IndexedDB with JavaScript/HTML/CSS

Browser-Based Database Storage & CRUD Operations

- What is IndexedDB?
  - Architecture Overview
  - Web Browser as a Platform
  - Webserver to run web applications (HTML)
  - Install and Run local web server
- /indexeddb/javascript/foobar.html
  - Using IndexedDB
  - Four steps to use IndexedDB
  - Warning: No automatic IndexedDB updates
- CRUD Operations in foobar.html
  - CREATE
  - READ
  - UPDATE
  - DELETE
- Transaction Management
- foobar-crud.html
  - GUI (HTML and CSS)
  - Initialization
  - Unique ID in a Record
  - CREATE
  - READ
  - UPDATE
  - DELETE
- students.html
  - Utility functions
  - Record in the JSON format
    - Querying with Indexes
  - KeyPath

What is IndexedDB?

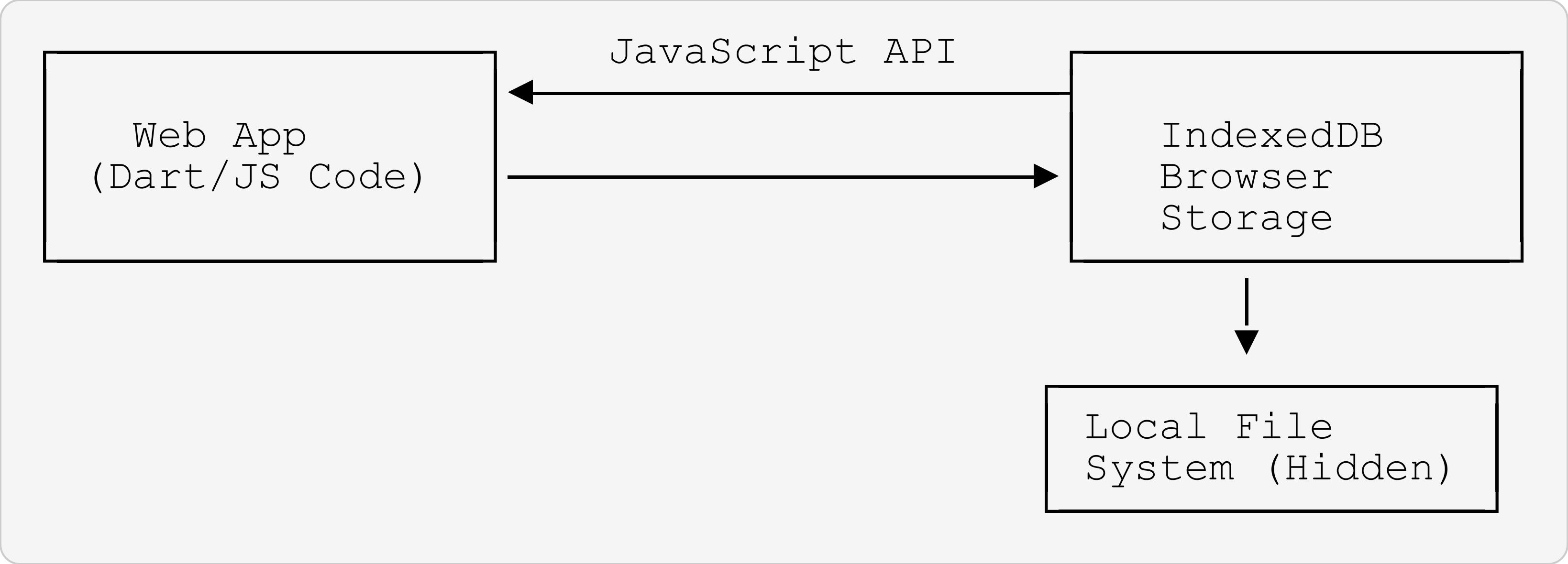
- 🌐 **Browser-native database** - No server required
- 📊 **NoSQL object database** with JavaScript objects
- 🔄 **Asynchronous API** with transaction support
- 💾 **Large storage capacity** (much more than localStorage)
- 🔒 **Same-origin policy** - Secure by default

