Fachhochschule Graubünden
University of Applied Sciences

Fahrplan-App

nick.schneeberger@fhgr.ch

Fahrplan-App

Bern

Chur

VERBINDUNG SUCHEN | ALLES LÖSCHEN

Abfahrt: 11:02 Uhr
Ankunft: 13:23 Uhr

Gleis: 2

Dauer: 02:21h

👉 Live Demo auf
interaktivemedien.nickschnee.ch/fahrplan

Themen

Arbeit mit öffentlichen ÖV-Daten der
Swiss public transport API in Echtzeit

Eventlistener, Bedingungen, Schleifen, Funktionen

Dynamisches Einfügen von HTML mit JavaScript

Wähle deinen Weg

All in One

Baue die Fahrplan-App mithilfe der Transport API nach.

- Das ist alles. Let's go!

Step by Step

Löse Schritt für Schritt die Übungen auf den folgenden Seiten.

- Verwende die gleichen Funktions- und Variablennamen, um Verwirrung vorzubeugen. Für jede Teilaufgabe ist die Lösung auf Moodle abgelegt.

Step by Step

Fahrplan-App

00.01 Lerne die API kennen

In dieser Übung arbeiten wir mit der Swiss Public Transport API. Deshalb ist es wichtig, dass wir deren Funktionsweise verstehen.

Dafür gibt es 2 Möglichkeiten:

- Die Dokumentation lesen
- Mit verschiedenen Anfragen experimentieren

Auch erfahrene Programmierer verstehen eine API nur, wenn sie die Dokumentation überfliegen und mit der API experimentieren.

00.02 Lerne die API kennen

Stöbere in der Dokumentation

transport.opendata.ch/docs.html

Experimentiere mit folgenden Abfragen (in Firefox):

http://transport.opendata.ch/v1/locations

http://transport.opendata.ch/v1/locations?query=Basel

http://transport.opendata.ch/v1/connections

http://transport.opendata.ch/v1/connections?from=Lausanne&to=Genève

http://transport.opendata.ch/v1/locations?x=47.476001&y=8.306130

00.03 Lerne die API kennen

Beantworte folgende Fragen nur mithilfe der API und Firefox:

- Wie lange dauert die Fahrt von Basel nach Zürich?
- Auf welchem Gleis fährt der nächste Zug von Genf nach St. Gallen?
- Welche Station befindet sich in der Nähe der Koordinaten 46.948832 | 7.439136 ?

- Wie kann man in der API nach Stationsnamen suchen?

01.01 Vorbereitung

Öffne die Vorlage Fahrplan_01_Aufgabe, um zu beginnen.

In dieser Übung arbeiten wir nur in control.js.

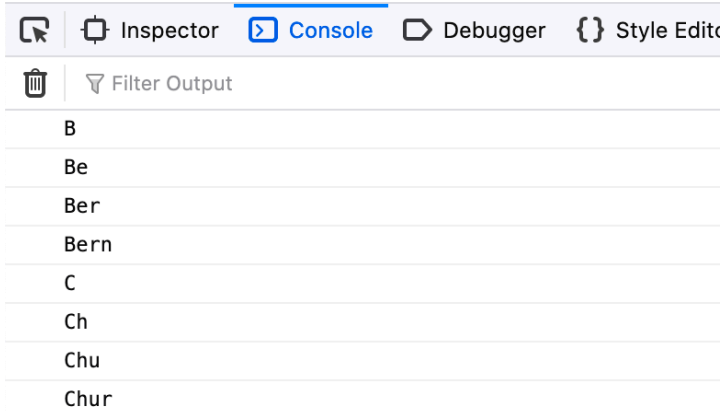
Alle HTML und CSS Dateien sind vorbereitet und final.

Überfliege index.html und style.css, um den Aufbau der Übung zu verstehen.

☝ In den Teilaufgaben musst du viele neue Funktionen erstellen. Damit bleibt der Code übersichtlich. Damit du weisst, welche Funktionen es gibt und wie diese in der Lösung angeordnet sind, gibt es in funktionen.js eine Übersicht.

Fachhochschule Graubünden
University of Applied Sciences

01.02 EventListener



Für beste User-Experience sollen während dem Tippen bereits Ortsvorschläge gemacht werden. Dafür müssen die zwei Input Felder merken, wenn getippt wird.

Speichere die Werte der Inputfelder bei jedem Tastenanschlag in einer Variable meinInput und gib den Wert in der Konsole aus.

- Welchen EventListener brauchen wir?
- Füge in control.js je einen EventListener pro Input-Feld hinzu
- Speichere den Wert in der Variable meinInput
- Gib meinInput in der Konsole aus

02 Funktionen

Die gespeicherten Suchbegriffe in meinInput wollen wir an die API senden, um mögliche Treffer abzufragen. Zuerst müssen wir unseren Code dafür vorbereiten.

