Implementation
TravelCompats

Technology
Arts Sciences
TH Köln

Installationsanleitung TravelCompats

Server

1. **NodeJS** downloaden und installieren.
   - https://nodejs.org/en/download/
2. **MongoDB** downloaden und installieren.
   - https://www.mongodb.com/download-center
   a. **den Download starten (eine Registrierung ist nicht erforderlich)**
   b. **die heruntergeladene Datei entpacken und den Ordner "mongodb" in das Homeverzeichnis kopieren z.B. "nicoferdinand/mongodb"**
   c. **vom System-Terminal aus zum Installationsordner navigieren**
   d. **"cd mongodb/bin" eingeben**
      Unter Windows: MongoDB/Server/VERSIONSNUMMER/bin
   e. **eine Shell mit Admin-Rechten erstellen: "sudo bash" (Befehl im Terminal eingeben)**
   f. **einen neuen Ordner erstellen: "mkdir -p /data/db"**
   g. **die Rechte des Ordners ändern: "chmod 777 /data/db"**
   h. **die Shell wieder verlassen: "exit"**
   i. **mit "./mongod" starten**
      Unter Windows: "mongod.exe"
      Wichtig bei Windows: Der Ordner "/data/db" muss laut mongodb-Konsole unter C: (dem aktuellen Laufwerk) angelegt werden: *C:\data\db.* Ist dies nicht der Fall wird die Mongodb-Anwendung automatisch beendet.
3. **Dann den Server starten**
   a. **zweites Terminalfenster öffnen**
   b. **vom Terminal in das Verzeichnis MS3/Prototyp/Server navigieren**
   c. **node app.js eingeben und bestätigen.** Bei Erfolg läuft der Server auf Port 3000.

   d. **Sollten die Module nicht installiert sein, im Terminal “npm install *” eingeben**
4. **Damit der Client betriebsfertig ist und die Testdaten in die Datenbank eingepflegt werden, im Browser folgende Ressourcen ansprechen (ein “Okay” gibt es nicht zurück, beim gestarteten Server bekommt man eine Rückmeldung darüber, dass die Daten eingefügt wurden)**
   a. **“http://localhost:3000/testDaten”**
   b. **“http://localhost:3000/testRatings”**
   c. **“http://localhost:3000/atmosphere”**

* jeweils für: “express”, “body-parser”, “mongodb” und “xmlhttprequest”

Client

1. **Android Studio installieren**
   - https://developer.android.com/studio/index.html
2. **Projekt Dateien (MS3/Prototyp/Client) in den Projektordner von Android Studio kopieren**
3. **Beim Android Smartphone den Debug-Modus aktivieren. (i.d.R. bei “Einstellungen -> Über das Telefon” zu finden und dann ganz nach unten scrollen und 5-7 Mal auf “Build-Nummer” klicken)**
4. **Dann bei den neuen Einstellungen den “USB-Debugging”-Modus aktivieren!**
5. **Beim PC: IP-Adresse herausfinden (im Terminal “ifconfig -a” und die eigene IP-Adresse raus suchen (oft bei eth0 oder en0))**
   Unter Windows: “ipconfig” in der CMD eingeben und nach ipv4 schauen.
6. **Bei folgenden Dateien die Variable “server_url” anpassen (die herausgefundene IP-Adresse hinterlegen und den Port dran hängen. Bsp.: “private String server_url = "http://192.168.0.248:3000";”): Recommend.java, Register.java, Sign.java, Upload.java, Profile.java**

7. **Ein Smartphone sollte zur Verwendung im *Debug-Modus* bereitliegen und angeschlossen sein (Gleiches für einen Emulator).**
8. **Der Client sollte nun ausgeführt werden können (es könnte in Android Studio oben rechts ein Fenster auftauchen, wo steht “Sync Now”, das bitte, wenn gefordert, auch noch machen)**
9. **Angemeldet werden kann man sich mit dem Benutzer “test” und dem Passwort “test”. Ein eigener “Account” darf ebenfalls gerne angelegt und genutzt werden.**

