommunizieren können, wird die App das Mobilfunknetz nutzen, um Daten für die Events und unterschiedliche Konto Informationen auszutauschen. Dazu sollen bestimmte Frameworks ausgewählt und der Datenversand bzw. Datenempfang sichergestellt werden. Es sollen neben der Durchführung der eigentlichen Mobile-App auch Klassentests entwickelt und implementiert werden.
Darüber hinaus soll es möglich sein, dass der Zustand der App zwischengespeichert werden kann, sofern keine Internetverbindung besteht. Bei erneuter Verbindung sollen diese Daten gespeichert werden können.

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 ausführlich dokumentiert werden. Es werden durch die Entwicklung der Mobile-App weder Anschaffungskosten neben dem Arbeitslohn des Anwendungsentwicklers noch laufende Kosten anfallen. Die Rechnung entspricht $70Euro * 70Std. = 4900Euro$. Die Entwicklungskosten für dieses Projekt belaufen sich auf 4900€. Die Events sind kein Teil des Projektes und werden von Dritten erstellt, deswegen fließen sie nicht mit in die Projektkosten ein.
Die Installierung und Einrichten von der App sollte nicht mehr als 5 Minuten dauern. Danach sollte jeder Mitarbeiter ein Konto erstellen und seine Informationen ausfüllen. Dieser Ablauf wird kaum 5 Minuten dauern. Um ein Event zu erstellen, braucht jede Person 5-10 Minuten abhängig von der Information, die auszufüllen ist. Die Zeit-Ersparnis bei der Benutzung der Mobile-App ist ungenau zu berechnen, da jeder Mitarbeiter unterschiedlich viele Informationen einträgt bzw. zu einem späteren Zeitpunkt Informationen bearbeitet. Die Vorteile solch einer App gegenüber MS-Outlook oder Doodle sind, dass jeder Teilnehmer, der einen Zugang zu der Veranstaltung hat, kann die Informationen selber bearbeiten und muss nicht immer den Administrator kontaktieren.

2.4 Testplanung

Die Testfälle für diese Android-App sind so erstellt, dass es für jede Java Klasse 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, ein Event suchen, ändern oder löschen, Benutzer für ein Event einladen, die Floating Action Buttons (FAB Menu) testen. Dabei werden ebenfalls auch triviale Tests durchgeführt, die Tests sollen überprüfen, ob jede Aktion, die an einen Button gebunden ist, erfolgreich ausgeführt wird. Darüber hinaus soll überprüft werden, ob die zusätzlich implementierte Navigation korrekt funktioniert. Es werden noch verschiedene UI Elemente getestet, u.a.:

- Layout
- Button
- Edit Text
- List View
- Search View

Weitere Informationen bzgl. der Tests, siehe Unterabschnitt 3.5.

#	Testszenarien	einzelne Testfälle
1	Start der App	Funktionieren die Buttons korrekt?
2	Anmelden	Ist eine Anmeldung erfolgreich oder nicht? Ist die Email/Passwort gültig?
3	Registrieren	Ist die Registrierung erfolgreich oder wurde die Email schonmal benutzt?
4	Menü	Funktionieren des Abmeldungsbuttons und das Menü fehlerfrei?
5	Profil	Funktionieren die Texteingaben und speichern sie die neuen Informationen?
6	Benutzer einladen	Kann jeder Benutzer für einen Termin eingeladen werden?
7	Event erstellen	Kann ein Mitarbeiter ein Event erstellen?
8	Events	Ist das Suchen eines Termins möglich?
9	Event ändern	Event bearbeiten/verlassen vom Benutzer oder löschen vom Administrator
10	Benutzer suchen	Nach einem Mitarbeiter suchen und das erste Ergebnis öffnen
11	Visitenkarte öffnen	Funktioniert die Navigation?

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 [5] ist eine sehr beliebte Umgebung für die Erzeugung von Android Applikationen und wird meist empfohlen. Außerdem ist die Dokumentation des Firebase [6] Frameworks, die auch von Google stammt, sehr verständlich und ausführlich beschrieben, sowie öffentlich zugänglich und kommt mit vielen Beispielen.

3.1.1 Auswahl der Entwickungsumbebung

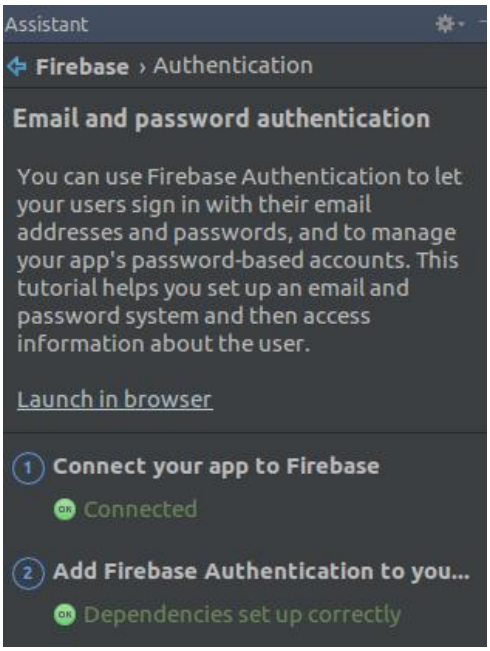
Für die Implementierung der Mobile-App wurde sich für die Entwicklungsumgebung Android Studio entschieden. Diese Umgebung wird empfohlen für den Einsatz der Android SDK und bringt bereits Gradle [7] mit, das für die Anwendung der Mobile-App benötigt wird.

