Zusammenfassung: «Reveal the Pain»

Web Programming Lab

Patrick Bucher

28.09.2019

1 Projektidee

Ein körperliches Leiden, z.B. Kopfschmerzen, kann viele Ursachen haben. Mithilfe von Self-Tracking einer Vielzahl möglicher Einflussgrössen können die Rohdaten zur Ursachenforschung erhoben werden. Mithilfe eines Phi-Koeffizienten kann anhand dieser Rohdaten die Korrelation zwischen Einflussgrössen und Krankheitssymptom berechnet werden. Damit können für ein Krankheitssymptom möglicherweise förderliche Einflüsse sondiert werden.

Die «Reveal the Pain»-Web-Applikation hilft einerseits beim täglichen Self-Tracking und führt andererseits die Berechnung der Korrelationskoeffizienten automatisch durch. Um etwa den Ursachen für Kopfschmerzen auf den Grund zu gehen könnten folgende Einflussgrössen getrackt werden: Alkoholkonsum, Bewegung, Stress, Wetterumschwung, Koffein, Rauchen, Lärm, usw.

Die Idee für diese Anwendung stammt aus dem Buch «Eloquent JavaScript», wo die Phi-Korrelation zwischen Pizzakonsum und der nächtlichen Verwandlung in ein Eichhörnchen untersucht wird (Haverbeke, Kapitel 4, Seite 66: *The Lycanothrope's Log*).

2 Technische Dokumentation

Die Applikation besteht aus drei Teilen: Ein Frontend, das als Single Page Application umgesetzt ist; ein Backend, das eine RESTful API anbietet; und den Key-Value-Store Redis für die Datenhaltung.

Das Frontend wurde mit reinem JavaScript («*Vanilla JS*») entwickelt. Es wurden weder externen Dependencies verwendet, noch gibt es einen Build-Prozess. Die ursprüngliche Idee, Web Components zu verwenden, wurde aus zeilichen Gründen aufgegeben. Das Design-Ziel im Frontend war Einfachheit.

Im Backend hingegen sind verschiedene Libraries verwendet worden: `express` und `cors` für die RESTful API, `jsonwebtoken` für das Ausstellen und Validieren der Tokens – d.h. zum Schützen der API, `bcryptjs` zum Verschlüsseln von Passwörtern, `redis` zum Ansprechen des Key-Value-Stores und `jest` für Unit Tests.

Im Key-Value-Store (Redis) werden die Daten als Sets abgelegt. Der Key hat die Form `username:date`, und der Wert ist ein Set von Tags, also z.B. `johndoe:2019-09-28 = [Beer, Walk, Coffee, Work, Stress, Insomnia]`.

Das Backend wurde einerseits mit Unit Tests (Berechnung der Korrelation, Validierungsfunktionen), andererseits mit einem Python-Skript (End-to-End-Tests der API) getestet. Das Frontend wurde ausschliesslich manuell getestet.

Um die Applikation produktiv betreiben zu können, müsste die statische Datei (`credentials.js`) mit Klartext-Credentials durch ein Registrierungssystem ersetzt werden. (Innerhalb des Backends werden bereits die verschlüsselten `bcrypt`-Tokens geprüft.) Der `secretKey` müsste als Umgebungsvariable injiziert werden.

Die Applikation kann mittels `docker-compose` gestartet werden und ist unter der URL http://localhost:8080/ verfügbar. Sie greift auf das Backend (Port `8000`) zu, welches wiederum auf Redis zugreift (Port `6379`).

2.1 Berechnung der Korrelation

Der Benutzer hat während einer Woche seine Verhaltensweisen getrackt, um auf die Ursache für sein Rückenleiden zu kommen. Das Journal könnte folgendermassen aussehen (die Zielvariable *Rückenschmerzen* wird wie jeder andere Tag erfasst):

- Montag: Ausschlafen, Gewichtheben, Büroarbeit, Gartenarbeit, Rückenschmerzen.
- Dienstag: Büroarbeit, Alkoholkonsum, Einkaufen, Rückenschmerzen.
- Mittwoch: Büroarbeit, Müdigkeit, Einkaufen, Rückenschmerzen.
- Donnerstag: Ausschlafen, Gartenarbeit, Alkoholkonsum, Rückenschmerzen.
- Freitag: Büroarbeit, Müdigkeit, Einkaufen.
- Samstag: Ausschlafen, Spaziergang, Putzen, Gartenarbeit, Rückenschmerzen.
- Sonntag: Ausschlafen, Alkoholkonsum, Spaziergang, Velotour.

