Technology
Arts Sciences
TH Köln

Entwicklungsprojekt interaktive Systeme

Wintersemester 2018/2019

Implementationsdokumentation

Meet & Remind

Dozenten

Prof. Dr. Gerhard Hartmann

Prof. Dr. Kristian Fischer

Betreuer

Daniela Reschke

Markus Alterauge

Team

Johanna Mayer

Julian Schoemaker

Inhaltsverzeichnis

1. Implementierung

In folgendem Dokument wird die Implementierung von Meet & Remind dokumentiert. Hierbei wird darauf eingegangen, wo sich die Komponenten befinden, wie sie aufgebaut sind und welche Anforderungen bestehen. Außerdem wird aufgezeigt, wo sich das Alleinstellungsmerkmal und die Server- und Clientseitige Anwendungslogik befinden.

Sowohl Client, als auch Server befinden sich außerhalb des Meilenstein 3 Ordners, da wir diese Teile des Systems in allen Teilen bearbeitet haben. Dadurch wird Redundanz im GitHub vermieden, welche entstehen würde, wenn in allen Meilenstein Ordnern Implementierungen zu finden wären.

2. Client

2.1. Installation des Client

Die Installationsdatei (APK-File) befindet sich in GitHub unter dem Ordner MS3.
Die Datei auf dem Android Smartphone oder Tablet öffnen. Ggf. muss in den Einstellungen die “Installation von Apps aus anderen Quellen” zugelassen werden. Danach kann die Datei installiert werden.

Der komplette Code befindet sich im GitHub unter dem Ordner “Client”. Über diesen Ordner kann ein Android Studio Projekt angelegt werden. Auch darüber ist eine Installation der App möglich, indem die App auf einem angeschlossenen Android Gerät ausgeführt wird. Die Nutzung eines Virtuellen Devices wird nicht empfohlen, da aufgrund der fehlenden Bluetooth Funktionen der Umfang hier minimiert ist.

2.2. Anforderungen an den Client

MinSdkVersion: 23 (Android 6.0)
Es wird empfohlen Android Geräte ab Version 7.0 zu verwenden, da damit getestet wurde.

Für die Verwendung der App müssen folgende Berechtigungen erlaubt werden:

- Bluetooth
  Für das Hinzufügen von Kontakten und der Erkennung der Nähe zu den Kontakten.
- GPS
  Für manche Methodenaufrufe von Bluetooth muss die GPS Berechtigung vorliegen.
- Internet
  Für GET und POST Requests an den Heroku Server.

2.3. Codestruktur

Activities sind für die Benutzerinteraktionen zuständig. Zu ihnen gibt es jeweils ein Layout im Resource Ordner.

- ContactList - activity_contact_list.xml
- ContactDetail - activity_contact_detail.xml
- ReminderDetail - activity_reminder_detail.xml
- AddContact - activity_add_contact.xml
- Suggestion - activity_suggestion.xml

Adapter sind die “Hilfsklassen” für ListViews und ähneln sich in ihrer Grundstruktur. Das Aussehen der Items innerhalb der ListView wird im jeweiligen layout-file im Resource Ordner festgelegt.

- BluetoothDevicesAdapter - adapter_devices_view.xml
- AdapterContactList - contact_item.xml
- AdapterLabelList - label_item.xml
- AdapterReminderList - remind_item.xml

Die Java Klasse **BingWebSearch** ist größtenteils aus der API Dokumentation von Microsoft Azure [1].
Sie ist notwendig, um die JSON Datei aus der API mittels eines Links und dem Ocp-Apim-Subscription-Key zu bekommen.

Die Klasse **BluetoothConnectionService** erweitert die Standard Klasse IntentService und überprüft asynchron, ob ein Bluetooth Gerät aus der Liste der gekoppelten Geräte in der Nähe ist. Hier befinden sich auch die Klassen **ClientClass** und **ServerClass**, die für die Socket-Bluetooth Verbindung zuständig sind.

2.4. Implementiertes Alleinstellungsmerkmal

Das Alleinstellungsmerkmal ist die Verwendung von Bluetooth für die Erkennung, dass sich zwei Kontakte in der Nähe befinden. In der clientseitigen Anwendungslogik wurde dies implementiert. Außerdem bietet die App selbst auch die Standardeinstellungen von Bluetooth, also Bluetooth aktivieren, Sichtbarkeit aktivieren, Geräte in der Nähe scannen und sich mit einem Gerät koppeln.