3.1.2 Auswahl der Frameworks

Am Anfang des Projektes wurde sich für bestimmte Frameworks entschieden, die für die Durchführung des Projektes fundamental sind. Dazu zählt Firebase als ein integriertes Framework im Android Studio, siehe Abbildung 1, das für die Authentifizierung der registrierten/angemeldeten Benutzer, Aufbewahrung von Bildern und der Datenbank Anbindung die für restliche Informationen benötigt wird. Damit die ausgewählten Frameworks benutzt werden können, müssen sie in der Gradle-Konfiguration eingerichtet werden. Dies ist nur möglich, wenn die **build.gradle** Konfigurationsdatei mit der unter stehenden Zeilen erweitert werden. Nach der Implementierung einer Dependency haben alle Klassen und Methoden Zugriff auf die enthaltenen Funktionalitäten in den Frameworks. Momentan ist die Version 11.0.4 aktuell und sie sollte für alle Firebase Frameworks benutzt werden.

```
implementation 'com.google.android.gms:play-services-auth:11.0.4'
implementation 'com.google.firebase:firebase-auth:11.0.4'
implementation 'com.google.firebase:firebase-storage:11.0.4'
implementation 'com.google.firebase:firebase-database:11.0.4'
implementation 'com.google.firebase:firebase-core:11.0.4'
```

Abbildung 1: Firebase Authentifikation



3.1.3 Auswahl der UI-Frameworks

Für das UI Design werden andere Bibliotheks-Abhängigkeiten (Dependency – pl. Dependencies) wie CardView, Glide und ConstraintLayout genutzt. Das erste wurde für das Hauptmenü angewendet. Das Menü besteht aus 6 Bildschaltflächen, siehe Abbildung 4, aber nur 4 von den haben eine Funktionalität. Die anderen 2 sind momentan ohne Funktion, aber die werden zu einem späteren Zeitpunkt weiterentwickelt. Glide wurde nur für das Einspielen von Profilbildern verwendet, die in der Firebase Lagerung gespeichert wurden. Das ConstraintLayout Dependency wurde für das Design die Aktivitäten genutzt.

```
implementation 'com.android.support:cardview-v7:26.0.1'
implementation 'com.github.bumptech.glide:glide:4.3.1'
annotationProcessor 'com.github.bumptech.glide:compiler:4.3.1'
implementation 'com.android.support.constraint:constraint-layout:1.1.0-beta4'
```

3.1.4 Auswahl der Tests Frameworks

Außerdem wird Firebase TestLab [8] für einen Robo-Test benutzt. Dieser Test überprüft, ob eine bestimmte App auf einem bestimmten Smartphonemodell fehlerfrei funktioniert. Siehe Abbildung 9. Neben den allen bisher genannten Frameworks wurde ein weiteres genutzt, das Framework Espresso [9]. Dieses Framework wurde für die Erstellung der automatisierten Tests benutzt. Diese UI-Tests liefen 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.

```
androidTestImplementation 'com.android.support.test:runner:1.0.1'
androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.1'
```

Die verwendeten Frameworks sind Open Source Software und es fallen keine Kosten hierfür an.

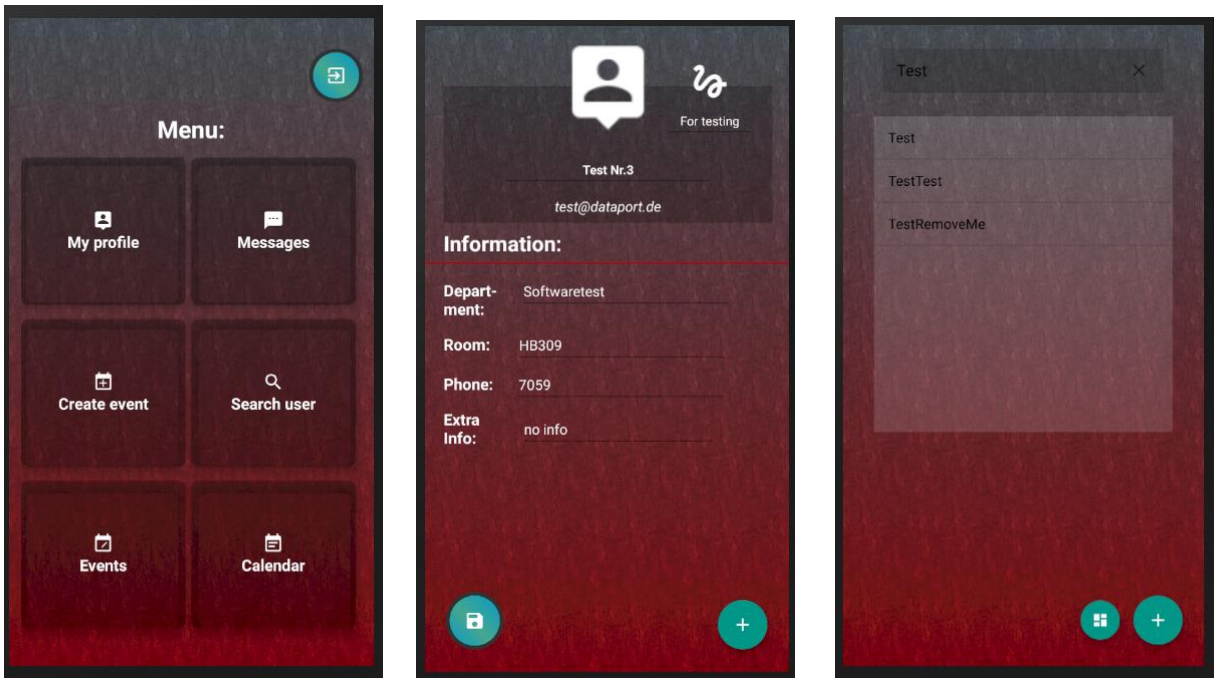
3.2 Entwicklung des UI Designs der Mobile-App

Als Erstes wird ein sogenanntes Mockup der App erstellt. Ein Mock-Up bedeutet einen Wegwerfprototyp der Benutzerschnittstelle einer zu erstellenden Software. Die Abbildung 8 stellt ein Beispiel für ein Mockup dar, die für das Projekt angefertigt wurde. 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 8 repräsentiert. Für das Design der App wurden viele freie Icons von Google Material [10] benutzt. 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 Navigationsmenü (FAB Buttons) mit einer Animation hinzugefügt, 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 [11]. Nach dem Laden der Applikation sollte eine Anmeldungsmaske erzeugt werden. Um Mitarbeiter anzumelden, sollten sie vorher registriert werden. Dazu wird es beim Starten von der Applikation eine Aktivität geben, wo beide Optionen auftauchen. Wenn ein Konto erstellt wurde, wird die Person zu einer neuen Aktivität weitergeleitet, wo sich das Hauptmenü befindet.

