

Projekt Last Minute Camping – Tent Edition

Florian Hannich - Patric Düntzsch - Tobias Beckmann

Zielsetzung

Für die Umsetzung unseres Projekts haben wir uns erneut an folgendem Modell orientiert:



Ziel war eine Datenbank mit der Programmiersprache Python zu erstellen sowie diese ebenfalls via Python zu verwalten. Anschließend sollten die eingepflegten Daten veranschaulicht und mehr oder minder sinnvoll verwendet werden.

Planung und Entscheidungsfindung

Im ersten Schritt der Planung befassten wir uns mit der Frage, was wir darstellen wollen.

Da im ersten Halbjahr bereits ein Datenbank-Projekt vorausgegangen war in welchem wir eine MySQL-Datenbank zum Thema „Last-Minute-Camping“ erstellten, war schnell klar, dass wir hier auch wieder ansetzen könnten.

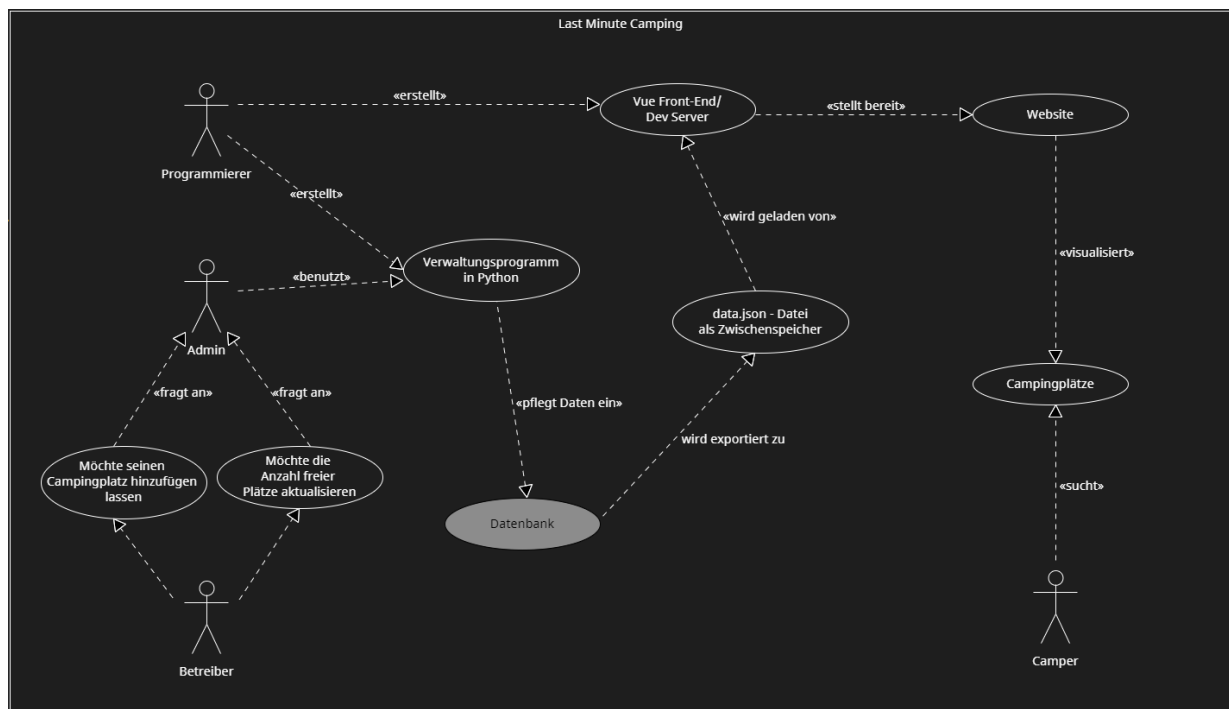
Die Entscheidung war also gefallen.

