Maturaarbeit

Programmieren eines Frameworks für digitale Schnitzeljagden als Progressive Web App



Nicolas Tscherter

N19c

Betreut von R. Kretzschmar

Kantonsschule Solothurn

2022

Inhalt

[Kurzfassung iii](#_Toc118492563)

[Danksagung iii](#_Toc118492564)

[1 Einleitung 1](#_Toc118492565)

[2 Bestehende Angebote und Stand der Technik 2](#_Toc118492566)

[2.1 Geocaching 3](#_Toc118492567)

[2.2 Actionbound 3](#_Toc118492568)

[2.3 Analyse 4](#_Toc118492569)

[3 Zielsetzung 4](#_Toc118492570)

[4 Vorgehensweise 5](#_Toc118492571)

[4.1 Agiles Projektmanagement 5](#_Toc118492572)

[4.2 Herausforderungen 6](#_Toc118492573)

[5 Funktionen und Funktionsweise 7](#_Toc118492574)

[5.1 Postentypen 8](#_Toc118492575)

[5.2 Das Renderingsystem 9](#_Toc118492576)

[5.3 Interaktionen und Spiellogik 10](#_Toc118492577)

[5.4 Speicher 11](#_Toc118492578)

[5.5 Offlinebetrieb 12](#_Toc118492579)

[5.6 Backend / API 13](#_Toc118492580)

[6 Programmbibliotheken 14](#_Toc118492581)

[7 Diskussion 15](#_Toc118492582)

[7.1 Mögliche Ausbauschritte 15](#_Toc118492583)

[7.2 Alternative Verwendungen 16](#_Toc118492584)

[7.3 Bekannte Mängel 17](#_Toc118492585)

[7.4 Sicherheitsaspekte 17](#_Toc118492586)

[7.5 Fazit 18](#_Toc118492587)

# Kurzfassung

Das Ziel dieser Arbeit war das Realisieren von Online-Schnitzeljagden, bei denen die Spielenden während ihrer Suche zu Orten geführt werden und Aufgaben lösen. Dazu wurde ein Framework erstellt, welches es erlaubt, ohne Programmierkenntnis Posten und Routen festzulegen. Dieses wurde so konzipiert, dass Posten beliebig miteinander verknüpft werden können. Die Reihenfolge der Posten und die Hilfestellungen können dem Spielverlauf angepasst werden. Die entstandene Plattform stellt einen Proof of Concept dar, welcher die Umsetzbarkeit eines solchen Projekts demonstriert. Die Plattform kann grundsätzlich auch als touristischer Städte-Guide, als Begleit-App für ein Museum oder im Schulkontext eingesetzt werden.

# Danksagung

Zuerst möchte ich meinem Betreuer Ralf Kretzschmar für die Ermöglichung dieser Arbeit danken. Er hat mich während der langen gemeinsamen Zeit kompetent beraten und mich mit viel Geduld davon abzuhalten versucht, mir zu hohe Ziele zu setzen. Ein grosses Merci geht auch an all meine Freunden an der Kantonsschule Solothurn, die mich moralisch unterstützt haben. Ich mochte mich auch bei meiner Familie bedanken, insbesondere meinen Eltern Vincent Tscherter und Elisabeth Enggist. Sie haben nicht nur diese Maturaarbeit unzählige Male korrekturgelesen und wertvolle Anregungen geliefert, sondern mich in schulischen und nichtschulischen Angelegenheiten immer wieder unterstützt und ermutigt.

# Einleitung

Schnitzeljagden, bei denen man Rätsel lösend Hinweisen zu einem Ziel folgt, faszinieren mich schon lange. Besonders spannend sind sie, wenn sie wie Spielbücher[[1]](#footnote-1) aufgebaut sind, wo man einen direkten Einfluss auf den Ablauf hat und wenn komplexe Interaktionen zwischen Gruppen ermöglicht werden. Da mich komplexe Systeme interessieren und ich die existierenden Angebote mit einer meist linearen Postenabfolge (Abbildung 1) für zu eingeschränkt halte, beschloss ich, selbst eine digitale Plattform für Schnitzeljagden zu erstellen. Diese Arbeit beschreibt die Plattform und das dafür programmierte Framework. Das Framework ist erweiterbar konzipiert, sodass es um neue Aufgabentypen und Logikkomponenten erweitert werden kann. Ausserdem erlaubt es nicht lineare und vom Benutzereingaben abhängige Aufgabenabfolgen (Abbildung 2). Für die Nutzerinnen und Nutzer präsentiert sich die Plattform als Progressive Web App[[2]](#footnote-2) (PWA). Das ermöglicht es, Schnitzeljagden auf den meisten mit einem modernen Webbrowser ausgestatteten mobilen Geräten abzurufen und nach der Installation ohne Internetverbindung, also offline, zu verwenden. Neue Schnitzeljagden können ohne Programmierkenntnisse erstellt werden, indem Parameter für verschiedene Aufgabentypen und Abfolgen in eine Konfigurationsdatei geschrieben werden.

|  |  |
| --- | --- |
| © swisstopo, map.geo.admin.ch | © swisstopo, map.geo.admin.ch |
| Abbildung : Lineare Postenabfolge | Abbildung : Nicht lineare oder chaotische Postenabfolge |

# Bestehende Angebote und Stand der Technik

Zunächst wurden bereits existierende Schnitzeljagd-Plattformen und -Anbieter untersucht, um eine fundierte Entscheidung über die erforderlichen Funktionen der in dieser Arbeit entwickelten Plattform treffen zu können. Der Fokus lag bei Plattformen, wo Nutzerinnen und Nutzer eigene Schnitzeljagden erstellen und publizieren können. Unter diese Kategorie fallen Actionbound (siehe Abschnitt 2.2) und Scavify[[3]](#footnote-3), beides Plattform mit eigener App, auf der eigene interaktive Schnitzeljagden oder touristische Guides erstellt und veröffentlicht werden können, sowie Geocaching (siehe Abschnitt 2.1), eine Form der Schnitzeljagd, bei der hauptsächlich darum geht, versteckte Behälter mit einem Logbuch zu finden.