Key Characteristics:

- Stores JavaScript objects (JSON) directly
- Supports indexes for fast queries
- Transaction-based operations
- Works offline completely
- Available in all modern browsers

Used by: Gmail, Google Drive, WhatsApp Web, Discord Web, VS Code Web

Architecture Overview



JavaScript API

Web App
(Dart/JS Code)

IndexedDB
Browser
Storage

Local File
System (Hidden)

Local IndexedDB File Location

- The exact physical location and file format depend on the browser and operating system.
  - For Safari/macOS, it is `~/Library/Safari/Databases/`.
- Each “origin” (website) gets its own folder.

Benefits:

- No network latency
- Works completely offline
- Automatic persistence
- Browser handles storage management

Web Browser as a Platform

- In this environment, A HTML file (with JavaScript and CSS) is one GUI application (Single Page Application).
- We can make any application using JavaScript.
- Most web browsers support developer tools to debug the web application.

Opening the developer mode

- In Chrome, click Alt-Cmd-I (Mac) or Alt-Ctl-I (PC) to open the Developer Tools.
- Or use the menu: View -> Developer -> Developer Tools
- You can open a terminal or see the IndexeDB storage.

Webserver to run web applications (HTML)

- To use IndexedDB, we should access the web applications using http:// protocol.
    - When we open the HTML using web browser, we use file:// protocol.
- To use the http:// protocol, we should use a local web server.

Install and Run local web server

- Install VSCode Live Server Extension.
- Click the Go Live button at the right buttom.
  - Check your web browser opens
  - Open HTML from the browser to run web applications.

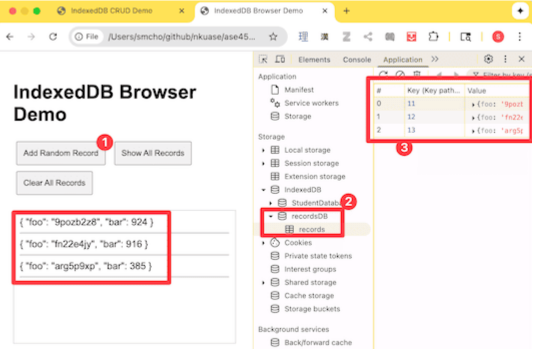
/indexeddb/javascript/foobar.html

1. Open the "indexeddb/javascript" directory in VSCode and click `Go Live` button.
2. Open the "foobar.html".
3. Open Developer Tools.
4. Click the "Application Tab".

Using IndexedDB

1. Click "Add Records" button (HTML).
2. Click the created recordsDB (Web Browser).

- Check the Records.



{ "foo": "9pozb2z8", "bar": 924 }

{ "foo": "fn22e4jy", "bar": 916 }

{ "foo": "arg5p9xp", "bar": 385 }

#	Key (Key path...	Value
0	11	▸{foo: '9pozt
1	12	▸{foo: 'fn22e
2	13	▸{foo: 'arg5p

Four steps to use IndexedDB

- Open Database
- Create Object Store (collection) in the Database
- Create Transaction (we can get the store from the transaction).
- Create a Record in JSON format

1. Open Database

```javascript
let db;
const dbName = 'recordsDB';
const request = indexedDB.open(dbName, 1);
request.onerror = (event) => {
    console.error("Database error:", event.target.error);
};
request.onsuccess = (event) => {
    db = event.target.result;
    console.log("Database opened successfully");
};
```

2. Create Object Store (≈ Table or Collection)

```
const storeName = 'records';
db.createObjectStore(
  storeName,
  { keyPath: 'id', autoIncrement: true });
```

- This is equivalent to collection.
- We need to specify keyPath.

3. Create Transactions

```
const transaction = db.transaction(
    [storeName], 'readwrite');
const store = transaction.objectStore(storeName);
```

- Ensure data consistency
- We can get store from the transaction.

