Entwicklung interaktiver Systeme
-
Aquaapp
-
Implementationsdokumentation

Moritz Müller

16. Januar 2017

Inhaltsverzeichnis

1 Einleitung

Da sich Johannes nach der Abgabe der Projekt-Dokumentation dazu entschieden hat, das Projekt abzubrechen, habe ich (Moritz) das Projekt alleine zuende gemacht. In diesem Dokument werde ich zunächst die Änderungen bzgl. der Systemdokumentation und dem Design erläutern und begründen und danach den Installations-Vorgang dokumentieren sowie genauer auf die Implementierung eingehen.

2 Systemdokumentation

2.1 Iteration des Architekturmodells

Die Systemarchitektur hat sich während der Programmierung ein wenig geändert bzw. erweitert. Auf diese Änderungen werde ich nun genauer eingehen. In Abbildung 2.1 befindet sich das aktuelle Modell der Architektur.

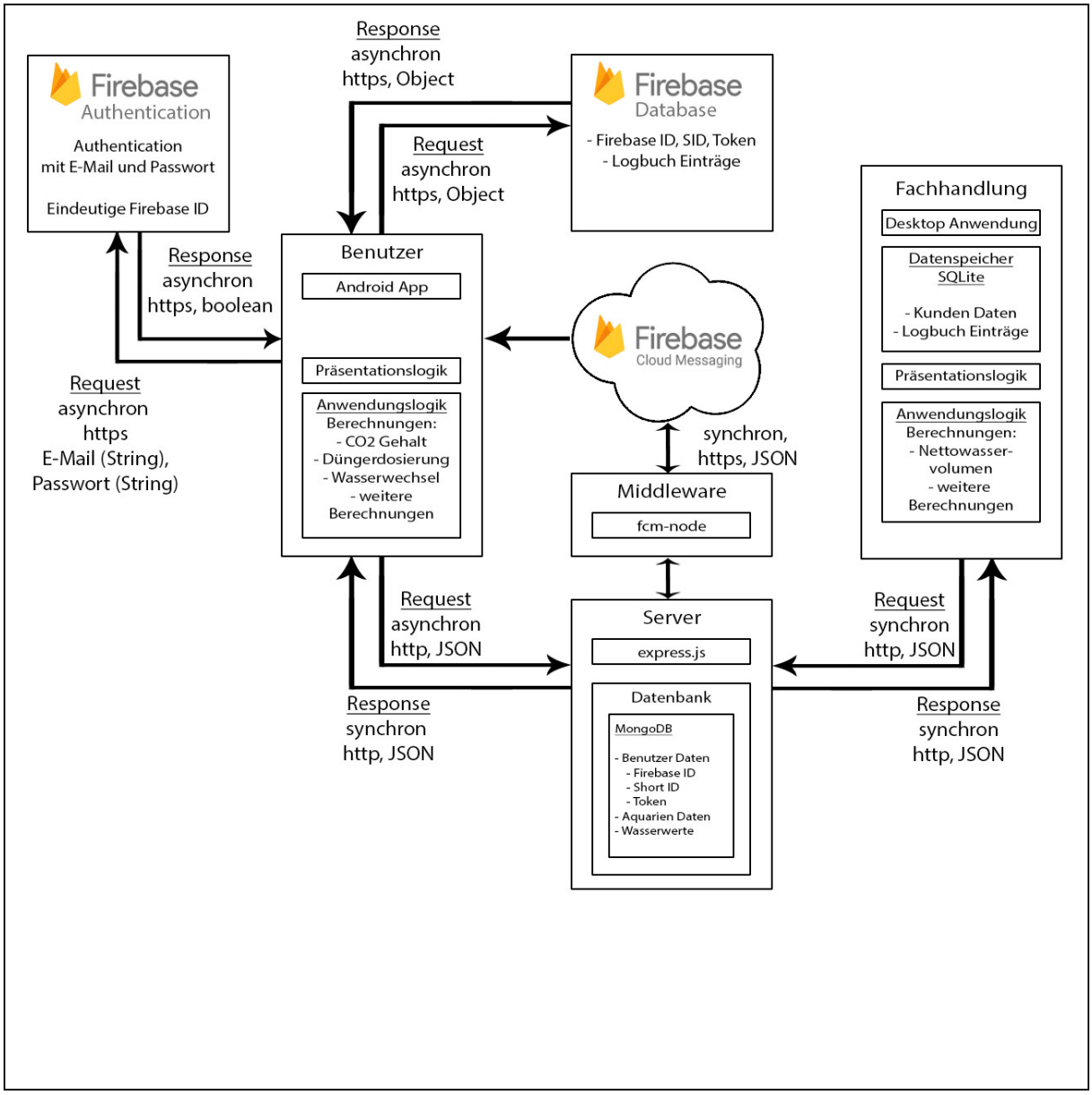

Abbildung 2.1: Architekturmodell

2.1.1 Firebase Authentication

Neu dazu gekommen ist u.a. die Firebase Authentication, womit ich einen Login ermögliche. Da der Token, welcher für das Firebase Cloud Messaging generiert wird, zum Beispiel bei einer Neu-Installation neu generiert wird, eignet sich dieser nicht um Daten einem Benutzer zuordnen zu können. Deshalb war die Umsetzung eines Logins nötig, welcher mit Hilfe der Firebase Authentication recht einfach umzusetzen war. Zur Anmeldung benötigt man lediglich eine E-Mail Adresse und ein Passwort. Wenn man sich registriert hat, bekommt man eine eindeutige 28-stellige ID zugewiesen. Diese ID kann für die Zuordnung der Aquarium- und Wasserwerte-Daten benutzt werden. Neben dieser ID wird noch eine zusätzliche "Short-ID" generiert, welche nur aus vier Zeichen besteht. Diese dient dazu, dass der Benutzer sie an den Fachhändler weiter geben kann, sodass er den Kunden zu seinem System hinzufügen kann. Somit muss der Benutzer nicht eine 28-stellige ID weiter geben.

2.1.2 Firebase Database