Daneben wurde der in der Schweiz bekannte kommerzielle Anbieter Foxtrail[[4]](#footnote-4) untersucht. Foxtrails sind Schnitzeljagden, die in Tourismusregionen zur Erkundung von Städten und Landschaften angeboten werden. Zudem nahm ich die Gelegenheit wahr, vergleichbare Angebote von «Basel Tourismus» und der Stadt Porrentruy im Jura auszuprobieren. Foxtrails wie auch die Städteangebote funktionieren mit Papierunterlagen und festinstallierten Anlagen. Ähnlich verhält es sich mit dem Geocaching. Die Instruktionen werden lediglich über das Web verteilt. Bei der Suche können die Instruktionen aber ebenso gut auf Papier festgehalten sein, da sie nur aus Text- und Bildinhalten bestehen. Anders ist es bei Actionbound und Scavify. Beide machen extensiven Gebrauch der digitalen Möglichkeiten des Smartphones. Darunter fallen Bildaufnahmen, automatische Verarbeitung von GPS- sowie Antwortdaten, und vieles mehr. Dies erlaubt interaktivere Aufgaben und Antworten, die automatisch geprüft werden. Spielerinnen und Spieler haben zudem jederzeit Zugriff auf Hilfestellungen. Antwortabhängige oder zufällige Aufgabenabfolgen machen den Spielablauf interessanter.

Aufgrund ihrer grossen Verbreitung wurde die Geocaching- und die Actionbound-Plattform genauer untersucht.

## Geocaching

Geocaching[[5]](#footnote-5) ist ein Schatzsuche-Spiel, bei dem Spieler anhand von Anweisungen einen Behälter suchen, in dem sich ein Logbuch und Tauschgegenstände befinden. Diese Behälter, sogenannte Caches, und die dazugehörigen Anweisungen werden von Freiwilligen erstellt und unterhalten. Will man einen Cache suchen, ruft man die Informationen des Cache über Webseiten wie geocaching.com oder einer anderen Geocaching-Applikation auf. Das Verwenden von Applikationen erlaubt in vielen Fällen das Herunterladen der Cache-Informationen und somit die spätere Schatzsuche ohne mobilen Internet-Zugang. Die Anweisungen beginnen üblicherweise mit den Koordinaten des ersten Postens. Im Fall von sogenannten Multicaches mit mehreren Stationen wird der Weg zum nächsten Posten so beschrieben, dass man den Ort des letzten Postens nicht herausfinden kann, ohne die vorherigen Posten besucht zu haben. Um den Cache zu finden, müssen alle Aufgaben gelöst werden. Das Spannende an den Geocaches ist, dass sie vielfach Geschichten des Ortes einbeziehen. So erfährt man einiges über historische Persönlichkeiten, Hintergründe zu Orts- und Flurnamen oder Gruseliges aus lokalen Legenden und Sagen. Da man beim Geocaching alle Anweisungen von Anfang an erhält, müssen die Anweisungen, die zu weiteren Posten führen, so formuliert sein, dass man sie nur befolgen kann, wenn die vorherigen Aufgaben gelöst wurden. Eine oft verwendete Möglichkeit ist das Verstecken von Anweisungen vor Ort. Dasselbe trifft auf allfällige Hilfestellungen zu, die üblicherweise mit ROT13[[6]](#footnote-6) verschlüsselt werden.

## Actionbound

Actionbound zeichnet sich anders als das Geocaching durch seinen interaktiven Charakter aus. Der Fokus liegt weniger bei der Suche, sondern bei den interaktiven Aufgaben. Ein Postenlauf, genannt Bound, besteht aus einer Reihe von Aufgaben respektive Posten, bei denen man etwa einen bestimmten QR-Code scannen, sich genug nah an einen Ort begeben oder Fragen beantworten muss.

Die Aufgaben werden je nach Einstellungen in zufälliger Reihenfolge oder geordnet wiedergegeben. Es gibt die Möglichkeit, gewisse Aufgaben nur dann einzublenden, wenn andere richtig gelöst wurden oder wenn eine gewisse Zeit abgelaufen ist.

Die Bounds werden wie beim Geocaching auf einer Webplattform erstellt. Gespielt werden die Bounds über eine mobile App, die dafür auf dem Smartphone installiert werden muss.

## Analyse

Das Angebot der analysierten Schnitzeljagd-Plattformen unterscheidet sich stark. Davon inspiriert wurde ein bunter Strauss von vielfältigen Funktionen zusammengestellt, die ein vielfältiges Spielerlebnis ermöglichen sollen.

1. Damit die Plattform auf möglichst vielen Geräten verwenden werden kann, sollte sie als Webanwendung umgesetzt werden, denn gerade junge Nutzerinnen und Nutzer verfügen nicht immer über das Recht, selbst Apps auf ihren mobilen Geräten zu installieren.
2. Ebenfalls im Hinblick auf jugendliche Spielerinnen und Spieler sollte die Nutzung kein mobiles Internet erfordern, welches Kosten verursacht. Das Durchlaufen einer Schnitzeljagd sollte nach einem einmaligen Download auch ohne Internetverbindung möglich sein.
3. Es sollte ein System von logischen Instruktionen geben, das komplizierte und reaktive Abfolgen von Aufgaben erlaubt. Damit ermöglicht man ein breiteres Anwendungsfeld und interessantere Spielverläufe.
4. Der Zustand der Schnitzeljagd soll so gespeichert werden, dass er auch beim Schliessen der Webanwendung erhalten bleibt. Beim Öffnen im Browser ist es möglich, dass ein Nutzer den Tab aus Versehen schliesst.

# Zielsetzung

Ziel der Arbeit war es, eine Webanwendung zu programmieren, die den Anforderungen aus der Analyse oben genügt. Diese Applikation soll als Plattform für digitale Schnitzeljagden dienen. Schnitzeljagden sollten zudem in einer vereinfachten Form gespeichert und von der Applikation in interaktive Elemente übersetzt werden, um das Erstellen von neuen Schnitzeljagden auch ohne Programmierkenntnisse zu ermöglichen. Allerdings muss hier auch festgehalten werden, dass im Rahmen der Arbeit kein fertiges Produkt erwartet wurde, sondern mit einem Prototypen die Machbarkeit einer solchen Plattform geprüft werden sollte.