4. Create a Record in JSON format

- All the Record is stored and shared in JSON.

```
const record = {
  foo: "Hello",
  bar: 100
};
store.add(record);
```

Warning: No automatic IndexedDB updates

- IndexedDB does not automatically update the structure of an object store once it has been created.
- Even if you modify your code, any existing database will keep its original structure.

Three ways to solve this issue

- Update DB version `const DB_VERSION = 2`.
- Developer Tools → Application → Storage → IndexedDB → FooBar2 (this database)
  - Click "Delete" button.
- Run JavaScriptCode

```
indexedDB.deleteDatabase('YOUR_DB');
```

CRUD Operations in foobar.html

- CREATE: store.add(data)
- READ: store.get(studentId)
- UPDATE:
- DELETE: store.clear()

Initialize Database

```
let db;
const dbName = 'recordsDB';
const storeName = 'records';
const request = indexedDB.open(dbName, 1);
```

- The value `1` sets the version of the IndexedDB database.
- If the version is higher than the current one, `onupgradeneeded` runs to update the schema; otherwise, the database just opens.
- If the database doesn’t exist, version 1 is created.

```
request.onerror = (event)
  => {
    console.error("Database error:",
      event.target.error);
};
request.onsuccess = (event)
  => {
    db = event.target.result;
    console.log("Database opened successfully");
};
request.onupgradeneeded = (event)
  => {
    const db = event.target.result;
    if (!db.objectStoreNames.contains(storeName)) {
        db.createObjectStore(storeName,
          { keyPath: 'id',
            autoIncrement: true });
    }
};
```

Helper function

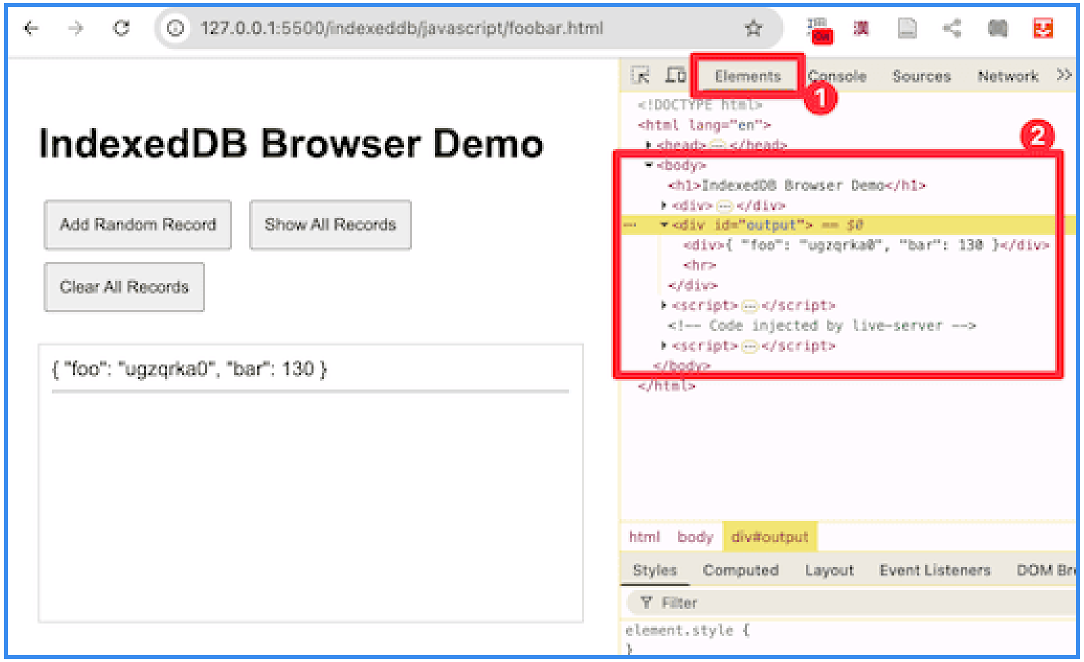
- HTML: Placeholder

```
<div id="output"></div>
```

- JavaScript: Display information at the placeholder

```
// Helper function to show output
function showOutput(message) {
    const output =
      document.getElementById('output');
    output.innerHTML +=
      `<div>${JSON.stringify(message, null, 2)}</div><hr>`;
}
```

- In the Developer Tools, click the "Elements" tab.
- We can see the JavaScript updates the HTML element.