Abweichungen vom Konzept

Von Open Street Maps zu Google Places

Ressourcen die zur Ermittlung der Atmosphäre von Orten gesammelt und verarbeitet werden, müssen laut Konzept aus der Bibliothek von Open Street Maps stammen. Zu Beginn war geplant alle Informationen von OSM über REST Methoden abzufragen. Nach einigen Evaluationsschritten während der praktischen Programmierarbeit wurde allerdings leider ersichtlich, dass dies scheinbar nur über schwierige Umwege möglich ist. So konnte man in den Testphasen mit dem OSM-Webservice keine Informationen als JSON-Objekt beziehen. Dies könnte nur umgangen werden, wenn ein OSM Dienst auf dem Client im Hintergrund ausgeführt wird, Informationen an den Server schickt, wo dieser diese verarbeitet und im Anschluss zurück an den Client schickt. Diese Alternative ist allerdings sehr unperformant und wird aus dem Entwicklungsprozess ausgeschlossen. Außerdem vergeben Nutzer (wie in der Dokumentation angegeben) ungerne Rechte an Anwendungen, vor allem, wenn diese permanent im Hintergrund und ohne visuelles Feedback gebraucht werden.

Als Konsequenz werden nun Daten von der Google Places API bezogen. Diese bietet zwar nicht alle benötigten Informationen an und manche sind erst nach Zahlung mit dem Premium Modell erhältlich, dennoch eignet sich dieses Verfahren, um mit der Implementation fortzufahren.

Aktuelle Attribute der Atmosphäre

Durch die Profilangaben eines Nutzers kann eine gewünschte Atmosphäre ermittelt werden. Ein Ort verfügt durch seine Attribute bereits über diese. So prüft die Anwendungslogik, ob sich die jeweiligen Atmosphären abgleichen. Passt eine Ort-Atmosphäre zu einer gewünschten Nutzer-Atmosphäre wird dieser Ort mit hoher Wahrscheinlichkeit empfohlen.

Aktuelle Attribute sind:

- **Geruch** eines Ortes.
- **Geschmack** des Essens z.B.
- **Freundlichkeit** der Menschen und des Personals
- **Geeignete Aktivitäten.** Stammt aus Analyse von App Daten
- **Geräusche** (Umgebungslautstärke) stammen aus Netatmos und eigenen Aufnahmen, um zu prüfen wann Richtwerte überstiegen werden.
- **Gesprochene Sprachen** an einem Ort und eines Landes werden berücksichtigt.
- **Wetter**. Informationen stammen von Open Weather Map.
- **Familientauglichkeit**
- **Sauberkeit** von Straßen und Gebäuden.
- **Behindertenfreundlichkeit/ Barrierefreiheit** (Differenzierbar in verschiedenen Bereichen wie Farbenblindheit und Rollstuhltauglichkeit)
- **Sehenswürdigkeiten.**
- **Textuelle Bewertungen** von anderen Nutzern über gelistete Orte in Google Places werden bei Empfehlungen mit angezeigt. Ein Keyword-Filter wird hierbei eingesetzt, um aus einem Fließtext wertvolle Informationen zu sammeln.

Diese Attribute werden in dezimalen Werten beschrieben - von 0 bis 5. 0 ist dabei ein Platzhalter, falls dieses Attribut gar nicht vertreten ist. 1 steht für den stärksten positiven Einfluss (Schulnoten-System).

```
LocationAtmosphere:{
  Geruch: [0-5],
  Geschmack: [0-5],
  Freundlichkeit: [0-5],
  Aktivitaeten: [(App-Analyse:) Pokemon Go, Fitness, etc.],
  Umgebungslautstärke: [0-5],
  Sprachen: [Deutsch, Englisch, Spanisch, Italienisch, Türkisch, ...]
  Wetter: ["Gebäude","Wetter betroffen"],
  Familientauglichkeit: [0-5],
  Sauberkeit: [0-5],
  Barrierefrei: [(Unterstützende Mittel)],
  Google Bewertung: [0-5],
}
```

Pseudocode Atmosphäre

Daten aus installierten Nutzer-Apps sammeln

Wie im Konzept beschrieben, sollen Informationen von installierten Apps eines Nutzers dazu beitragen die Empfehlungs-Algorithmen zu verfeinern.
Um dies zur Code-Implementierung zu simulieren, werden nun keine Daten aus MyFitnessPal bezogen, sondern aus der Google Fit App. Grund dafür ist der fehlende API-Schlüssel von Seiten MyFitnessPals. Dieser wurde zwar beantragt, doch noch nicht ausgeliefert und wird offensichtlich nicht rechtzeitig eintreffen. Die Funktionalität wurde allerdings vorerst auskommentiert, da das System für den Prototypen kompakter gemacht wurde (siehe Systemänderungen).