Nun möchte der Benutzer die Korrelationen verschiedener möglicher Einflussgrössen für die Zielvariable *Rückenschmerzen* berechnen lassen. Für jeden Tag (jede erklärende Variable im Journal, ausgenommen die Zielvariable *Rückenschmerzen*), wird für jedes Datum folgende Klassifizierung vorgenommen:

- Gruppe 1: Weder Ziel- noch erklärende Variable eingetragen.
- Gruppe 2: Zielvariable fehlt, erklärende Variable vorhanden.
- Gruppe 3: Zielvariable eingetragen, erklärende Variable fehlt.
- Gruppe 4: Ziel- und erklärende Variable eingetragen.

Für obiges Journal ergäbe sich dadurch folgende Klassifizierung der erklärenden Variablen *Büroarbeit* (EV) und Zielvariablen *Rückenschmerzen* (ZV):

	EV fehlt	EV vorhanden
ZV fehlt	1 (So)	1 (Fr)
ZV vorhanden	2 (Do, Sa)	3 (Mo, Di, Mi)

Die Gruppen können nun folgendermassen bezeichnet und mit Werten belegt werden: EV und ZV fehlt: $n_{00} = 1$, EV vorhanden/ZV fehlt: $n_{01} = 1$, EV fehlt/ZV vorhanden:

$n_{10} = 2$, EV und ZV vorhanden: $n_{11} = 3$. Der Phi-Koeffizient kann nun folgendermassen berechnet werden:

$$\phi = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{(n_{10} + n_{11})(n_{00} + n_{01})(n_{01} + n_{11})(n_{00} + n_{10})}}$$

Mit obigen Werten eingesetzt ergibt das:

$$\underline{\underline{\phi}} = \frac{3 \times 1 - 2 \times 1}{\sqrt{(2+3)(1+1)(1+3)(1+2)}} = \frac{1}{\sqrt{5 \times 2 \times 4 \times 3}} = \frac{1}{\sqrt{120}} = \underline{\underline{0.0913}}$$

Die Skala reicht von -1 (negative Korrelation) bis +1 (positive Korrelation). Für Werte mit einem Betrag ab 0.5 kann von einer Korrelation die Rede sein. Die erfassten Daten ergeben somit keine Korrelation zwischen Büroarbeit und Rückenschmerzen. Der Benutzer muss weiter Daten erfassen, um seinem Rückenleiden auf die Spur zu kommen.

3 Fazit

- Die erstellte Applikation ist für die lokale Anwendung durchaus praktisch einsetzbar. Ich werde sie versuchshalber einsetzen, um möglichen Ursachen für meine Kopfschmerzen auf die Spur zu kommen.
- Ob die Applikation auch für mehrere Benutzer auf einem Server praktikabel verwendbar ist (Lasttest), konnte nicht getestet werden. Im Frontend wird es sich zeigen, ob das komplette Neuladen des Journals beim Erfassen eines Tags auch bei grösseren Journalen genug performant ist.
- Das User-Interface wurde bewusst minimalistisch gehalten. Auf Fragestellungen wie Accessibility und Responsiveness konnte aus zeitlichen gründen nicht eingegangen werden.
- Die ursprünglich geplante Umsetzung mit der redundanten Datenspeicherung konnte umgangen werden. Dafür müssen mehr Abfragen gegen den Key-Value-Store getätigt und ausgewertet werden. Mit kleineren Datenmengen hat sich das nicht als problematisch ausgestellt.
- Zur praktischen Anwendung auf einem Server müsste die Möglichkeit einer Registrierung eingebaut werden. Die Credentials könnten in Redis gespeichert werden.
- Ein Studienkollege merkte an, dass sich der Report auch gut mit einem Diagramm visualisieren liesse. Da dies den Projektumfang sprengen würde, möchte ich diese Erweiterung im Rahmen des Moduls *Datenvisualisierung*, das ich dieses Semester besuche, mit D3.js umsetzen.

4 Reflexion

- Die Blockwoche und v.a. die Projektarbeit haben nicht nur mein Interesse an der Web-Entwicklung wieder geweckt, sondern auch an der funktionalen und asynchronen Programmierung. Ich möchte mich weiter mit diesen Themen befassen.

- Ursprünglich wollte ich das Backend mit der Programmiersprache Go umsetzen, weil ich damit schon verschiedene HTTP-Server-Anwendungen entwickelt habe. Nach der Einführung in Node.js und Express habe ich mich aber dazu entschlossen, diese Technologien einzusetzen, um dabei etwas Neues zu lernen.
- JavaScript und Express erlauben es die Endpoints sehr einfach und mit wenig Code zu definieren und zu implementieren. Gewöhnungsbedürftig war hingegen, dass viele Libraries (z.B. die Redis-Library zum Zugriff auf den Key-Value-Store) asynchron arbeiten, was jedoch nötig ist, um den globalen Event-Loop von Node.js nicht mit unnötig grossen Aufgaben zu blockieren. Hat man sich aber erst einmal an den Promise-Mechanismus gewöhnt, kann man damit schnell übersichtlichen und hochwertigen Code schreiben.
- Beim Frontend musste ich aus zeitlichen Gründen von der Idee, Web Components zu verwenden, abrücken. Ich habe mich dazu entschieden, das Frontend mit HTML, CSS und Vanilla JS zu entwickeln, d.h. ohne externe Dependencies und ohne Build-Prozess.
- Beim Backend habe ich andere Design-Ziele verfolgt: Ich habe Libraries verwendet, um mir die Entwicklung möglichst zu erleichtern.
- Für den Umfang der Applikation ist ein Frontend mit Vanilla JS praktikabel. Bei grösseren Frontends gibt es mehr Statusübergänge, was ohne MVC-Architektur schnell unübersichtlich werden könnte.

