Platzhalter Titel

Lennart Ploog

8. Juli 2020

IS Medieninformatik
Fakultät 4
Hochschule Bremen

Inhaltsverzeichnis

1 Abstract

TODO: Acronyms, Figures

2 Einleitung

2.1 Problemfeld

Ob auf Reisen, im Supermarkt oder im Fahrstuhl - Situationen in denen mobile Endgeräte keine stabile Internetverbindung haben, kommen im Alltag häufiger vor als gewünscht. In vielen Entwicklungsländern und auch in ländlichen Gegenden entwickelter Industriestaaten fehlt dafür gar die komplette Infrastruktur. Im Laufe der letzten Jahre eröffneten Innovationen im Bereich der Browser-Technologien, allen voran der Service-Worker, neue Möglichkeiten für die Webentwicklung. Als Offline-First-Applikationen werden Webanwendungen bezeichnet, die ihre Funktionalität so weit es geht behalten, wenn die Verbindung zum Internet getrennt ist.

Eine der Kernherausvorderungen der Entwicklung von Offline-First Applikationen ist die Synchronisation von Daten. Werden offline Änderungen vorgenommen, sollen diese nicht verloren gehen. Hat sich der Zustand der Applikation, beispielsweise durch Modifikationen eines anderen Nutzers, in der Zwischenzeit jedoch geändert, müssen beide Änderungen zusammengebracht, also synchronisiert werden. Der Prozess der Synchronisation ist oft aufwendig, denn zum Einen muss ermittelt werden, wo sich beide Replikationen unterscheiden und zum Anderen muss vermieden werden, dass die Änderungen sich in die Quere kommen.

Gängige Lösungen zu einer solchen Zusammenführung von Daten Umfassen die Nutzung bestimmter Datenbanken-Technologien. Dazu gehören Datenbanken mit implementierter Synchronisation wie CloudDB oder auch Backend-as-a-Service Produkte wie Firebase oder IBM Cloudant, welche ebenfalls eine solche Funktionalität anbieten. Um dies zu vermeiden, Verzichten viele Applikationen bei Konflikten auf eine Synchronisation und einer der beiden Nutzer verliert seine vollbrachte Arbeit.

2.2 Ziel der Arbeit

Ziel dieser Arbeit ist es, eine Lösung zur Synchronisation von Daten in Offline-First Anwendungen mit Hilfe von konfliktfreien replizierten Datentypen, kurz CRDTs, umzusetzen. Der Einsatz von CRDTs ermöglicht, dass Daten in einem verteilten System in beliebiger Reihenfolge ausgetauscht werden können und dennoch zum gleichen Zustand aller Replikationen führen. Durch die Implementierung einer Datenstruktur, welche auf CRDTs aufbaut, kann der Prozess der Synchronisierung somit vermieden werden. Diese Lösung soll unabhängig von der gewählten Datenbank sein.

So ergeben sich folgende Forschungsfragen:

- Wie können CRDTs in Offline-First Applikationen verwendet werden?
- Welche CRDTs bieten sich zur Umsetzung von Offline-First Applikationen an und wie werden diese in die Datenbanken (Client und Server) implementiert?
- Welche Vor- und Nachteile bietet die Nutzung von CRDTs im Vergleich zu anderen Optionen zur Synchronisierung von Daten in Offline-First Applikationen.

2.3 Vorgehen

Es gibt verschiedene Möglichkeiten, wie CRDTs in Webapplikationen eingesetzt werden können. Bevor die Implementation der Datenstruktur beginnen kann, muss ermittelt werden, welche CRDTs sich am besten für die Daten des Prototypen eignen. Um dies herauszufinden eignet sich die Recherche in den im Abschnitt Verwandte Arbeiten erwähnten Publikationen. Darüber hinaus lohnt es sich an dieser Stelle auch in Erfahrung zu bringen, welche CRDTs bis heute in fertigen Applikationen verwendet wurden. Auch Literatur zum Austausch von Daten in Applikationen zu kollaborativem Editieren zu P2P Netzwerken bietet sich zur Recherche an, denn in diesen Bereichen sind CRDTs schon weiter verbreitet als in anderen Anwendungsgebieten.

Damit der Prototyp als praxisnahes Beispiel dienen kann, sollte auch der Stand der Technik im Themenbereich der Offline-First Anwendungen ermittelt werden. Um einzuordnen, an welcher Ebene der Architektur der Applikation sich die umzusetzende Funktionalität zur Zusammenführung der Daten am besten einbauen lässt, lohnt sich

auch ein Blick auf bestehende Lösungen, welche die Synchronisation nicht direkt auf der Datenbankebene durchführen, sondern zwischen Applikation und Datenbank.