Android Versionskontrolle

Im Code ist zu erkennen, dass sich die Versionsabfrage auf Android 6 bezieht. Trotzdem ist die Software und die spezifische Methode auch auf älteren Systemen, z.B. 5.1. funktionstüchtig.
Der Grund für die Abfrage der neuesten Versionsnummer ist lediglich notwendig, weil die Methode und Syntax der aktuellen Android Studio Version danach verlangt, wenn man die Ortungsfunktion nutzen möchte.
Leider fehlten uns während den Tests Geräte mit älteren Android Versionsnummern als 5.0.1. Deshalb können wir nicht gewährleisten, dass die Anwendung unter der Version funktionstüchtig ist.

CompyMarks

Weil sich der Workload auf die Anwendungslogik der Empfehlungssysteme und Atmosphären-Ermittlung konzentriert hat, werden CompyMarks noch nicht implementiert. Dafür soll eine zweite App-Version für wirtschaftliche Nutzer hinzugefügt werden.

Systemänderungen

Ursprünglich war ein universal Werkzeug zur Aktivitäts- und Urlaubsplanung geplant. Allerdings sprengte dies den Rahmen der Programmierkenntnisse und

der Zeit. Deshalb wurde das System so universell wie möglich entwickelt, aber die Abfrage für die Empfehlung bezieht sich nur noch auf Restaurants im Umkreis von 500 Metern von der TH Köln Mensa (in Gummersbach). Die Entscheidung hatte viele Gründe, einer der wichtigsten Gründe dafür war allerdings, dass man (un)endlich viele Testdaten für sämtliche Standorte und Etablissements erstellen und speichern müsste. Außerdem werden die Testdaten ausgewertet und abgewandelt. (siehe aktuelle Attribute der Atmosphäre). Wir haben nun Daten für folgende Orte hinterlegt: “Thairestaurant Bambusgarten Gummersbach”, “Mensa TH Köln”, “32 Süd”, “Rhodos” und “Dornseifers”. Der wichtigste Punkt ist allerdings, dass die Daten dennoch durch das System verarbeitet werden. Werden die Stellen, die im Quellcode des Servers markiert wurden, wieder in den Code integriert (Kommentare müssen dafür gelöscht werden), sollte es möglich sein sämtliche Restaurants zu bewerten.

Ein weiterer wichtiger Punkt sind die REST-Ressourcen, die in der Dokumentation spezifiziert wurden. Diese wurden ebenfalls angepasst und geändert. Zum Beispiel wurden sämtliche POSTs zu PUTs gewandelt, da dies REST-Konform(er) ist. Der ursprüngliche Gedanke bei den meistens POSTs war es, dass Ressourcen angelegt werden (in der Datenbank). Manche GETs wurden allerdings auch zu POSTs umgewandelt, da das GET-Request von dem auf Java installierten Plugin Volley, keine Übertragung von Daten an der Server erlaubt. Deshalb wurden die GETs zu POSTs und damit zweckentfremdet.
Bei der endgültigen Implementierung wurden jedoch alle POSTs, die keine Ressourcen erstellen, zu PUTs gewandelt (damit sind die REST-Ressourcen wieder REST-konform(er)).

Im folgenden steht die abgeänderte REST-Spezifikation:

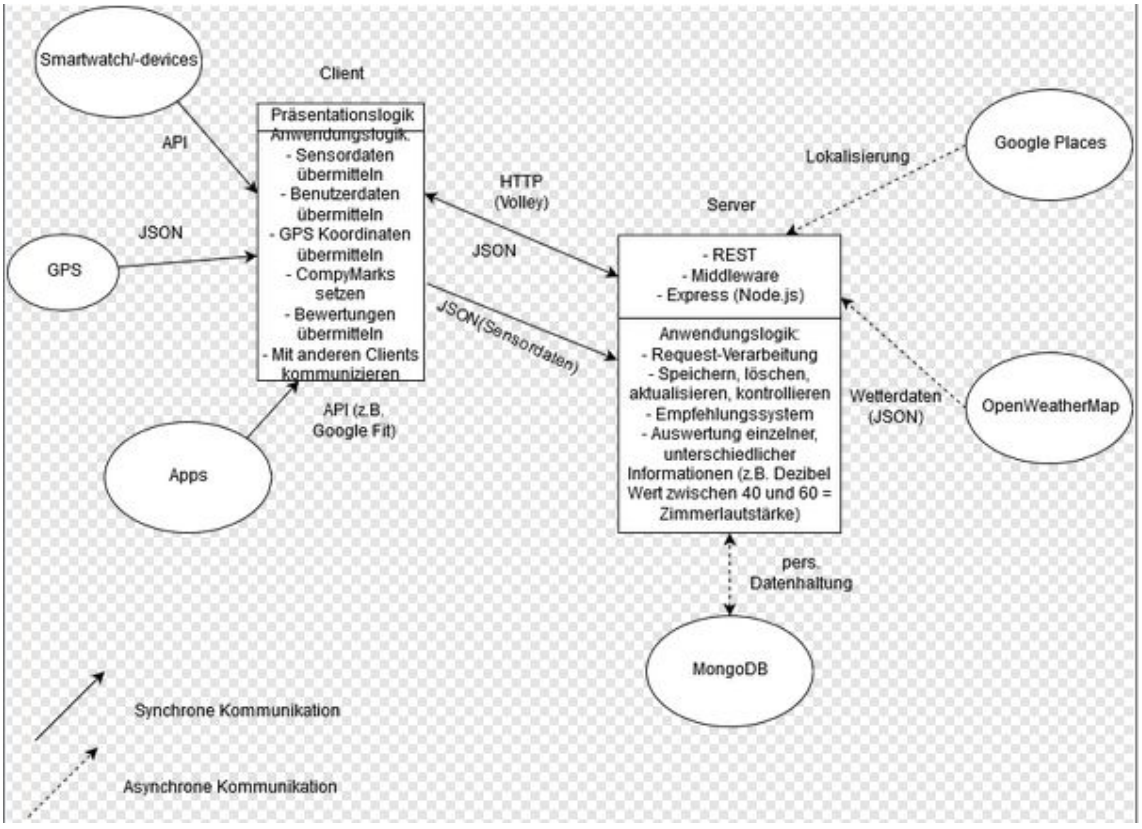
Ressource	Methode	Semantik	(req)	(res)
/userData	PUT	Dient zum registrieren (Benutzerdaten	application/json	application/json

		werden in der Datenbank gespeichert) und zum anmelden		
/userData/config	PUT	Hat der Benutzer bereits ein Profil? Ja: die Daten werden aktualisiert. Nein: Übergebene Werte werden in der Datenbank gespeichert.	application/json	-
/audioData	PUT	Empfängt Audio und Textuelle Daten und speichert diese in der Datenbank. (Ein Request kann nur gemacht werden, wenn der aktuelle Standort ermittelt wurde)	application/json	application/json
/recommendation	PUT	Der Benutzer übermittelt notwendige Informationen, wie seinen Benutzernamen und sendet an den Client eine Empfehlung für seinen aktuellen Standort	application/json	application/json
/testDaten	GET	Speichert Testdaten in der Datenbank. (GET wurde zweckentfremde	-	-

		t: Allerdings ist der Aufruf über den Browser ohne weiteres realisierbar)		
/testRatings	GET	Die gespeicherten Testdaten werden nun analysiert und interpretiert	-	-
/atmosphere	GET	Die gespeicherten testRatings und testDaten werden in das Schulnotensystem gebracht.	-	-

Synchronität der Architektur

Wie in der Code Implementation besprochen wurde findet der Datenaustausch zwischen Client und Server über HTTP synchron statt und nicht asynchron wie zunächst angenommen. Grund für die Annahme war bei den Tests die Beobachtung, dass der Server immer noch gesendete Informationen des Clients bearbeitete und zurücksenden wollte, obwohl die Client nicht mehr verbunden war. Lediglich die Informationen, die über die APIs an den Server laufen (Google Places und OWM) asynchron.



Architekturdiagramm Stand 10.07.2017