Da wir in den Wochen zuvor bereits daran gearbeitet hatten eine Datenbank in Python zu erstellen war auch der Ausgangspunkt für das Projekt schnell gefunden.

Um das Projekt überschaubarer und anschaulicher zu halten, als auch aufgrund des knappen Zeitrahmens, entschieden wir uns, keine Kopie der Datenbank aus dem ersten Halbjahr zu erstellen, sondern eine neue, kompaktere Version dieser zu entwickeln. Die „Tent Edition“ war geboren, welche sich ausschließlich auf Camper mit Zelt fokussieren sollte.

Als Nächstes beschäftigten wir uns mit der sinnvollen Verwendung unserer Datenbank und der enthaltenen Daten. In unserem Projekt im ersten Halbjahr wurden viele Informationen in eine Datenbank eingepflegt und konnten auch jederzeit vom betreffenden Administrator verändert werden, aber von außen einsehen, geschweige denn als möglicher Interessent hiermit zu interagieren war leider nicht möglich. Aber wie wollten wir unsere Daten weiterverarbeiten? Da es sich um eine Übersicht zu Campingplätzen sowie deren freie Kapazitäten handelt war der logische Entschluss unsere Daten für interessierte Camper und Camperinnen zur Verfügung zu stellen. Als eine der geläufigsten Varianten wählten wir eine browserbasierte Webansicht, welche dem zukünftigen Kunden eine intuitive und übersichtliche Darstellung unserer Daten vermitteln würde. Die Umsetzung sollte in **Vue.js** erfolgen. Dabei handelt es sich um ein sogenanntes JavaScript-Framework, welches die Verknüpfung von HTML-, CSS- und JavaScript-Elementen stark vereinfacht und so auch den Entwicklungsprozess beschleunigt. Der große Vorteil der von uns gewählten Darstellungsform ist zudem, dass unsere Daten in jedem gängigen Internetbrowser visualisiert werden können. Abschließend stellte sich die Frage, wie wir ein clientseitiges Frontend, welches durch den Webbrowser dargestellt wird mit unserer Datenbank verbinden könnten. Im Normalfall regelt ein Backend den Austausch der Daten zwischen Frontend und der Datenbank. Dies geschieht z.B., um für Sicherheit zu sorgen, da der Quellcode hierdurch nicht einsehbar und manipulierbar wäre oder aber auch um Datenintegrität zu gewährleisten wobei das Backend jeden Datenaustausch validieren würde und dadurch verhindert, dass es bei zeitgleichen Zugriffen verschiedener Benutzer zu Dateninkonsistenzen kommen kann. Da die zeitlichen Fristen für das Projekt schon sehr knapp erschienen, verwarfen wir aber schnell den Plan ein echtes Backend zu integrieren und suchten nach einer Behelfslösung. Die Wahl fiel auf eine JSON-Datei (JavaScript Object Notation) als Zwischenspeicher. Mit dem Entschluss war die grobe Planung abgeschlossen und die Umsetzung konnte beginnen.

UML-Modell der Projektstruktur:



Umsetzung

Wie in der Planung bereits erwähnt, war unser Ausgangspunkt eine im Voraus erstellte „Personendatenbank“ in welcher wir eine mySQL-Datenbank mit Python erstellen und verwalten. Diese Verknüpfung wurde durch die Python-Bibliothek `mysql-connector` realisiert.

Um die Administration der Datenbank in Python zu ermöglichen wurden folgende Funktionalitäten implementiert:

- Eine Datenbank namens `lastMinuteCampingTentEdition` wird erstellt, sofern sie noch nicht existiert
- Eine Tabelle namens `dataTable` wird erstellt, sofern sie in der Datenbank noch nicht existiert
- Ein Datensatz kann hinzugefügt werden, sofern der gewählte Name in der Datenbank noch nicht existiert
- Ein Campingplatz kann anhand des Namens in der Datenbank gesucht werden (zur Steigerung der Übersichtlichkeit werden die Attribute bei der Ausgabe sinnvoll beschränkt)
- Alle vorhandenen Datensätze können ausgegeben werden (zur Steigerung der Übersichtlichkeit werden hierbei die Attribute sinnvoll beschränkt)
- Die Anzahl freier Campingplätze bereits in der Datenbank existierender Datensätze kann geändert werden
- Datensätze können gelöscht werden
- Zur Veranschaulichung können Testdaten in die Datenbank eingetragen werden