# Vorgehensweise

Um die Herausforderung zu bewältigen, die die Umsetzung der gestellten Zielsetzungen darstellt, waren vorgängige Erfahrungen in der Webentwicklung unerlässlich. Dazu kam eine straffe Planung, die einen strukturiertes Vorgehen ermöglichte.

## Agiles Projektmanagement

Der Fokus bei der programmatischen Umsetzung lag primär bei der Erstellung eines minimalen, aber dennoch erweiterbaren Frameworks zur Ausführung von Schnitzeljagden in einer PWA. Weitere Anforderungen sollten nach der Methodik des agilen Projektmanagements während dem Programmieren ergänzt werden.

Das agile Projektmanagement gibt vier Schritte vor, die wiederholt durchgelaufen werden:

1. **Planung**: In der Planungsphase entscheidet man, welche Funktionalität man als Nächstes implementiert und wie sie grob funktionieren sollen. Ein Beispiel für das Ergebnis einer Planungsphase war der Entscheid, dass Schnitzeljagden als JSON[[7]](#footnote-7)-Datei konfiguriert und gespeichert werden.
2. **Implementierung**: Programmieren macht zusammen mit dem Testen den Hauptteil des Arbeitsaufwands aus. Das Programmieren selbst lässt sich in vier Teile unterteilen. Das Problem auf einer abstrakten Ebene lösen oder eine existierende Lösung suchen. Die (Teil-)Lösungen in die bereits vorhandene Software integrieren. Den Code testen und zuletzt gefundene Fehler beheben.
3. **Überprüfen**: Nachdem der Code hinreichend getestet wurde, muss im nächsten Arbeitsschritt die Machbarkeit der aktuellen Projektpläne überprüft werden.
4. **Anpassen**: Die Planung ist gegebenenfalls anzupassen, wenn sich ein Vorhaben als unrealistisch erweist. In manchen Fällen führen solche Planänderung dazu, dass bereits programmierte Funktionen für ein besseres Zusammenspiel der einzelnen Komponenten erneut angepasst werden müssen. Das Backend-Daten-System musste in meinem Fall mehrmals überarbeitet und das grafische User-Interface ebenfalls komplett neu geschrieben werden, um die gewünschte Modularität zu erreichen.

## Herausforderungen

Im Verlauf der Umsetzung sind verschiedene, durchaus zu erwartende Schwierigkeiten aufgetreten.

1. Fehleinschätzung von Zeitaufwand: Personen mit wenig Erfahrung unterschätzen den nötigen Zeitaufwand beinahe immer. Zum einen waren die eingesetzten Bibliotheken[[8]](#footnote-8) (siehe Kapitel 6) und die damit verbundenen Vorgehensweise Neuland für mich. Für die serverseitige Programmierung musste ich mich mit PHP in eine neue Programmiersprache eindenken. Ein weiterer Grund war der benötige Aufwand, zum Auffinden und Beheben von Programmfehlern. Mit besseren Methoden zur Fehlersuche könnte der Zeitaufwand sicher verringert werden. Es ist jedoch nicht unüblich, dass auch professionelle Entwickler bis zur 50 % der Zeit für das Aufspüren von Fehlern aufwenden[[9]](#footnote-9).
2. Zusammenspiel aller Komponenten in einem komplexen System: In dieser Arbeit wurden grosse Teile des Frontends geschrieben, bevor das Backend funktionsfähig war[[10]](#footnote-10). Um die Funktionen des Frontends zu testen, wurde zu Beginn eine erste Schnitzeljagd statisch im Code als POJO[[11]](#footnote-11) integriert. Um auf JSON-Daten, die vom Backend geliefert werden umzusteigen, musste nachträglich Teiles des Speichersystems angepasst werden. Hauptsächlich, um die Speicherstände der verschiedenen Schnitzeljagden zu trennen. Da das Laden von Daten vom Server mit einer gewissen Latenz verbunden ist, waren weitere Anpassungen nötig, damit dieser Code asynchron und damit nicht-blockierend ausgeführt werden kann. Der Umbau von einer «normalen» Website in eine PWA war ebenfalls Neuland und brachte ähnliche Schwierigkeiten mit sich.
3. Für unbekanntes Vorausplanen: Um die Erweiterbarkeit sicherzustellen, habe ich den Interpreter, der die Konfigurationsdatei übersetzt, modularisiert. Die Beschreibung der verschiedenen Module sollte so weit wie möglich parametrisiert sein, ohne übermässig kompliziert zu werden. Die Notation, die verwendet wird, um Schnitzeljagden zu beschreiben und ihre Zustände zu speichern, wurde parallel zu den wachsenden Bedürfnissen entwickelt. Einfacher ist es, wenn die Datenstrukturen bereits feststehen. Damit lassen sich Kompatibilitätsprobleme zwischen Modulen minimieren. Um möglichst viele dieser Ziele zu erreichen, mussten mehrere Programmteile mehrmals überarbeitet oder komplett neugeschrieben.

Die aufgetretenen Schwierigkeiten waren vielfach zeitraubend, aber auch stets ein wertvoller Lernprozess.

# Funktionen und Funktionsweise

In diesem Kapitel werden Funktionalität und Funktionsweise der entstandenen Plattform erläutert. Die Plattform ist über die URL <https://ompha.li> abrufbar. Der Programmcode sowie Beispielkonfiguration sind über das GitHub-Repository unter:

<https://github.com/TscherterJunior/Maturaarbeit>

abrufbar. Eine interaktive Demonstration der Plattform gibt es unter <https://ompha.li/#!/trail/?trailid=test> . Der Omphalos war im antiken Griechenland ein Stein, der den Nabel der Welt markiert. Über diesen Namen soll verdeutlicht werden, dass eine Schnitzeljagd zeitweilig einen Ort zum Mittelpunkt des Geschehens werden lässt.

Beim erstmaligen Aufrufen der Webanwendung bekommen die Spielerinnen und Spieler eine Liste möglicher Schnitzeljagden angezeigt, aus der sie später eine oder mehrere zur offline Nutzung auswählen können (siehe Abschnitt 5.5 Offlinebetrieb).

