

## Warum IndexedDB?

Aus Sicherheitsgründen darf ein Browser aus einer JavaScript Anwendung heraus nicht einfach Daten auf dem Client des Users speichern und lesen. Stattdessen stehen uns APIs zur Verfügung, die wir zu diesem Zweck verwenden können.

**Cookies**, **LocalStorage** und **SessionStorage** haben wir schon in anderen Arbeitsblättern behandelt. In diesem Arbeitsblatt geht es um die Datenbank **IndexedDB**, die in jedem Browser eingebaut ist. Diese Datenbank ist viel leistungsfähiger als **localStorage**.

Beispiel: Die maximale Datenmenge bei **localStorage** ist auf **5 MB** per App und Browser beschränkt. Die maximale Datenmenge bei IndexedDB ist hingegen riesig. Sie hängt hauptsächlich nur von 2 Dingen ab: **Browser** und **Festplattenplatz**.

Bei Chrome und den meisten Chromium-basierten Browsern können 80% des Speicherplatzes genutzt werden, davon 75% für jede Origin<sup>1</sup>. Das heißt, wenn du einen Festplattenspeicherplatz von 100 GB hast, können 80 GB für die Speicherung von Daten in der indexedDB verwendet werden und davon 60 GB für eine Origin.

Firefox erlaubt die Speicherung von 2 GB Daten pro Origin, und Safari 1 GB.

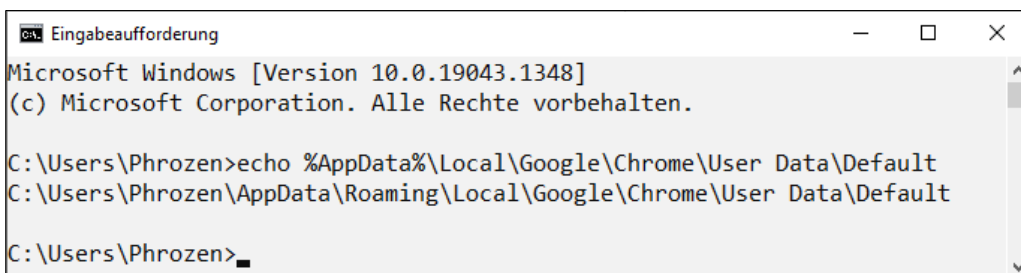
**Aufgabe 1:** Ermittle, wie viel Speicher bei dir für die IndexedDB verfügbar ist! Verwende folgendes JavaScript:

```
const estimate = await navigator.storage.estimate();  
const totalSpace = (estimate.quota / 10**9).toFixed(2); // in GB
```



Aber wo befindet sich die Datenbank?

Bei Chrome unter Windows: %AppData%\Local\Google\Chrome\User Data\Default



<sup>1</sup> Die Origin (Ursprung) von Webseiten wird durch das Schema (Protokoll), den Hostnamen (Domäne) und das Port der URL definiert, über die sie aufgerufen werden. Zwei Objekte haben nur dann dieselbe Origin, wenn das Schema, der Hostname und das Port übereinstimmen.

Robert Baumgartner

Beispiel von meinem Windows Rechner für Origin: <http://127.0.0.1:3000>:

« Benutzer > Phrozen > AppData > Local > Google > Chrome > User Data > Default > IndexedDB > http_127.0.0.1_3000.indexeddb.leveldb			
Name	Änderungsdatum	Typ	Größe
000004.log	08.12.2021 16:32	Textdokument	215 KB
000005.idb	08.12.2021 16:26	Microsoft Access ...	238 KB
CURRENT	08.12.2021 15:08	Datei	1 KB
LOCK	08.12.2021 15:06	Datei	0 KB
LOG	08.12.2021 16:26	Datei	1 KB
LOG.old	08.12.2021 15:07	OLD-Datei	0 KB
MANIFEST-000001	08.12.2021 16:26	Datei	1 KB

Bei Chrome unter MacOS (Monterey): ~/Library/Application Support/Google/Chrome/Default/IndexedDB