Definiere zwei neue Funktionen fetchStationenVon() und fetchStationenNach() ganz unten im Dokument. Rufe die neuen Funkionen innerhalb der Eventlistener auf und übergib ihnen den Wert von meinInput als Parameter. Teste den Code, in dem du meinInput innerhalb der jeweiligen Funktion in der Konsole ausgibst.

- Wie kannst du einer Funktion eine Variable als Parameter übergeben?
- Damit es mit den Namen der Funktionen kein Durcheinander gibt, sind diese in funktionen.js aufgelistet. Kopiere neue Funktionen am besten von dort.

03 Fetch!

Nun haben wir alles vorbereitet, um die erste API Abfrage zu machen. Dafür senden wir den Wert von meinInput an die API und speichern das Resultat, welches uns diese zurückgibt in einer Variable.

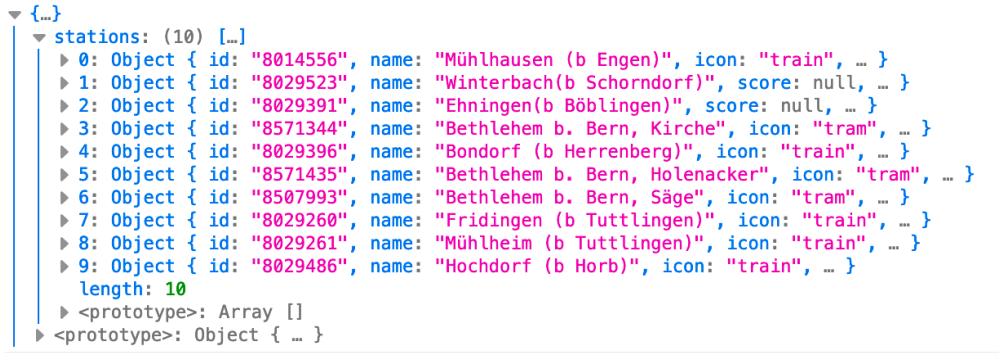
Innerhalb von fetchStationenVon(): Trage meinInput als URL in der Methode fetch() ein und speichere das Resultat als json in der Variable data. Gib data in der Konsole aus.

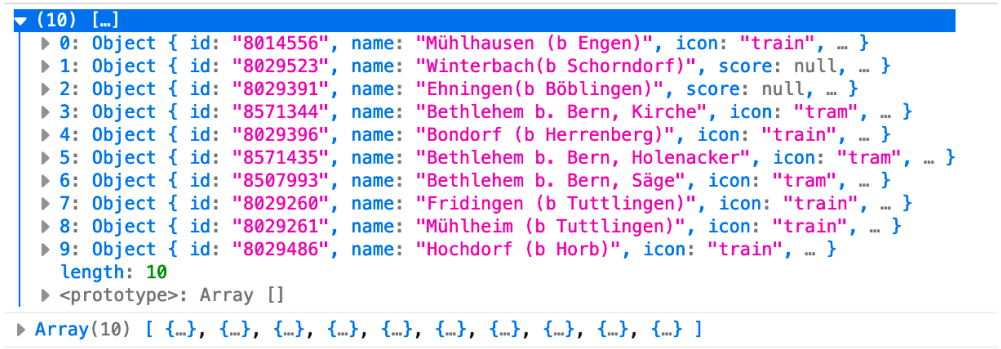
- Ohne Hilfe der Dokumentation kannst du diese Aufgabe nicht lösen.
- Kopiere die URL aus der Dokumentation im Abschnitt /locations, um zu beginnen. Ersetze den Query durch deine Variable.
- Setze den 'Type' im Query auf "station", um Adressen und Sehenswürdigkeiten auszublenden.

04 Explore!

Wir wollen bloss die Namen der Stationen im HTML anzeigen. Mit unserer Abfrage erhalten wir aber ein ganzes Paket zurück. Wo verstecken sich die Stationsnamen im Paket?

Speichere alle 10 Resultate im Array stationen. Erstelle der Übersicht halber eine neue Funktion zeichneDropdownVon() und übergib ihr den Array stationen. Rufe die neue Funktion bei erfolgreichem fetch() auf und gib den Array stationen innerhalb der neuen Funktion in der Konsole aus.





05 Loop!

Um alle Stationsnamen des Arrays stationen im HTML anzuzeigen, brauchen wir eine Loop.

Loope durch den Array stationen und füge alle Stationsnamen als Listen-Elemente <li> ins HTML ein. Gib den Elementen die Klasse .dropdown-von . Füge jedes neue Element mittels appendChild() dem Parent-Element mit der ID #dropdown-von an.