Abbildung 2: Screenshots: Hauptmenü, Profileseite, Events



(a) Hauptmenü (b) Profileseite (c) Events

Jeder angemeldete Nutzer hat die Möglichkeit seine Profil-Daten und Bilder zu ändern. Ein Nutzer kann auch ein Event erstellen oder bearbeiten. Dazu werden noch zwei neue Aktivitäten realisiert. Bei der Erstellung von einer Veranstaltung soll der Nutzer zuerst die Teilnehmer auswählen, die zu diesem Event eingeladen werden sollen. Danach gibt der Nutzer die Informationen für das Event ein. Alle Termine, die von dem eingeloggten Konto zu sehen sind, werden auf die zweite Aktivität - **Events** als eine Liste auftauchen. Dort können die Mitarbeiter ein Event nach dem Name suchen und öffnen. Wenn der Ersteller der Veranstaltung die angemeldete Person ist, dann hat diese Person die Berechtigung Informationen zu ändern, eine neue Frist anzusetzen oder sogar das ganze Event zu löschen. Ein eingeladener Nutzer kann die Texteingaben bearbeiten oder das Event verlassen.
Es wurde noch eine vierte Aktivität implementiert. Dort werden alle registrierten Konten aufgelistet. Jede Person kann ein Konto anhand einer Email-Adresse suchen. Beim Öffnen von einer Visitenkarte hat der Benutzer zugriff zu den Daten seiner Kollegen. Am Ende sollte ein Abmeldungsbutton hinzugefügt werden.

3.3.1 Implementierung der Login/Register

Die Implementierung der Anmeldungs- und Registrierungsaktivitäten fand mit dem Einrichtung des Firebase Frameworks Abbildung 1 statt. Für die Anmeldung und Registrierung von Mitarbeitern wurde Firebase Authentifikation benutzt. Damit die App einen Zugriff mit dieser Anmeldung realisieren kann, sollte eine Instanz zu einem **FirebaseAuth**-Objekt umgesetzt werden, wie unten abgebildet:

```
24 private FirebaseAuth mAuth;
25 ...
26 @Override
27 protected void onCreate(Bundle savedInstanceState) {
28   mAuth = FirebaseAuth.getInstance();
29   ...
30 }
```

Für die Registrierung sollte jede Person ihre Email und zweimal das gleiche Passwort eingeben. Wenn das Passwort übereinstimmt und die Email valide ist, wird auf dem **mAuth**-Objekt die Methode - **createUserWithEmailAndPassword** aufgerufen. Wenn diese Methode erfolgreich ausgeführt wurde, wurde ein neues Konto mit der eingegebenen Email-Adresse und Passwort erstellt. Jeder registrierte Nutzer erhält eine User-ID, die auch in der Firebase Authentifikation gespeichert wird. Firebase Authentifikation erlaubt keine doppelten Anmeldungen mit der gleichen Email-Adresse, deswegen ist jede Email-Adresse genauso wie die UID universell und eindeutig.

Für die Anmeldung sollte die gleiche Prozedur durchgeführt werden. Hier wurde die Methode **signInWithEmailAndPassword** auf dem **mAuth**-Objekt aufgerufen. Bei einer erfolgreichen Anmeldung sollte eine neue Aktivität geöffnet werden.

3.3.2 Implementierung der Mainpage/Profile

Nach der Implementierung des Anmeldebildschirms sollte das Hauptmenü implementiert werden. Für dieses wurde eine CardView-Ansicht erstellt [12]. Das Menü wurde auf 6 Felder aufgeteilt und jedes Feld erhält eine Abbildung und einen Text.

```
58 <GridLayout ... >
59   <!-- Row 1, Column 1 -->
60   <android.support.v7.widget.CardView ... >
61       <LinearLayout ... >
62           <ImageView ... />
```

```
63            <TextView ... />
64        </LinearLayout>
65    </android.support.v7.widget.CardView>
66
67    <!-- Row 1, Column 2 -->
68    <android.support.v7.widget.CardView ... >
69      ...
70    </android.support.v7.widget.CardView>
71  </GridLayout>
```

In Java würde das Menü so aussehen:

```
20  GridLayout mainGrid;
21  ...
22  @Override
23  protected void onCreate(Bundle savedInstanceState) {
24    mainGrid = findViewById(R.id.mainGrid);
25    ...
26  }
27
28  private void setSingleEvent(GridLayout mainGrid){
29          for(int i = 0; i < mainGrid.getChildCount(); i++){
30              CardView cV = (CardView)mainGrid.getChildAt(i);
31              final int var = i;
32              cV.setOnClickListener(new View.OnClickListener() {
33                  @Override
34                  public void onClick(View view) {
35                      if(var == 0){
36                          Intent i = new Intent(Mainpage.this, Profile.class);
37                          startActivity(i);
38                      }else if(var == 4){
39                        Intent i = new Intent (Mainpage.this, Events.class);
40                        startActivity(i);
41                      }else{
42                        ...   //die andere Teile der Menü
43                      }
44                  }
45              });
46          }
47      }
```

