

Dokumentation der Proof of Concepts

Kommunikation zwischen Client und Server

Server Aufbau und Start

Zunächst wird das 'net' Modul importiert, um eine asynchrone Netzwerk API zum Erstellen eines TCP-Servers und Clients zu erstellen. Die Clients, die sich verbinden werden, zwischengespeichert damit immer wieder darauf zugegriffen werden kann. Zum Schluss der Function wird noch ein .listen aufgerufen damit der Server direkt gestartet wird und auf der angegebenen Adresse inkl. Port auf Verbindungen wartet.

```
var net = require('net');

// stores all logged in users
var clients = [];

var server = net.createServer(function(socket) {
  clients.push(socket); // stores the connecting client

  // socket.write('THIS came from the server.\r\n');

  // client connected
  console.log(socket.remoteAddress + ' just connected.');
```

```
  socket.on('error', (err => {
    throw err;
  }));

  socket.on('close', () => {
    console.log(socket.remoteAddress + ' just disconnected.');
```

```
  });

  // redirects everything back to the socket
  socket.pipe(socket); //
}).listen(1337, '127.0.0.1');
```

Verbindungsaufbau des Clients

Es wird eine Socket Variable angelegt, um die Verbindung zum Server herzustellen und aufrecht zu erhalten.

```
const net = require('net');
let client = new net.Socket();
|
client.connect(1337, '127.0.0.1', function() {
  console.log('Connected');
```

Client Verbindung aufbauen und trennen

```
Connected
Do you want to
(1)log in
(2)create a new user
(3)Exit
exit
Connection closed
```

Server Ausgabe

```
127.0.0.1 just connected.
127.0.0.1 just disconnected.
|
```

Firestore initialisieren

```
// FIREBASE INIT
const admin = require('firebase-admin');
const functions = require("firebase-functions");
const serviceAccount = require("../serviceKey.json");

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
  databaseURL: "https://scrapofood.firebaseio.com"
});

// FIREBASE INIT END

// FIRESTORE ADD USER
//let db = admin.database(); // realtime Database
let db = admin.firestore();
let docref = db.collection('userCol').doc(getIdInCollection('userCol'));

let setArthur = docref.set({
  name: 'Arthur T',
  email: "any1@mail.com",
  adress1: "An der Delle 7",
  adress2: "12345",
  adress3: "Gummersbach"
});

setArthur.then(function () {
  console.log("user added");
}).catch(function (error) {
  console.log("Error occurred");
});
```

Daten hinzufügen

Die User werden unter der Sammlung 'userCol' mit allen notwendigen Informationen abgelegt. Unter der Sammlung eines jeden Nutzer wird zukünftig ein Verweis der Gruppen, denen ein einzelner Nutzer zugehört sowie die verschiedenen Inserate die dieser erstellt oder abonniert hat abgelegt.

scrapofood	userCol	D7RyAnyLovhMouMFN8Wa
+ Sammlung hinzufügen	+ Dokument hinzufügen	+ Sammlung hinzufügen
userCol >	1S1JP7C43o55eQ0Pbrsh	+ Feld hinzufügen
	D7RyAnyLovhMouMFN8Wa >	adress1: "An der Delle 7"
	GAVHYJufTzY9a1NvUNSt	adress2: "12345"
	Uf759FVZErwgNtKhefC4	adress3: "Gummersbach"
		email: "any1@mail.com"
		name: "Arthur T"

Generieren der UserIDs

Firebase bietet eine Möglichkeit userIDs automatisch generieren zu lassen sodass es keine ID ein zweites Mal geben wird und somit keine Fehler entstehen können.

```
// generate random ID
function getIdInCollection(collectionName) {
  let ref = db.collection(collectionName).doc();
  let id = ref.id;
  return id;
}
```

Wetterdaten über externen Dienst abfragen und speichern

OpenWeatherMap API Call

Der API-Call besteht aus dem API-Key (welcher hier jetzt aus Sicherheitsgründen nicht zu sehen ist) und verschiedenen weiteren Attributen, um genauere Informationen über einen Ort oder eine Stadt zu filtern.

```
function weatherCall() { |
  let url = 'https://api.openweathermap.org/data/2.5/forecast?q=cologne,DE&units=metric&a
  request(url,function (err, response, body) {
    if(err){
      console.log('error: ',error);
    } else {
      //console.log('body:', body);
      let weatherData = JSON.parse(body);
      console.log(weatherData);
    }
  })
}
```

OpenWeatherMap Result

Mit der API Abfrage wird einem ein Bulk von Daten geschickt, welchen man zunächst in das JSON Format konvertieren kann um sie besser lesen zu können. Unter anderem werden Längen- und Breitengrade für die Stadt ausgegeben sowie Zeitzone und in welchem Land sich der Ort befindet. Wenn verfügbar, sogar Einwohnerzahlen für eine Stadt.

```
    wind: [Object],
    sys: [Object],
    dt_txt: '2019-06-06 12:00:00' } ],
city:
{ id: 2886242,
  name: 'Cologne',
  coord: { lat: 50.9384, lon: 6.96 },
  country: 'DE',
  population: 963395,
  timezone: 7200 } }
```