Der Aufbau einer Schnitzeljagd wird in einer Konfigurationsdatei festgehalten. Jede Schnitzeljagd hat hier eigene Konfigurationsdatei. Diese enthalten:

1. Ein Titel sowie eine inhaltliche und funktionale Beschreibung aller Posten.
2. Eine initiale Startkonfiguration mit einer Liste der vorerst sichtbaren Posten sowie Vorgabewerte für allfällige globale Variablen. Dies wird in Abschnitt 5.4 erläutert.
3. Ein Regelwerk von Bedingungen und Befehlen, die bei jeder Änderung der globalen Variablen automatisch ausgewertet und umgesetzt wird. Dies wird in Abschnitt 5.3 erläutert.

Wird eine Schnitzeljagd angewählt, ändert die Webanwendung das Seitenlayout in den Schnitzeljagdmodus. Welche Posten angezeigt werden, hängt vom individuellen Spielstand ab. Ist für die gewählte Schnitzeljagd noch kein Spielstand vorhanden, wird der Initialspielstand aus der Konfigurationsdatei der Schnitzeljagd ausgelesen und dafür eingesetzt.

Sobald ein Spielstand verfügbar ist, werden die darin als sichtbar aufgelisteten Posten als DOM[[12]](#footnote-12)-Elemente grafisch dargestellt.

Der individuelle Spielstand (siehe auch Abschnitt 5.4) einer Schnitzeljagd enthält eine geordnete Liste der sichtbaren Posten, Variablen, die über Befehle umgeschrieben werden können, und die inneren Zustände der einzelnen Posten. Jedes Mal wenn ein Befehl die globalen Variablen ändert, werden die in der Konfigurationsdatei beschriebenen Logikinstruktionen überprüft. Eine Logikinstruktion besteht aus einer Bedingung und einer Liste von Befehlen. Ist die Bedingung erfüllt, werden die Befehle, die zu dieser Bedingung gehören, ausgeführt.

## Postentypen

Es gibt fünf Arten von Posten:

**Bestätigungs-Posten** enthalten Text und einen Startknopf. Der Bestätigungs-Posten wird üblicherweise als Startposten verwendet und dient dazu, vor dem Start des Postenlaufs anfällige Informationen, Warnungen oder Nutzungsbedingungen anzuzeigen. Beim Drücken des Startknopfes wird eine bei der Konfiguration festgelegte Liste von Befehlen ausgeführt, die den Postenlauf starten. Bestätigungs-Posten können inaktiv gemacht werden.

**Text-Posten** dienen als Informationsblock und zeigen Text an.

**Multiple Choice-Posten** bestehen aus einer Aufgabe und einer Auswahl mit verschiedenen Antworten, von denen eine angewählt werden kann[[13]](#footnote-13). Die Antwort wird erst nach einem «Submit» geprüft. Die Anzahl Versuche wird protokolliert. Multiple Choice-Posten speichern ihren Zustand automatisch, auch wenn noch kein «Submit» erfolgt ist. Dieser Zustand gibt an, welche Antwort angewählt ist und wie häufig eine Prüfung der Antwort stattgefunden hat. Für jede Antwort sowie für jede Anzahl Versuche können bei der Konfiguration des Postens eine Liste von Befehlen angegeben werden, die den Zustand des Postenlaufs verändern. So kann das Spiel so konfiguriert werden, dass nach einer gewissen Anzahl Fehlversuche Hilfestellungen angezeigt oder Strafpunkte vermerkt werden.

In **QR-Code-Posten** können direkt über die Webanwendung QR-Codes gescannt werden. Nachdem die nötigen Berechtigungen erteilt wurden, erscheint ein Video-Element, welches das Bild, das die Geräte-Kamera liefert, abbildet. Sobald der Scan erfolgreich war, verschwindet das Video-Element wieder.. Wird ein QR-Code erfolgreich eingelesen, wird das Ergebnis des Scans in die in der Konfiguration festgelegten Zielvariable abgelegt. QR-Code-Posten können inaktiv gemacht werden.

**Resultate-Posten** zeigen einen Text und einen vorgängig gespeicherten Wert an, wie das Resultat eines QR-Scans.

## Das Renderingsystem

Eine Rendering Funktion erzeugt ausgehend von der Konfiguration und dem Spielspielstand eine DOM kompatible Darstellung der Schnitzeljagd. Nach jeder Interaktion wird das DOM virtuell neu berechnet. Aus Gründen der Effizienz werden anschliessend nur die effektiv nötigen Veränderungen am sichtbaren DOM vorgenommen, die über einen Abgleich mit einer der virtuellen Kopie des DOM bestimmt werden[[14]](#footnote-14). Dafür kommt die Programmbibliothek Mithril.js (siehe Kapitel 6) zur Anwendung. Mithril.js stellt eine Rendering-Funktion zur Verfügung, über die die besagten virtuellen DOM-Knoten für jeden Posten erzeugt werden. Mithril.js übernimmt auch den Prozess aus dem neu erstellten virtuellen DOM die Änderungen in das angezeigte DOM zu übertragen. Über die Rendering-Funktion werden die DOM-Elemente einerseits definiert und optional mit CSS gestaltetet sowie für die Benutzerinteraktion mit Event Listenern[[15]](#footnote-15) versehen.

|  |  |
| --- | --- |
|  |  |
| Abbildung : Auszug des Konfigurations- Objekts eines Multiple Choice-Postens | Abbildung : Darstellung eines Multiple Choice-Postens |

## Interaktionen und Spiellogik

Posten wurden so konzipiert, dass sie nur ihren eigenen Zustand direkt umschreiben können (siehe Kapitel 5.4). Diese Restriktion vereinfacht es, neue Posten zu implementieren, ohne dass es zu Kompatibilitätsproblemen kommt. Der Schreibzugriff erfolgt indirekt über einen eigens erstellten **Befehlsinterpreter**. Befehle werden in der Konfigurationsdatei als Objekt notiert. Damit sollen Cross-Site-Scripting[[16]](#footnote-16) Angriffe verhindert werden, da der Interpreter nur gültige Befehle jedoch keinen Code auswertet. Soll ein Befehl ausgeführt werden, wird die Interpreter-Funktion des Moduls mit dem Befehlsobjekt als Argument aufgerufen. Die Interpreter-Funktion liest dann den Befehlstyp und genaue Parameter aus und setzt den Befehl um. Die aktuell verfügbaren Befehlsobjekte sind:

1. Posten zur Anzeigeliste hinzufügen oder aus ihr entfernen
2. Interaktivität eines Postens aktivieren oder deaktivieren
3. Postenübergreifende Variablen schreiben.

