favBB Dokumentation

Die Idee hinter FavBB ist es das Suchen nach ÖV Verbindungen zu vereinfachen. Durch Speichern von Favoriten, die in einem Dashboard angezeigt werden. Ausserdem ist es das Ziel ein Tech Stack auszuprobieren der beim Arbeitgeber zur Diskussion steht um eine interne Applikation zu entwickeln.

Tech Doku

Die Tech Doku beinhaltet den Scope und die Dokumentation der Umsetzung des Frontend und Backend.

Scope

Die Webseite soll drei Seiten beinhalten. Zwei Seiten sind dafür da Verbindungen zu suchen, wobei die einte eine textbasierte und die zweite eine kartenbasierte Suche anbietet. Die dritte Seite ist das Dashboard, wo gespeicherte Verbindungen angezeigt werden und aktuelle Verbindungen auflistet. Als Technologie wird Mithril verwendet. Mithril ist eine sehr schlanke UI Library die einen virtuellen DOM pflegt und nur veränderte Teile neu zeichnet. Ausserdem ist eine Request API vorhanden die es einem erlaubt ein Backend einfach anzusprechen. Ausserdem wird für das definieren des HTML JavaScript verwendet, was aber gar nicht kompliziert ist, da es sehr an ein HTML Dokument erinnert.

```
m("span",
    m("i", { class: "fas fa-map-marker-alt" }),
    "  " + section.departure.station.name + "  "
)
```

Beispiel eines 'span' welches ein Icon und dynamischer Text enthält

Als Backend soll im Projekt Postgres mit PostgREST (eine automatische REST API für Postgres Datenbanken) eingesetzt werden. Dies ist als kleines Proof-of-Concept gedacht, um vor allem PostgREST einmal zu testen. Alternativ zum Backend ist eine Client seitige Speicherung vorgesehen. Dies bringt den Vorteil, dass auf dem Backend nur ein kleiner Webserver notwendig ist der das HTML, JavaScript und CSS ausliefert.

Backend

Als Backend wird Postgres mit PostgREST verwendet. Das Ganze läuft in zwei Docker Container. Zusätzlich dazu läuft SwaggerUI um die REST API direkt zu Visualisieren. Das Ganze wurde ausgiebig lokal getestet und sieht vielversprechend aus. Die Funktionalität von PostgREST überzeugt vollumfänglich.

Datenbank Schema

Für den Anfang reicht ein simples Datenbankschema. Es gibt Benutzer die eine gewisse Anzahl Verbindungen speichern können. Nachdem das Schema in der Postgres Datenbank erstellt wurde (via PgAdmin 4), wurde ein Backup Script erstellt, dass von da an dafür gebraucht werden konnte, um die Datenbank zu sichern und auch wiederherzustellen.

Frontend

Die Frontend Toolchain besteht hauptsächlich aus NPM, Mithril, Bulma (CSS), TypeScript und browserify. Mithril beinhaltet einen Router, der es erlaubt alle Seiten an einem Ort zu definieren.

```
m.route(appDiv as Element, '/', {
  '/': Dashboard,
  '/dashboard': Dashboard,
  '/search': Search,
  '/mapsearch': MapSearch
});
```

Dashboard

Das Dashboard zeigt pro gespeicherte Verbindung eine Liste der aktuell verfügbaren ÖV Verbindungen zwischen zwei Orten.

Search

Die Search Seite hat zwei Felder, um Verbindungen zu suchen. Beim Tippen wird dynamisch nach Stationen gesucht. Sobald diese ausgewählt sind werden auch schon Verbindungen angezeigt ohne dass diese gespeichert werden müssen.

MapSearch

Die MapSearch Seite ist noch nicht ganz fertig. Das Ziel ist es das man den Abfahrtsort und Ankunftsort in einer Map auswählen kann und optional Speichern kann. Momentan funktioniert das noch nicht.

Components

Für das UI sind Komponenten notwendig. Für dieses Projekt wurde zum Beispiel die "Searchable List" entwickelt. Zusätzlich noch einige kleine Komponenten für die Darstellung von Details wie Verbindungen und der Sektionen (Search Seite).

Searchable List

Die "Searchable List" ist die grösste Komponente, die entwickelt wurde. Es wurde versucht das UI und die Logik zu trennen. Um das zu erreichen bietet die "Searchable List" Callback Funktionen an, die es erlauben den Inhalt anzupassen sobald etwas eingetippt wird oder auf die Auswahl eines Elementes zu reagieren. Alle Informationen kommen vom Parent Element und somit ist die Kopplung zwischen UI und der Logik minimal.

Services

Für die Anbindung an die TransportAPI und für das Speichern der Verbindungen auf dem Client wurden zwei Services implementiert.

