Entwicklung einer Web-API zur Indoor-Ortung basierend auf WiFi-Fingerprints

Im Modul

Ausgewählte Kapitel mobiler Anwendungen

Angewandte Informatik (B. Sc.)

Von
Tom Morelly (561440)
Sewanstraße 56
10319 Berlin

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Hochschule für Technik und Wirtschaft
Treskowallee 8
10318 Berlin

Inhaltsverzeichnis

1. Einleitung

Das folgende Dokument beschreibt die Realisierung einer Web-API zur Indoor-Ortung anhand WiFi-Fingerprints.

Dieses Projekt wurde im Rahmen der Lehrveranstaltung „Ausgewählte Kapitel mobiler Anwendungen" im fünften Semester des Bachelor Studienganges Angewandte Informatik angefertigt.

1.1 Projektziel

Ziel ist es, eine Web-API bereitzustellen, welche in der Lage ist WiFi-Fingerprints zu verarbeiten gemäß der sogenanten CRUD Operationen:

- **C**reate (HTTP Post)
- **R**ead (HTTP Get)
- **U**pdate (HTTP Put)
- **D**elete (HTTP Delete)

Darüber hinaus, soll eine Methode entwickelt werden, den bestpassendsten Fingerprint für Clients anhand dessen verfügbaren WLAN (BSSID) und der jeweiligen Signalstärke (RSSI) zu bestimmen.

1.3 Projektumsetzung

Das Projekt wird unter Einsatz sogenannter Microservices[1] entwickelt. Bei Microservices kann jede Entität (Datenbank, API) von einem oder mehreren Containern dargestellt werden. In diesem Projekt werden Docker Container[2] eingesetzt. Die Verbindung und Orchestrierung der einzelnen Container wird mithilfe Docker-Compose[3] realisiert. Die API soll auf Grundlage des Webframeworks Flask[4] in Python entwickelt werden. Als Speicherinstanz wird die Dokumentenorientierte Datenbank MongoDB[5] benutzt.

Die Applikation soll nach dem Prinzip der testgetriebenen Entwicklung erstellt werden. Für die Tests wird das Python Framework pytest[6] eingesetzt. Als CI-Tool wird Travis[7] benutzt, welches nach jedem Commit oder Merge die Tests automatisch ausführt umso den aktuellen Status der Applikation bereitzustellen.

Der Code wird unter dem folgenden GitHub Repository aufgeführt:

- https://github.com/FalcoSuessgott/AccessPointFingerprintAPI

[1] https://en.wikipedia.org/wiki/Microservices
[2] https://www.docker.com/
[3] https://docs.docker.com/compose/
[4] https://www.palletsprojects.com/p/flask/
[5] https://www.mongodb.com/
[6] https://docs.pytest.org/en/latest/
[7] https://travis-ci.com/

2. Applikationsarchitektur

Im Folgenden wird die Applikation und deren Funktionsweise erläutert.

2.1 Aufbau

Die Applikation besitzt folgende Schnittstellen:

Schnittstelle	Beschreibung
localhost:5000/fingerprint	HTTP Endpunkt zum Verwalten von WiFi-Fingerprints
localhost:5000/localize	HTTP Endpunkt zum Bestimmen des bestpassendsten Fingerprints
localhost:3100	MongoDB Webinterface
mongodb:27017	Datenbank

2.2 HTTP Endpunkte

2.2.1 /fingerprint

HTTP Methode	*Beschreibung*	*Parameter*	*HTTP Status Code*
GET	Liefert den Fingerprint mit der jeweiligen ID zurück	Fingerprint ID	200 -> Erfolgreich 404 -> Kein Fingerprint unter dieser ID
POST	Erstellt einen Fingerprint mit den übergebenden Informationen	Fingerprint im JSON-Format	201 -> Erfolgreich 400 -> Fingerprint mit dieser ID existiert bereits 404 -> Fingerprint hat ein ungültiges Format
PUT	Aktualisiert den Fingerprint mit der jeweiligen ID	Fingerprint ID	201 -> Erfolgreich 404 -> Kein Fingerprint unter dieser ID
DELETE	Löscht den Fingerprint unter dieser ID	Fingerprint ID	201 -> Erfolgreich 404 -> Kein Fingerprint unter dieser ID

2.1.2 /localize

HTTP Methode	*Beschreibung*	*Parameter*	*HTTP Status Code*
GET	Bestimmt den bestpassendsten Fingerprint	Je WLAN den BSSID und RSSI Wert	200 -> Erfolgreich 400 -> Ungültiger Request 404 -> Kein Fingerprint unter dieser ID

2.2 Datenbank

Die Datenbank lauscht standardmäßig auf Port 27017 des mongodb-Container.