2.3.1 Prototyp

Platzhalter, genauer nach Fertigstellung des Prototypen Als Prototyp wird ein Online-Kochbuch mit folgenden Funktionen umgesetzt:

- Anlegen, Bearbeiten und Löschen von Rezepten mit Namen, Zutaten und Beschreibung
- Zugriff auf die Gleichen Rezepte von verschiedenen Clients
- "Liken" der Rezepte

3 Verwandte Arbeiten

Entstanden in der Forschung an Datenstrukturen für kollaboratives Editieren formulieren Shapiro u. a. (2011) die theoretischen Grundlagen von CRDTs um Strong Eventual Consistency, kurz SEC, in großen verteilten Systemen zu garantieren. SEC erweitert den bis dahin verbreiteten Ansatz der Eventual Consistency, kurz EC. Während EC nur garantiert, dass sämtliche Updates schlussendlich alle Replizierungen der Datenbank erreichen, garantiert SEC zusätzlich, dass Updates unabhängig von Reihenfolge und Zeitpunkt immer zum gleichen Zustand der Replizierungen führen.

Seitdem hat sich die Verwendung von CRDTs in verschiedenen Bereichen der Webentwicklung verbreitet. Kleppmann und Beresford (2017) entwerfen eine library CRDT konformer JSON Datenstrukturen, genannt "automerge", die beliebig verschachtelte Listen und Maps unterstützt. Mit "Hypermerge" entstand auch eine spezielle Version für Peer-to-Peer Netzwerke.

Mit Woot(Oster u. a. 2006), Logoot(Weiss, Urso und Molli 2009), LSEQ(Nédelec u. a. 2013) sind bereits CRDTs speziell für den Bereich des kollaborativen Editierens entwickelt worden.

Der Anzahl an Quellen und Resourcen rund um CRDTs mangelt es weder an theoretischen noch an praktischen Beispielen. Während einige Arbeiten der Nutzung von CRDTs in für Offlinefunktionalität empfehlen, und die Umgebung von Offline-First Applikationen sehr den verteilten Netzwerken ähnelt, für die CRDTs konzipiert sind, sind mir keine Arbeiten über den konkreten Einsatz von CRDTs in Offline-First Applikationen bekannt.

Ziel dieser Arbeit ist es deshalb, die Umfangreich erforschten Grundlagen zum Einsatz von CRDTs in einer Offline-First Applikation umzusetzen, und zu ermitteln, welche eigenen Herausforderungen diese Umgebung aufweist.

4 Grundlagen

4.1 Definition: Offline-First

Als Offline-First wird ein Vorgehen bezeichnet, bei welchem eine Applikation den Fall der unterbrochenen Internetverbindung nicht als Ausnahme, sondern als Standard ansieht. Teilweise wird der Begriff auch anders interpretiert, im Rahmen dieser Arbeit sei Offline-First jedoch unter folgenden Kriterien zu verstehen: Die Applikation geht davon aus, dass die Verbindung mit dem Internet nach dem ersten Laden der Seite stets unterbrochen werden kann. Auch im Falle von Verbindungsproblemen, welche nicht vom Endgerät des Nutzers als solche erkannt werden, z.B. wenn das Endgerät mit dem Internet verbunden ist, aber die Route zur Website an anderer Stelle unterbrochen ist. Sämtliche Use-Cases werden so geplant, dass dem Nutzer auch offline so viele Funktionalitäten wie möglich zur Verfugung stehen.

4.2 Service Worker

Ein Service Worker ist ein sogenannter Web Worker. Web Worker sind Skripts, die unabhängig von anderen Skripts, welche auf Interaktionen mit der Benutzeroberfläche reagieren, im Hintergrund der Webanwendung laufen(WHATWG, 2019).

In traditionellen Webanwendungen werden alle benötigten Dateien, Markups, Skripte und Assets über HTTP-Requests an den Server geladen. Der Service Worker ist ein event-basiertes Skript, welches als Proxy zwischen Client und Server agiert. Damit diese Tatsache kein Sicherheitsrisiko darstellt, funktionieren Service Worker nur, wenn die Applikation HTTPS nutzt. Requests, welche üblicherweise direkt an den Server gehen würden, werden erst vom Service Worker verarbeitet. Entwickler können gezielt Entscheiden welche Netzwerk-Requests auf welche Art und Weise verarbeitet werden sollen. Mithilfe dieser Funktionalität können Entwickler sogenannte Caching-Strategien für den Service Worker implementieren, womit das Verbindungsverhalten der Applikation festgelegt werden kann(Jaiswal, 2019). Eine für Offline-First Applikationen essenzielle

Funktionalität, denn so kann garantiert werden, dass die Applikation auch ohne Internetverbindung funktionsfähig ist.