Transport API

Die TransportAPI wird von "Opendata Schweiz" zur Verfügung gestellt und beinhaltet sämtliche ÖV Informationen. Für diese Projekt sind vor allem die Verbindungen interessant und die möglichen Abfahrts- bzw. Ankunftsorte. Dafür reichen zwei Requests.

```
import m from 'mithril';

let query = "Luzern";
let result = await m.request({
    method: "GET",
    url: "http://transport.opendata.ch/v1/locations?query="+query
});

let from = "Luzern";
let to = "Basel SBB";
let result = await m.request({
    method: "GET",
    url: `http://transport.opendata.ch/v1/connections?
from=${from}&to=${to}`
});
```

Storage

Für das Speichern der Verbindungen auf dem Client wird "localforage" eingesetzt. Das ist eine Library die Abstraktionen über all die Möglichen Speichermöglichkeiten auf dem Client anbietet. Es kann ausgewählt werden wo gespeichert werden soll oder es kann ganz automatisch passieren je nach den verfügbaren Möglichkeiten. Das Interface ist ziemlich Einfach und besteht aus einem Key / Value Store.

Fazit

Das Projekt war extrem spannend und es wurden wichtige Erkenntnisse bezüglich dem Tech Stack gesammelt. Leider hat am Schluss einiges noch gefehlt für eine voll funktionsfähige Webseite. Die Backend Anbindung wurde nicht gemacht und nur ein Client seitiger Storage implementiert. Auch die einzelnen Seiten müssen noch verbessert werden. Jedoch wurde mit Mithril eine sehr interessante Bibliothek ausprobiert, die überblickbar, schnell und vielseitig ist. Zusätzlich wurde mit der "Searchable List" eine sehr generische Komponente entwickelt die in anderen Projekten wieder eingesetzt werden kann. Durch den Einsatz der TransportAPI konnte zusätzlich eine REST API Anbindung realisiert werden, was eine gute Erfahrung war und auch eine gute Architektur forderte, um das UI und die REST API Anbindung gut zu trennen.

Reflexion

Das Projekt war von Anfang an sehr ambitioniert und somit wurde ein gewisses Risiko eingegangen nicht alles zu erreichen was geplant war. In Zukunft gilt es realistischer zu planen und lieber einige Features als Optionen offen zu lassen. Trotzdem wurde extrem viel gelernt und das ist wohl das wichtigste bei so einem Projekt. Im Backend mit Postgres & PostgREST und im Frontend mit Mithril und der TransportAPI. Nicht zu vergessen mit der Toolchain rund um Bulma (CSS), NPM, browserify und TypeScript. Schlussendlich bleibt eine gewisse Verunsicherung was den Tech Stack angeht. Das Backend macht sicherlich Sinn wenn eine Postgres Datenbank eine REST API Schnittstelle bekommen soll. Von Anfang an solch eine Lösung anzustreben ist diskussionswürdig, wenn nicht zwingend eine relationale Datenbank im Hintergrund benötigt wird. Im Frontend ist die Frage aber nicht so einfach zu beantworten. Es gibt so viele Frameworks. Im Unterricht wurde mit Angular ein recht populäres Framework angeschaut und auch andere wie React und Vue sind sehr populär. Was aber auffällt ist, dass diese Frameworks recht gross sind und einen in eine gewisse Abhängigkeit bringen. Um dies zu Umgehen wurde Mithril evaluiert. Der Vorteil ist, dass der Source Code recht klein ist und somit auch selber verwaltet werden kann, falls das Projekt eingestellt wird oder gravierende Änderungen passieren die nicht eingepflegt werden können. Das bringt eine gewisse Sicherheit, da die Abhängigkeit nicht mehr so gross ist. Als Entwickler sollte man sowieso aufpassen sich nicht zu abhängig von Frameworks und Libraries zu machen, die man im Notfall nicht selber verwalten kann.

Arbeitsjournal

- Setup Postgres, PostgREST and Swagger UI with Docker ~ 2h
- Create first DB Schema and Script to Backup/Restore ~ 1h
- Mithril & Bulma.io Learning ~ 3h
- Searchable List ~ 3h
- Transport API Service ~ 4h
- Sections ~ 2h
- Map ~ 4h
- Dashboard ~ 4h
- Backend Tests ~ 4h
- Doku & Präsentation ~ 3h

Total

Im gesamten wurden rund 30 Stunden in das Projekt investiert. Nicht eingerechnet sind einige zusätzliche Recherchen und konsultieren der Dokumentation.

Abschluss

Der Ganze Source Code und die Dokumentation ist auf Github verfügbar.

Author

Steve Ineichen, steve.ineichen@stud.hslu.ch