{ "foo": "ugzqrka0", "bar": 130 }

```
<!DOCTYPE html>
<html lang="en">
 ▸ <head> … </head>
 ▾ <body>
    <h1>IndexedDB Browser Demo</h1>
  ▸ <div> … </div>
  ▾ <div id="output"> == $0
      <div>{ "foo": "ugzqrka0", "bar": 130 }</div>
      <hr>
    </div>
  ▸ <script> … </script>
    <!-- Code injected by live-server -->
  ▸ <script> … </script>
  </body>
</html>
```

```
element.style {
}
```

Buttons

```
<div>
    <button onclick="addRecord()">Add Random Record</button>
    <button onclick="getAllRecords()">Show All Records</button>
    <button onclick="clearAllRecords()">Clear All Records</button>
    <button onclick="updateLastRecord()">Update Last Record</button>
</div>
```

output

```
#output {
  margin-top: 20px; padding: 10px;
  border: 1px solid #ccc; min-height: 200px;
}
<div id="output"></div>
```

CREATE

Creating a Single Record

```
const request = store.add(data);
```

```
function addRecord() {
    const data = {
        foo: Math.random().toString(36).substring(2, 10),
        bar: Math.floor(Math.random() * 1000)
    };
    const transaction = db.transaction([storeName], 'readwrite');
    const store = transaction.objectStore(storeName);
    const request = store.add(data);
    request.onsuccess = () => {
        showOutput(data);
    };
}
```

READ

Reading All Records

```
store.getAll();
```

```
function getAllRecords() {
    const transaction = db.transaction([storeName], 'readonly');
    const store = transaction.objectStore(storeName);
    const request = store.getAll();
    request.onsuccess = () => {
        showOutput({ data: request.result });
    };
}
```

UPDATE

Accessing the record using a cursor

```
// Open a cursor in reverse order
// to get the last (highest id) record
store.openCursor(null, 'prev');
request.onsuccess = (event) => {
  const cursor = event.target.result;
  if (cursor) {
    const record = cursor.value;
    record.foo = `P-${record.foo}`;
    record.bar += 1;
```

Update the Record

```
cursor.update(record)
```

```
function updateLastRecord() {
    const transaction = db.transaction([storeName], 'readwrite');
    const store = transaction.objectStore(storeName);
    const request = store.openCursor(null, 'prev');
    request.onsuccess = (event) => {
        const cursor = event.target.result;
        if (cursor) {
            const record = cursor.value;
            record.foo = `P-${record.foo}`;
            record.bar += 1;
            // Update the record
            const updateRequest = cursor.update(record);
            updateRequest.onsuccess = () => {
                showOutput({ message: 'Record updated', updated: record });
            };
        } else {
            showOutput({ message: 'No records to update' });
        }
    };
}
```

DELETE

```
store.clear();
```

```
function clearAllRecords() {
    const transaction = db.transaction([storeName], 'readwrite');
    const store = transaction.objectStore(storeName);
    const request = store.clear();
    request.onsuccess = () => {
        showOutput({ message: 'All records cleared' });
    };
}
```

Transaction Management

Transaction Types

```
// Read-only transaction (for SELECT operations)
const readTransaction =
  db.transaction(['students'], 'readonly');

// Read-write transaction (for INSERT, UPDATE, DELETE)
const writeTransaction =
  db.transaction(['students'], 'readwrite');
```

foobar-crud.html

- This web application uses the same foobar record.
- However, it supports better GUI and better CRUD service functions.