4.3 Caching

Es folgen einige grundlegende Caching-Strategien, mit Beispielen, für welche Art Requests sie sich eignen könnten.

Netzwerk dann Cache

Abbildung 4.1 beschreibt einen Request den der Service Worker erst über das Netzwerk delegiert. Falls die Kommunikation mit dem Internet unterbrochen ist, beispielsweise wenn der Nutzer offline ist, leitet der Service Worker den Request an den Cache weiter. Diese Methode eignet sich für Requests bei denen aktuelle Daten bevorzugt sind, dem Nutzer aber eine ältere Version zur Verfügung gestellt werden soll, wenn die Internetverbindung unterbrochen ist.

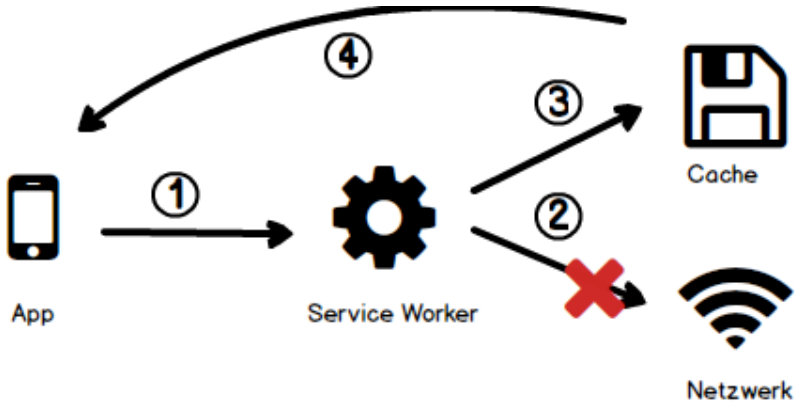

Abbildung 4.1: Netzwerk dann Cache

Cache dann Netzwerk

Wie Abbildung 4.2 zeigt, wird der Request hier zuerst an den Cache weitergeleitet. Befindet sich die angefragte Datei nicht im Cache, wird die Anfrage als HTTP-Request and den Server weitergeleitet. Dies ist die bevorzugte Strategie für die meisten Requests in Offline-First Anwendungen.(Ater, 2017, Kapitel 05).

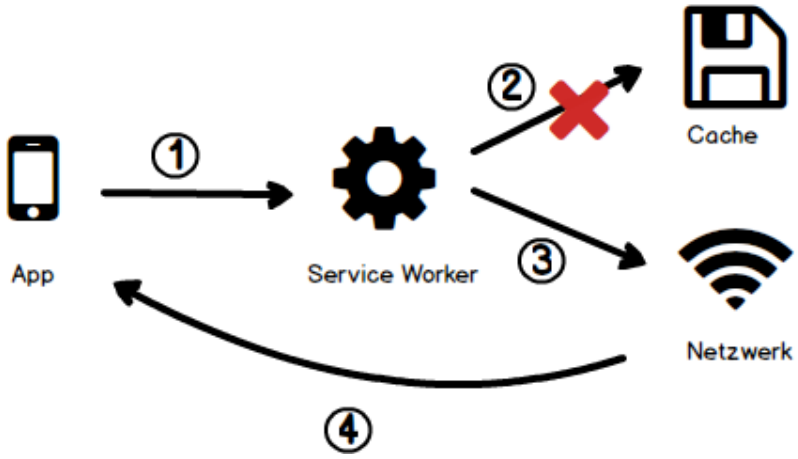

Abbildung 4.2: Caching Strategie: Cache dann Netzwerk

Nur Netzwerk

Für Aufgaben die nur Online zu erfüllen sind, eignet sich die in Abbildung 4.3 bezeichnete Strategie. Hier leitet der Service Worker den Request nur an das Netzwerk und nie an den Cache weiter. Offline-First Applikationen sollten so Konzipiert sein, dass diese Art Requests im Falle einer unterbrochenen Internetverbindung nachgeholt werden können, wenn der Nutzer wieder online ist.

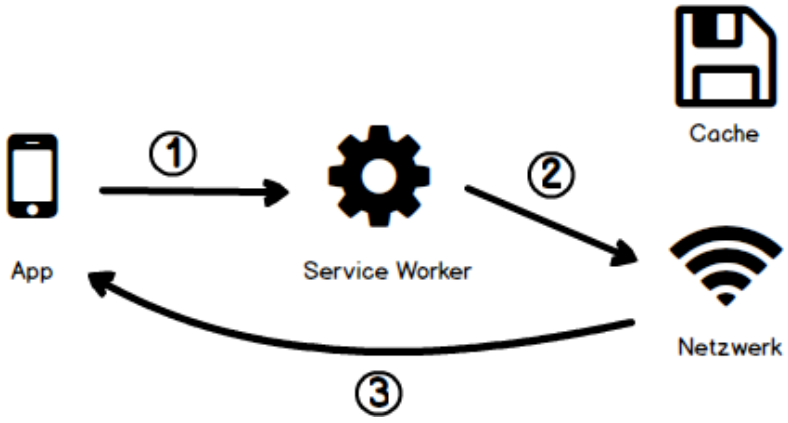

Abbildung 4.3: Nur Netzwerk

Nur Cache:

Abbildung 4.4 zeigt, wie die angefragte Ressource nur im Cache abgefragt wird. Diese Strategie ist nur dann Sinnvoll, wenn die betroffenen Daten in einem vorherigen Schritt, beispielsweise beim Installieren des Service Workers, mit gecached wurden.

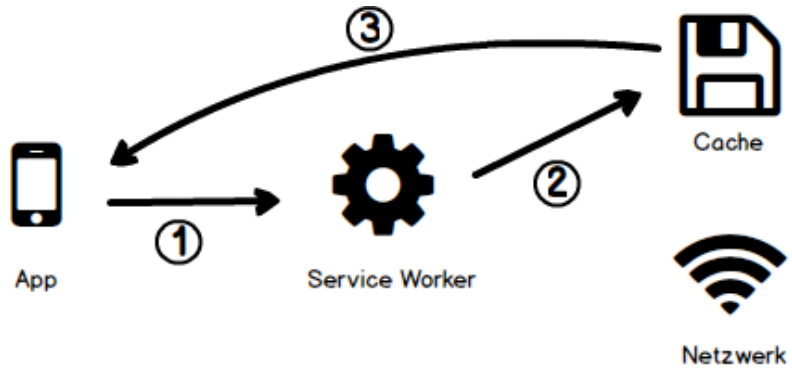

Abbildung 4.4: Nur Cache

test

4.4 Offline First Applikationen als verteiltes System

(Steen und Tanenbaum, 2016) charakterisieren verteilte Systeme wie folgt: “Ein verteiltes System ist eine Sammlung von autonomen Rechenelementen, die den Benutzern als ein einziges kohärentes System erscheint.” Weitere erwähnte Charakteristikam verteilter Systeme lassen sich wie Folgt zusammenfassen: https://medium.com/system-design-blog/key-characteristics-of-distributed-systems-781c4d92cce3 file:///home/lenni/Downloads/Distribute

4.5 Synchronisation in Offline First Applikationen

4.6 CRDTs

4.6.1 Register

4.6.2 Counter

4.6.3 Grow-Only Set

4.7 Hybrid-Logical Clocks

Literatur

[1] Marc Shapiro u. a. “Conflict-Free Replicated Data Types”. In: *Stabilization, Safety, and Security of Distributed Systems*. Hrsg. von Xavier Défago, Franck Petit und Vincent Villain. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 386–400. ISBN: 978-3-642-24550-3.

[2] M. Kleppmann und A. R. Beresford. “A Conflict-Free Replicated JSON Datatype”. In: *IEEE Transactions on Parallel and Distributed Systems* 28.10 (2017), S. 2733–2746.

[3] Gérald Oster u. a. “Data Consistency for P2P Collaborative Editing”. In: *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*. CSCW ’06. Banff, Alberta, Canada: Association for Computing Machinery, 2006, S. 259–268. ISBN: 1595932496. DOI: 10.1145/1180875.1180916. URL: https://doi.org/10.1145/1180875.1180916.

[4] Stéphane Weiss, Pascal Urso und Pascal Molli. “Logoot: A scalable optimistic replication algorithm for collaborative editing on p2p networks”. In: *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE. 2009, S. 404–412.

[5] Brice Nédelec u. a. “LSEQ: an adaptive structure for sequences in distributed collaborative editing”. In: *Proceedings of the 2013 ACM symposium on Document engineering*. 2013, S. 37–46.

[6] WHATWG. *HTML Living Standard*. 2019. URL: https://html.spec.whatwg.org/multipage/workers.html#workers (besucht am 05. 07. 2020).

[7] Aayush Jaiswal. *Understanding Service Workers and Caching Strategies*. 2019. URL: https://blog.bitsrc.io/understanding-service-workers-and-caching-strategies-a6c1e1cbde03 (besucht am 05. 07. 2020).

[8] Tal Ater. *Building Progressive Web Apps*. O’Reilly Media, Inc., 2017.

[9] Maarten Steen und Andrew S. Tanenbaum. “A Brief Introduction to Distributed Systems”. In: *Computing* 98.10 (Okt. 2016), S. 967–1009. ISSN: 0010-485X. DOI: 10.1007/s00607-016-0508-7. URL: https://doi.org/10.1007/s00607-016-0508-7.