2.5. Anwendungslogik

Die Clientseitige Anwendungslogik besteht darin, dass eine Benachrichtigung an den Benutzer geschickt wird, wenn ein Bluetooth Kontakt, der auch die App besitzt, sich in der Nähe befindet.

Zunächst ist es wichtig, dass beide Benutzer die App besitzen. Danach kann in der App der Kontakt mittels Bluetooth zu den gekoppelten Geräten hinzugefügt werden.

Für die Erkennung der Nähe der Bluetooth Geräte müssen Server und Client Sockets erstellt werden. Der Server Socket wird beim Aufruf der App erzeugt und das eigene Gerät ist für Anfragen vom Client bereit.

Damit die Verbindung in einem Hintergrundprozess ausgelagert ist, musste ein IntentService erzeugt werden. Somit ist es auch möglich die Anwendungslogik auszuführen, wenn die App geschlossen wird.
Im Prozess wird die Liste der gekoppelten Geräte durchiteriert und für das erste Gerät der Liste ein Client erstellt. Server und Client versuchen sich zu verbinden. Für einen garantierten Statuswechsel wird der Thread für 15 sekunden schlafen gelegt, danach wird der Status überprüft. Ist die Verbindung fehlgeschlagen, wird mit dem nächsten Device derselbe Prozess durchlaufen. Falls die Verbindung erfolgreich war, wird die Benachrichtigung auf dem eigenen Gerät angezeigt.

2.6. Implementierte Use Cases

Kontakt zu den gekoppelten Geräten hinzufügen
Auf der Startseite den Button mit dem Plus Icon klicken. Man wird auf die “Kontakt hinzufügen” Seite weitergeleitet. Der Benutzer muss Bluetooth aktivieren. Danach auf den Button “Geräte suchen” klicken. Die Geräte in der Nähe, sowie die bereits gekoppelten Geräte werden angezeigt.
Der Kontakt, der zu den gekoppelten Geräten hinzugefügt werden soll, muss sicherstellen, dass Bluetooth und Sichtbarkeit aktiviert sind. Dann sollte dieses Gerät in der Liste der Geräte in der Nähe erscheinen. Beim Klick auf das gewünschte Gerät wird die Bluetooth Kopplung eingeleitet. Weitere Anweisungen befolgen.

Erinnerungen ansehen
Auf der Startseite das Element mit der ID “54:27:58:24:B2:7F” anklicken. Danach wird man auf die Seite weitergeleitet, die alle erstellten Erinnerungen zu 54:27:58:24:B2:7F beinhaltet. Beim Klick auf eine beliebige Erinnerung wird man auf die “Erinnerung bearbeiten” Seite weitergeleitet und sieht den Titel, sowie die gesamte Beschreibung.

Erinnerung erstellen
Ausgang ist die Seite aller Erinnerungen zu einem Kontakt. Beim Klick auf den Button mit dem Plus Icon wird man auf die “Erinnerung erstellen” Seite weitergeleitet. Hier in das Feld Titel klicken und einen Titel mit der Tastatur eingeben. Dasselbe für die Beschreibung. Für ein Thema der Erinnerung ein Item anklicken und es erscheint ein blaues Häkchen.
Wenn alle Angaben gemacht wurden, auf den grünen Button mit dem Häkchen Icon klicken. Die Erinnerung wurde erstellt und dem Server übergeben. Man wird auf die Seite der erstellten Erinnerungen weitergeleitet, jedoch wird noch nicht die neu erstellte Erinnerung angezeigt. Hierzu muss auf die Startseite zurückgekehrt werden und der Kontakt mit der ID “54:27:58:24:B2:7F” erneut angeklickt werden, damit die Seite der Erinnerungen neu geladen wird.

Erinnerung bearbeiten
Ausgang ist die Seite aller Erinnerungen zu einem Kontakt. Beim Klick auf eine Erinnerung gelangt man auf die Seite “Erinnerung bearbeiten”. Danach ist der Ablauf wie beim oberen Punkt “Erinnerung erstellen”.