GUI (HTML and CSS)

```html
<h1>IndexedDB CRUD Operations Demo</h1>

<div class="container section">
    <h2>1. CREATE - Add Data</h2>
    <p>Add the sample data to the database:</p>
    <button onclick="createData()">
        Create Sample Data
    </button>
    <div class="output"
         id="createOutput">
           Click "Create Sample Data" to add data to IndexedDB...
    </div>
</div>
```

```css
.output {
    background: #f8f9fa;
    ...
    min-height: 50px;
}
```

JSON.stringify

This function transforms JSON object into JSON string.

```
const data = { id: 1, foo: 'hmeiijfc', bar: 580 };
// null => no filtering, 2 spaces
console.log(JSON.stringify(data, null, 2));
```

```
{
  "id": 1,
  "foo": "hmeiijfc",
  "bar": 580
}
```

Initialization

```
// Database configuration
const DB_NAME = 'Foobar2';
const DB_VERSION = 1;
const STORE_NAME = 'dataStore';

let db;
// Initialize the database when page oads
window.onload = function() {
    initDatabase();
};
```

Display function.

```
function updateOutput(elementId, message) {
  document.getElementById(elementId).textContent
    = message;
}
```

Open DB using "indexedDB.open"

- When there is no DB or the DB should be upgraded, a new DB is created.

```
function initDatabase() {
    const request = indexedDB.open(DB_NAME, DB_VERSION);
    // This event is only triggered
    // when the database is created or upgraded
    request.onupgradeneeded = function(event) {
        ...
        db.createObjectStore('myStore', ... )
    };
}
```

Unique ID in a Record

- Use keyPath or autoIncrement to define a unique primary key.
- autoIncrement generates a new and unique numeric ID for each record.

```
request.onupgradeneeded = function(event) {
  const db = event.target.result;
  // Create an object store with 'id'
  // as the keyPath and enable autoIncrement
  db.createObjectStore('myStore', { autoIncrement: true });
};
```

To manage the records, do not add primary key (id) so IndexedDB automatically generates one.

```
const data = {
    // id: 1,
    foo: 'hmeiijfc',
    bar: 580
};
const transaction = db.transaction([STORE_NAME], 'readwrite');
const objectStore = transaction.objectStore(STORE_NAME);
const addRequest = objectStore.add(data);
```

CREATE

- Check if DB is valid reference
- Transaction, Store, and use `add` method.

```
function createData() {
    if (!db) {
        updateOutput('createOutput', 'Database not initialized');
        return;
    }
    const transaction = db.transaction([STORE_NAME], 'readwrite');
    const objectStore = transaction.objectStore(STORE_NAME);

    // Our sample data with a unique ID
    const data = { ... }

    const request = objectStore.add(data);
```

READ

Create transaction with "readonly".

```
const transaction = db.transaction([STORE_NAME], 'readonly');
```

Retrieve data from the database

```
function readData() {
    if (!db) { ... }
    const transaction = db.transaction([STORE_NAME], 'readonly');
    // Get data with ID = 1
    const request = objectStore.get(1);
}
```

Get all data in the store

- Step1: get keys and record

```
function readAllData() {
    ...
    // Get all data and all keys simultaneously
    const dataRequest = objectStore.getAll();
    const keysRequest = objectStore.getAllKeys();
    ...
}
```

When the getAll() and getAllKeys() are finished, checkComplete() is invoked.

```
dataRequest.onsuccess = function event) {
    dataResults = event.target.result;
    checkComplete();
};
keysRequest.onsuccess = function event) {
    keyResults = event.target.result;
    checkComplete();
};
```

Each operation increases completed variable by 1, and when both of them are completed, we can combine the arrays.

```
function checkComplete() {
    completed++;
    if (completed === 2) {
      ...
    }
}
```

- Step2: combine them when display

For each dataResults with autogerated index, we prepend "id: keyResuls[index]".

```
const combinedResults = ataResults.map(
  (data, index) => ({
    id: keyResults[index],
    ...data
}));
```

UPDATE

We get the users' input from HTML elements.

```
const newFoo =
  document.etElementById('newFoo').value;
const newBar =
  parseInt(document.etElementById('newBar').value);
const updateId =
  parseInt(document.etElementById('updateId').value);
```