Beispiel von meinem Mac Rechner für Origin: <https://baumi-employees.herokuapp.com>

```

robertbaumgartner@Roberts-MacBook-Pro: ~/Library/Application Support/Google/Chrome/Default/IndexedDB/https_baumi-employees.herokuapp.com_0.i...
+ Default/IndexedDB/https_baumi-employees.herokuapp.com_0.indexeddb.leveldb ls -lisa
total 64
7821237 0 drwx-----@ 8 robertbaumgartner staff 256 12 Dez 12:11 .
1474540 0 drwx-----@ 19 robertbaumgartner staff 608 12 Dez 11:40 ..
7921308 40 -rw-----@ 1 robertbaumgartner staff 17834 12 Dez 12:12 000003.log
7921307 8 -rw-----@ 1 robertbaumgartner staff 16 12 Dez 12:11 CURRENT
7821239 0 -rw-----@ 1 robertbaumgartner staff 0 12 Dez 11:40 LOCK
7921305 8 -rw-----@ 1 robertbaumgartner staff 409 12 Dez 12:11 LOG
7878691 0 -rw-----@ 1 robertbaumgartner staff 0 12 Dez 12:01 LOG.old
7921306 8 -rw-----@ 1 robertbaumgartner staff 23 12 Dez 12:11 MANIFEST-000001
+ Default/IndexedDB/https_baumi-employees.herokuapp.com_0.indexeddb.leveldb

```

## Eigenschaften von IndexedDB

Für eine Browser Datenbank sind die Eigenschaften von IndexedDB beeindruckend. **IndexedDB ist eine transaktionsfähige Key-Value Objektdatenbank.**

Ein Key-Value Objektdatenbank gehört zum Typus der NoSQL Datenbanken und basiert auf einer Hashtabelle. Sie wird vor allem dann verwendet, wenn der gesamte Zugriff auf die Datenbank über den Key erfolgen soll. Jeder Eintrag besteht aus einem Key/Objekt Paar. Das Objekt kann von Eintrag zu Eintrag unterschiedlich sein. Die Datenbank ist also schemalos, es gibt keine Tabellenstruktur. Alle Einträge haben untereinander keine Beziehung und sind unabhängig. Also gibt es auch keine Fremdschlüssel!

Key	Value
1	{ age: 47, name: { first: 'Sarah', last: 'Bridges', }, broke: true, }
2	{ age: 26, name: { first: 'Madge', last: 'Shaw', }, }

Robert Baumgartner

Darüber hinaus besitzt IndexedDB wichtige Eigenschaften:

- IndexedDB läuft vollständig im Browser. Auf alle darin gespeicherten Daten kann unabhängig von der Netzwerkverbindung des Benutzers zugegriffen werden.
- IndexedDB folgt der **Same Origin Regel**. Mit anderen Worten: Benutzer können Daten innerhalb der gleichen Origin lesen und schreiben. Sie können nicht auf Daten zugreifen, die von einer anderen Origin in die IndexedDB geschrieben wurden.
- In IndexedDB kannst du mehrere Datenbanken erstellen (obwohl die meisten Anwendungen normalerweise nur eine Datenbank erstellen).
- Jede Datenbank kann mehrere Objectstores (aka Stores) enthalten (entspricht Tabellen in SQL Datenbanken).
- Jeder Objectstore enthält eine bestimmte Art von Daten (z.B. Benutzer, Chat-Nachrichten, Reservierungen usw.).
- Aktionen (einfügen, löschen, lesen, etc.), die du in IndexedDB durchführst, werden in Transaktionen gebündelt. IndexedDB besitzt eine asynchrone API. Das bedeutet, die Calls sind nicht blockierend, was sich positiv auf die Leistung und die Benutzerfreundlichkeit auswirkt. Ursprünglich hatte IndexedDB auch ein synchrones API, das aber irgendwann aus der Spezifikation entfernt wurde.

IndexedDB ist unter anderem wegen des fehlenden Promise-Supports etwas mühsam zu verwenden, deswegen verwenden wir die Library **idb** von **Jake Archibald**.



Hello, I'm Jake and that is my tired face.  
I'm a developer advocate for Google  
Chrome.

**idb** ist eine winzige Library (~1k), die größtenteils das IndexedDB API widerspiegelt, aber mit kleinen Verbesserungen, die einen großen Unterschied in der Benutzerfreundlichkeit machen.