Themenvorschlag ansehen
Auf der Startseite befindet sich beim Kontakt mit der ID “54:27:58:24:B2:7F” ein blaues Symbol. Beim ersten Klick auf das Symbol, erkennt das System ein Klick auf das gesamte Item. Dies ist ein Fehler. Beim Zurückkehren auf die Startseite und einem erneuten Klick auf das blaue Symbol, wird man auf die richtige Seite, dem Themenvorschlag, weitergeleitet. Hier wird ein Artikel der News Bing Search API geladen. Je nach Uhrzeit und Thema gibt es neue Artikel und somit neue URLs. Manche URLs können direkt in die App geladen werden, aber bei manchen wird eine neue Internetseite im Browser geöffnet. Außerdem sind nicht alle URLs für eine Webansicht ausgelegt, jedoch haben wir darauf wenig Einfluss.
Das Thema des Themenvorschlag ergibt sich aus der Anwendungslogik des Servers, der die Themen der Erinnerungen matched.

2.7. Abweichungen von Meilenstein 2

2.7.1. Abweichungen der Anwendungslogik

Die Benachrichtigung bei der Erkennung der Nähe wird verschickt, egal ob zu dem Kontakt eine Erinnerung erstellt wurde oder nicht. Außerdem zeigt die Benachrichtigung nicht den Titel und die Beschreibung einer Erinnerung an.
Für diese Funktionen müsste aus der Liste der gekoppelten Geräte eine neue Liste erstellt werden, in der die Kontakte, mit denen man die App benutzen möchte, enthalten sind. Danach müssten noch zahlreiche Informationen zu den IDs der Benutzer zum IntentService übergeben werden, sowie eine GET Abfrage für die Informationen der Erinnerung ausgeführt werden.

2.7.2. Abweichungen im Layout

Im Design war geplant auf der Startseite in der oberen Leiste 3 Menüpunkte anzuzeigen. Hier sollten die Menüpunkte “Impressum” und “Datenschutz” zu finden sein.
Da aber eine Toolbar zur Implementierung genutzt wurde, ist dies nicht einfach zu realisieren. In Hinblick auf Optimierung der Anwendungslogik und Qualität des Codes wurde das Menü nicht implementiert, auch weil es keinen Mehrwert für den Client bildet.

3. Server

3.1. Installation des Server

Der Server befinden sich auf oberster Ebene im GitHub im Ordner Server. Da es sich um einen NodeJS Server handelt, muss in diesem Ordner der Befehl “npm install” ausgeführt werden. Dadurch werden die Node Module angelegt und installiert, die zum Aufruf benötigt werden. Der Server kann dann durch den Aufruf “node server.js” im Server-Ordner gestartet werden. Es sollte eine Nachricht zur Verfügbarkeit und eine Bestätigung des Ausführens der Anwendungslogik erscheinen.

Der Client greift jedoch auf den in Heroku liegenden Code zu, sodass Codeänderungen auf dem Server nicht von der App erkennbar sind. Die URL zu Heroku lautet https://eisws1819mayerschoemaker.herokuapp.com/. Dies wurde implementiert um die Verteiltheit unabhängig von Geräten zu gewährleisten und sicherzustellen, dass nur release-reife Implementierungen des Servers mit dem Client genutzt werden.

Die Dokumentation des Server-Codes sollte auch über eine JavaDoc bzw. in diesem Fall JSDoc automatisiert werden. Dafür wurden bereits ausgiebige Kommentare in den jeweiligen JavaScript Dateien eingefügt, die zusätzlich zu dieser Dokumentation betrachtet werden sollten.

3.2. Codestruktur

3.2.1. Server.js

Die Server.js bildet die Hauptdatei des Servers. Mit ihr wird sowohl der Server, als auch die serverseitige Anwendungslogik aufgerufen. Dies geschieht mit hilfe der Node Moduls “express”. Außerdem sind hier die Datenhaltung, die Anwendungslogik und die Routen zu den REST Ressourcen initialisiert.

Die serverseitige Anwendungslogik ist im Server ist einerseits so hinterlegt, dass sie ausgeführt wird, sobald man den Server startet. Andererseits ist aber auch schon ein Cronjob eingebunden, der die serverseitige Anwendungslogik dann später einmalig pro Nacht ausführt. Dies wird über das Node Modul “node-schedule” gewährleistet.

3.2.2. topics.js

Diese Datei weist nur eine einzige REST Ressource mit einer Methode auf. Die vordefinierten Topics, die als Label in den Erinnerungen gesetzt werden können, können hier per GET gezogen werden. Hierfür wird sowohl die Initialisierung von FIrebase als auch von Express benötigt.