Parameter:

Parameter	Wert
Datenbank	testdb
Collection	Fingerprint
Root Username	test
Root Password	test123
Datenverzeichnis	/data/db
Datenbankbenutzer	Flask
Password	flask123

Für Debug Zwecke wird Mongoku[8] benutzt, welches ein Webinterface für die Datenbank bereitstellt. Mongoku ist über Port 3100 auf dem Localhost erreichbar.

[8] https://github.com/huggingface/Mongoku

3. WiFi-Fingerprints

In diesem Abschnitt sollen Aufbau und Erzeugung der WiFi-Fingerprints erläutert werden.

Zu erwähnen ist, dass der Autor das Format nicht selber ausgesucht hat, sondern auf bestehende Arbeiten aufsetzt. Frau Walter hat bereits in Ihrer Master-Thesis[9] ein Verfahren mittels WiFi-Fingerprints entwickelt.

Weiterhin haben Studenten haben auf Grundlage dessen, eine Android Applikation[10] entwickelt, welche Fingerprints von aktuellen Positionen ermitteln kann. Und abspeichern kann.

Die im Folgenden beschriebenen Eigenschaften basieren aus Kompatibilitätsgründen auf den vorangegangenen Arbeiten.

Ein Fingerprint ist ein in JSON[11] formatierte Datei, welche neben einer ID, Beschreibung und weiteren Informationen, für jede Sekunde der Messung die verfügbaren WLANs (BSSID) und die jeweilige Signalstärke (RSSI) speichert. Im Rahmen dieses Projektes, betrug eine Messung 60 Sekunden. Unter A.1 Bespiel WiFi-Fingerprint ist ein Beispiel eines Fingerprints hinterlegt.

3.1 Erzeugung

Zur Erzeugung eines aktuellen Fingerprints, einfach die BVGDetection Applikation starten und die benötigten Informationen (Name des Ortes, Beschreibung) ausfüllen. Mit Drücken des Fingerprints Icons, werden anschließend 60 Sekunden lang alle verfügbaren WLANs und deren Signalstärke gescannt und im internen Speicher des Gerätes abgelegt.

Abbildung 2: BVGDetection Fingerprint Erzeugung

[9] http://www.sharksystem.net/htw/FP_ICW_BA_MA/2017_Carola%20Walter%20Masterarbeit.pdf
[10] https://github.com/OpenHistoricalDataMap/BVGDetection
[11] https://www.json.org/json-en.html

Derzeit müssen die Fingerprints noch händisch aus dem Internen Speicher des Gerätes auf einen Computer kopiert werden, um sie dann anschließend mittels HTTP Get in die Web-API zu laden. Dies ist eine denkbare Erweiterung für die Zukunft.

4. Indoor Ortung von Clients

Clients, welche keine Möglichkeit haben einer Lokalisierung mittels GPS, können nun mithilfe der API und der verfügbaren WLANs positionsbestimmt werden.

In diesem Abschnitt wird der Algorithmus zur Bestimmung des Fingerprints erläutert. Anschließen werden Versuchsaufbau und Ergebnisse analysiert. Als Clients wurde ESP8266[12], ein WiFi-Microcontroller, genutzt.

4.1 Bestimmung des passenden Fingerprints

Wie erwähnt, liefert der „localize"-Endpunkt mittels HTTP Get den bestpassendsten Fingerprint zurück anhand der verfügbaren WLANs und der Signalstärke des Clients. Diese Informationen werden in der URL als Parameter eingehängt.