Ebenfalls neu hinzugekommen ist die Firebase Database. Diese bietet neben dem Server und dem lokalen Speicher eine zusätzliche Möglichkeit Daten zu speichern. Die Idee dahinter war, dass Daten, die nur der Benutzer benötigt, wie zum Beispiel sein Logbuch, in der Firebase Database gespeichert werden und Daten, auf die sowohl Benutzer als auch Fachhandlung zugreifen können, wie zum Beispiel die Aquarium-Daten, auf dem Server gespeichert werden. Der Vorteil daran ist, dass es sehr leicht zu implementieren war und man als Administrator im Dashboard von Firebase einen guten Überblick über die Daten hat und diese auch bearbeiten kann. Ansonsten hätte aber auch nichts dagegen gesprochen, alle Daten auf dem Server zu speichern.

2.1.3 Datenspeicher im Allgemeinen

Ein weiterer Unterschied zum vorherigen Modell ist, dass ich den Datenspeicher beim Benutzer entfernt habe. Dieser wird durch die Firebase Database nicht mehr benötigt. Man hätte dort zwar auch eine SQLite Datenbank verwenden können, allerdings wäre dadurch unnötiger Speicher angefallen. Bei der Fachhandlung habe ich eine SQLite Datenbank für die Logbuch-Einträge und Kunden verwendet, da es meines Wissens nach bis jetzt noch nicht möglich ist Firebase für eine Desktop Anwendung zu verwenden. Bei den jeweiligen Datenspeichern habe ich noch zusätzlich die Daten mit bei geschrieben, die jeweils gespeichert werden.

2.2 Anwendungslogik

2.2.1 Benutzer App

Bei der App haben wir wie geplant verschiedene Berechnungen als Anwendungslogik implementiert. Dazu gehören folgende Berechnungen:

- Volumen des Aquariums
- Netto-Wasser-Volumen im Aquarium

- Zielgerichteter Wasserwechsel
- Düngerdosierung
- CO2-Gehalt

Eine Berechnung für den täglichen Nährstoffverbrauch habe ich für die Benutzer App nicht direkt implementiert, allerdings habe ich für jeden Wasserwert einen Graph implementiert, in dem man den Verlauf des Wertes aus den verschiedenen Messungen sehen kann. Außerdem sieht man gleichzeitig den Durchschnittswert aller anderen App-Nutzer angezeigt, um einen Vergleich haben zu können.