Um die Interaktion angenehmer zu gestalten läuft das Programm in einer `while`-Schleife die erst durch eine manuelle Eingabe beendet werden kann.

Innerhalb dieser Schleife befindet man sich in einem Optionsmenü.

Durch Eingabe einer Zahl wird eine bestimmte Option ausgeführt, nachdem anschließend (falls nötig) weitere Parameter hierzu abgefragt werden.

Welche Option sich hinter welcher Zahl verbirgt wird nach jedem Durchlauf, aufgelistet, angezeigt.

Da es sich bei den Optionen um sich wiederholende Abläufe handelt, wurden sie in Form von Funktionen in das Programm implementiert.

Mögliche, folgende Parameter hingegen sind variabel und werden der Funktion daher bei der Ausführung bereitgestellt und berücksichtigt.

```
Was kann ich für dich tun?
1 - Um alle Campingplätze anzuzeigen.
2 - Um einen Campingplatz zu suchen.
3 - Um einen Campingplatz hinzuzufügen.
4 - Um die Anzahl freier Plätze zu bearbeiten.
5 - Um einen Campingplatz zu löschen.
6 - Um Testdaten in die Datenbank einzufügen.
7 - Um das Programm zu beenden.

Welche Option möchtest du durchführen?
Option: █
```

Der Datenbank-Server wird in Form eines **MySQL**-Servers über das frei verfügbare Programm **XAMPP** lokal bereit gestellt.


Sobald eine Funktion aufgerufen wird, welche mit der Datenbank arbeitet, soll unser Python Programm mit Hilfe des **mysql-connectors** mit diesem **MySQL**-Server kommunizieren.

Als relevante Attribute für unsere Datenbank, haben wir uns für die Folgenden entschieden:

- Name des Campingplatzes
- Adresse des Campingplatzes (atomar)
- Telefonnummer
- Öffnungs- und Schließzeiten
- Bewertung
- Preis
- Anzahl freier Zeltplätze
- diverse Extras (WC, dusche, spielplatz, tiereErlaubt, barrierefrei, bademöglichkeit, kiosk, WLAN, strom, waschmaschine)
- Eine BildURL für die spätere Darstellung auf der Webseite

Der Einfachheit halber hat jeder Zeltplatz auf einem Campingplatz zu jeder Zeit den selben Preis, zudem gelten die identischen Öffnungs- und Schließzeiten des ganze Jahr über.

Das Resultat des ersten Aufrufes des Python-Scripts sah nun wie folgt aus:

#	Name	Typ	Kollation	Attribute	Null	Standard	Kommentare	Extra
1	id 	int(11)			Nein	kein(e)		AUTO_INCREMENT
2	name	varchar(50)	utf8mb4_general_ci		Ja	NULL		
3	postleitzahl	varchar(5)	utf8mb4_general_ci		Ja	NULL		
4	ort	varchar(30)	utf8mb4_general_ci		Ja	NULL		
5	straße	varchar(30)	utf8mb4_general_ci		Ja	NULL		
6	hausnummer	varchar(15)	utf8mb4_general_ci		Ja	NULL		
7	telefonnummer	varchar(30)	utf8mb4_general_ci		Ja	NULL		
8	öffnungszeitenAnfang	varchar(5)	utf8mb4_general_ci		Ja	NULL		
9	öffnungszeitenEnde	varchar(5)	utf8mb4_general_ci		Ja	NULL		
10	bewertung	varchar(3)	utf8mb4_general_ci		Ja	NULL		
11	preis	varchar(7)	utf8mb4_general_ci		Ja	NULL		
12	anzahlFreierPlätze	smallint(6)			Ja	NULL		
13	WC	tinyint(1)			Ja	NULL		
14	dusche	tinyint(1)			Ja	NULL		
15	spielplatz	tinyint(1)			Ja	NULL		
16	tiereErlaubt	tinyint(1)			Ja	NULL		
17	barrierefrei	tinyint(1)			Ja	NULL		
18	bademöglichkeit	tinyint(1)			Ja	NULL		
19	kiosk	tinyint(1)			Ja	NULL		
20	WLAN	tinyint(1)			Ja	NULL		
21	strom	tinyint(1)			Ja	NULL		
22	waschmaschine	tinyint(1)			Ja	NULL		
23	bildLink	varchar(255)	utf8mb4_general_ci		Ja	NULL		