Ein Beispielhafter HTTP-Request, könnte somit so aussehen:

http://API_IP:5000/localize?mac1=00:AA:11:BB:CC:44&strength1=-58&mac2=00:AA:11:BB:DD:44&&strength2=-34

Die API iteriert nun durch alle verfügbaren Fingerprints und berechnet den Grad der Übereinstimmung. Der Grad der Übereistimmung wird folgendermaßen Ermittelt:

Iteriere durch alle Fingerprints:

1. Parse alle MAC-Adressen eines Fingerprints zu einem Array.
2. Entferne Alle MAC-Adressen, welche weniger als 1/3 (= 20) in der Messung vorkommen.
3. Berechne den Durchschnitt der Signalstärke zu jeder MAC-Adresse.
4. Parse alle in der URL übergebenden MAC-Adressen und Signalstärke-Werte zu einem Array.
   Iteriere durch alle MAC-Adressen aus den URL Parametern:
   a. Wenn die aktuelle MAC auch im Fingerprint vorkommt, dann Berechne die Übereinstimmung der jeweiligen Durchschnitte mit:

$$100 - \frac{(b-a) * 100}{a}$$

   Wobei a die Signalstärke der URL MAC und b der Durchschnitt der Signalstärke MAC aus dem Fingerprint ist.
   Das Ergebnis wird gespeichert und mit jedem weiteren Durchlauf hinzuaddiert. Am Ende wird der Wert durch die Anzahl der Durchläufe geteilt. Man erhält einen Prozentwert, der die Übereinstimmung der Werte aus den URL Request und den aktuellen Fingerprint beschreibt.

5. Gebe den Fingerprint mit der höchsten Übereinstimmung wieder.

[12] https://en.wikipedia.org/wiki/ESP8266

4.2 Client Code

Der ESP8266 wurde so programmiert, dass er alle 30 Sekunden die „localize"-Schnittstelle über HTTP Get kontaktiert und die WLANs samt Signalstärke in der URL anhängt. Der zurückgelieferte Fingerprint wird dann auf einen Webserver auf dem ESP8266 dargestellt.

Abbildung 3: ESP8266 WebServer

4.3 Validierung

4.3.1 Versuchsaufbau

Um nun die Genauigkeit der in 4.1 Bestimmung des passenden Fingerprints Methode zu erhalten, wurde in der Wohnung vom Autor in jeden Raum Fingerprints mithilfe der BVGDetection Applikation erzeugt. Diese sind im Repository unter „tests/fingerprints" einzusehen.

Anschließend wurde der ESP8266 an einer der Stellen positioniert. Der Fingerprint wurde dann im Browser unter de IP des ESPs verglichen. Als Beispiel kann unter A.2 Log von einem „localize"-Request ein Ausschnitt aus dem Log während eines Requests eingesehen werden.

4.3.2 Ergebnisse

Für alle 5 Fingerprints lauten die Ergebnisse wie folgt:

Raum / Fingerprint	**Ermittelter Fingerprint**	**Übereinstimmung**
Flur (00_fp_floor.json)	00_fp_floor.json	Ja
Küche (01_fp_kitchen.json)	01_fp_kitchen.json	Ja
Raum_1 (02_fp_room_pw.json)	02_fp_room_pw.json	Ja
Raum_2 (03_fp_room_tm.json)	02_fp_room_pw.json	Nein
Raum_3 (04_fp_room_pw.json)	04_fp_room_pw.json	Ja

4.3.3 Auswertung

In 4 von 5 Versuchen wurde der korrekte Fingerprint von der API ermittelt.

Es wird vermutet, dass der fehlerhafte Versuch durch die geringe Distanz (ca. 2m) zu dem anderen Fingerprint (02_fp_room_pw & 03_fp_room_tm) entstanden ist.

Bis zu einer Genauigkeit von ca. 2 Metern, ist die API in der Lage eine Indoor Ortung anhand von WiFi-Fingerprints vorzunehmen.

Recherchen[13] bestätigten die Annahme mittels verschiedener Formeln um die Distanz zweier RSSI Werte in Metern zu errechnen.

13
https://www.researchgate.net/post/How_can_one_calculate_distances_using_the_RSSI_value

4. Fazit

Im letzten Abschnitt des Dokuments soll ein Fachliches, sowie ein Persönliches Resümee gezogen werden.