2.2.2 Fachhändler Anwendung

Die Fachhandlung hat ebenfalls die Berechnungen für das Volumen, das Netto-Wasser-Volumen und die CO2 Berechnung zur Verfügung. Berechnungen für den zielgerichteten Wasserwechsel und die Düngerdosierung waren hier nicht erforderlich.

2.2.3 Server

Als Anwendungslogik für den Server war geplant, den täglichen Nährstoffverbrauch anhand von steigender Masse an Daten berechnen zu können. Aus Zeitgründen hatte ich das bis zum Tag der Abgabe noch nicht umgesetzt gehabt. Durch ein Tipp von Herrn Proffessor Hartmann bei der Code Inspektion habe ich mich dann aber doch noch dazu entschieden, diese Berechnung umzusetzen. Allerdings konnte ich sie nicht mehr sinnvoll in einen der Clients einbauen. Somit ist es jetzt möglich diese Berechnung auf dem Server ausführen zu lassen, in dem man die Route /naehrstoff-verbrauch aufruft. Die Ausgabe findet dann im Browser als HTML Seite statt. Herr Professor Hartmann hatte mir den Tipp gegeben, das ganze mit der Extrapolation zu lösen. Da das Ziel aber nicht direkt war eine Prognose für die Zukunft zu machen, sondern den täglichen Verbrauch zu ermitteln, hat sich die Interpolation besser geeignet. Dabei geht es darum, dass man einen Wert zwischen zwei gegebenen Werten ungefähr berechnen kann. Somit habe ich das so programmiert, dass immer zwei Wassermessungen miteinander verglichen werden. Dabei wird die Differenz der Tage sowie der Verbrauch des Nährstoffes berücksichtigt. Nach der Anwendung der Formel hat man den ungefähren Verbrauch nach einem Tag. Wenn man das mit jedem möglichem Mess-Paar macht und die Ergebnisse zusammen rechnet und das dann durch alle teilt, bekommt man einen durchschnittlichen Verbrauch des Nährstoffes pro Tag. Dabei gilt: Je mehr Wasserwerte durch Benutzer und Fachhandlungen eingetragen werden, desto genauer wird das Ergebnis dieser Berechnung. Unter der oben angegebenen Route habe ich das ganze erstmal nur für den CO2-Gehalt umgesetzt, man könnte es aber genau so gut auf alle anderen Nährstoffe anwenden.

2.3 Architekturmerkmale

2.3.1 Ressourcen

Bei den Ressourcen gab es kleine Änderungen an den jeweiligen Eigenschaften. Diese werde ich hier vorstellen.

Benutzer

Beim Benutzer werden auf dem Server nur noch die beiden IDs und der Token gespeichert. Vorname, Nachname und Geburtsdatum werden nur noch lokal bei der Fachhandlung gespeichert. Diese Daten muss der Benutzer dann angeben, wenn er sich bei der Fachhandlung registrieren lässt. Die Adressinformationen habe ich nicht unbedingt als nötig empfunden und nicht implementiert.

```
user{
    uid: String,
    sid: String
    token: String
}
```

Virtuelles Aquarium

Beim Aquarium sind die Werte Glasstärke, Kieshöhe und Füllstanddifferenz hinzugekommen. Diese helfen beim Berechnen des Netto-Wasser-Volumen. Da der Benutzer sowieso Maße wie Länge, Breite und Höhe angeben muss, finde ich es passend, dass die drei neuen Daten ebenfalls direkt beim Erstellen des Aquariums eingetragen werden und nicht erst bei Ausführen der Berechnung. Weggelassen habe ich hingegen Fische, Pflanzen und Geräte. Diese haben wir zwar am Anfang als wichtig empfunden, aber es stellte sich heraus, dass sie eigentlich nur zu einem geringen Anteil beim Netto-Wasser-Volumen beitragen. Bei den anderen Berechnungen haben sie keine Rolle gespielt, weswegen ich mir ein aufwendiges Design zum Eintragen der jeweiligen Sachen gespart habe.

```
aquarium{
    uid: String,
    bezeichnung: String,
    laenge: Number,
    breite: Number,
    hoehe: Number,
    glasstaerke: Number,
    kieshoehe: Number,
    fuellstanddifferenz: Number
}
```