Siehe: <https://github.com/jakearchibald/idb>

## Unsere Beispiel-Applikation

Im Folgenden verwenden wir eine kleine Applikation, mit der wir CRUD Operationen ausprobieren.

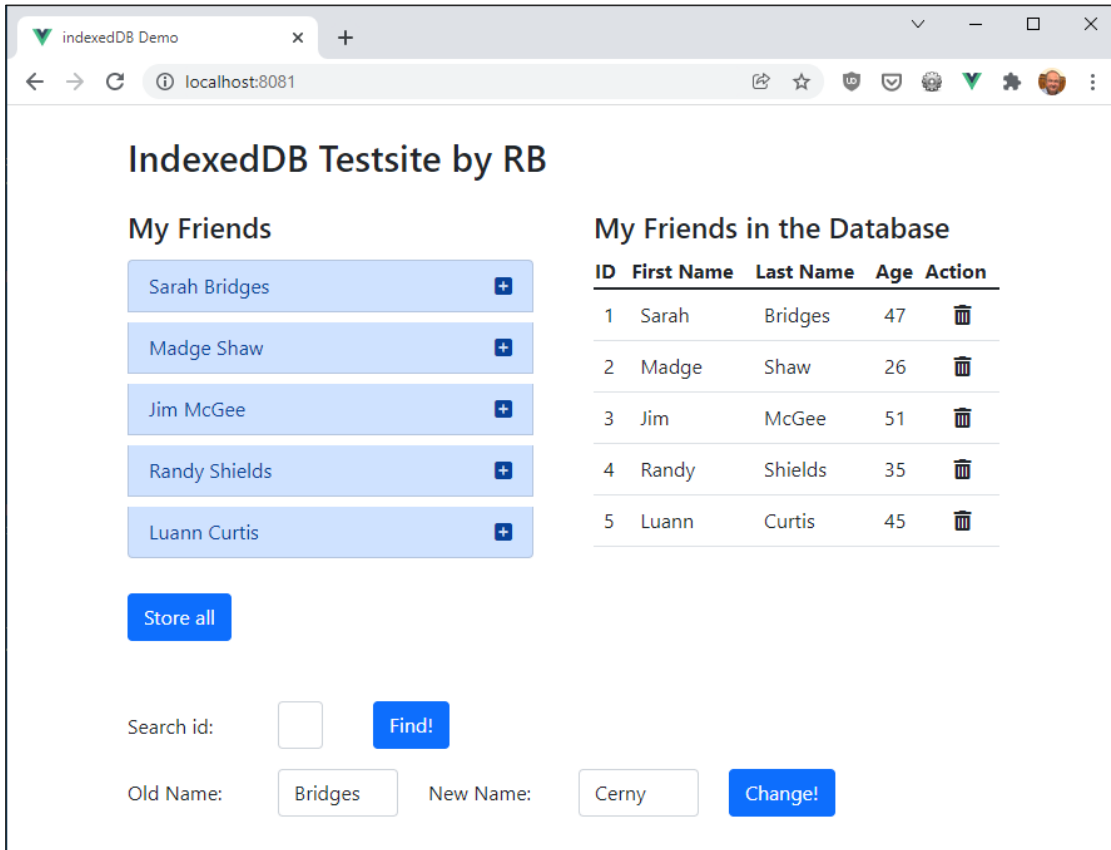
GUI: nächste Seite!

Funktionalität: Links ist eine Liste von Freunden dargestellt. Durch Klicken auf das "+" Icon wird der Freund, die Freundin in der Datenbank gespeichert und gleichzeitig in der Tabelle aufgeführt. Der Button **Store all** speichert alle Freunde und Freundinnen auf einmal in der Datenbank.

Wird in der Tabelle auf das **Delete** Icon geklickt, wird die Person aus der Datenbank gelöscht.

Es kann nach einer Person via **id** gesucht werden (Button **Find!**).

Es kann der Nachname geändert werden (Button **Change!**).



