Technology
Arts Sciences
TH Köln

Entwicklungsprojekt interaktive Systeme

Wintersemester 2018/2019

Fazit

Meet & Remind

Dozenten

Prof. Dr. Gerhard Hartmann

Prof. Dr. Kristian Fischer

Betreuer

Daniela Reschke

Markus Alterauge

Team

Johanna Mayer

Julian Schoemaker

Inhaltsverzeichnis

1. Schwierigkeiten und Herausforderungen **3**
1.1. Client 3
1.1.1. Erkennung der Nähe bei Klick auf gekoppeltes Gerät 3
1.1.2. Erkennung der Nähe in einem Hintergrundprozess 3
1.1.3. API und WebView 3
1.1.4. GET Abfrage 3
1.1.5. Android Studio Layout 4
1.2. Server 4

2. Diskussion des Zielerreichungsgrades **5**
2.1. Client 5
2.2. Server 5

3. Ausblick **6**
3.1. Release-fähige Version 6
3.2. Wünschenswerte Funktionen 6

4. Quellenverzeichnis **7**

1. Schwierigkeiten und Herausforderungen

1.1. Client

1.1.1. Erkennung der Nähe bei Klick auf gekoppeltes Gerät

Grundlage für die Verbindung über Bluetooth war ein Google Sample [1].
Das Beispiel enthält viele komplexe Funktionen und konnte vereinfacht werden, da in unserem System die Bluetooth Verbindung nur zum Zweck hat, die Nähe zu erkennen. Daten zwischen den Geräten müssen nicht verschickt werden. Generell agiert ein Gerät als Server und eins als Client. Realisiert wird dies über BluetoothSockets.
Zum Testen wurde in der AddContact activity ein OnItemClickListener, sowie ein Statusfeld eingebaut.

1.1.2. Erkennung der Nähe in einem Hintergrundprozess

Nun sollte die Erkennung der Nähe nicht über ein aktives Klicken vom Benutzer getriggert werden, sondern permanent im Hintergrund laufen – auch wenn die App von einem Benutzer geschlossen wird. Es war schwierig herauszufinden, welche Methode sich dafür eignet, weswegen sich erstmal durch die Dokumentationen von AsyncTask, Service, IntentService und JobScheduler gelesen wurde. Zum Schluss ergab sich, dass ein IntentService die richtige Wahl für die Anforderungen war. Es wurde eine neue Klasse erstellt, die von IntentService erbt. Danach musste der Code umstrukturiert werden, damit er in den IntenService integriert werden konnte. Die Abhängigkeiten und Erreichbarkeiten innerhalb des Android Studio Projektes sind eingeschränkt, weshalb man oft in Sackgassen landete.
Zum Schluss wurde es so gelöst, dass im Hintergrundprozess die Liste der gekoppelten Geräte neu generiert wurde und danach für jedes Gerät die Überprüfung der Nähe erfolgt.

1.1.3. API und WebView

Die Dokumentation für die Bing News API war hilfreich, aber an manchen Stellen etwas ungenau. Als diese so übernommen wurde und anhand der main-Methode ausgeführt wurde, hat alles geklappt. Eine JSON mit verschiedenen Metadaten und den verschiedenen URLs wurde ausgegeben. Nun sollte diese aber nicht als main ausgeführt werden, sondern in der Suggestion Activity integriert werden. Also wurde die main zu einer getJSON Methode umgeschrieben und aufgerufen. Jedoch kam dann ein Fehler. Diese Abfrage hat für den normalen UI Thread zu lange gedauert, weshalb sie gestoppt wurde. Die Lösung bestand darin, einen AsyncTask einzubauen, der diese Abfrage in einen seperaten Thread auslagert. Nun musste aus der JSON Datei noch die richtige URL extrahiert werden und diese in eine WebView geladen werden.

1.1.4. GET Abfrage

Zunächst musste recherchiert werden, welche Mittel es gibt, HTTP Abfragen über Android Studio zu machen. Varianten wie Spring, Retrofit oder android-async-http library haben nicht funktioniert. OkHttp3 hat dann letztendlich funktioniert und kann GET, POST und PUT Befehle ausführen.