Um komplexe Postenabfolgen zu ermöglichen, wurde die Plattform um eine zusätzliche Komponente, dem **Bedingungsinterpreter**, ergänzt. Wird über einen Befehl eine der nicht postenspezifischen Variablen verändert, werden die in den Logikinstruktionen festgehaltenen Bedingungen getestet. Ist eine Bedingung erfüllt, wird die zur Bedingung gehörende Liste von Befehlen ausgeführt. Bedingungen sind gleich wie Befehle als Objekt notiert. Implementierte Bedingungen sind:

1. Gleichheit: Überprüft, ob die Werte, die an zwei Adressen gespeichert sind, gleich sind.
2. Konkrete Gleichheit: Überprüft, ob der Wert an einer Adresse einem konkreten Wert entspricht.
3. Alle wahr: Die Alle wahr Bedingung enthält eine Liste von weiteren Bedingungen, die alle erfüllt sein müssen, damit sie als erfüllt gilt.
4. Nicht wahr: Die Nicht wahr-Bedingung ist erfüllt, wenn eine bestimmte Bedingung nicht erfüllt ist

## Speicher

Der Zustand einer Schnitzeljagd ist in einem Objekt gespeichert (siehe Beispiel in Abbildung 5). Das Zustandsobjekt besteht aus folgenden Komponenten:

1. **board** enthält eine Liste der Posten, die aktuell angezeigt werden.
2. **quest** enthält ein Objekt, welches zu jedem Posten die lokalen Zustandsdaten speichert.
3. **keys** ist ein Objekt, das die globalen postenübergreifenden Variablen enthält. Diese globalen Variablen können in einem Spiel wie bei einem Adventure Game die Funktion eines Inventars übernehmen. So können für erfolgreich gelöste Aufgaben Münzen oder andere Items als Belohnung vergeben werden.

|  |
| --- |
| Ein Bild, das Text enthält.  Automatisch generierte Beschreibung |
| Abbildung : Beispiel eines Speicherstands |

Eine Version dieses Zustandes wird in der Konfigurationsdatei vorgegeben. Dieser Zustand wird als Startzustand geladen, falls noch kein Speicherstand existiert. Der Speicherstand wird über eine Funktion verändert, die sowohl den Speicherstand im RAM als auch im nichtflüchtigen lokalen Web Storage[[17]](#footnote-17) – konkret im Local Storage – nachführt. Um das Umschreiben des Speicherstands zu vereinfachen, kommt die Programmbibliothek Mergerino (siehe Kapitel 6) zum Einsatz. Der im Local Storage gespeicherte Zustand bleibt auch nach dem Schliessen des Browsers erhalten.

## Offlinebetrieb

Ein in der Webanwendung integrierter Service Worker[[18]](#footnote-18) ermöglicht den Offlinebetrieb. Beim ersten Aufruf der Webanwendung dafür ein Service Worker automatisch installiert. Der Service Worker fungiert als Proxy zwischen Server und Client und lädt benötigte Dateien in den Browser-Cache[[19]](#footnote-19). Wird eine Schnitzeljagd «installiert», wird die entsprechende Konfigurationsdatei durch den Service Worker in den Cache abgelegt. Dadurch kann sie offline genutzt werden. Spätere Anfragen auf diese Datei werden vom Service Worker abgefangen. Die Konfigurationsdateien für Schnitzeljagden werden allerdings nicht automatisch zwischengespeichert. Dies erfolgt erst, wenn ein Spieler oder eine Spielerin es wünscht. Beim Installieren führt das Programm zudem eine interne Liste der verfügbaren Schnitzeljagden nach. Diese Liste ersetzt die Liste der verfügbaren Schnitzeljagden, wenn die komplette Liste nicht vom Server geliefert wird.