**IndexedDB Testsite by RB**

**My Friends**

- Sarah Bridges +
- Madge Shaw +
- Jim McGee +
- Randy Shields +
- Luann Curtis +

**Store all**

Search id:  **Find!**

Old Name:  New Name:  **Change!**

**My Friends in the Database**

ID	First Name	Last Name	Age	Action
1	Sarah	Bridges	47	
2	Madge	Shaw	26	
3	Jim	McGee	51	
4	Randy	Shields	35	
5	Luann	Curtis	45	

## Implementierung der Basisfunktionalität

**Aufgabe 1:** Führe die folgenden Schritte durch:

Entpacke die Datei **IndexedB START.zip**. Mache dich mit dem Code vertraut.

Die allgemeinen Schritte bei der Verwendung von IndexedDB sind:

1. Prüfe, ob IndexedDB verfügbar ist.
2. Öffne die Datenbank.
3. Erstelle einen Objektspeicher in der Datenbank.
4. Starte eine Transaktion und stelle Anfragen (z.B. Hinzufügen/Abrufen von Daten).
5. Warte, bis die Vorgänge abgeschlossen sind und verarbeite die Ergebnisse.

Diese Schritte sehen wir uns nun im Einzelnen an.

1. Ob der Browser IndexedDB zur Verfügung stellt, prüfst du mit:

```
created() {
  if (!window.indexedDB) alert('IndexedDB is not available!');
},
```

2. Installiere die Library von Jake: `npm install idb`

In **App.vue** importiere die Funktion `openDB`:

```
import { openDB } from 'idb';
```

Robert Baumgartner

Sowohl eine nicht existierende als auch eine existierende DB wird mit **openDB** geöffnet. Dabei gibst du neben dem Namen der DB, die Versionsnummer an, die du verwenden möchtest. Existiert die DB nicht, ist die Version 0, also solltest du eine Versionsnummer > 0 angeben, um eine neue DB anzulegen. Dadurch wird der Event **onupgradeneeded** erzeugt. Für diesen Event stellst du eine Methode zur Verfügung, die **upgrade** heißt. Ist die Versionsnummer gleich, gibt es keinen Event, deine Methode wird nicht aufgerufen und die DB wird einfach nur geöffnet.

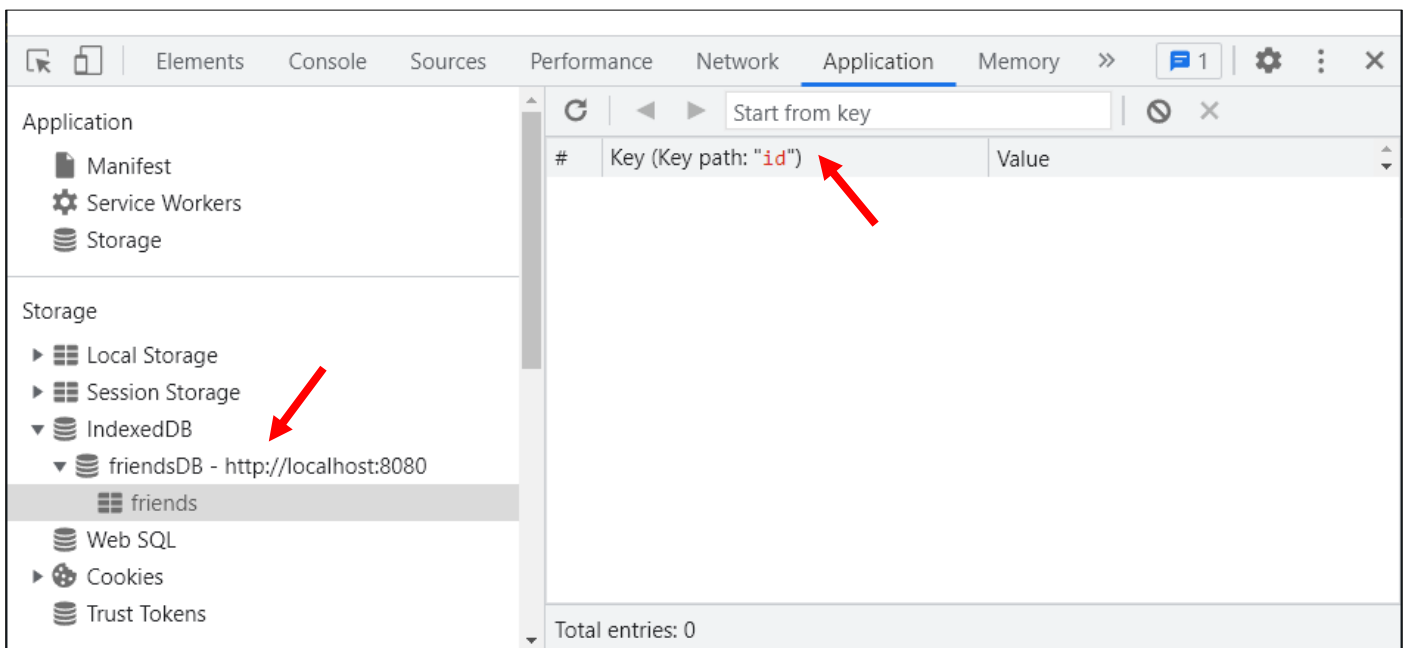
In unserem Beispiel sieht das so aus (der Name der Datenbank ist **friendsDB**):

```
this.db = await openDB('friendsDB1', 1, {
  upgrade(db, oldVersion, newVersion) {
    db.createObjectStore('friends', { keyPath: 'id' });
  },
});
```