- Füge vor der Loop eine Zeile ein, die den Inhalt von #dropdown-von bei jeder neuen Anfrage löscht. Sonst wird die Liste der Resultate immer länger.
- Schau dir in index.html die Struktur an. Diese ist schon vorbereitet und muss nicht bearbeitet werden.

06 Klick mich

Damit Userinnen und User das richtige Suchresultat aus der Liste auswählen können, müssen wir die Stationsnamen in die Input-Felder eintragen, sobald jemand darauf klickt.

Erstelle einen Klick-Eventlistener für alle Stationsnamen, die wir vorher ins HTML geschrieben haben. Alle benötigten Elemente wählen wir über die Klasse .dropdown-von aus.

Ersetze den vom User eingetippten Value im Inputfeld #input-von mit dem angeklickten Stationsnamen.

- Erstelle dafür der Übersichtlichkeit halber eine neue Funktion addKlickListenerVon().
- Rufe die neue Funktion nach der Loop innerhalb der Funktion zeichneDropdownVon() auf.

07 Double Up

Bisher haben wir nur das Feld #input-von interaktiv gemacht.

Kopiere und adaptiere alle Funktionen, so dass ebenfalls im Feld #input-nach Stationen gesucht und ausgewählt werden können.

08 Verbindungen suchen

Mit einem Klick auf den Button VERBINDUNG SUCHEN sollen die nächsten Verbindungen angezeigt werden.

Erstelle einen Eventlistener auf dem Button VERBINDUNG SUCHEN. Überprüfe, ob Abfahrts- und Ankunftsort eingegeben wurden. Wenn ja, rufe eine neue Funktion fetchVerbindungen() auf, fetch() die Verbindungen und gib die Ergebnisse in der Konsole aus. Wenn keine Orte eingegeben wurden, gib einen Fehler in der Konsole aus.

- Ohne Hilfe der Dokumentation kannst du diese Aufgabe nicht lösen.
- Kopiere das Beispiel aus der Dokumentation im Abschnitt /connections, um zu beginnen. Ersetze den Query durch deine Variable.
- Es lohnt sich, den Query für die API bereits im EventListener zusammenzusetzen und als ganzes der neuen Funktion fetchVerbindungen() zu übergeben.

09 Verbindungen anzeigen

Nun müssen wir die Verbindungen nur noch ins HTML übertragen. Gehe hierfür analog zu den Teilaufgaben 04 und 05 vor.

Erstelle eine neue Funktion zeichneVerbindungen(). Loope durch den Array verbindungen und füge für jede Verbindung einen Div-Container ins HTML ein. Gib dem Container die Klasse .verbindung . Füge jeden neuen Container mittels appendChild() dem Element mit der ID #verbindungen an.

- Damit die Container angezeigt werden, brauchen sie einen Inhalt. Speichere dafür den Wert platform aus der API in der variable gleis und schreibe diese mittels innerHTML in die generierten Div-Container.

10 Daten lesbar machen

Leider können wir nicht alle Werte so einfach wie das Gleis ins HTML schreiben. Abfahrt, Ankunft und die Fahrzeit müssen wir zuerst lesbar machen. Hier hilft uns eine kurze Google-Suche.

Definiere die Variablen abfahrt, ankunft, dauer und gleis und speichere darin die entsprechenden Resultate aus der API. Wandle departureTimestamp, arrivalTimestamp und duration in Lesbare Werte um.

Füge alle Variablen zu einem String zusammen und schreibe den ganzen String mittels innerHTML in die jeweiligen Elemente.

- Stackoverflow hilft bei verbreiteten Problemen, wie der Umwandlung von Timestamp zu lesbarem Datum / Zeit.
- Findest du heraus, wie du die dauer auf die relevanten Teile trimmst?

11 Aufräumen & Abschluss

Zum Schluss soll die Fahrplan-App für die nächsten Verbindungsabfragen wieder zurückgesetzt werden können.

Programmiere den Button ALLES LÖSCHEN so, dass die Inhalte der beiden Inputfelder, der Dropdowns und der Verbindungen gelöscht werden.

Ebenfalls sollen bei erneutem Klicken auf den Button VERBINDUNG SUCHEN die vorherigen Inhalte aus dem Container #verbindungen gelöscht werden. Schreibe die entsprechende Code-Zeile vor der Loop in der Funktion zeichneVerbindungen().

12 Ausblick

- *Um den Code möglichst wenig komplex zu halten, haben wir einigen doppelten Code produziert. Wie liessen sich die Doppelspurigkeiten vermeiden?*
- *Welche Features könnten wir mit den Daten aus der API noch in die Fahrplan-App einbauen?*