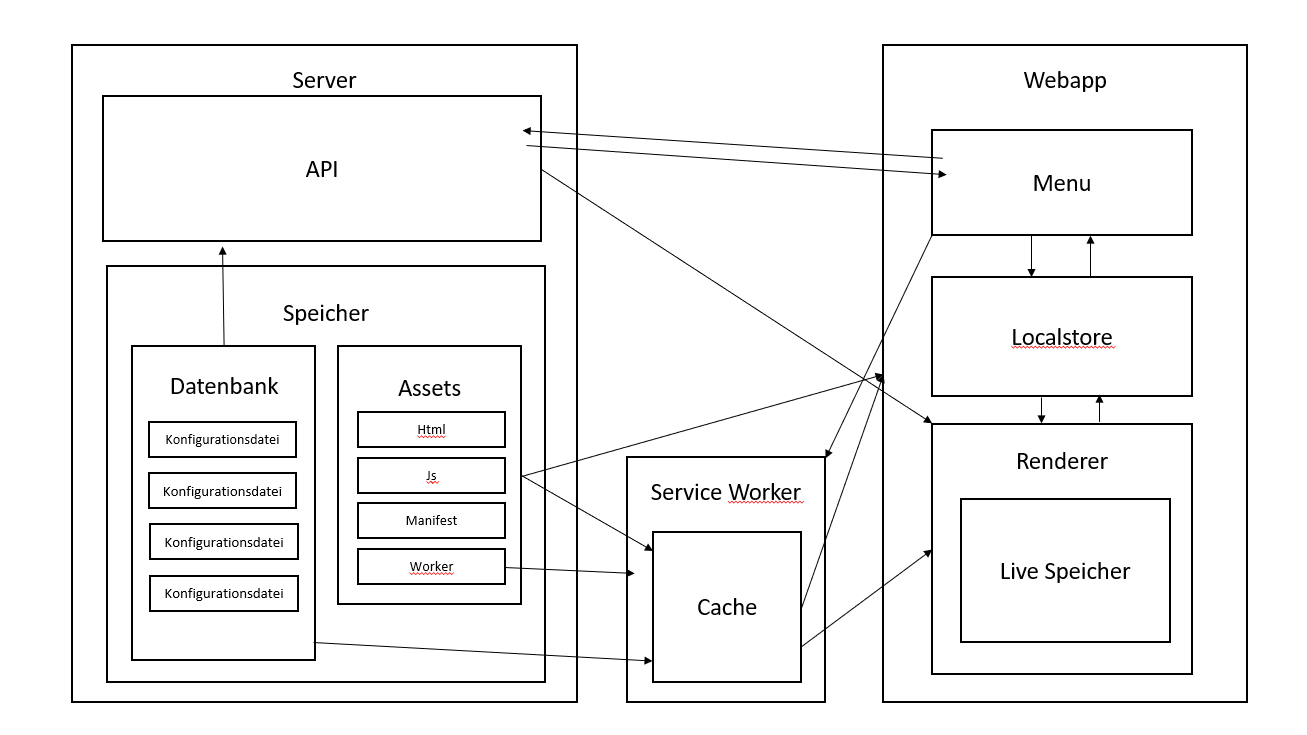


Abbildung : Schematische Darstellung des Datenflusses

## Backend / API

Das Backend dieses Projekts ist eine in PHP geschriebene API[[20]](#footnote-20). Programme und Webseiten können über das Internet mit dieser API kommunizieren. Wird eine Anfrage für Informationen ohne weitere Parameter ausgeführt, antwortet sie mit einer Liste aller verfügbaren Schnitzeljagden. Ist in der URL-Query[[21]](#footnote-21) die ID einer Schnitzeljagd angegeben, antwortet die API mit der Konfigurationsdatei dieser Schnitzeljagd.

# Programmbibliotheken

Beim Programmieren von grösseren Projekten ist es üblich Programmbibliothek zu verwenden. Programmbibliotheken dienen dazu, häufig aufkommende Aufgaben zu vereinfachen. In dieser Arbeit wurden vier Programmbibliotheken verwendet.

1. **Mithril.js** dient dazu, das Bearbeiten des DOMs über JavaScript zu vereinfachen und zu optimieren.   
   Quelle: <https://mithril.js.org> (letzter Abruf 05.11.2022)
2. **BSS** ist eine mit Mithril.js kompatible Programmbibliothek, die es erlaubt über JavaScript erstellte Elemente im JavaScript selbst mit CSS in kompakter Schreibweise zu stylen.  
   Quelle: <https://github.com/porsager/bss> (letzter Abruf 05.11.2022)
3. **Mergerino** unterstützt beim Zustandsmanagement. Mittels Mergerino können Objekte über sogenannten Patches umgeschrieben werden. Solch ein Patch kann eine Funktion beinhalten, was unter anderem erlaubt, den Wahrheitswert eines Objektattributs zu invertieren oder eine Liste gezielt zu verändern.  
   Quelle: <https://github.com/fuzetsu/mergerino> (letzter Abruf 05.11.2022)
4. Ein **QR-Scanner** vereinfacht das Scannen von QR-Codes. Der Videofeed der Gerätekamera wird ausgelesen und über einen Web Worker verarbeitet, damit die Webseite ungehindert weiterläuft.   
   Quelle: <https://github.com/nimiq/qr-scanner> (letzter Abruf 05.11.2022)

Jede in einem Projekt eingesetzt Programmbibliothek löst einerseits ein Problem, in dem es anspruchsvolle Funktionen einbindet, für die man selbst weder das nötige Know-how noch die Zeit hätte es zu implementieren. Anderseits ist mit jeder Bibliothek auch ein nicht zu unterschätzender Einarbeitungsaufwand verbunden. Man sollte auch darauf achten, dass die verwendeten Bibliotheken aktiv gepflegt werden. Über den Fremdcode könnten auch potenzielle Sicherheitslücken entstehen.

# Diskussion

Zu Beginn meiner Arbeit hatte ich mir folgendes Ziel gesetzt: ich wollte demonstrieren, dass sich eine flexible Plattform für digitale Schnitzeljagden als PWA umsetzen lässt. Die jetzt vorliegende Plattform ist so konzipiert, dass sie nicht nur für klassische Schnitzeljagden verwendet werden kann, sondern auch für zahlreiche andere Zwecke verwendet werden könnte. Dass mein Produkt fortgeschrittene Logik unterstützt, konnte durch das Nachbauen eines «Rule 110 cellular automaton»[[22]](#footnote-22) gezeigt werden. Somit ist das Logiksystem Turing-vollständig.

Dank dem modularen Design, den standardisierten Protokollen und dem erweiterbaren Datensysteme ist es vergleichsweise einfach neue Aufgabentypen oder Logikkomponenten hinzuzufügen.

## Mögliche Ausbauschritte

Das entstandene Framework stellt einen Proof of Concept dar. Um als Produkt eingesetzt zu werden, sind weitere Ausbauschritte nötig:

1. **Neue Module**: Der bestehende Erweiterungsbedarf ist stark von der Aufgabe abhängig, die die Plattform in Zukunft erfüllen soll. Am ausgeprägtesten ist diese Diversität bei den neuen Modulen. Entscheidet man sich z. B. die Plattform in einem Schulkontext für Übungen einzusetzen, wäre es vorteilhaft, wenn Teile von Übungen zufällig generiert werden könnten. Bei Befehlen und Bedingungen gibt vermutlich mehr generell anwendbare Erweiterungen. Das Ausführen eines Bedingungstests über einen Befehl anstatt infolge jeder Zustandsänderung würde es vereinfachen, den Programmfluss zu kontrollieren. Zurzeit fehlen auch Möglichkeiten, Variablen im Speicher zu kombinieren oder effizient zu verändern. Zu diesem Zweck wird es für sinnvoll erachtet, neue Befehle einzuführen.
2. **Editor**: Ein Schnitzeljagd-Editor vergleichbar zu dem der bei Actionbound anzutreffen ist, würde das Erstellen von neuen Schnitzeljagden vereinfachen und beschleunigen, indem er das Schreiben von gleichbleibender Syntax automatisiert. Ein fortgeschrittener Editor könnte das Schreiben von Logikinstruktionen für simple Postenabfolgen komplett automatisieren und so das Erstellen von einfachen Postenläufen auch Laien zugänglich machen. Der Editor von Actionbound zeigt allerdings deutlich, dass eine hohe Flexibilität bei der Ausführungslogik zu sehr komplexen und schwierig zu verstehenden Benutzerschnittstellen führen kann.
3. **User Identifikation**: Um zu kontrollieren, wer auf Schnitzeljagden zugreifen und sie erstellen darf, ist es unerlässlich eine Form von Benutzerkonto einzuführen. Die Benutzeridentifikation würde auch die Tür für Bewertungen und Kommentaren zu Trails öffnen.
4. **Suchfunktion**: Sobald die Anzahl an Schnitzeljagden zunimmt, wird eine benutzerfreundliche Suchfunktion nötig, um den Nutzerkomfort aufrechtzuerhalten. Die Suche nach Schnitzeljagden in der Umgebung des Anwenders so wie die Kategorisierung nach Inhalt scheinen vorteilhaft.
5. **Ansprechendes Design**: Das grafische Design der aktuellen Version ist noch bewusst simpel gehalten. Verschiedene Knöpfe sind ungünstig positioniert.