3. In der Methode **upgrade** erzeuge den Objectstore **friends**. In diesem Store wollen wir unsere Objekte speichern. **keyPath** ist notwendig, denn es ist ja ein Key/Value Store und IndexedDB muss wissen, wo der eindeutige Key zu finden ist.

Für die anderen Events lies die Dokumentation von **idb**.

Baue das alles im **created** Hook ein und checke die DevTools:



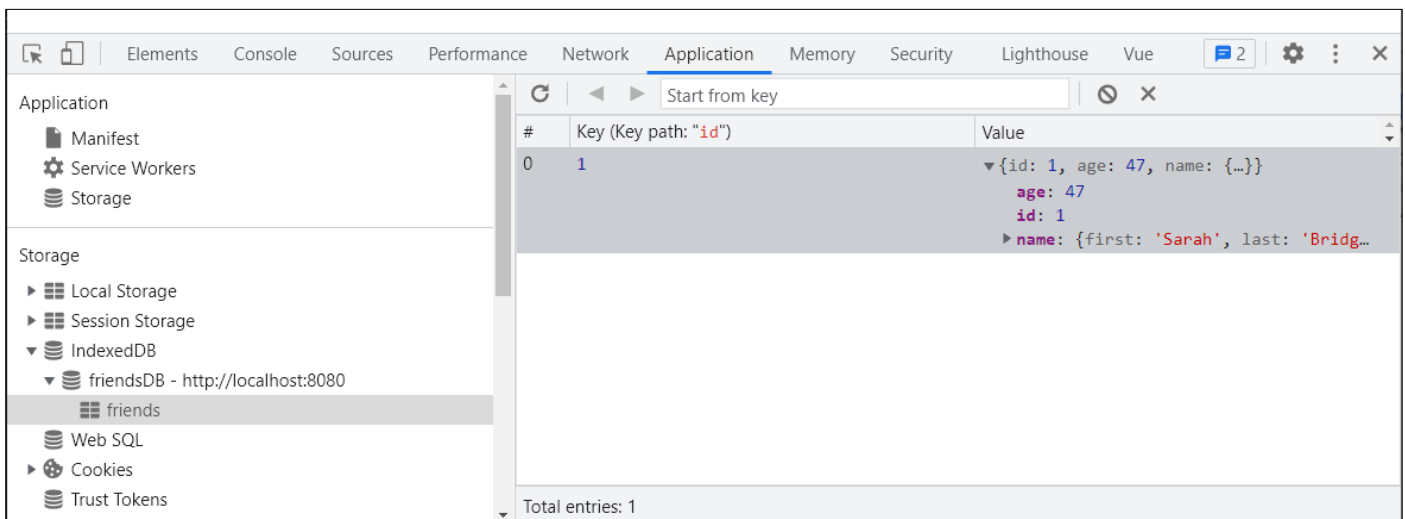
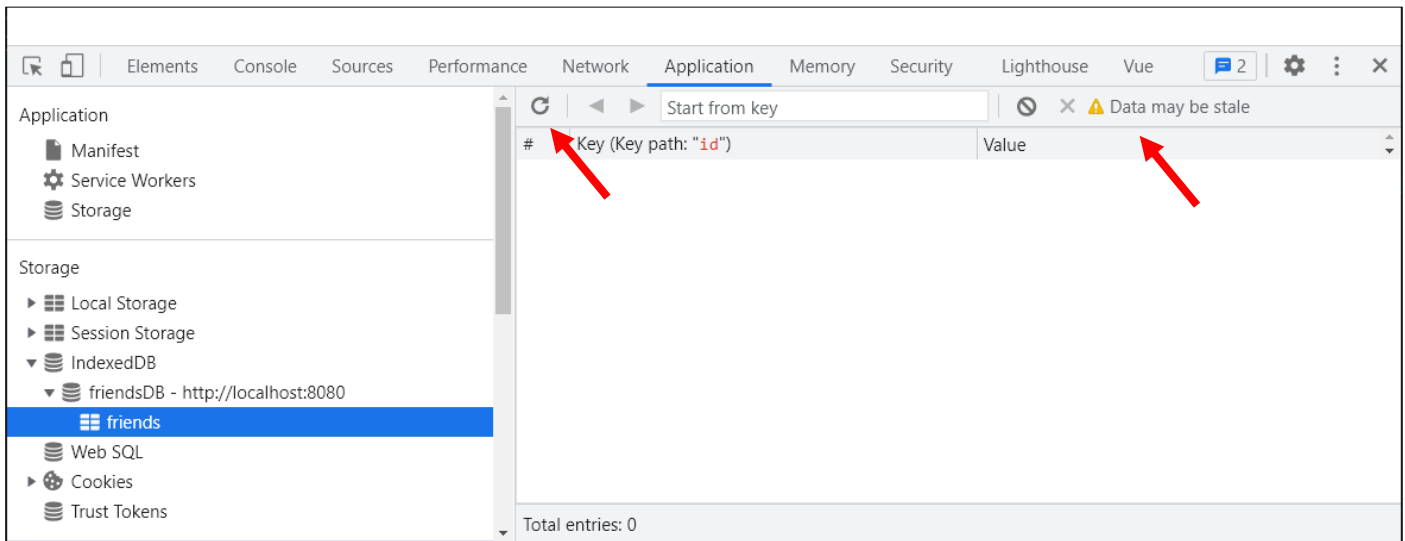
4. Klickt der Benutzer auf das "+" Icon soll der Name der Person im Objectstore **friends** gespeichert werden.