Für die Erstellung der Profilseite wurde eine Datenbank für die Speicherung der Informationen genutzt. Die Datenbank für diese Applikation ist wieder von Firebase und es werden die Daten in einer JSON-Datei gespeichert. Um eine Verbindung mit der Datenbank herzustellen, muss ein **mDatabase**-Objekt erstellt werden. Das Objekt ist vom Typ **FirebaseDatabase** und soll ähnlich wie bei der Firebase Authentifikation eine Instanz auf dem Objekt erzeugen. Für die Verwaltung von den Profilbildern wurden noch zwei Frameworks - Firebase Storage und Glide benutzt.
Um ein Bild hochzuladen, wurde die Funktion **uploadImage()** erstellt, die eine Instanz auf ein **Firebase Storage**-Objekt erzeugt. Danach wurde das Bild mit der Methode **putFile** hochgeladen.
Um das Bild zu speichern, sollte die Methode **setPhotoUri** auf dem Objekt vom Typ **UserProfileChangeRequest.Builder** genutzt werden. Danach sollten die neuen Änderungen mit der Methode **updateProfile** aktualisiert werden. Auf diese Weise soll das Bild erfolgreich gespeichert werden.

```
291 String imageUrl;
292 ...
293 UserProfileChangeRequest profile = new UserProfileChangeRequest.Builder()
```

```
                    .setPhotoUri(Uri.parse(imageUrl))
                    .build();
user.updateProfile(profile);
...
```

Um das Bild nach einem erneuten Öffnen der Profilseite zu laden, wurde Glide wie folgt benutzt:

```
ImageView profileInfoPhoto;
...
private void loadUserPhoto() {
    FirebaseUser user = mAuth.getCurrentUser();
      if (user != null) {
         if (user.getPhotoUrl() != null) {
           Glide.with(this)
                .load(user.getPhotoUrl().toString())
                .into(profileInfoPhoto);
         }
      }
}
```

Bei der Profilseite gibt es noch einige Texteingaben, wo jeder Benutzer mehr Informationen über sich selber eingeben kann. Um die Eingaben in einer JSON-Datenbank zu speichern, sollten sie in verschiedenen Variablen gespeichert werden und danach die Methode **child()** auf der **Database-Reference** benutzen, um die richtige Stelle für den Wert in der Datenbank zu finden.

```
FirebaseDatabase mDatabase = FirebaseDatabase.getInstance();
DatabaseReference mRef = mDatabase.getReference("Benutzer/");
EditText profileInfoName;
...
String name = profileInfoName.getText().toString();
mRef.child(userID).child("Name").setValue(name);
```

Um Informationen aus der Datenbank zu bekommen, muss die **DatabaseReference** der Methode **addValueEventListener** aufgerufen werden. Danach wurde auf die Funktion **onDataChange** mithilfe der *dataSnapshot* der richtige Text von der JSON-Datei ermittelt und die Profilseite aktualisiert.

```
DatabaseReference mRef = mDatabase.getReference("Benutzer/");
...
mRef.addValueEventListener(new ValueEventListener() {
      @Override
      public void onDataChange(DataSnapshot dataSnapshot) {
        String name = dataSnapshot.child("Name").getValue(String.class);
        profileInfoName.setText(name);
     }
}
```

3.3.3 Implementierung der anderen Klassen

Alle anderen Klassen wurden identisch zu der Profile-Klasse implementiert. Für jede Klasse können die Mitarbeiter Informationen von der Datenbank hinzufügen, ändern oder löschen. In der Events-/InviteMembers-/Search-Klassen wurden Listen mit allen Events/Benutzer aufgezeigt. Jeder kann ein Event suchen, wenn er Zugriff zu dieser Veranstaltung hat.

3.3.4 Implementierung anderer Komponenten

Es werden andere Komponenten wie Date Dialog und Floating Action Buttons implementiert. Die Aufgabe der FAB ist eine schnelle Navigation zu erzielen, während Date Dialog für die Auswahl der Frist eines Events zuständig ist.

3.4 Entwurf der UML-Klassendiagramme

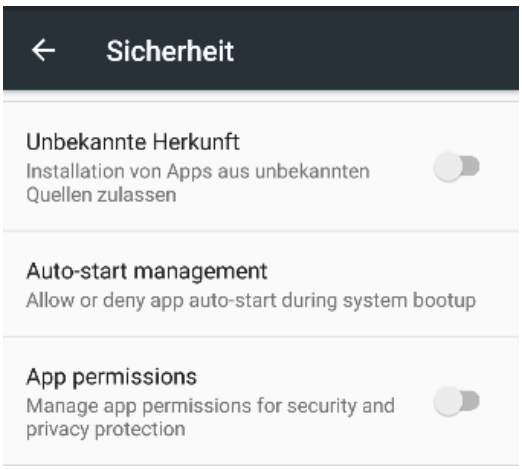
Vor der Implementierung der Klassen wurde ein UML-Klassendiagramm per Hand gezeichnet. Nach dem Entwurf aller Java Klassen mithilfe des Diagramms wurde ein Plugin im Android Studio benutzt, um alle Variablen und Funktionen jeder Klasse darzustellen. Diese werden in Abbildung 10 und Abbildung 11 gezeigt. Mithilfe eines Klassendiagramms konnte berücksichtigt werden, welche der geplanten Methoden implementiert und welche Variablen benötigt wurden.

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 automatisierte 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 nur das Navigationsmenü oder die Texteingaben und Buttons, während andere überprüfen, ob zum Beispiel die Anmeldung erfolgreich ist. Es prüft auch, ob ein Benutzer sich mit einer Email-Adresse registrieren kann, wenn die Email-Adresse schon benutzt wurde oder ob das Passwort die Bedingungen vom Firebase Authentifikation [13] erfüllt. Wenn ein Passwort nicht vorhanden ist oder weniger als sechs Zeichen hat, ist keine Registrierung möglich.

3.6 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 Dependency 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“) [14] 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“) [15] durchgeführt. Auf der Abbildung 9 ist ein Teil der Tests zu sehen. Die Durchführung der Testszenarien verlief wie geplant. Alle 41 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 Durchführung benötigten jeweils eine Stunde mehr Zeit als gedacht, wodurch die Projektplanung von 5 auf 6 Stunden und die Implementierung der App von 47 auf 48 Stunden angestiegen sind. Diese Stunden wurden in der Dokumentations-Phase eingespart, da die Tests schnell und fehlerfrei verliefen. Entgegen der Planung im Projektantrag konnten die Funktionen Versenden von Nachrichten und Kalender Vorschau nicht umgesetzt werden, da dieses den geplanten Zeitrahmen deutlich überschritten hätte. Somit wurden die 70 Arbeitsstunden eingehalten.