5 Arbeitsjournal

Datum	Stunden	Bereich	Tätigkeit
01.09.2019	1.0	Dok	Projektidee ausgearbeitet
01.09.2019	1.0	BE	Einarbeiten in Redis Sets
01.09.2019	1.0	Dep	initiale Konfiguration für Docker Compose erstellt
04.09.2019	1.0	Dok	Projektbeschreibung erstellt
05.09.2019	1.0	BE	Backend mit Node.js, Express, Redis-Client und Docker aufgesetzt
05.09.2019	0.5	Dok	Dokument für Zusammenfassung aufgesetzt (Makefile für Pandoc)
12.09.2019	0.5	BE	API-Design erarbeitet
13.09.2019	1.0	BE	Durchstich imt Express.js und Redis: Log-Eintrag erfassen
13.09.2019	1.0	BE	Validierungsfunktionen mit Testfällen entwickelt
15.09.2019	0.5	BE	Löschfunktion für Log-Einträge umgesetzt
15.09.2019	1.0	BE	Tags aller Log-Einträge eines Benutzers und pro Datum ermitteln
15.09.2019	1.5	BE	Map von Datum auf Tagliste erstellen
15.09.2019	0.5	BE	Testfall für Kategorisierung der Tag-Korrelation schrieben
17.09.2019	1.0	BE	Kategorisierung und Phi-Funktion für Tag-Korrelation umgesetzt
18.09.2019	0.5	BE	Korrekturen an Kategorisierung und Tag-Korrelation vorgenommen

Datum	Stunden	Bereich	Tätigkeit
19.09.2019	1.0	BE	Tag-Korrelation und dazugehörigen Endpoint umgesetzt
19.09.2019	0.5	BE	Python-Skript zur Erstellung von Testdaten über HTTP erstellt
20.09.2019	1.0	BE	Authentifizierung/Autorisierung mit bcrypt und JWT implementiert
20.09.2019	0.5	BE	Testdaten-Skripts um AuthN/AuthZ erweitert
22.09.2019	1.0	Dok	Berechnung des Phi-Koeffizienten dokumentiert
23.09.2019	1.0	Dep	Frontend-Deployment erstellt; Abhängigkeiten definiert
23.09.2019	1.0	Dep	Backend erst nach Redis aufstarten; Verbindung umkonfiguriert
24.09.2019	1.0	FE	User-Interface aufgebaut; Login-Formular erstellt; Login-Request
24.09.2019	0.5	BE	CORS konfiguriert für lokales Deployment
25.09.2019	1.0	FE	Login/Logout und Navigation (Single Pager) umgesetzt
26.09.2019	0.5	FE	Navigation und Session-Handling verbessert
26.09.2019	0.5	FE	Generische Funktion für Backend-Requests erstellt
26.09.2019	0.5	FE	Formular zum Erfassen von Journaleinträgen erstellt
26.09.2019	0.5	FE	Journaleinträge aufgrund neuer und bestehender Tags erstellen
26.09.2019	0.5	BE	Endpoint und Testskript für Journaleinträge-Tagesdaten erstellt
27.09.2019	1.0	FE	Journal laden und darstellen
27.09.2019	0.5	FE	Lösch-Funktion für bestehende Einträge implementiert
27.09.2019	1.0	FE	Report-Seite erstellt; Laden und Darstellen der Korrelationen
28.09.2019	0.5	FE	Korrelationen absteigend sortieren und bei jedem Aufruf leeren
28.09.2019	3.0	Dok	Techische Dokumentation (Architektur, Libraries, Deployment)
28.09.2019	1.0	Dok	Fazit und Reflexion
	31.0	**Total**	

(Legende: BE=Backend, FE=Frontend, Dok=Dokumentation, Dep=Deployment)

Total wurden 31 Stunden Arbeitsaufwand geleistet, verteilt auf Backend (13.5 Stunden), Frontend (7 Stunden), Dokumentation (7.5 Stunden) und Deployment (3.0).

6 Quellen

Marjin Haverbeke: *Eloquent JavaScript. A Modern Introduction to Programming. (Third Edition)*