In der zu erstellenden Methode **addFriend**, die vom Klick aufgerufen wird, führe folgende Schritte durch:

```
const tx = this.db.transaction('friends', 'readwrite'); // erstelle Transaktion im R/W Mode
const store = tx.objectStore('friends'); // hole Zugriff auf den Store
await store.put(friend); // put = neu oder update, je nach dem
await tx.done; // warte auf Ende der Transaktion
```

Überprüfe das Ergebnis in den DevTools!

Vergiss nicht, die Sicht auf die DB zu refreshen! Hinweis: Data may be stale<sup>2</sup>.



Wie du siehst, hat der `put` Befehl die Philosophie von IndexedDB studiert und verinnerlicht. Gibt es den Key noch nicht, wird das Objekt in den Store eingefügt, ansonst wird das Objekt mit dem Key geändert.

Da es üblich ist, eine Transaktion für eine einzelne Aktion zu erstellen, gibt es dafür Hilfsmethoden (danke Jake):

Statt:

```
const tx = this.db.transaction('friends', 'readwrite');
const store = tx.objectStore('friends');
await store.put(friend);
await tx.done;
```

Schreibe:

Objectstore

Object

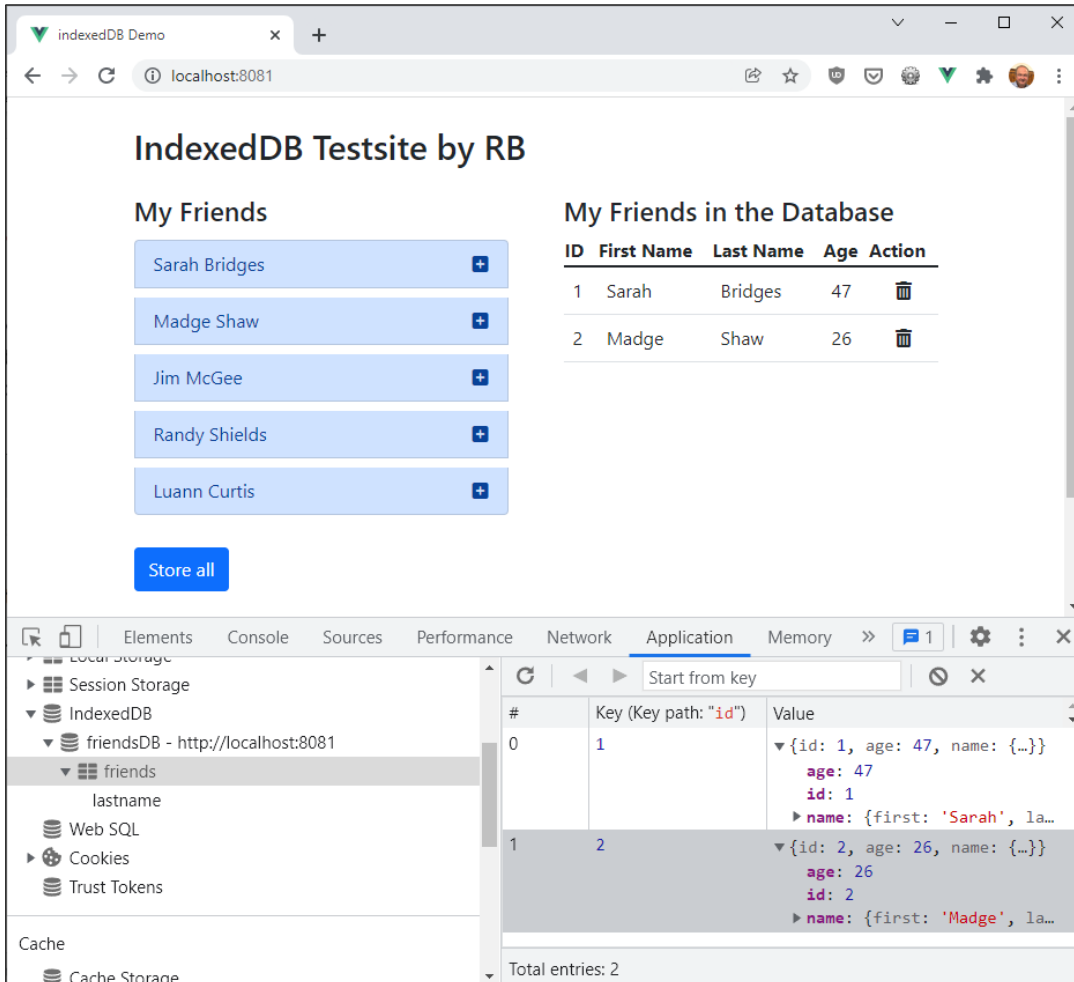
```
await this.db.put('friends', friend);
```

Weitere Shortcuts: `get`, `getKey`, `getAll`, `getAllKeys`, `count`, `put`, `add`, `delete`, and `clear`.

<sup>2</sup> stale = abgestanden, schal. Wird in der Informatik verwendet, um Daten zu bezeichnen, die nicht mehr aktuell sind.

Siehe Dokumentation: <https://github.com/jakearchibald/idb>

**Aufgabe 2:** Damit unsere Freunde in der Tabelle im GUI auftauchen, erstelle eine Methode `getStoredFriends`, welche mit dem Shortcut `getAll` alle Objekte aus dem Store holt und der Variablen `storedFriends` zuweist. Ändere die Methode `addFriend` und rufe dort `getStoredFriends` auf, sodass das GUI sofort die neuen Inhalte des Stores anzeigt.



The screenshot shows a web application titled "IndexedDB Testsite by RB" running on localhost:8081. It features two main sections: "My Friends" and "My Friends in the Database".