Die erste GET Abfrage sollte die Kontakte von “0C:8F:FF:C7:92:2C” zurückliefern. Die Struktur der JSON Datei ist etwas unüblich, da die IDs nicht innerhalb eines Arrays abgelegt sind, sondern einzeln direkt abgelegt sind. Somit konnte eine ID nicht mit array[0] abgefragt werden. Stattdessen erreicht man die IDs mit einem Iterator und der Methode JSONObject.keys().

1.1.5. Android Studio Layout

Insgesamt gab es bei der Implementierung des Layouts viele Aufgaben, die zunächst Leicht erschienen, aber dann doch größere Schwierigkeiten bereiteten.
Beispielsweise die Icons in den verschiedenen Listviews sind erst bei dem 2. Klick anklickbar. Zunächst wurde angenommen, dass die touchable Area der Icons zu klein ist, weshalb diese vergrößert wurde. Aber auch dies löste das Problem nicht.
Außerdem sollte bei der Auswahl eines Themas für eine Erinnerung ein blaues Häkchen erscheinen (statt Radio Buttons). Zunächst wurde mit einer boolean Variable eine Single Choice Liste simuliert. Sobald aber die Liste länger wurde, wurden auf einmal das Häkchen doppelt angezeigt. Nach weiterer Recherche wurde eine Variante gefunden, bei der bereits im Layout der ListView der Choice Mode festgelegt wird. Danach wird dann eine Xml Datei erstellt, die das Design (also das blaue Häkchen) festlegt und bei der OnItemClick Methode zum Einsatz kommt. Zunächst schien das die Lösung des Problems zu sein, jedoch gibt es Momente, in denen das Design nicht übernommen wird und ein grüner Kasten stattdessen angezeigt wird.

1.2. Server

Beim Server und der eng gekoppelten Datenhaltung bestand zunächst keine Schwierigkeit. Durch das Aufstellen der Ressourcen und damit verbundenen REST Methoden in Meilenstein 2 waren die zu implementierenden Funktionen klar.
Probleme tauchten hier nur bei der Verbindung zur Datenhaltung Firebase auf. Sie arbeitet in vielen Fällen mit Promises, die bereits aus dem Modul Web-basierte Anwendungen 2 bekannt waren. Trotzdem kam es beim Aufbau aufgrund der Komplexität immer wieder zu Herausforderungen. Die Firebase Firestore Dokumentation [2] liefert hier jedoch ausreichend Codebeispiele.

Auch bei der Anwendungslogik besteht der erste Teil aus dem Ziehen der Daten aus Firebase. Hier wäre mit normaler Implementierung von Promises eine große und komplizierte Verschachtelung nötig gewesen. Bei einer Recherche hat sich jedoch die Funktionalität über async/await [3] als gute Lösung für dieses Problem ergeben. Über async/await ist es möglich Promises in einer prozeduralen Weise abzuarbeiten und somit die Lesbarkeit des Codes zu verbessern. Einzelne Funktionen, die ein Promise liefern wurden modular aufgebaut und werden dann mittels await aufgerufen, was bedeutet, dass gewartet wird, bis das Promise einen return-Wert zurückliefert.
Im zweiten Teil der Anwendungslogik war dann das logische Abarbeiten des erstellten Daten-Arrays die größte Herausforderung. In dieser Phase wurden viele Skizzen erstellt, um die Abläufe zu visualisieren.

2. Diskussion des Zielerreichungsgrades

2.1. Client

Die wichtigsten Use Cases wurden im Client implementiert: Erinnerungen ansehen, erstellen, bearbeiten, sowie eine Benachrichtigung bei der Nähe eines Kontakts erhalten. Damit diese Use Cases garantiert zu implementieren waren, wurde immer von denselben Kontakten ausgegangen. Somit konnten auch die GET und POST Anfragen an den Server sicher ausgeführt werden. Weitere Use Cases wie “Erinnerung löschen” würden nach dem selben Schema ablaufen, jedoch wurde sich darauf konzentriert, dass die bestehenden Use Cases problemlos funktionieren und der Code aufgeräumt ist.