Wasserwerte

Bei den Wasserwerten sind lediglich die Werte “von” und “datum” hinzugekommen. “Von” gibt an, wer die Wasserwerte eingetragen hat, also entweder der Benutzer oder die Fachhandlung. Somit kann zum Beispiel unterschieden werden, ob eine Benachrichtigung an den Benutzer geschickt werden muss, falls die Fachhandlung die Daten eingetragen hat. Das Datum hilft bei der Einordnung der Wasserwerte.

```
wasserwerte{
    uid: String,
    von: String,
    datum: String,
```

```
    no3: String,
    po3: String,
    eisen: String,
    kalium: String,
    co2: String,
    gh: String,
    kh: String,
    ph: String
}
```

2.4 Richardson Maturity Model

Wie bereits in der Projekt-Dokumentation von uns vermutet haben wir Level 2 des Richardson Maturity Model erreicht, da wir die üblichen Standard-Methoden wie GET, POST, PUT und DELETE verwendet haben. Wir haben zwar nicht bei jeder Ressource alle vier Methoden benutzt, aber da wo es von Nutzen war haben wir die Methoden implementiert. Zum Beispiel war es nicht erforderlich die DELETE Mehtode für Benutzer zu implementieren, da nicht vorgesehen war, dass Benutzer gelöscht werden können. Level 3 des Modells konnte nicht erreicht werden, da kein Hypertext verwendet wurde und dieses auch nicht benötigt wurde.

3 Design Änderungen

Bei der App habe ich doch eine andere Struktur, als in den Detailed User Interfaces geplant war, umgesetzt. In der Projekt Dokumentation hatten wir uns entschieden, eine Tab-Struktur zu nehmen. Nach erneutem Überlegen habe ich mich aber dagegen entschieden und eine Sidemenu-Struktur genommen, da eine Tab-Strukur eher typisch für iOS gewesen wäre und nicht für Android. Das lässt sich u.a. damit begründen, dass Android Geräte meistens unten am Bildschirm zusätzliche "Buttons" haben, wie zum Beispiel den "Zurück-Button". Wenn die Tabs wie geplant am unteren Bildschirmrand platziert worden wären, hätte es leicht passieren können, dass man versehentlich auf einen Tab geklickt hätte, obwohl man den "Zurück-Button" antippen wollte oder umgekehrt. Da wir aber beim konzeptuellen Modell sowieso eine Sidemenu-Struktur als Alternative in Betracht gezogen hatten, konnte ich mich daran orientieren.

4 Installationsdokumentation

Bevor ich genauer auf die Implementierung eingehen werde, erkläre ich kurz den Installationsvorgang, um die Anwendungen zum Laufen zu bekommen. Die Ordner befinden sich jeweils in unserem Repository (https://github.com/jkimmeyer/EISWS1617MuellerKimmeyer) unter "MS3/Implementation/".

4.1 Server

Den Server habe ich auf uberspace.de gehostet, sodass er durchgehen läuft und von jedem Gerät aus erreichbar ist, ohne dass erst immer der Server gestartet werden muss. Der Server ist unter (http://eis1617.lupus.uberspace.de/nodejs) zu erreichen. Als Ressource kann zum Beispiel "users" angegeben werden. Um den Server zu testen können Requests zum Beispiel von der Seite https://www.hurl.it/ abgeschickt werden. Dabei sollte man beachten, dass der Header "Content-Type" gleich "application/json" gesetzt werden sollte. Im Body sollte ein JSON String mit den entsprechenden Parametern stehen. Wenn Sie den Server auch lokal testen wollen, müssen Sie folgende Installationsschritte vornehmen:

Vorbereitungen

1. MongoDB installieren (https://www.mongodb.com/download-center#community)
2. Folgenden Ordner manuell erstellen: C:\data\db
3. Die Datei C:\Program Files\MongoDB\Server\3.4\bin\mongod.exe ausführen
4. Der Server läuft nun