**My Friends**

- Sarah Bridges
- Madge Shaw
- Jim McGee
- Randy Shields
- Luann Curtis

**My Friends in the Database**

ID	First Name	Last Name	Age	Action
1	Sarah	Bridges	47	
2	Madge	Shaw	26	

Below the "My Friends" list is a "Store all" button. At the bottom, the Chrome DevTools Application tab is open, showing the IndexedDB database structure. The "friends" store contains two entries:

- Key (Key path: "id"): 1, Value: {id: 1, age: 47, name: {first: 'Sarah', last: 'Bridges'}}
- Key (Key path: "id"): 2, Value: {id: 2, age: 26, name: {first: 'Madge', last: 'Shaw'}}

Total entries: 2

**Aufgabe 3:** Ändere auch den `created` Hook, sodass die Freunde sofort aus dem Store ausgelesen werden, wenn die Seite geöffnet wird!

Implementiere `AddAll` (fügt alle Freunde auf einmal in den Store ein) und `RemoveFriend` (löscht einen Freund, eine Freundin aus dem Store).

Implementiere `findFriend` (sucht mittels `id` nach einem Eintrag im Store und zeigt den Eintrag an). Überprüfe, ob eine Exception auftritt.

## Indizes

Komplizierter wird es, wenn du nach einem Attribut suchen möchtest, das nicht der Key ist. Damit die DB den Datensatz bzw. die Datensätze schneller finden kann, wird ein Index benötigt. Diesen legst du am besten gleich beim Erstellen des Stores an.

In unserem Beispiel wollen wir den Nachnamen einer Person ändern. Dazu müssen wir erst die Person finden. Daher legen wir einen Index über das Attribut `name.last` und nennen ihn `lastname`. Außerdem geben wir an, dass das Attribut eindeutig sein muss.

```
store.createIndex('lastname', 'name.last', { unique: true });
```

Robert Baumgartner

**Aufgabe 4:** Erstelle eine Methode **updateFriend** und verknüpfe sie mit dem Button **Change!** Die Shortcut Funktionen helfen hier nichts mehr, da wir eine Transaktion benötigen, die aus zwei Schritten besteht:

- a) Finden und Holen des Objektes
- b) Ändern und erneutes Speichern des Objektes

Nach dem Starten der Transaktion (siehe obiges Transaktionsbeispiel), kannst du über den Index das Objekt bekommen.

```
const index = tx.store.index('lastname');  
const obj = await index.get(this.oldName);
```

Ändere das Objekt und speichere es wieder ab.

```
await this.db.put('friends', obj);
```

Warte auf das Ende der Transaktion und refreshe das GUI mit **getStoredFriends**.

Was aber wenn der Index auf ein nicht eindeutiges Attribut verweist, d. h., wenn mehrere Objekte den gleichen Attribut Wert haben können? In so einem Fall wird ein Cursor (Zeiger) verwendet, der ein Objekt nach dem anderen aus dem Store holt.

### Abfrage einer Key Range

Aufwendige Querys, so wie mit SQL, sind mit einer Key/Value Datenbank nicht möglich. Jedoch unterstützt IndexedDB das Abfragen einer Key Range. Zur Erinnerung: Keys können durch einen Keypath oder einen Index definiert werden.

Um einen Key Range zu definieren, gibt es ein Objekt **IDBKeyRange**, das Methoden für die Angabe des Bereiches definiert.

Siehe: <https://developer.mozilla.org/en-US/docs/Web/API/IDBKeyRange>

**Aufgabe 5:** Erweitere die bestehende Applikation, um eine Möglichkeit Freunde nach dem Alter zu filtern. Erstelle eine Methode **getFriendsWithAge**.

In der Methode verwende den Shortcut **getAllFromIndex**. Die Methode hat folgende Parameter:

1. Parameter: Storename
2. Parameter: Indexname
3. Parameter: Key Range



indexedDB Demo

localhost:8080

## IndexedDB Testsite by RB

### My Friends

Sarah Bridges

+

Madge Shaw

+

Jim McGee

+

Randy Shields

+

Luann Curtis

+

Store all

### My Friends in the Database

ID	First Name	Last Name	Age	Action
1	Sarah	Bridges	47	
2	Madge	Shaw	26	
3	Jim	McGee	51	
4	Randy	Shields	35	
5	Luann	Curtis	45	

Search id: 

Find!

 Found: Madge Shaw, 26

Old Name:  New Name: 

Change!

From Age:  To Age: 

Get!

- Found: Randy Shields, 35
- Found: Luann Curtis, 45
- Found: Sarah Bridges, 47