3.2.3. users.js

Über die users.js wird der größte Teil der REST Ressourcen angesprochen. Hier finden die Module für Firebase, Express und die JSON Schema Validierung Verwendung. Implementiert wurden die in Meilenstein 2 genannten Verben unter Berücksichtigung der nötigen Datenformate und Firebase Abfragen. Zusätzlich wurden mittels Setzen der erstellten/angesprochen URI in die “location” im Header bei POST und PUT gewährleistet, dass die Ebene des “Hypermedia” im REST erreicht wurde.

Error-Handling wurde eingebunden, um Clientseitig Fehlermeldungen ausgeben zu können, die auch zum Debugging genutzt werden können. Dies trägt, zusätzlich zur JSON-Schema-Validierung, dazu bei, die Datenhaltung konsistent zu halten.

3.2.4. json-schema.json

Zur Überprüfung der richtigen Übermittlung der Reminder an das System wurde eine JSON-Schema-Validierung eingebunden. Diese gibt vor, welche Werte beim POST eines Reminders benötigt werden, welche optional sind und welchen Datentyp sie aufweisen müssen. Diese Validierung sollte später für alle JSON eingebunden werden, da die Validierung hierfür sonst unzureichend ist.

3.2.5. applicationlogic.js

In der application.js befindet sich die Serverseitige Anwendungslogik. Die Hauptmethode ist die Methode setMainTopic() mit den beiden Methoden getAllLabelsInFB() und getLabelCount().

In getAllLabelsInFB() werden die Daten aller User aus Firebase gezogen und in ein Array gesetzt. Hier befinden sich dann die Daten zu den Usern, ihren Contacts und den zugehörigen Remindern. Dies geschieht über die drei einzelnen Methoden, die jeweils die vorangegangenen Arrays nutzen, um die Struktur zu befüllen. Folgende Struktur geht aus der Methode getAllLabelsInFB() hervor:

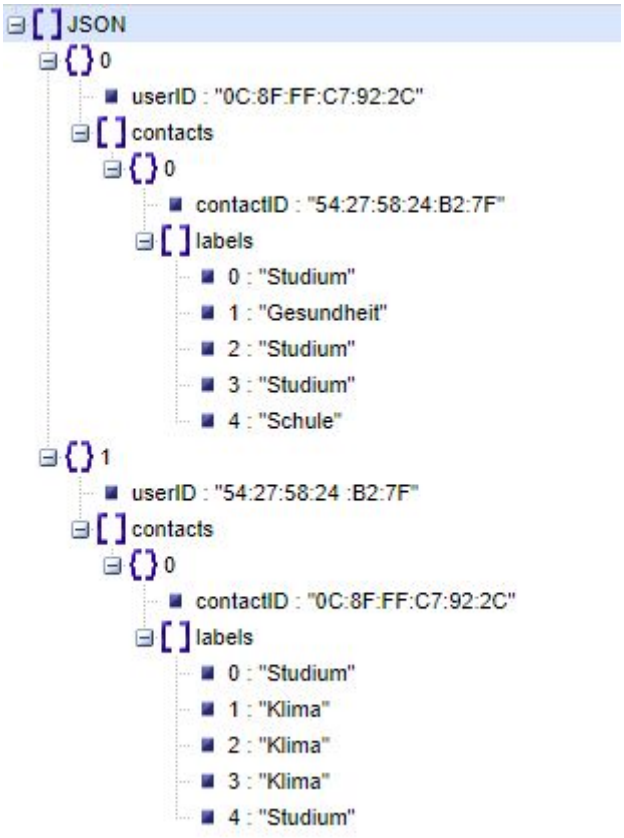

Abbildung 1: JSON der gesetzten Labels nach User und deren Contacts

Nun sind alle Labels des Users und seines Contacts klar, sodass dieses JSON darauf analysiert werden kann, welches Label bei beiden summiert am öftesten vorkommt.

Dafür werden in der getLabelCount() alle User und dessen Contact dieses JSON iteriert. Zunächst wird ermittelt, wie oft welches Label im User für einen Contact gesetzt wird.

Im Falle des Beispiels: {“Studium”: 3, “Gesundheit”: 1, “Schule”: 1}

Danach wird dann der Contact identifiziert, um auch von ihm die gesetzten Label für den User zu erhalten. Diese werden dann zu den ersten Anzahlen von Labels addiert, sodass ein Array von gematchten Labels entstanden ist.