Projektphase	Soll-Stunden	Ist-Stunden
Projektplanung	5	6
Projektdurchführung	47	48
Test und Dokumentation	18	16
Gesamtsumme	**70**	**70**

Tabelle 2: Soll-Ist-Vergleich

4.3 Fazit

Das Projekt „Entwicklung eines Mobile-App-Prototypen zur Terminabstimmung und -verwaltung von Veranstaltungen zwischen Mitarbeitern im Unternehmen“ konnte mit reduziertem Leistungsumfang erfolgreich abgeschlossen werden. Als Resultat ist eine Android-App entstanden, mit dessen Hilfe viele Mitarbeiter verschiedene Events bearbeiten 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.

Floating Action Buttons Der FAB ist ein dynamisches UI-Element, das die schnelle Navigation in der Applikation unterstützt und die Benutzerfreundlichkeit verbessert.

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

JSON JSON ist ein kompaktes Datenformat in einer einfach lesbaren Textform.

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 Dritter eingesehen und genutzt werden kann.

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

UID User-ID.

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

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%B6sungen/LC3%B6sungen-A-bis-Z.aspx. [Online, letzter Abruf: 12-April-2018].

[2] D. AöR, "Unternehmensportät." https://www.dataport.de/Seiten/Unternehmen/%C3%9Cber-uns.aspx. [Online, letzter Abruf: 12-April-2018].

[3] Google, "Android." https://www.android.com/. [Online, letzter Abruf: 16-April-2018].

[4] "Doodle." https://doodle.com/de/. [Online, letzter Abruf: 17-April-2018].

[5] "Android studio." https://developer.android.com/studio/index.html. [Online, letzter Abruf: 11-April-2018].

[6] Google, "Firebase documentation." https://firebase.google.com/docs/. [Online, letzter Abruf: 18-April-2018].

[7] G.Inc., "Gradle." https://gradle.org/. [Online, letzter Abruf: 18-April-2018].

[8] Google, "Firebase tests." https://firebase.google.com/docs/test-lab/. [Online, letzter Abruf: 18-April-2018].

[9] Google, "Ui-tests mit espresso." https://developer.android.com/training/testing/espresso/index.html. [Online, letzter Abruf: 18-April-2018].

[10] Google, "Material icons." https://material.io/icons/. [Online, letzter Abruf: 15-April-2018].

[11] "Splashscreen." https://www.skillnotch.com/images/spinner.gif. [Online, letzter Abruf: 22-March-2018].

[12] Android. https://developer.android.com/reference/android/support/v7/widget/CardView.html. [Online, letzter Abruf: 27-March-2018].

[13] Google, "Firebase authentifikation." https://firebase.google.com/docs/auth/. [Online, letzter Abruf: 16-April-2018].

[14] Google, "Android 7.0 nougat." https://www.android.com/intl/de_de/versions/nougat-7-0/. [Online, letzter Abruf: 19-April-2018].

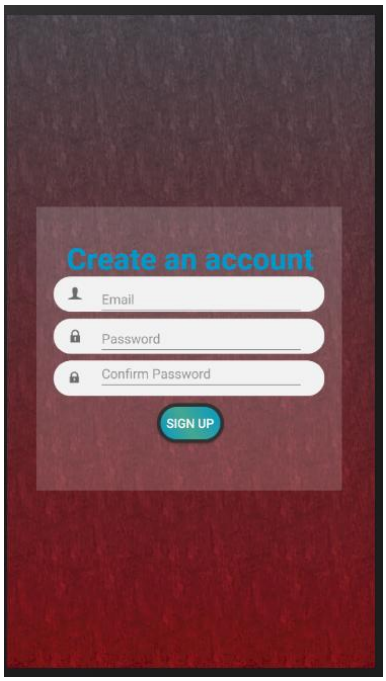
[15] Google, "Android 6.0 marshmallow." https://www.android.com/intl/de_de/versions/marshmallow-6-0/. [Online, letzter Abruf: 19-April-2018].