4.1 Fachliches Fazit

Eine Indoor-Ortung von Clients mithilfe der API ist möglich. In der 4 von 5 Versuchen wurde der korrekte Fingerprint ermittelt. Vermutet wird, dass die Signalstärke bis zu 2 Meter signifikant sind und somit zu einer Unterscheidung ausreichen.

4.2 Persönliches Fazit

Die Planung und testgesteuerte Entwicklung einer Web-API mit Datenbank verlief sehr positiv.

Zu einem konnte ich meine Kenntnisse in Projekten mit Containern vertiefen und festigen. Des Weiteren lernte ich wichtige Zusammenhänge in der API Entwicklung mit Python und dem Flask-Framework.

Abgerundet wurde das Projekt mit dem erfreulichen Resultat von 4 korrekten Fingerprints in 5 Tests.

A. Anhang

A.1 Bespiel WiFi-Fingerprint

```
{
      "id": "test",
      "description": "test desc",
      "coordinates": "",
      "additionalInfo": "",
      "fingerprint": [
        {
          "timestamp": "20-12-2019-02.31.29",
          "signalSample": [
            {
              "macAddress": "02:00:00:00:01:00",
              "strength": -50
            }
          ]
        },
        {
          "timestamp": "20-12-2019-02.32.32",
          "signalSample": [
            {
              "macAddress": "02:00:00:00:01:00",
              "strength": -50
            }
          ]
        }
  ]
}
```

Parameter	**Bechreibung**	**Format**
id	Eindeutige Id des Fingeprrints	String
Description	Beschreibung	String
Coordinates	Latitude und Longitude Werte des Fingeprints	String(Lat,Long)
Addiionalinfo	Zusätzliche Info	String
fingerprint[]	Array, mit EInträgen aller WLANS der jeweiligen Sekujnde	Array
Timestamp	Timestamp der Messung	Timestamp
signalSample[]	Array mit Werten aus allen verfügbaren WLANS	Array
macAddress	BSSID Wert des jeweiligen WLANS	String
Strength	RSSI Wert des jeweiligen WLANs	Integer