Durch die Implementierung einer **seeder**-Funktion ist es möglich vordefinierte Testdaten in die Datenbank zu integrieren, welche wir zur Anschauung verwenden wollen.

id	name	postleitzahl	ort	straße	hausnummer	telefonnummer	öffnungszeitenAnfang	öffnungszeitenEnde	bewertung	preis	anzahlFreierPlatze	WC	duche	spielplatz	tiereFlaubt	barrierefrei	bademöglichkeit	kiosk	WLAN	strom	waschmaschine	bildLink
1	Nature camping	18423	Nature	Naturestreet	5	0436543	08.00	24.00	4.3	22.00	11	1	1	0	1	1	1	1	0	1	1	https://www.unwes.com/objectImage/0000172/0a48...
2	Forest camping	24500	Neuwald	Neuer waldweg	13	04046307387438543	08.00	21.00	4.5	20.00	7	1	1	1	1	1	0	1	1	1	1	https://neuehacamping.johnsonoutdoors.com/sites/0e...
3	Nik camping	24788	Rendsburg	Kanalstraße	66a	04356374356	06.00	22.00	3.9	12.80	35	1	1	1	1	0	1	0	0	1	1	https://medialles.ufaebagaru.de/wp-content/uploa...
4	Neumünster camping	24536	Neumünster	Hauptstraße	18a	0785496732	07.00	21.00	3.3	16.50	52	1	1	0	0	1	0	1	1	1	1	https://www.trip.de/bilder/2020/09/04/9037396/2389...
5	Bordesholm camping	25235	Bordesholm	Dorfstraße	12	075837443543	08.00	20.00	4.6	13.50	24	1	1	1	1	1	1	0	0	0	0	https://blog-6a0.kocdn.com/wp-content/uploads/202...
6	Kieler campingparadies	23164	Kiel	Ächterbahn	65	04376587353	10.00	19.00	2.3	75.60	45	1	1	0	0	0	0	1	0	0	0	https://encrypted-tbn0.gstatic.com/images?q=tbn:ANL...

Als nächstes befassten wir uns mit der Überführung unserer Daten in ein Front-End. Bei uns sollte dies durch das JavaScript-Framework Vue realisiert werden.

Dadurch, dass das Front-End aber nicht direkt auf die Datenbank zugreifen kann, haben wir uns mit der Erstellung einer **data.json** Datei als Zwischenspeicher beholfen. Diese Datei wurde so implementiert, dass sie, jedes Mal wenn wir Daten in die Datenbank übertragen oder vorhandenen Datensätze verändern, geleert wird, die Datenbank erneut ausgelesen wird und die Daten neu in die data.json geschrieben werden.