D Abbildungen

Abbildung 3: Screenshots: Splashscreen, Startseite, Registrierung

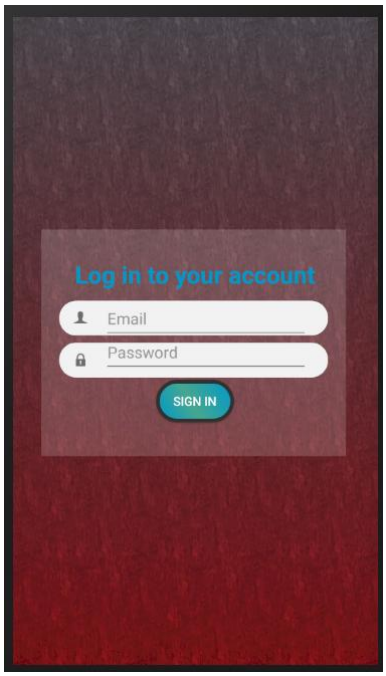
(a) Splashscreen

(b) Start der App



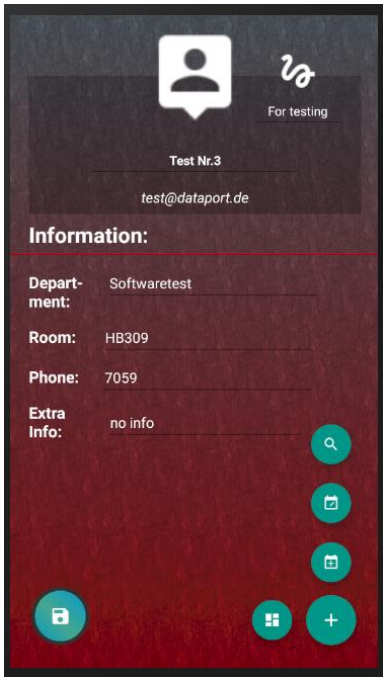
(c) Registrierung

Abbildung 4: Screenshots: Anmeldung, Hochladen vom Bild, Profileseite mit FAB



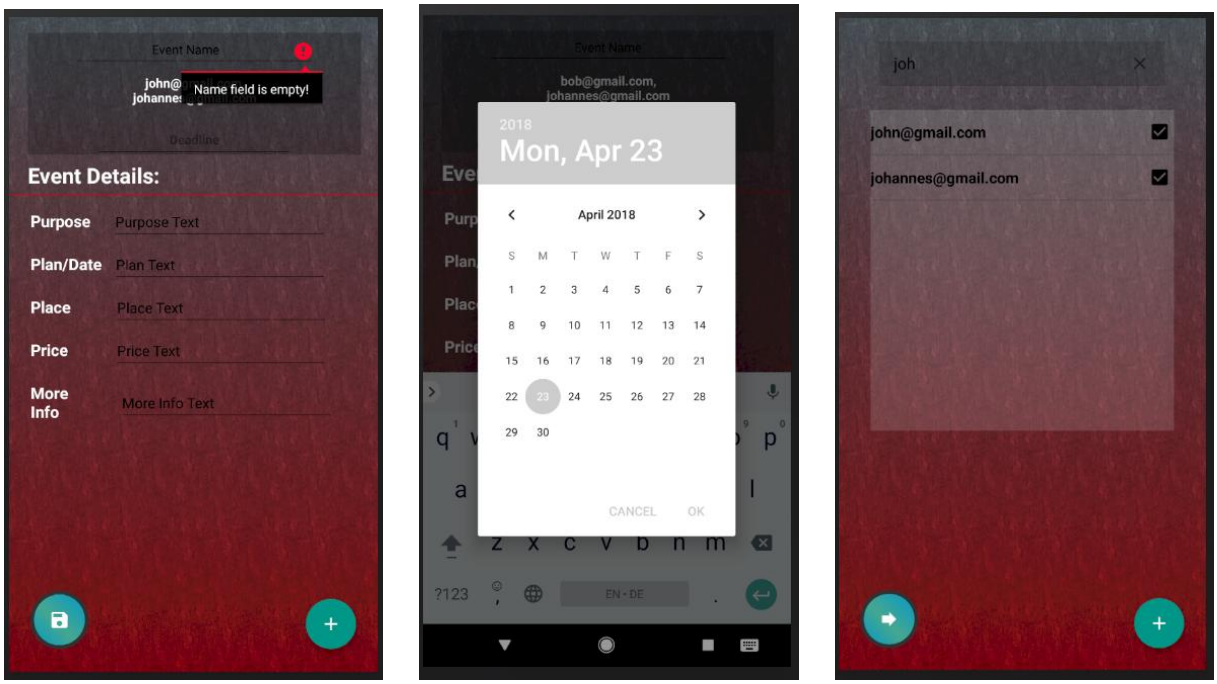
(a) Anmeldung

(b) Hochladen vom Bild



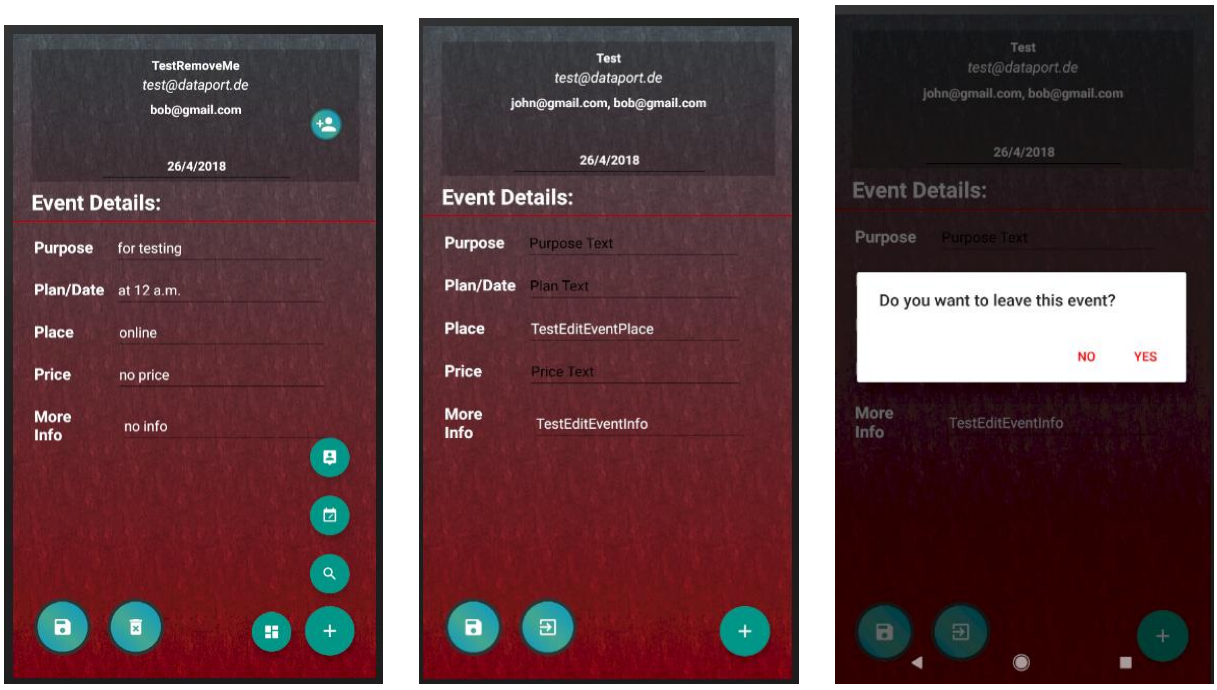
(c) Profileseite mit FAB

Abbildung 5: Screenshots: Event erstellen, Frist ansetzen, Teilnehmer einladen



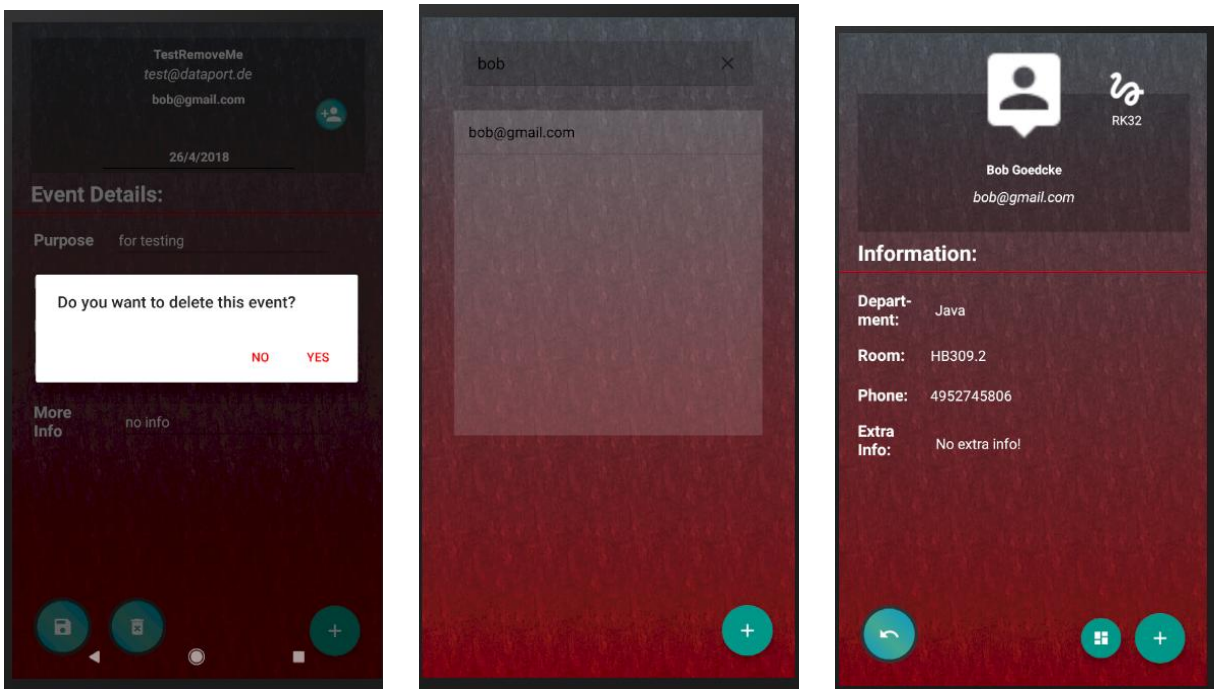