Im Falle des Beispiels: {“Studium”: 5, “Gesundheit”: 1, “Schule”: 1, “Klima”: 3}

In diesem wird nun betrachtet, welches Label die höchste Anzahl hat. Dadurch kann ein String, im Beispiel “Studium”, mit der höchsten Anzahl ausgegeben werden, der dann genutzt wird, um in Firebase sowohl das Topic für "0C:8F:FF:C7:92:2C" als auch für "54:27:58:24:B2:7F" als Contact auf Studium setzen zu können.

3.3. Anwendungslogik

Das generierte “mainTopic” ist also eine Datenanreicherung ohne Nutzer-Interaktion, die im Client für die Funktionalität der Themenvorschläge werden.
Würde einer der beiden User, ausgehend vom vorherigen Absatz zur application.js, nun also für den jeweils anderen den Themenvorschlag im Client aufrufen, erhält er einen Artikel zum Thema “Studium”, da dies das Thema ist, für das beide gegenseitig am meisten Erinnerungen erstellt haben. Wichtig hierbei ist, dass immer die Erinnerungen beider betrachtet werden und somit das Attribut “topic” immer identisch ist.

Wie bereits im Abschnitt zur server.js beschrieben, ist die Anwendungslogik als ein Cronjob angedacht, sodass dieser Teils hoch-aufwändige Algorithmus in einer Zeit abgearbeitet werden kann, in der wenig bis keine Benutzer mit dem System interagieren.

3.4. Abweichungen von Meilenstein 2

In Meilenstein 2 wurden mehr REST Ressourcen/Methoden benannt, als letztendlich benötigt wurden. Einzelne Anfragen von Attributen in Ressourcen wurden durch Anfragen auf die gesamte Ressource umgesetzt. Dies ermöglicht es mehrere Attribute mit einer Anfrage zu erhalten und im System nutzen zu können.

4. Datenhaltung

Die Datenhaltung wurde über Googles Firebase Firestore, wie in Meilenstein 2 beschrieben, umgesetzt.

4.1. Datenstruktur

Die Datenstruktur in Firebase Firestore wurde bereits in Meilenstein 2 festgelegt. Zum Verständnis der Anwendungslogik ist es jedoch vorteilhaft, die Struktur hier abzubilden:

Datenstruktur in Firebase Firestore

Collections	**Documents**	**Collections**	**Documents**	**Collections**	**Documents**
topics	<Generierte ID>				
	Fields: name : “Schule”				
users	<BluetoothID>	contacts	<BluetoothID>	reminder	<Generierte ID>
			Fields: name : “Michael”, topic : “Studium”		**Fields:** title : "EIS", description : "Wie geht es weiter?", label : "Studium" priority : 1

(Bei den Felder-Werten (Attributwerten) handelt es sich um Beispiele)

4.2. Zusammenhang mit Anwendungslogik

Das Feld “topic” in dem Document des Contacts, neben dem sich auch der Name des Contacts befinden, wird durch die serverseitige Anwendungslogik angereicht. Will man die serverseitige Anwendungslogik testen, können die Labels der einzelnen Reminder verändert werden oder das topic des Contacts auf irgendeinen String gesetzt werden. Die Anwendungslogik geht dann so vor, dass sie die Labels der Reminder sowohl für den User, als auch seinen Gegenüber, analysiert und dann das topic der beiden Kommunikationspartner dementsprechend anpasst. Wie diese Anreicherung funktional umgesetzt wurde, ist unter dem entsprechenden Absatz des Servers beschrieben.

4.3. Zugang zur Datenhaltung

Um das testen zu können, wurde ein Zugang angelegt, mit dem die Datenbank manipuliert werden kann. Die Zugangsdaten werden den beiden Mentoren per Mail zugesandt. Es sollten nur die Werte für die Felder “topic” und “label” in den Contacts angepasst werden. Es wird darum gebeten keine weiteren Daten zu manipulieren, um einen reibungslosen Prototypen zu gewährleisten.

4.4. Abweichungen von Meilenstein 2

Die in Meilenstein 2 Absatz 15.3. angedachten Datensätze sind leicht verändert worden. Die oben stehende Tabelle der Datenstruktur hat nun einen größeren Fokus auf die User und dessen Kontakte. Die Abhängigkeiten sind nun besser gekapselt, da sie einen stringenten Verlauf haben:

users - UserID - contacts - ContactID - reminder - ReminderID

Es gibt also immer nur eine Abhängigkeit für jede Collection.

5. Quellenverzeichnis

1. Bing News Search API: https://docs.microsoft.com/en-us/azure/cognitive-services/bing-news-search/java. (abgerufen am 20.01.2019)