Die Funktion welche die Daten aus der Datenbank in data.json schreibt sieht dabei folgendermaßen aus:

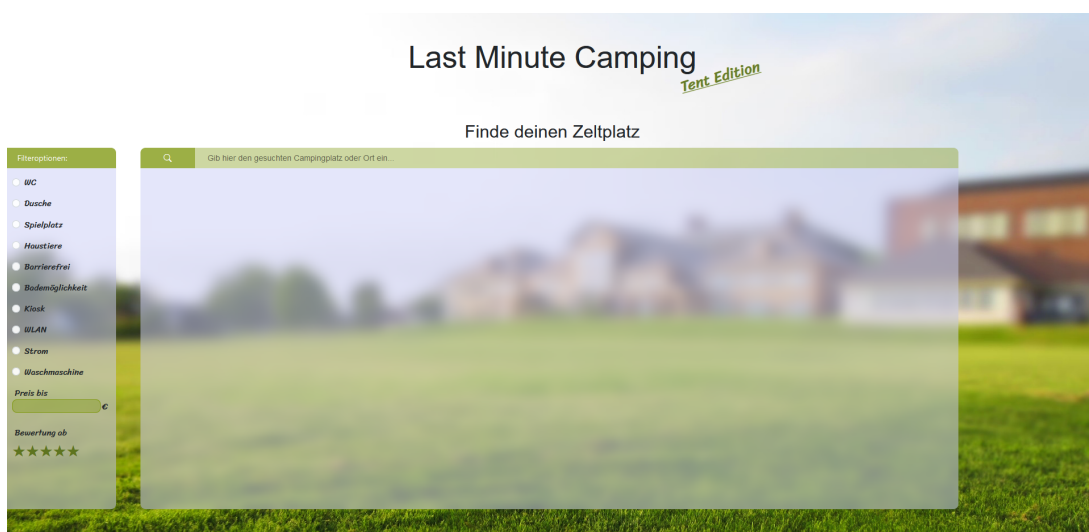
```
def writeDatabaseToJSON():
    mydb = mysql.connector.connect(
        host="localhost", user="root", database="lastMinuteCampingTentEdition"
    )
    mycursor = mydb.cursor()
    mycursor.execute("SELECT * FROM dataTable")
    tabelle = mycursor.fetchall()
    data = []
    for datensatz in tabelle:
        dateaObj = {
            "id": datensatz[0],
            "name": datensatz[1],
            "plz": datensatz[2],
            "ort": datensatz[3],
            "straße": datensatz[4],
            "hausnummer": datensatz[5],
            "telefonnummer": datensatz[6],
            "öffnungszeitenAnfang": datensatz[7],
            "öffnungszeitenEnde": datensatz[8],
            "bewertung": datensatz[9],
            "preis": datensatz[10],
            "anzahlFreierPlaetze": datensatz[11],
            "WC": datensatz[12],
            "duche": datensatz[13],
            "spielplatz": datensatz[14],
            "haustiere": datensatz[15],
            "barrierefrei": datensatz[16],
            "bademöglichkeit": datensatz[17],
            "kiosk": datensatz[18],
            "WLAN": datensatz[19],
            "strom": datensatz[20],
            "waschmaschine": datensatz[21],
            "bildURL": datensatz[22],
        }
        data.append(dateaObj)
    with open('data.json', 'w') as f:
        json.dump(data, f)
```

Als Ergebnis erhalten wir folgendes Datenformat mit welchem wir in Vue weiter arbeiten können:

```
{
  "id": 1,
  "name": "Nature camping",
  "plz": "18403",
  "ort": "Nature",
  "stra\u00dfe": "Naturestreet",
  "hausnummer": "5",
  "telefonnummer": "0436543",
  "oeffnungszeitenAnfang": "00:00",
  "oeffnungszeitenEnde": "24:00",
  "bewertung": "4.3",
  "preis": "22.00",
  "anzahlFreierPlaetze": "11",
  "wc": 1,
  "dusche": 1,
  "spielplatz": 0,
  "haustiere": 1,
  "barrierefrei": 1,
  "badem\u00f6glichkeit": 1,
  "kiosk": 1,
  "wlan": 0,
  "strom": 1,
  "waschmaschine": 1,
  "bildURL": "https://www.usnews.com/object/image/00000172-0a48-dd19-af73-dfc9adc60000/1-Intro.jpg?update-time=1589312815886&size=responsive640"
}
```