(a) Event erstellen (b) Frist ansetzen (c) Teilnehmer einladen

Abbildung 6: Screenshots: Event als Admin/Teilnehmer bearbeiten, Event verlassen



(a) Event als Admin bearbeiten (b) Event als Teilnehmer bearbeiten (c) Event verlassen

Abbildung 7: Screenshots: Event löschen, Mitarbeiter suchen, Visitenkarte

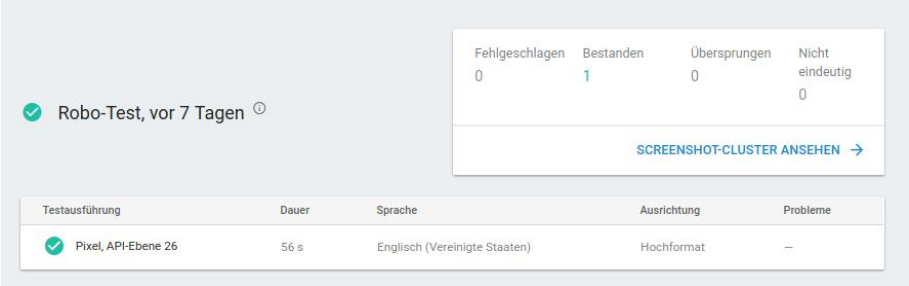


(a) Event löschen (b) Mitarbeiter suchen (c) Visitenkarte

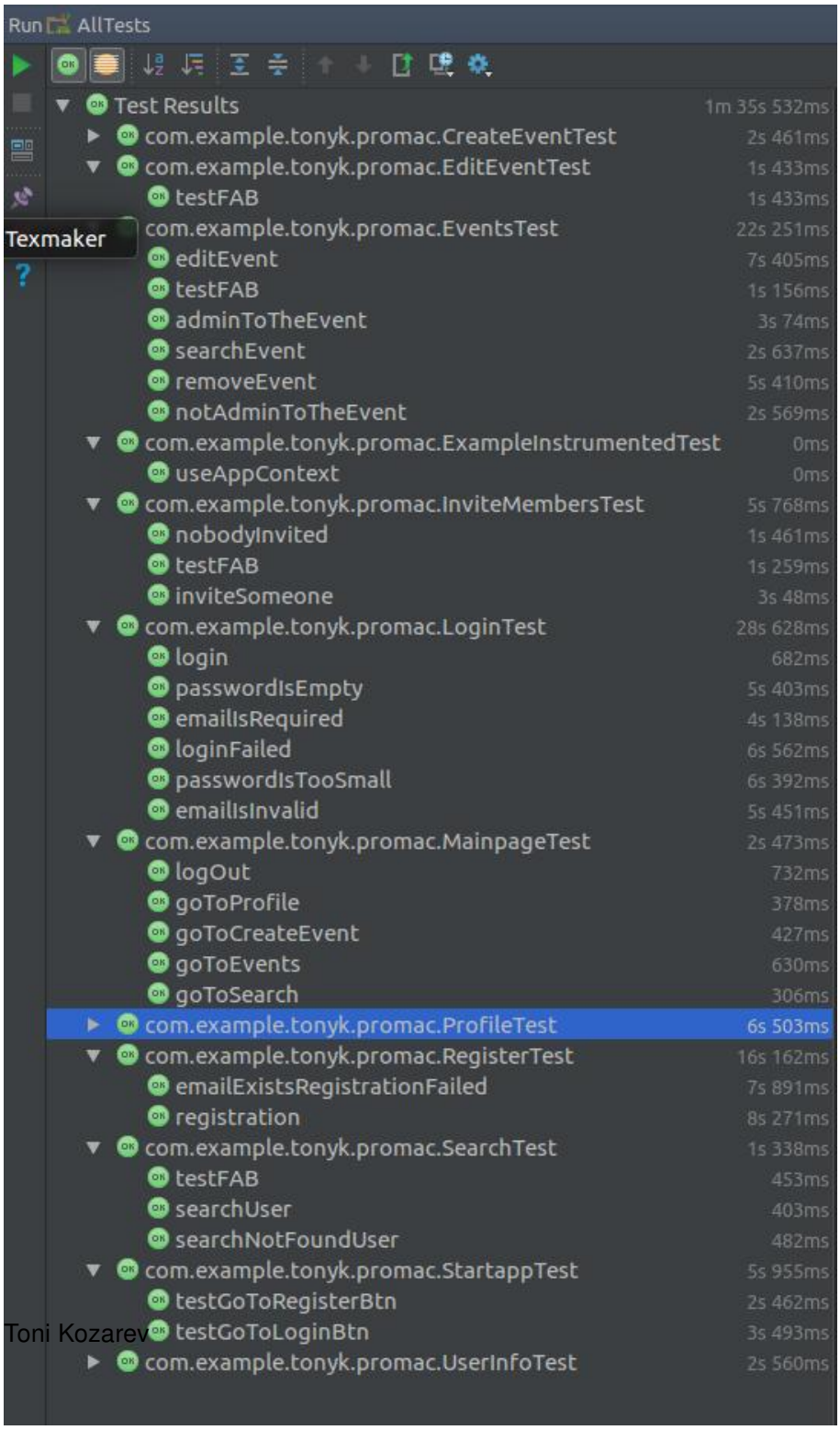
Abbildung 8: Screenshots: Hintergrundsbild, Mockup Beispiel, Mockup Realisierung