## Alternative Verwendungen

Durch seine grosse Anpassbarkeit lässt sich das in dieser Arbeit entwickelte Framework auch für andere Zwecke als für digital geleitete Schnitzeljagden verwenden. Spielbücher, in denen die Leserin oder der Leser zwischen verschiedenen Handlungssträngen auswählt, lassen sich ohne Modifikation implementieren. Ein Postentyp, der Würfelwürfe generiert und auswertet, könnte im Falle einer solchen Anwendung das Spielergebnis noch verbessern. Eine weitere mögliche Anwendung wären automatisch überprüfte Übungen für Unterrichtszwecke, wobei für solch eine Anwendung parametrisierte Posten und mehr Postentypen für diversere Aufgabentypen wohl sinnvoll wären. Die integrierte QR-Scan-Funktion liesse sich alternativ verwenden, um z. B. in einem Museum Information zu Ausstellungsstücken anzuzeigen. Besucher könnten eine Sprache auswählen und QR-Codes scannen. Die Webanwendung würde dann zum QR-Code gehörenden Text in der gewällten Sprache anzeigen. In diesem Anwendungsfall wäre das Implementieren von Wiedergabeelementen für alternative Medien wie Ton, Bild oder Video sinnvoll.

## Bekannte Mängel

Obwohl viel Zeit in das Beheben von Programmierfehlern investiert wurde, war es nicht möglich alle bekannten Fehler zu beheben. Bekannte Fehler sind:

1. Der Installierknopf funktioniert erst, nachdem die Webanwendung mit fünf oder mehr Minuten Abstand zum ersten Aufruf der Seite wieder geöffnet wird. Der Grund dafür ist, dass der Browser den Nutzer vor unnötigen Popups schützen soll. Der Installierknopf sollte erst sichtbar oder aktiv werden, wenn die Installation möglich ist.
2. Wird die Seite nach dem Löschen des Browser-Cache neu geladen, kann es passieren, dass der Service Worker seinen Dienst versagt. Tritt dieser Fall ein, können Schnitzeljagden bis zum Neuladen der Seite nicht mehr installiert oder deinstalliert werden.
3. Der Service Worker wurde bislang nicht so programmiert, dass er lokal gespeicherte Dateien im online-Betrieb automatisch aktualisiert.
4. Die Manifest-Datei[[23]](#footnote-23) enthält die Adresse eines Icons anstelle von Icons in mehreren Auflösungen. Aufgrund dessen kann es sein, dass auf manchen Geräten kein Icon angezeigt wird.

## Sicherheitsaspekte

Bis auf einen einfachen Zeichenfilter wurden in der API keine Vorkehrungen getroffen, um fremde Zugriffe auf das System zu verhindern. Es ist stets anzunehmen, dass über API-Anfragen Code auf dem Server ausgeführt werden kann. Eine API muss daher einem rigorosen Review unterzogen werden. Eingeschleuster Code könnte alle Daten auf dem Server auslesen, verändern oder die Rechenleistung des Servers zweckentfremden. Bevor man in dieser Arbeit erstellte Plattform in einem öffentlichen Rahmen anwendet, sollten weitere Schutzmassnahmen getroffen werden.

## Fazit

Durch die Erstellung eines flexiblen Frameworks für digitale Schnitzeljagden wurde die Machbarkeit einer derartigen Plattform demonstriert. Inzwischen ist die Plattform soweit ausgereift, dass sie von einer grösseren Nutzergruppe getestet werden könnte. Das erstellte Framework implementiert weiter alle in der Analyse (Abschnitt 2.3) vorgeschlagenen Funktionen. Bevor die Plattform marktreif wäre, müssten jedoch verschiedene Anpassungen vorgenommen werden. Die mit der Flexibilität der Plattform einhergehende Komplexität macht die Konfiguration von Schnitzeljagden im Vergleich zu verbreiteten Alternativen anspruchsvoll. Das Entwickeln eines solchen Systems brachte wie zu erwarten seine Schwierigkeiten mit sich. Hauptschwierigkeiten waren die hohe Komplexität des Systems, die Herausforderung mehrere Subsysteme zu konzipieren, ohne die Funktionsweise noch ungeplanter Subsysteme zu kennen und mangelnde Vertrautheit mit verwendeten Programmiersprachen und Programmbibliotheken. Im Rahmen der zur Verfügung stehenden Zeit konnte dennoch ein Produkt mit Potenzial entwickelt werden. Die bei der Erstellung eines grösseren Programmierprojekts gewonnenen Erfahrungen waren auf alle Fälle den Aufwand wert.