Durch die implementierung von **node.js** und des **node package managers** (npm) ist es möglich, lokal, einen live-Server zu starten, welcher das Vue "Template" (welches als eine Art HTML-Äquivalent fungiert) im Browser visualisiert. Dadurch kann jeder Entwicklungsschritt in Vue, live im Browser verfolgt werden was die Entwicklung selbst wiederum deutlich angenehmer gestaltet.

Die Zielsetzung unseres Front-Ends war es, einen Anlaufpunkt für Interessenten zu erstellen, durch welchen der Benutzer die Datenbank nach für ihn relevanten Einträgen durchsuchen und filtern kann. Gleichzeitig sollte auch das Layout den Benutzer nicht allzu sehr abschrecken. Hierfür wurde am linken Rand der Website ein Abschnitt erstellt, welcher nach den von uns in Python definierten Extras filtert (da es sich um Daten des Typs Boolean handelt wird hier zwischen True und False unterschieden). Zusätzlich beinhaltet der Abschnitt einen Filter im welchem Bewertungen von **1 - 5** Sternen auswählbar sind sowie einen Filter, in welchem ein maximaler Preis pro Tag und Zeltplatz eingegeben werden kann. Im oberen Bereich der Seite finden wir den Namen unseres Projekts sowie ein Suchfeld darunter. Dieses kann verwendet werden um nach Campingplatznamen sowie nach Orten zu suchen. Im unteren Bereich befindet sich der wichtigste Teil der Webseite. Ein Container welcher alle in der Datenbank enthaltenen Datensätze wiedergibt, sofern kein Filter angewählt oder Text in das Suchfeld eingegeben wurde.



Damit war die Umsetzung abgeschlossen.

Bewertung

Die Zielsetzung lautete:

- "Die Programmiersprache Python verwenden"
 - Wurde erfüllt. Python fungiert bei uns als Eingabe- und Verwaltungswerkzeug z.B. für einen Administrator
- "Speichern von Daten in einer Datenbank (SQLite3, MySQL oder ähnliches)"
 - Wurde ebenfalls erfüllt. Unsere Datenbank wurde in **MySQL** erstellt.
- "sinnvolle Verwendung der Daten"
 - Auch dieses Ziel wurde mehr oder minder erfüllt. Unsere Daten dienen als Informationsquelle für Camper die auf der Suche nach einem Zeltplatz sind. Die Website existiert allerdings nur lokal, zu Anschauungszwecken.

Wie zu erwarten lief aber auch bei diesem Projekt nicht alles so wie geplant:

- Für die Uhrzeiten der Attribute **ÖffnungszeitenAnfang** und **ÖffnungszeitenEnde** hätte man besser den Datentypen **TIME** verwendet. Die Integration in die data.json Datei führte hierbei allerdings zu Fehlern, sodass wir uns für den Datentypen **VARCHAR(5)** entschieden.
- Den Bewertungen hätte man außerdem besser den Datentypen **DECIMAL(3, 1)**, sowie dem Preis besser den Datentypen **DECIMAL(7, 2)** zugewiesen, aber auch hier führte die Integration in die data.json Datei zu Fehlern, sodass wir uns hier für die Datentypen **VARCHAR(3)** bzw. **VARCHAR(7)** entschieden.
- Da alle Bild-URLs unserer Testdaten weniger als 255 Zeichen besitzen, haben wir hierfür **VARCHAR(255)** gewählt. In der Realität gibt es aber teilweise auch deutlich längere URLs, sodass man hierfür eventuell besser den Datentypen **TEXT** hätte verwenden sollen. Dies hat jedoch auch Einfluss auf die Performance des Datenaustausches.