Bei der Anwendungslogik der Erkennung der Nähe funktioniert gut; es fehlen lediglich die zu übermittelnden Daten. Diese können jedoch auch erst ermittelt werden, sobald die Dummy Kontakte durch die Echten ersetzt wurden. Dies wäre der nächste Schritt in der Implementierung, der jedoch viele Funktionen voraussetzt:
Zunächst müsste der Benutzer einen Kontakt aus der Liste der gekoppelten Geräte hinzufügen (POST). Danach müsste sichergestellt werden, dass beide Kontakte sich gegenseitig hinzugefügt haben. Dann müssten im Code über Intents bei jedem Klick eines Items die IDs übergeben werden, um die richtigen GET und POST Befehle auszuführen. Erst danach könnte in der Anwendungslogik die Liste der Kontakte (statt der Liste der gekoppelten Geräte) durchiteriert werden und geprüft werden, ob zu einem Kontakt eine Erinnerung erstellt wurde. Wenn ja, können die Informationen der Erinnerung in der Benachrichtigung angezeigt werden. Außerdem würde auch erst hier die Prioritätszahl einer Erinnerung eine Rolle spielen.

2.2. Server

Der Server beinhaltet alle angedachten Funktionen. Die Implementierung der REST Methoden und der Anwendungslogik konnte, wie in Meilenstein 2 angedacht, umgesetzt werden. Damit wurden die nötigen Voraussetzungen geschaffen, die App mit allen Daten befüllen und die Daten auf die Datenbank schreiben zu können. Zusätzlich konnten noch Fehlermeldungen (Error-handling) und eine beispielhafte JSON-Schema-Validierung implementiert werden, sodass erste Schritte für eine Qualitätssicherung gemacht wurden.

3. Ausblick

3.1. Release-fähige Version

Für eine Release-fähige Version, die im Google Play Store angeboten werden kann, werden noch ein paar Dinge benötigt. Aus Sicht des Projektes wäre es jedoch lohnenswert möglichst zeitnah einen Prototypen zu veröffentlichen, um die Resonanz der Zielgruppe aufnehmen zu können.
Damit diese Version den nötigen Funktionsumfang hat, werden vor allem noch weitere Methoden auf dem Client benötigt. Hier sind noch nicht alle Server Methoden (REST Verben) integriert. Zum Beispiel ist es aktuell noch nicht möglich Kontakte hinzuzufügen oder Erinnerungen zu löschen.
Außerdem müsste die Clientseitige Anwendungslogik noch weiter mit Funktionen versehen werden, um das Alleinstellungsmerkmal – nämlich die Benachrichtigung zu einer Erinnerung auf dem mobilen Gerät – in vollem Umfang nutzen zu können.
Abschließend sollten auch die kleinen Layoutanpassungen implementiert werden, sodass das in Meilenstein 2 erstellte User Interface aus dem dem high-fidelity Prototype hervorgeht.

3.2. Wünschenswerte Funktionen

Ein großer Vorteil unseres Systems besteht darin, dass Firebase integriert wurde. Ein weiterer Nutzen kann daraus gezogen werden, die Analyse-Tools von Firebase zu nutzen, um z.B. erkennen zu können, zu welcher Zeit die meisten Benutzer mit der App interagieren. Somit ist es uns möglich zu analysieren, wann wir den Cronjob der Serverseitigen Anwendungslogik laufen lassen können, ohne hohe Last auf dem System zu verursachen.

4. Quellenverzeichnis

1. Google Beispiel Bluetooth Chat: https://github.com/googlesamples/android-BluetoothChat/blob/master/Application/src/main/java/com/example/android/bluetoothchat/BluetoothChatService.java (abgerufen am 20.01.2019)
2. Google Firestore: https://firebase.google.com/docs/firestore/ (abgerufen am 20.01.2019)
3. Async-await: https://javascript.info/async-await (abgerufen am 20.01.2019)