1. Ein **Spielbuch** ist ein gedrucktes literarisches Werk, das dem Leser oder der Leserin die Möglichkeit gibt, den Geschichtsverlauf zu beeinflussen, indem er oder sie Entscheidungen trifft. Spielbücher waren in den 80er Jahren sehr beliebt. [↑](#footnote-ref-1)
2. Eine **Progressive Web** **App**, oder kurz **PWA**, ist eine Art von Anwendungssoftware, die über das Web bereitgestellt wird und mit gängigen Webtechnologien wie HTML, CSS und JavaScript erstellt wurde. PWA laufen im Kontext eines Webbrowsers und bieten einen Funktionsumfang, der mit nativen Applikationen vergleichbar ist. [↑](#footnote-ref-2)
3. **Scavify**, ist eine kommerzielle Schnitzeljagd-App die speziell für Gruppenaktivtäten und Teambuilding in Unternehmen entwickelt wurde. <https://www.scavify.com/> (letzter Abruf 05.11.2022) [↑](#footnote-ref-3)
4. **Foxtrail** ist eine kommerzielle GmbH, die touristische Schnitzeljagden an attraktiven Orten anbietet. <https://www.foxtrail.ch/> (letzter Abruf 05.11.2022) [↑](#footnote-ref-4)
5. **Geocaching** steht als Überbegriff für diverse Angebote in dem Bereich. Der bekannteste Anbieter ist unter <https://www.geocaching.com/> (letzter Abruf 05.11.2022) zu finden. [↑](#footnote-ref-5)
6. **ROT13** ist eine einfache Substitution-Chiffre, die einen Buchstaben durch den 13. nachfolgenden Buchstaben im Alphabet ersetzt. ROT13 kann von Hand entschlüsselt werden. [↑](#footnote-ref-6)
7. **JSON** steht für JavaScript Object Notation und ist ein standardisiertes, menschenlesbares Datei- bzw. Datenaustauschformat für Datenobjekte. Objekte beinhalten eine Sammlung von Attribut-Wert-Paaren, wobei Werte wieder Objekte oder Listen (Arrays) sein können. [↑](#footnote-ref-7)
8. Eine Programm**bibliothek** ist eine Sammlung von Funktionen und Subroutinen die Lösungswege für häufige Problemstellungen bieten. [↑](#footnote-ref-8)
9. Devon H O'Dell, 2017, The Debugging Mindset. <https://queue.acm.org/detail.cfm?id=3068754> (letzter Abruf 05.11.2022) [↑](#footnote-ref-9)
10. Im einem Client-Server-Modell wird der Client (hier der Webbrowser) in der Regel als **Frontend** und der Server als **Backend** bezeichnet. [↑](#footnote-ref-10)
11. **POJO** steht für Plain Old Java Object, also ein «ganz normales» Objekt-Literal. Die verwendete Notation ist vielen Teilen kompatibel zu den später verwendeten JSON-Dateien. [↑](#footnote-ref-11)
12. Das Document Object Model (**DOM**) ist eine Schnittstelle im Browser über das aktuell angezeigte Dokument programmatisch verändert werden kann. Die dargestellte Webseite wird in einer baumartigen Struktur repräsentiert, die alle Elemente des Dokuments enthält oder darauf verweist. [↑](#footnote-ref-12)
13. **Multiple Choice** wird hier im Sinne von «Einfachauswahl» verwendet. Dies entspricht der im Englisch üblichen Konvention, dass bei «Multiple Choice» nur eine Antwort der Auswahl gültig ist. Aufgaben, bei denen mehrere Optionen ausgewählt werden können, werden mit «Multiple Response» bezeichnet (Quelle: <https://de.wikipedia.org/wiki/Multiple_Choice> (letzter Abruf 05.11.2022)). [↑](#footnote-ref-13)
14. Zahlreiche JavaScript Bibliotheken nutzen zur Erstellung und Aktualisierung der Oberfläche ein virtuelles DOM. Oft handelt es ich um eine vereinfachte, rein objektbasierte Darstellung des DOM, das von einer «pure» Funktion aus den Zustandsdaten extrem schnell erzeugt wird. Bekannt Vertreter dieses Ansatzes sind Angular und React. [↑](#footnote-ref-14)
15. Ein **Event Listener** ist eine Rückruffunktion (engl. Callback) in einem Computerprogramm, die auf das Eintreten eines Ereignisses wartet. Beispielsweise legt man mit einem Event Listener fest, was beim Klicken auf einen Knopf passieren soll. Für den Aufruf der Funktion ist die Ereignisschleife (engl. event loop) des Browsers zuständig. [↑](#footnote-ref-15)
16. **Cross-Site-Scripting** (XSS) ist eine in Webanwendungen ausgenutzte Sicherheitslücke, bei der es Angreifern gelingt, clientseitige Skripte in Webseiten einzuschleusen. Da die Schnitzeljagden konfiguriert und nicht programmiert werden, sollten XSS-Angriffe daher ausgeschlossen sein. [↑](#footnote-ref-16)
17. Der **Web Storage** ist ein vom Webbrowsern bereitgestellten Speicher. Er ermöglicht es Webanwendungen, dauerhaft Daten zu speichern. Im Unterschied zu Cookies werden dafür keine Informationen an den Server geschickt, und es steht eine deutlich grössere Speichermenge pro Domain zur Verfügung. [↑](#footnote-ref-17)
18. Ein **Worker** ist ein JavaScript Programm, das von einer Webseite im Hintergrund laufend ausgeführt wird. Als Hintergrundprozess läuft es in einem separaten Thread und blockiert bei der Ausführung nicht die ganze Webseite. Worker werden benutzt, um zeitintensive Berechnungen auszuführen oder für asynchrone nichtblockierende Aufgaben. [↑](#footnote-ref-18)
19. Im **Browser-Cache** legt der Webbrowser einmal geladene Ressourcen in einem Pufferspeicher ab, um sie beim nächsten Aufruf schneller oder auch offline verfügbar zu haben. [↑](#footnote-ref-19)
20. Eine Web-**API** ist eine Anwendungsprogrammierschnittstelle für einen Webserver oder einen Webbrowser. Üblicherweise liefert eine Web-API unformatierte Daten aus, die im Hintergrund von der Webseite nachgeladen werden. Über eine Web-API können auch Benutzerinteraktionen laufend an den Server zur persistenten Speicherung geschickt werden. [↑](#footnote-ref-20)
21. Ein **URL-Query** ist eine Zeichenkette über die, als Teil eines Uniform Resource Locator (URL), Serveraufrufe parametrisiert werden. [↑](#footnote-ref-21)
22. «Rule110» ist ein eindimensionaler zellulärer Automat für den die Turing-Vollständigkeit bewiesen wurde (Quelle: <https://en.wikipedia.org/wiki/Rule_110> (letzter Abruf 05.11.2022)). Dafür müssen unendliche Speicherkapazität und Rechenleistung angenommen werden. [↑](#footnote-ref-22)
23. Die Manifest-Datei ist eine PWA spezifische Datei. In dieser wird werden Metadaten zur Webanwendung hinterlegt. Sie ist auch nötig, damit die Webanwendung wie eine native App installiert werden kann. [↑](#footnote-ref-23)