Server Installation

1. Repository (https://github.com/jkimmeyer/EISWS1617MuellerKimmeyer) runterladen und in den Ordner /MS3/Implementation/server gehen
2. Die Konsole in diesem Ordner öffnen und folgenden Befehl eingeben: npm install
3. Die Datei server.js öffnen und Zeile 23 auskommentieren und Zeile 25 entfernen
4. Nun in der Konsole eingeben: node server.js
5. Jetzt können Sie im Browser die Adresse localhost:61000 aufrufen oder mit dem REST Client Ihrer Wahl Requests an den Server schicken

4.2 Benutzer Client - Android App

folgt

4.3 Fachhändler Client - Desktop Anwendung

Vorbereitung

1. Eclipse installieren (https://www.eclipse.org/downloads/)

Projekt importieren

1. Repository (https://github.com/jkimmeyer/EISWS1617MuellerKimmeyer) runterladen
2. Bei Eclipse unter File->Open Project from File System... und dann auf Directory... den Ordner "fachhandlungClient" in unserem Repository auswählen und auf "Finish" klicken
3. Unter Aquaapp/src/de.eis.muellerkimmeyer können nun die Java Dateien geöffnet werden
4. Anwendung kann jetzt gestartet werden

4.4 Testdaten

5 Implementierung

5.1 Benutzer App

Die App besteht aus vier verschiedenen Activities, und zwar für die Registrierung, für den Login, für den ersten Login und für Hauptbereich der App. Wenn man sich das erste mal einloggt, muss man erstmal seine Aquarium-Maße eintragen und danach kommt man zum Hauptbereich der App. Wie bereits erwähnt, hat die App eine Sidemenu-Struktur. Die App (nach dem Login) besteht quasi nur aus einer Activity, die verschiedene Fragmente lädt, je nachdem welche Button im Sidemenu angetippt wurde. Für jedes Fragment wurde eine eigene Klasse und eine eigene Layout Datei angelegt, sodass alle Fragmente individuell angepasst werden konnten. Lange Texte und Beschriftungen wurden außerdem in die strings.xml Datei ausgelagert, da dies zu einer besseren Übersicht führt. Kurze Beschriftungen wurden direkt in den Quellcode geschrieben. Aus Zeitgründen musste ich den Menüpunkt "Fachhandlung" auslassen, über den noch Kontakt zu der Fachhandlung hätte aufgenommen werden können. Da man dies aber auch mit einem einfachen Anruf erledigen kann, war das für mich nicht sehr wichtig.

5.2 Fachhandlung Client

Den Fachhandlung Client habe ich vom Design her wie geplant umgesetzt. Die Hauptklasse der Anwendung ist die AppFrame Klasse. Diese wird in der Main Klasse ein mal instanziiert. In der AppFrame Klasse werden die verschiedenen Ansichten, wie zum Beispiel die Kundenübersicht und die einzelnen Kunden Tabs erstellt. Es gibt Tabs für verschiedene Kunden, zwischen denen man schnell hin und her wechseln kann. Wenn man sich auf einem Tab von einem Kunden befindet wird dort das Accordion mit den Menüpunkten dargestellt. Da ich dafür keine optimale Java Library gefunden habe, habe ich mir die Klasse dafür selbst programmiert. Als Menüpunkte gibt es "Kundeninformationen" und "Wasseranalyse". Normalerweise waren noch zwei Punkte mehr geplant, diese konnten aber ebenfalls aus Zeitgründen nicht umgesetzt werden. Einer davon wäre die Kaufberatung gewesen. Da ich aber das Eintragen von Fischen, Pflanzen und Geräten sowieso in der App nicht umgesetzt hatte, konnte ich auch diesen Punkt bei der Fachhandlung auslassen. Der andere Punkt wäre die Problemanalyse gewesen. Hier hätte der Fachhändler einen Ablaufplan zur Fehlerfindung vorfinden können. Da ein Fachhändler dies aber auch ohne Hilfestellung drauf haben sollte, konnte ich diesen Punkt auch weg lassen.

5.3 Server

Der Server wurde soweit wie geplant umgesetzt. Den Teil zur Anwendungslogik habe ich ja bereits weiter oben beschrieben.

Tabellenverzeichnis

Abbildungsverzeichnis