(a) Hintergrundsbild (b) Mockup Beispiel (c) Mockup Realisierung

Abbildung 9: Screenshots: Firebase TestLab - Robo-Test, alle erzeugte Tests



(a) Firebase TestLab - Robo-Test



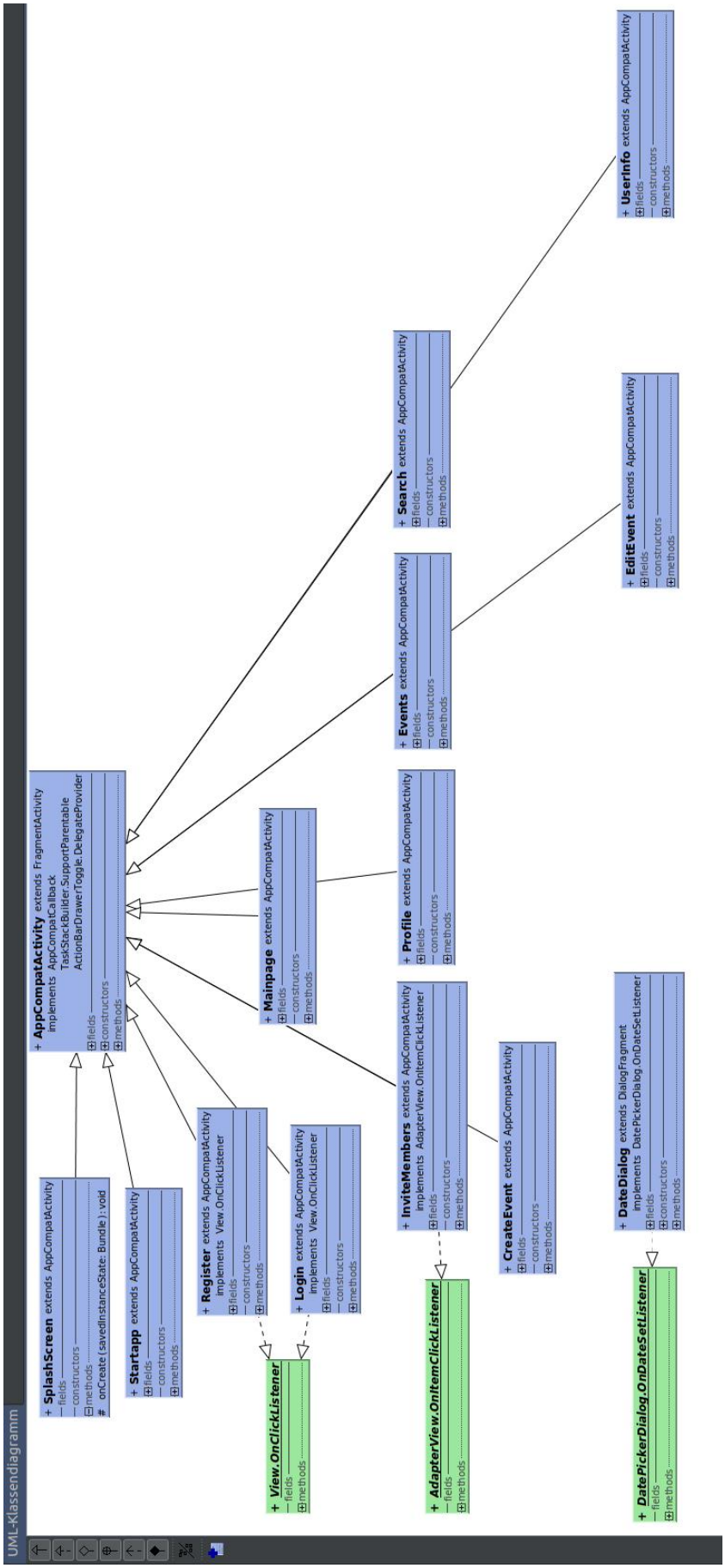


Abbildung 10: Alle UML-Klassen

Abbildung 11: Alle Klassen, Funktionen und Variablen