We update the record with the ID using the updatedData.

```
// Updated data
const updatedData = {
    foo: newFoo,
    bar: newBar
};
const request = objectStore.put(
  updatedData, updateId);
```

DELETE

Remove specific data

- We get the ID of the record to delete from users' input.

```
function deleteData() {
    if (!db) { ... }
    const deleteId =
      parseInt(document.getElementById('deleteId').value);

    ...
    const request = objectStore.delete(deleteId);
}
```

DELETE ALL - Clear entire database

- We can use `objectrStore.clear()` to clear the DB.
- We can use `indexedDB.deleteDatabase('YOUR_DB')` to delete the DB.

```
function clearDatabase() {
    if (!db) { ... } ...
  const request = objectStore.clear();
}
```

students.html

- We implement the Student DB using IndexedDB.

```
let db;
const dbName = 'UniversityDB';
const dbVersion = 1;
const storeName = 'students';
```

Utility functions

```
function log(message) {
    const output = document.getElementById('output');
    const timestamp = new Date().toLocaleTimeString();
    output.textContent += `[${timestamp}] ${message}\n`;
    output.scrollTop = output.scrollHeight;
}
```

Record in the JSON format

```
const student = {
    name: name,
    major: major,
    age: age,
    createdAt: new Date().toISOString()
};
```

- We have name, major, and age: we can make index for each of them to speedup the search.

Querying with Indexes

JavaScript Index Creation

```
// During database upgrade
request.onupgradeneeded = function(event) {
    const db = event.target.result;
    const objectStore = db.createObjectStore(...);

    // Create indexes for fast searching
    objectStore.createIndex(
      'nameIndex', 'name', { unique: false });
    objectStore.createIndex('majorIndex',
      'major', { unique: false });
    objectStore.createIndex(
      'ageIndex', 'age', { unique: false });
};
```

- We can use the index to search and get results.

```
function searchByName(name) {
    const transaction = ...
    const objectStore = ...
    const index = objectStore.index('nameIndex');
    const request = index.getAll(name);
    ...
}
function getStudentsByMajor(major) {
    const transaction = ...
    const objectStore = ...
    const index = objectStore.index('majorIndex');
    const request = index.getAll(major);
    ...
}
```

KeyPath

- We didn't use keyPath for the `foobar-crud.html` for ID.
  - The key is separate from the stored object
  - IndexedDB automatically generates sequential numeric keys
  - You store just the data object, and the key is handled externally

```
const objectStore = db.reateObjectStore(STORE_NAME, {
    autoIncrement: true
});
```

```
// Storing data
const transaction = db.transaction(['students'], 'readwrite');
const store = transaction.objectStore('students');

// Key will be auto-generated (1, 2, 3, etc.)
store.add({ name: 'John', age: 20, major: 'CS' });
store.add({ name: 'Jane', age: 22, major: 'Math' });

// Retrieving data
store.get(1).onsuccess = (event) => {
  // { name: 'John', age: 20, major: 'CS' }
    console.log(event.target.result);
};
```

- In this example, we use keyPath.
  - The key is a property within the stored object
  - The object must have (or will get) an id property
  - The entire object structure includes the key

```
const objectStore = db.createObjectStore(storeName,
    keyPath: 'id',
    autoIncrement: true
});
```

```javascript
// Storing data
const transaction = db.transaction(['students'], 'readwrite');
const store = transaction.objectStore('students');

// The 'id' will be auto-generated and added to the object
store.add({ name: 'John', age: 20, major: 'CS' });
// Stored as: { id: 1, name: 'John', age: 20, major: 'CS' }

store.add({ name: 'Jane', age: 22, major: 'Math' });
// Stored as: { id: 2, name: 'Jane', age: 22, major: 'Math' }

// Retrieving data
store.get(1).onsuccess = (event) => {
// { id: 1, name: 'John', age: 20, major: 'CS' }
    console.log(event.target.result);
};
```