A.2 Log von einem „localize"-Request

```
api        | 192.168.178.38 - - [14/Jan/2020 16:18:16] "GET
/localize?mac1=44:4E:6D:53:88:C1&strength1=-82&mac2=38:10:D5:B2:64:10&strength2=-
59&mac3=E8:DF:70:CD:61:1B&strength3=-73&mac4=B4:A5:EF:43:54:24&strength4=-
70&mac5=82:CE:62:74:E2:FF&strength5=-87&mac6=E8:DF:70:FA:D8:A3&strength6=-88 HTTP/1.1" 200 -
api        | [2020-01-14 16:18:28,218] INFO in app: Url params: {'mac1': '38:10:D5:B2:64:10',
'strength1': '-58', 'mac2': '44:4E:6D:53:88:C1', 'strength2': '-83', 'mac3':
'B4:A5:EF:43:54:24', 'strength3': '-66', 'mac4': '82:CE:62:74:E2:FF', 'strength4': '-88',
'mac5': '18:83:BF:47:33:23', 'strength5': '-80', 'mac6': 'E8:DF:70:CD:61:1B', 'strength6': '-
73', 'mac7': '84:A9:3E:CF:E6:79', 'strength7': '-85'}
api        | [2020-01-14 16:18:28,246] INFO in app: Current Best match: None
api        | [2020-01-14 16:18:28,247] INFO in app: Compare url {'38:10:D5:B2:64:10': '-58',
'44:4E:6D:53:88:C1': '-83', 'B4:A5:EF:43:54:24': '-66', '82:CE:62:74:E2:FF': '-88',
'18:83:BF:47:33:23': '-80', 'E8:DF:70:CD:61:1B': '-73', '84:A9:3E:CF:E6:79': '-85'} &
fingerprint: Id: 00_fp_floor
api        | [2020-01-14 16:18:28,247] INFO in app: {'38:10:D5:B2:64:10': '-58',
'44:4E:6D:53:88:C1': '-83', 'B4:A5:EF:43:54:24': '-66', '82:CE:62:74:E2:FF': '-88',
'18:83:BF:47:33:23': '-80', 'E8:DF:70:CD:61:1B': '-73', '84:A9:3E:CF:E6:79': '-85'}
api        | [2020-01-14 16:18:28,248] INFO in app: Current url 1. mac: 38:10:D5:B2:64:10
api        | [2020-01-14 16:18:28,250] INFO in app: Fingerprints mac average strength: -39
api        | [2020-01-14 16:18:28,251] INFO in app: Current url 2. mac: 44:4E:6D:53:88:C1
api        | [2020-01-14 16:18:28,253] INFO in app: Fingerprints mac average strength: -72
api        | [2020-01-14 16:18:28,254] INFO in app: Current url 3. mac: B4:A5:EF:43:54:24
api        | [2020-01-14 16:18:28,256] INFO in app: Fingerprints mac average strength: -61
api        | [2020-01-14 16:18:28,257] INFO in app: Current url 4. mac: is not in fingerprint
00_fp_floor
api        | [2020-01-14 16:18:28,258] INFO in app: Current url 5. mac: is not in fingerprint
00_fp_floor
api        | [2020-01-14 16:18:28,259] INFO in app: Current url 6. mac: E8:DF:70:CD:61:1B
api        | [2020-01-14 16:18:28,261] INFO in app: Fingerprints mac average strength: -57
api        | [2020-01-14 16:18:28,261] INFO in app: Current url 7. mac: 84:A9:3E:CF:E6:79
api        | [2020-01-14 16:18:28,264] INFO in app: Fingerprints mac average strength: -74
api        | [2020-01-14 16:18:28,264] INFO in app: Relation: 56.11066803282706
api        | [2020-01-14 16:18:28,273] INFO in app: Current Best match: ('00_fp_floor',
56.11066803282706)
api        | [2020-01-14 16:18:28,274] INFO in app: Compare url {'38:10:D5:B2:64:10': '-58',
'44:4E:6D:53:88:C1': '-83', 'B4:A5:EF:43:54:24': '-66', '82:CE:62:74:E2:FF': '-88',
'18:83:BF:47:33:23': '-80', 'E8:DF:70:CD:61:1B': '-73', '84:A9:3E:CF:E6:79': '-85'} &
fingerprint: Id: 01_fp_kitchen
api        | [2020-01-14 16:18:28,275] INFO in app: {'38:10:D5:B2:64:10': '-58',
'44:4E:6D:53:88:C1': '-83', 'B4:A5:EF:43:54:24': '-66', '82:CE:62:74:E2:FF': '-88',
'18:83:BF:47:33:23': '-80', 'E8:DF:70:CD:61:1B': '-73', '84:A9:3E:CF:E6:79': '-85'}
api        | [2020-01-14 16:18:28,275] INFO in app: Current url 1. mac: 38:10:D5:B2:64:10
api        | [2020-01-14 16:18:28,278] INFO in app: Fingerprints mac average strength: -49
api        | [2020-01-14 16:18:28,279] INFO in app: Current url 2. mac: 44:4E:6D:53:88:C1
api        | [2020-01-14 16:18:28,282] INFO in app: Fingerprints mac average strength: -76
api        | [2020-01-14 16:18:28,283] INFO in app: Current url 3. mac: B4:A5:EF:43:54:24
api        | [2020-01-14 16:18:28,287] INFO in app: Fingerprints mac average strength: -58
api        | [2020-01-14 16:18:28,288] INFO in app: Current url 4. mac: is not in fingerprint
01_fp_kitchen
api        | [2020-01-14 16:18:28,288] INFO in app: Current url 5. mac: 18:83:BF:47:33:23
api        | [2020-01-14 16:18:28,291] INFO in app: Fingerprints mac average strength: -67
api        | [2020-01-14 16:18:28,292] INFO in app: Current url 6. mac: E8:DF:70:CD:61:1B
api        | [2020-01-14 16:18:28,294] INFO in app: Fingerprints mac average strength: -62
api        | [2020-01-14 16:18:28,295] INFO in app: Current url 7. mac: 84:A9:3E:CF:E6:79
api        | [2020-01-14 16:18:28,297] INFO in app: Fingerprints mac average strength: -79
api        | [2020-01-14 16:18:28,298] INFO in app: Relation: 74.42040861485725
```